



25.06.25
09h à 19h



4 place Jussieu
75005 Paris

RUST PARIS 2025



Langage de programmation
sûr et performant

En partenariat avec



Mot d'accueil

Florence Blazy

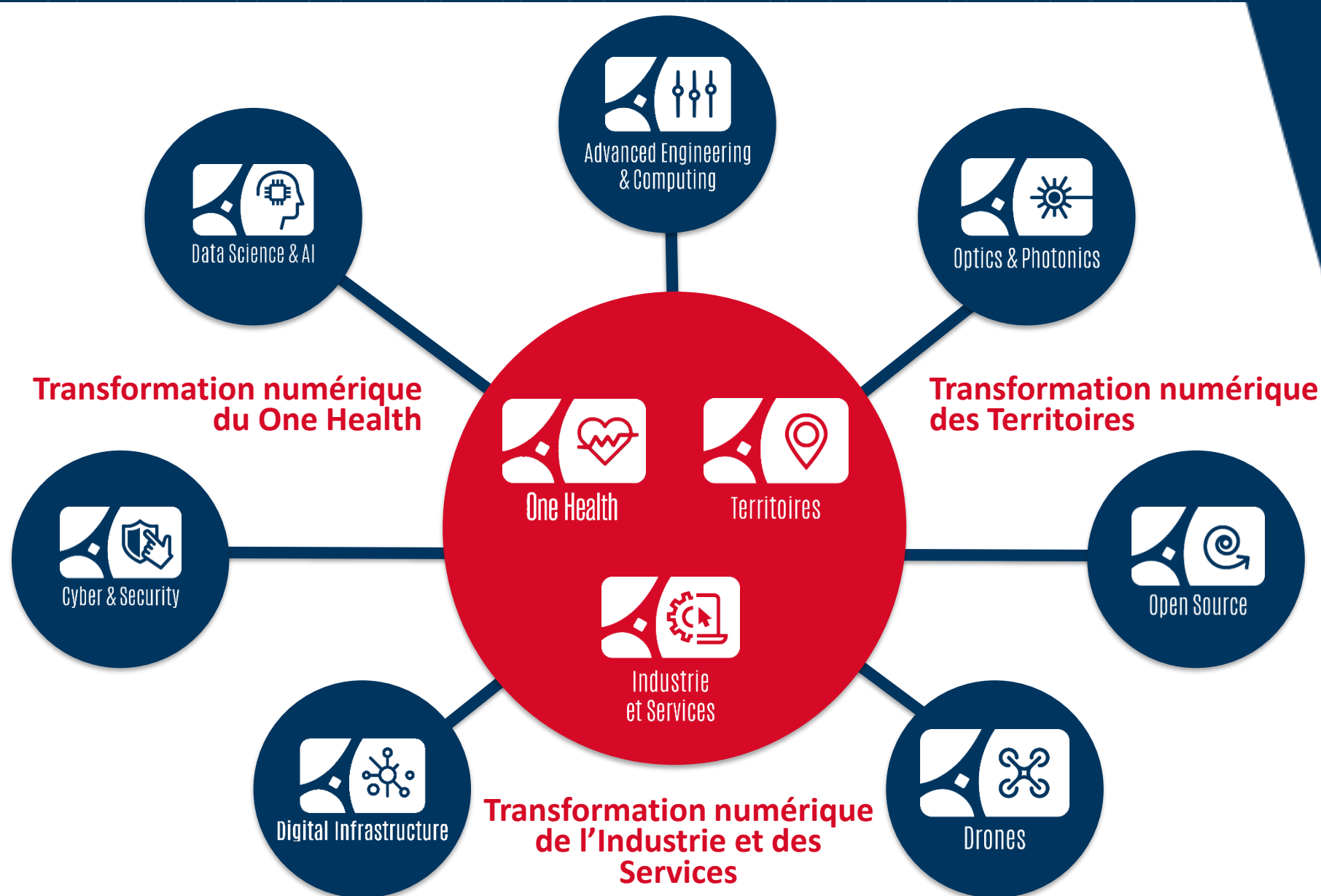
Coordinatrice du Hub Open Source

Systematic Paris-Region

Un écosystème d'excellence de + de 900 membres



Systematic : organisation



**7 Hubs
technologiques
+
3 Enjeux
transverses
économiques &
sociétaux**

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



15 ans d'existence



150 membres :
PME, ETI,
Grands groupes,
académiques

Membres du Hub Open Source



Le Hub Open Source



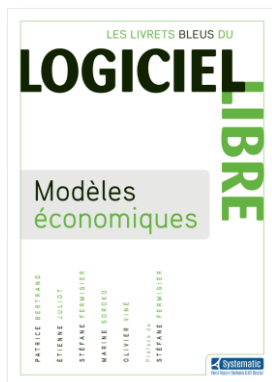
Open Source Experience 2025



Livret bleu open source pour l'IoT



Livret Bleu du Logiciel Libre : Les modèles économiques



Ateliers Alternatives open source à VMware



Atelier sur les enjeux juridiques de l'open source



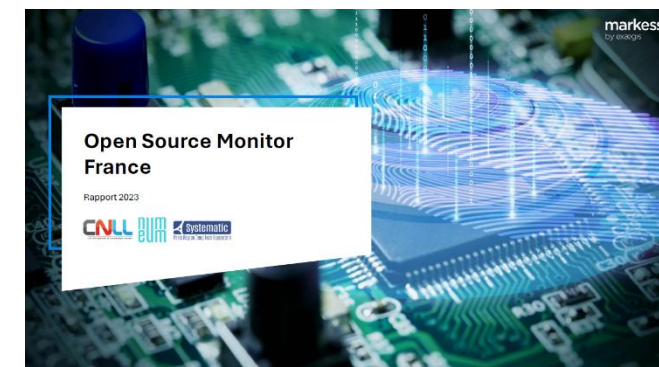
Sécurité de la supply chain des logiciels open source



Cycle de conférences « Open Source Business Lab »



Étude annuelle marché/filière Open Source



PyParis



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



GOLD



SILVER

AdaCore

CÉNOTÉLIE



RUST PARIS 2025

Notre partenaire



Emmanuel Chailloux
Professeur émérite
LIP6, IRILL, Sorbonne Université

Membre du comité de pilotage du
Hub Open Source de Systematic



RUST PARIS 2025

Sponsor Gold



OCaml PRO

Fabrice Le Fessant
CTO, OcamlPro

Membre du comité de pilotage du
Hub Open Source de Systematic



OCaml PRO

The Art of Programming



Experts en langages de programmation depuis 2011

Un ADN recherche

80% de nos ingénieurs *ont une thèse* (mais 95% sont quand même sympas)

Des projets innovants et reconnus (ICO Tezos 250M€, Kopek, spin-off après un AP du CNC)

Appliqué aux problèmes de l'industrie

- Compilateurs, interpréteurs, analyseurs et outils de développement
- Conception de langages métiers
- Regain de maîtrise sur du code legacy

Sur des technologies variées

Wasm OCaml Rust COBOL SQL JS Solidity C Méthodes formelles

Depuis 2020, OCamlPro fait aussi du Rust

Accompagner vers Rust

- Formations
- *Senior as a Service*
- Prototypage
- Accompagnement à la migration

Construire en Rust

- DSLs
- Générateur de code depuis SysML
- Serveur de transactions
- Micro-services
- Contributions au compilateur

Rencontrez-nous !



<https://ocamlpro.com>



RUST PARIS 2025

Sponsor Gold

THALES
Building a future we can all trust

Sébastien Lejeune
Open Source Advocate @Thales

Membre du comité de pilotage du
Hub Open Source de Systematic





THALES – Gold Sponsor

www.thalesgroup.com



Pourquoi Rust?

Thales est un leader mondial en sécurité et biométrie.

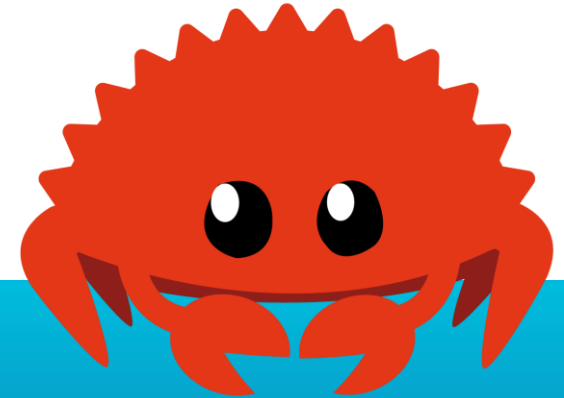
Rust est fiable et memory safe

90% des produits Thales contiennent du logiciel.

Rust est de plus en plus adopté

Beaucoup de code legacy en ADA

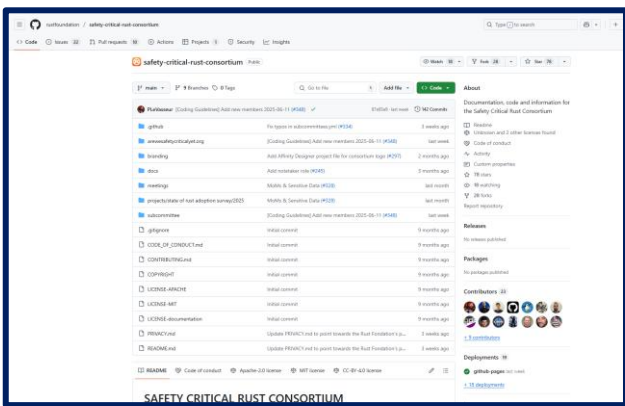
Plus facile de recruter de jeunes devs Rust



Initiatives Thales autour de Rust

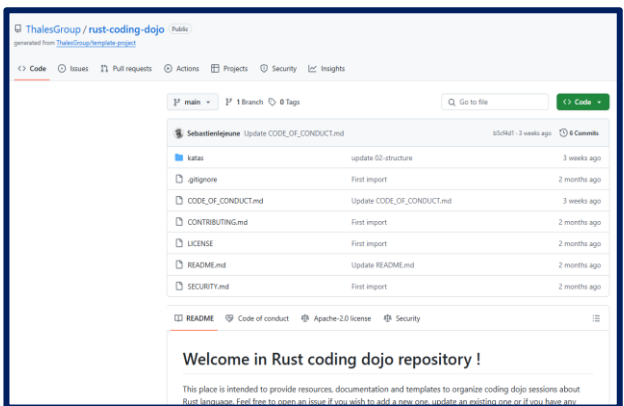
1

Rust Critical Consortium



2

Dissémination active



3

Événements





Contact

Sébastien LEJEUNE

Thales Open Source Advocate

 06 20 50 40 02

 sebastien.lejeune@thalesgroup.com

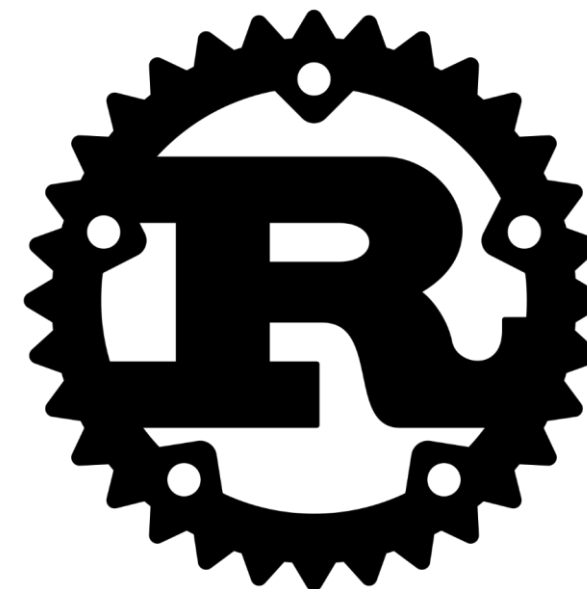


Merci

www.thalesgroup.com

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

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



Nos sponsors :



■ Merci au comité de programme :



Florian BENNETOT
DevOps Engineer
Parsec



Florence BLAZY
Coordinatrice du
Hub Open Source
Systematic



Arthur CARCANO
Ingénieur R&D Rust
Docteur en mathématiques et informatique
Ocamlpro



Emmanuel CHAILLOUX
Professeur émérite
LIP6, IRILL,
Sorbonne Université



Thierry LEBLOND
Co-Founder & CEO



Fabrice LE FESSANT
Président Fondateur
Ocamlpro



Sébastien LEJEUNE
Open Source Advocate
Thales



Laurent WOUTERS
CEO
Cénotélie

Nos sponsors :



Agenda – Amphithéâtre 25

Matin

- **09h45 – 10h30** : Keynote : A Tale of Oxidation and Witchcraft
Jonathan Brossard | CNAM et Moabi
- **10h35 – 11h05** : Rust pour le trading haute fréquence
Nicolas Kateb | XBTO
- **11h25 – 11h55** : Rust et WebAssembly, levier de frugalité et de robustesse pour les traitements de données
Reyyan Tekin | Thales
- **12h00 – 12h30** : Rust & WebAssembly : Sécurité et Portabilité du Cloud aux Enclaves Etienne Bossé & Jean-Jacques Lafay | Secretarium France

Nos sponsors :



Agenda – Amphithéâtre 25

Après-midi

- **14h00 – 14h30** : Rust dans l'automobile : efficacité, sécurité, sûreté, et attractivité
Jean-François Bortolotti | Ampere (Renault Group)
- **14h35 – 15h05** : Rust dans l'embarqué : retours sur le développement d'une borne de recharge pour véhicules électriques
Hugo Mercier | IoT.bzh
- **15h10 – 15h40** : Ariel OS: a Rust embedded OS standing on the shoulders of giants
Romain Fouquet, Ingénieur de recherche | Inria Saclay
- **16h00 – 16h30** : Une démarche de certification industrielle avec Ferrocene et TrustInSoft Analyzer
Aïssata Maïga, Ingénieure logiciel | Ferrous Systems
Sasha Pourcelot, Ingénieure Rust & OCaml | TrustInSoft
- **16h35 – 17h05** : Vérification Formelle de Rust pour Ethereum
Guillaume Claret, Fondateur / Ingénieur en méthodes formelles | Formal Land

Nos sponsors :



Agenda – Amphithéâtre 25

Pauses

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



Nos sponsors :



RUST PARIS 2025



Sébastien Lejeune

Open Source Advocate @Thales

Membre du comité de pilotage du
Hub Open Source de Systematic



OCaml PRO

THALES
Building a future we can all trust

AdaCore

CÉNOTÉLIE

 zetta
scale



XBTO

Jonathan Brossard



Enseignant-Chercheur, **CNAM**
CTO, **MOABI**

■ A Tale of Oxidation and Witchcraft



le cnam

A Tale of Oxidation and Witchcraft

Jonathan Brossard
June 2025, Paris, France



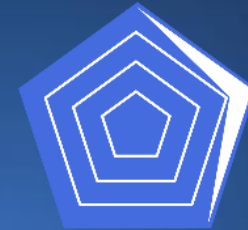
Who am I ?

Security Researcher
ORCID: 009-0004-8031-1438

CTO @ MOABI

Previously: Principal Engineer & Director
Security at Salesforce
(San Francisco)

My work is partially sponsored by:



MOABI

salesforce

bpifrance

le cnam

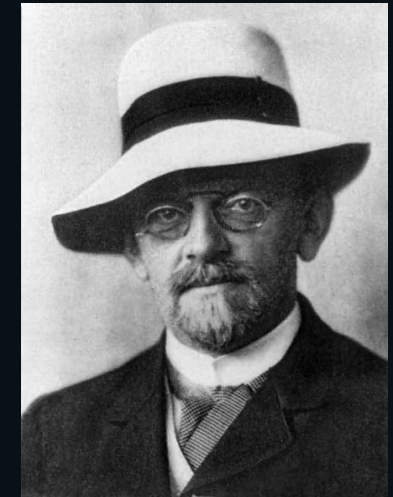


A Malicious Counter History of the Mathematics of the XXth Century

Once Upon a Time...

2nd Congress of Mathematicians (Paris, 1900):

- 23 Problems (Matrixes of Lie, Zeros of the Riemann's Zeta function, axiomatization of Physics...)
- Secret goal : Axiomatization of all branches of Mathematics
- Then deduce all possible theorems from those axioms



David Hilbert

Had 69 PhD students, including John von Neumann

HILBERT, David. Mathematische probleme. *Nachrichten von der Koniglichen Gesellschaft der Wissenschaften zu Gottingen*, 1900.

And then...

PhD thesis in 1931 (at 25 years old):

First **incompleteness** theorem:

"Any consistent formal system F within which a certain amount of elementary arithmetic can be carried out is incomplete; i.e. there are statements of the language of F which can neither be proved nor disproved in F ." (Raatikainen 2020)

Those statements are named **undecidable**.



Kurt Gödel

GÖDEL, Kurt. On formally undecidable propositions of principia mathematica and related systems i 1 (1931).
In : Godel's Theorem in Focus. Routledge, 2012. p. 17-47.

Algorithmic equivalents

PhD thesis (1953):

Any nontrivial semantic property about the language recognized by a Turing machine is **undecidable**.

Corollary:

- Deciding if a piece of software contains malware is **undecidable**
- Deciding if a piece of software contains vulnerabilities is **undecidable**



Henry Gordon Rice

RICE, Henry Gordon. Classes of recursively enumerable sets and their decision problems. *Transactions of the American Mathematical society*, 1953, vol. 74, no 2, p. 358-366.

Modern Equivalent

AGI (2020) is going to solve all our problems and magically solve undecidable problems.



Sam Altman

Modern Equivalent

AGI (2020) is going to solve all our problems and magically solve undecidable problems.



Sam Altman

Rust from a Security Perspective

Memory Corruptions

Memory Corruption vulnerabilities count for about 70% of exploitable vulnerabilities in modern software

Source: Google Project Zero, Microsoft MSRC)

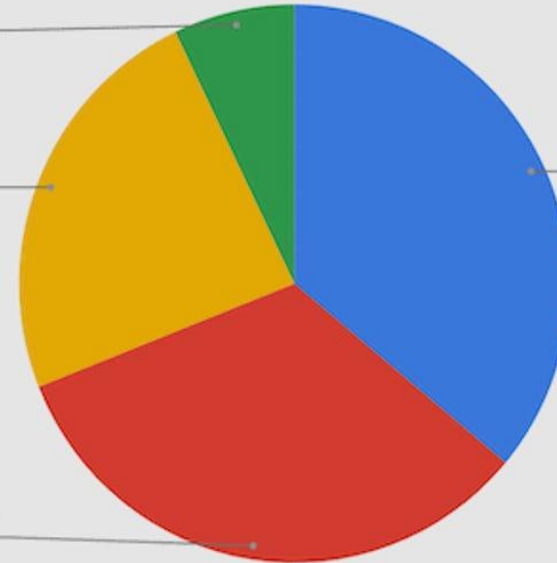
High+, impacting stable

Security-related assert
7.1%

Other
23.9%

Other memory unsafety
32.9%

Use-after-free
36.1%



Exploitable bugs in Chrome. Source: Google Project Zero

Memory Corruptions : Rust binary Hardening (default)

```
jonathan@blackbox: ~/rust/openssl
Fichier  Édition  Affichage  Rechercher  Terminal  Aide
jonathan@blackbox:~/rust/openssl$ checksec.sh --file target/release/openssl-test
RELRO          STACK CANARY      NX            PIE            RPATH          RUNPATH         FILE
Full RELRO     No canary found   NX enabled    PIE enabled    No RPATH       No RUNPATH      target/release/openssl-test
jonathan@blackbox:~/rust/openssl$
```

Memory Corruptions : Adding Stack Cookies

Adding Hardening Flags:

Method 1: Passing CFLAGS

RUSTFLAGS="-C target-feature=+stack-protector-strong" cargo build --release

Method 2: edit .cargo/config.toml

```
[build]
rustflags = ["-C", "target-feature=+stack-protector-strong"]
```

Memory Corruptions : MOABI Recommended Flags (x86_64)



CFLAGS="-fPIC -pie -fPIE -fstack-protector-all -O2 -D_FORTIFY_SOURCE=2 -Wl,-z,now "

**CFLAGS_EXTRA="-fstack-clash-protection -fzero-call-used-regs=used-gpr -fcf-protection
-fstack-check"**

Brief Taxonomy of Memory Corruptions

Overflow : in any wr

Stack overflow, heap overflows,
.bss/.data overflows

```
jonathan@blackbox: ~/rust/openssl
Fichier  Édition  Affichage  Rechercher  Terminal  Aide
jonathan@blackbox:~/rust/openssl$ eu-readelf -S target/release/openssl-test
There are 34 section headers, starting at offset 0x6af98:

Section Headers:
[Nr] Name                               Type            Addr             Off             Size            ES Flags Lk   Inf Al
[ 0]                               NULL            0000000000000000 00000000 00000000 0      0  0  0
[ 1] .interp                               PROGBITS        0000000000000350 00000350 0000001c 0  A      0  0  1
[ 2] .note.gnu.property                    NOTE            0000000000000370 00000370 00000020 0  A      0  0  8
[ 3] .note.gnu.build-id                    NOTE            0000000000000390 00000390 00000024 0  A      0  0  4
[ 4] .note.ABI-tag                         NOTE            00000000000003b4 000003b4 00000020 0  A      0  0  4
[ 5] .gnu.hash                             GNU_HASH        00000000000003d8 000003d8 00000024 0  A      6  0  8
[ 6] .dynsym                               DYNSYM          0000000000000400 00000400 00000078 24  A      7  1  8
[ 7] .dynstr                               STRTAB          0000000000000a78 00000a78 00000045a 0  A      0  0  1
[ 8] .gnu.version                          GNU_verSYM      0000000000000ed2 00000ed2 0000008a 2  A      6  0  2
[ 9] .gnu.version_r                        GNU_verneed     0000000000000f60 00000f60 00000130 0  A      7  4  8
[10] .rela.dyn                             RELA            0000000000001090 00001090 00004b30 24  A      6  0  8
[11] .rela.plt                             RELA            0000000000005bc0 00005bc0 00000030 24  AI     6 27  8
[12] .init                                 PROGBITS        0000000000006000 00006000 0000001b 0  AX      0  0  4
[13] .plt                                  PROGBITS        0000000000006020 00006020 00000030 16  AX      0  0 16
[14] .plt.got                              PROGBITS        0000000000006050 00006050 00000008 8  AX      0  0  8
[15] .text                                 PROGBITS        0000000000006060 00006060 00040d8b 0  AX      0  0 16
[16] .fini                                 PROGBITS        00000000000046dec 00046dec 0000000d 0  AX      0  0  4
[17] .rodata                               PROGBITS        00000000000047000 00047000 000055d8 0  A      0  0 16
[18] .eh_frame_hdr                         PROGBITS        0000000000004c5d8 0004c5d8 0000104c 0  A      0  0  4
[19] .eh_frame                             PROGBITS        0000000000004d628 0004d628 00005a00 0  A      0  0  8
[20] .gcc_except_table                     PROGBITS        00000000000053028 00053028 00001090 0  A      0  0  4
[21] .ldata                               PROGBITS        00000000000055ca8 00055ca8 00000020 0  WAT     0  0  8
[22] .tbss                                NOBITS          00000000000055cc8 00055cc8 00000040 0  WAT     0  0  8
[23] .init_array                           INIT_ARRAY      00000000000055cc8 00055cc8 00000010 8  WA      0  0  8
[24] .fini_array                           FINI_ARRAY      00000000000055cd8 00055cd8 00000008 8  WA      0  0  8
[25] .data.rel.ro                          PROGBITS        00000000000055ce0 00055ce0 00002888 0  WA      0  0  8
[26] .dynamic                              DYNAMIC         00000000000058568 00058568 00000230 16  WA      7  0  8
[27] .got                                  PROGBITS        00000000000058798 00058798 00000868 8  WA      0  0  8
[28] .data                                 PROGBITS        00000000000059000 00059000 00000978 0  WA      0  0  8
[29] .bss                                  NOBITS          00000000000059978 00059978 000000f8 0  WA      0  0  8
[30] .comment                              PROGBITS        00000000000000000 00058978 00000057 1  MS      0  0  1
[31] .symtab                               SYMTAB          00000000000000000 000589d0 00004e60 24      32 494 8
[32] .strtab                               STRTAB          00000000000000000 0005d830 0000d625 0      0  0  1
[33] .shstrtab                             STRTAB          00000000000000000 0006ae55 0000013d 0      0  0  1

jonathan@blackbox:~/rust/openssl$
```

Overflow : Rust itself is immune

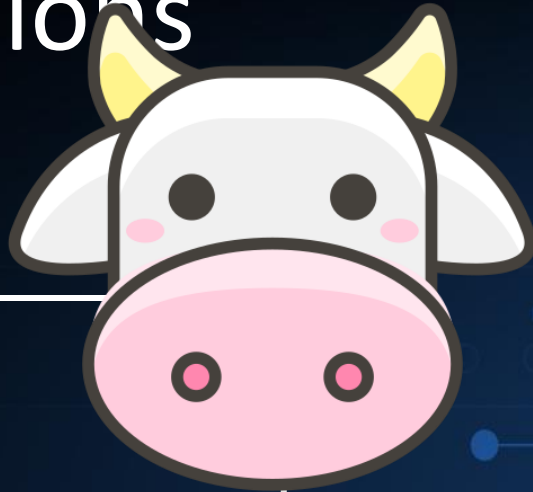
The mechanism of memory borrowing + ownership + bound checking
Ensures memory safety.

EPIC WIN !!!

Type confusions

```
typedef struct cow {  
    char name[16];  
    int (*f)(void);  
}
```

```
typedef struct pig {  
    char name[16];  
    char brand[8];  
};
```



```
int do_milk (struct cow *v)  
{  
    return v->f();  
}
```

What happens if brand is user controlled and we try to do_milk() a pig ?
(very common in Javascript)

Casting Errors/Overflows

Exempli Gratia:

1) sign extension:

```
movsxd rdx, eax
```

2) Integer overflow:

```
mul r8
```

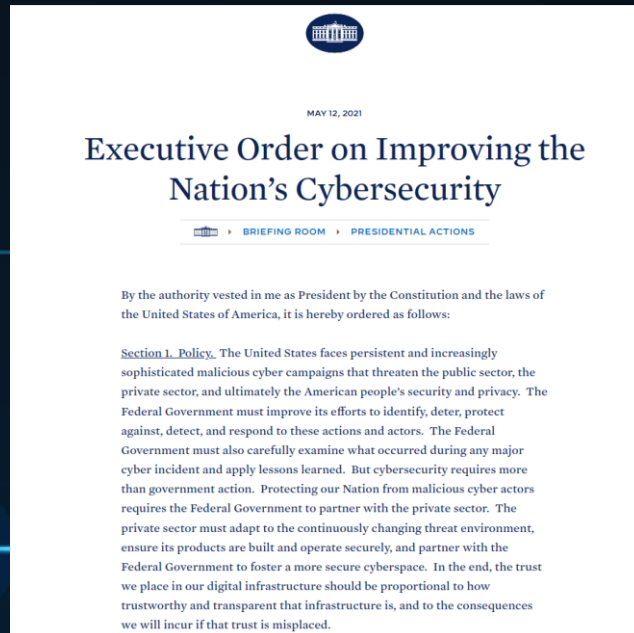

Type Confusions : Rust itself is immune

Affine typing prevents type confusions vulnerabilities.

EPIC WIN !!!

Partial Proofs of Vulnerabilities : Motivation

Motivation : SBOMs vulnerability assessments don't scale well



Executive Order on Improving the Nation's Cybersecurity

MAY 12, 2021

By the authority vested in me as President by the Constitution and the laws of the United States of America, it is hereby ordered as follows:

Section 1. Policy. The United States faces persistent and increasingly sophisticated malicious cyber campaigns that threaten the public sector, the private sector, and ultimately the American people's security and privacy. The Federal Government must improve its efforts to identify, deter, protect against, detect, and respond to these actions and actors. The Federal Government must also carefully examine what occurred during any major cyber incident and apply lessons learned. But cybersecurity requires more than government action. Protecting our Nation from malicious cyber actors requires the Federal Government to partner with the private sector. The private sector must adapt to the continuously changing threat environment, ensure its products are built and operate securely, and partner with the Federal Government to foster a more secure cyberspace. In the end, the trust we place in our digital infrastructure should be proportional to how trustworthy and transparent that infrastructure is, and to the consequences we will incur if that trust is misplaced.



EU Cyber Resilience Act

New EU cybersecurity rules ensure safer hardware and software.

From baby-monitors to smart-watches, products and software that contain a digital component are omnipresent in our daily lives. Less apparent to many users is the security risk such products and software may present.

The [Cyber Resilience Act \(CRA\)](#) aims to safeguard consumers and businesses buying or using products or software with a digital component. The Act would see inadequate security features become a thing of the past with the introduction of mandatory cybersecurity requirements for manufacturers and retailers of such products, with this protection extending throughout the product lifecycle.

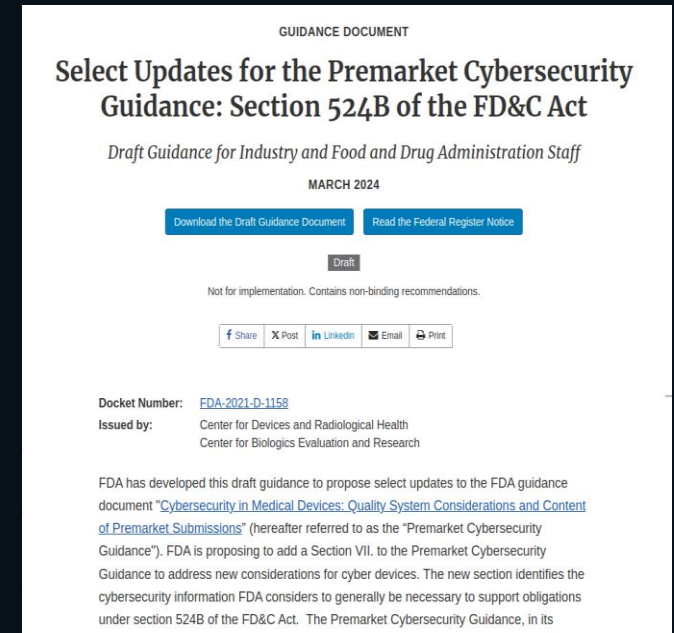
The problem addressed by the Regulation is two-fold.

First is the inadequate level of cybersecurity inherent in many products, or inadequate security updates to such products and software.

Second is the inability of consumers and businesses to currently determine which products are cybersecure, or to set them up in a way that ensures their cybersecurity is protected.

The Cyber Resilience Act will guarantee:

- harmonised rules when bringing to market products or software with a digital component;



GUIDANCE DOCUMENT

Select Updates for the Premarket Cybersecurity Guidance: Section 524B of the FD&C Act

Draft Guidance for Industry and Food and Drug Administration Staff

MARCH 2024

[Download the Draft Guidance Document](#) [Read the Federal Register Notice](#)

[Draft](#)

Not for implementation. Contains non-binding recommendations.

[f Share](#) [X Post](#) [in LinkedIn](#) [Email](#) [Print](#)

Docket Number: [FDA-2021-D-1159](#)

Issued by: Center for Devices and Radiological Health
Center for Biologics Evaluation and Research

FDA has developed this draft guidance to propose select updates to the FDA guidance document "[Cybersecurity in Medical Devices: Quality System Considerations and Content of Premarket Submissions](#)" (hereafter referred to as the "Premarket Cybersecurity Guidance"). FDA is proposing to add a Section VII. to the Premarket Cybersecurity Guidance to address new considerations for cyber devices. The new section identifies the cybersecurity information FDA considers to generally be necessary to support obligations under section 524B of the FD&C Act. The Premarket Cybersecurity Guidance, in its



Software Bill of Materials are becoming mandatory

Motivation : SBOMs vulnerability assessments don't scale well

Software Bill of Materials (SBOMs) contain lists of CPEs or Package URLs (purl) describing all the components of a given Software.

They allow to perform vulnerability assessments by comparing the CPEs to the dictionaries published by the NIST for each CVE.

```
"components": [
  {
    "type": "application",
    "name": "sshd",
    "hashes": [{
      "alg": "MD5",
      "content": "0e6e1c30f8e10e45647ba5d9d4ea6948"
    }, {
      "alg": "SHA-1",
      "content": "399e5d00e2d91628792ee066489f64a6310486"
    }, {
      "alg": "SHA-256",
      "content": "5bd9544d2da6daf7519ee8d6efcd71d6edc16a507f86a90e233174bbebb843c2"
    }]
  },
  {
    "type": "application",
    "description": "sshd",
    "group": "openbsd",
    "name": "openssh",
    "version": "9.6p1",
    "cpe": "cpe:2.3:a:openbsd:openssh:9.6p1:*:*:*:*:*:*:*",
    "externalReferences": [{
      "type": "website",
      "url": "https://www.openssh.com/"
    }],
    "licenses": [{
      "license": {
        "id": "BSD-2-clause"
      },
      "license": {
        "id": "BSD-3-clause"
      }
    }],
  }
]
```

Software Bill of Materials are becoming mandatory

Motivation : SBOMs vulnerability assessments don't scale well

243 SOFTWARE MONITORED

SOFTWARE	SCAN DATE	NEW CVEs	Last Modified
 Hyundai.update.zip	10 August 2022	1230	8 August 2024
 psa_rcc.zip	10 August 2022	1151	8 August 2024
 Firmware.bin	17 October 2022	952	8 August 2024
 Tesla_2019.20.4.2.model3.squashfs	20 June 2023	467	8 August 2024

SBOMs provide possible CVES.
For each vulnerability : is it true ?

CONTINUOUS MONITORING

1019 new vulnerabilities identified for software Tesla_2019.20.4.2.model3.squashfs

Updated report for [dnsmasq](#)

 CVE-2023-50387

CVSS v3 : 7.5

 CVE-2023-50387

Description

Certain DNSSEC aspects of the DNS protocol (in RFC 4033, 4034, 4035, 6840, and related RFCs) allow remote attackers to cause a denial of service (CPU consumption) via one or more DNSSEC responses, aka the "KeyTrap" issue. One of the concerns is that, when there is a zone with many DNSKEY and RRSIG records, the protocol specification implies that an algorithm must evaluate all combinations of DNSKEY and RRSIG records.

Impact

CVSS v3 : 7.5 HIGH

Type

CWE-770

Attack Vector

CVSSv3 Vector : CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:H

Motivation (Bis Repetita Placent)

Zerodays : No patch, am I vulnerable ?

Taint analysis on binary
(no source code)

Calling convention SYSTEM V:
RDI, RSI, RDX, RCX, R8, R9

```

000000000000011e7 <function1>:
11e7: f3 0f 1e fa      endbr64
11eb: 55              push    rbp
11ec: 48 89 e5        mov     rbp,rsi
11ef: 48 81 ec 20 01 00 00 sub     rsp,0x120
11f6: 89 bd ec fe ff ff mov     DWORD PTR [rbp-0x114],edi
11fc: 48 89 b5 e0 fe ff ff mov     QWORD PTR [rbp-0x120],rsi
1203: 64 48 8b 04 25 28 00 mov     rax,QWORD PTR fs:0x28
120a: 00 00
120c: 48 89 45 f8      mov     QWORD PTR [rbp-0x8],rax
1210: 31 c0          xor     eax,eax
1212: 48 8b 85 e0 fe ff ff mov     rax,QWORD PTR [rbp-0x120]
1219: 48 83 c0 08      add     rax,0x8
121d: 48 8b 00        mov     rax,QWORD PTR [rax]
1220: 48 89 c7        mov     rdi,rax
1223: e8 78 fe ff ff  call   10a0 <strlen@plt>
1228: 48 89 c2        mov     rdx,rax
122b: 48 8b 85 e0 fe ff ff mov     rax,QWORD PTR [rbp-0x120]
1232: 48 83 c0 08      add     rax,0x8
1236: 48 8b 08        mov     rcx,QWORD PTR [rax]
1239: 48 8d 85 f0 fe ff ff lea     rax,[rbp-0x110]
1240: 48 89 ce        mov     rsi,rcx
1243: 48 89 c7        mov     rdi,rax
1246: e8 85 fe ff ff  call   10d0 <memcpy@plt>
124b: 48 8d 85 f0 fe ff ff lea     rax,[rbp-0x110]
1252: 48 89 c6        mov     rsi,rax
1255: 48 8d 05 aa 0d 00 00 lea     rax,[rip+0xdaa]
125c: 48 89 c7        mov     rdi,rax
125f: b8 00 00 00 00  mov     eax,0x0
1264: e8 57 fe ff ff  call   10c0 <printf@plt>
1269: b8 00 00 00 00  mov     eax,0x0
126e: 48 8b 55 f8      mov     rdx,QWORD PTR [rbp-0x8]
1272: 64 48 2b 14 25 28 00 sub     rdx,QWORD PTR fs:0x28
1279: 00 00
127b: 74 05          je      1282 <function1+0x9b>
127d: e8 2e fe ff ff  call   10b0 <__stack_chk_fail@plt>
1282: c9             leave
1283: c3             ret
  
```



Zerodays : No patch, am I vulnerable ?

Taint analysis on binary
(no source code)

ASLR	YES
FORTIFY	NO
FULL RELOCATIONS	YES
NX (STACK)	YES
STACK COOKIES	YES
W^X	YES

Vulnerability		CWE-120 - BUFFER OVERFLOW	Score : 7
	Impact: 9	Confidence: 9	Risk: 5
Type	CWE-120 - BUFFER OVERFLOW		
Address	00001246		
Function	000011e7		
Description	Vulnerability when calling function memcpy() : * function argument 3 has the following properties: user controlled, command line argument, arithmetic operation, upper function argument 2, stack variable at offset -0x30, return value from strlen(), * source has the following properties: user controlled, command line argument, arithmetic operation, upper function argument 2, stack variable at offset -0x30, * destination has the following properties: stack variable at offset -0x10, ↪ buffer overflow in third argument		
Backtrace	<pre>#00 <1246> void * __cdecl memcpy(__out_bcount_full_opt(_Size) void * _Dst, __in_bcount_opt(_Size) const void * _Src, __in size_t _Size) at: /home/jonathan/sample/sample.c:22 #01 <11e7> function1() at: /home/jonathan/sample/sample.c:19 #02 <1284> int main (int argc, char **argv, char **envp) at: /home/jonathan/sample/sample.c:29</pre>		

Problem Statement :

Partial Proofs of Vulnerabilities through Procedural Debugging

Proving Exploitability

Industry standard to Prove exploitability : Write an exploit

This bar is too high.

If we decompose an exploit into 3 problems:

- Reach the vulnerable function
- Trigger the vulnerability
- Achieve code execution/Weaponize

The first step alone is already undecidable ("reachability problem").



DANGER

Undecidable

Proving Exploitability



Entry point

Vulnerable function

The "reachability problem" is undecidable.

Proving Exploitability : Partial Proof of Vulnerability

Let's do only step 2:

If we decompose an exploit into 3 problems:

- ~~— Reach the vulnerable function~~
- **Trigger the vulnerability**
- ~~— Achieve code execution/Weaponize~~

This a reasonable heuristic to determine vulnerability of the application.

We'd like to be able to call the vulnerable function directly.

Problem : How to do this out of context ?

Proposal : Let's turn the vulnerable application into a shared library !

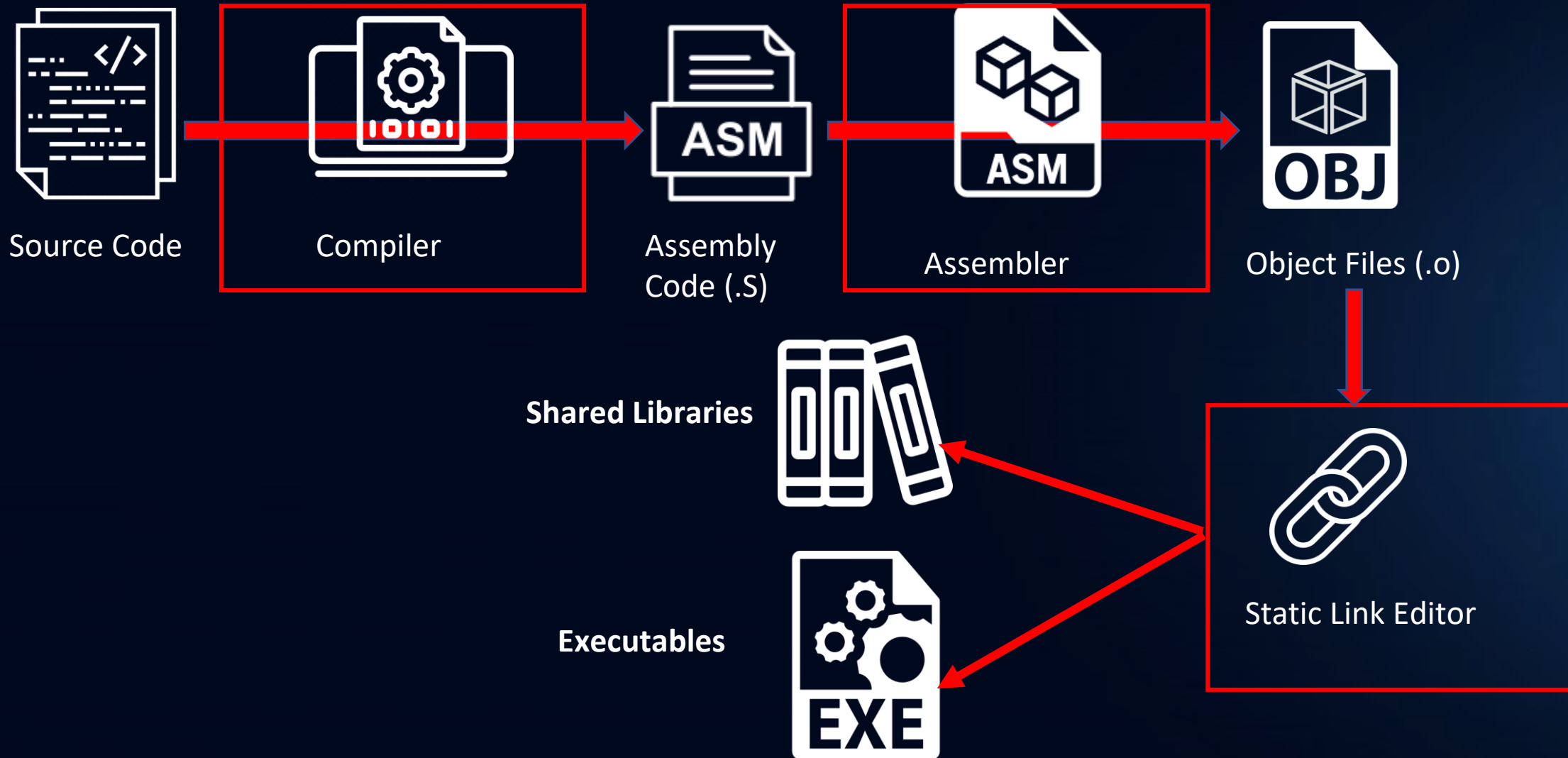


DANGER

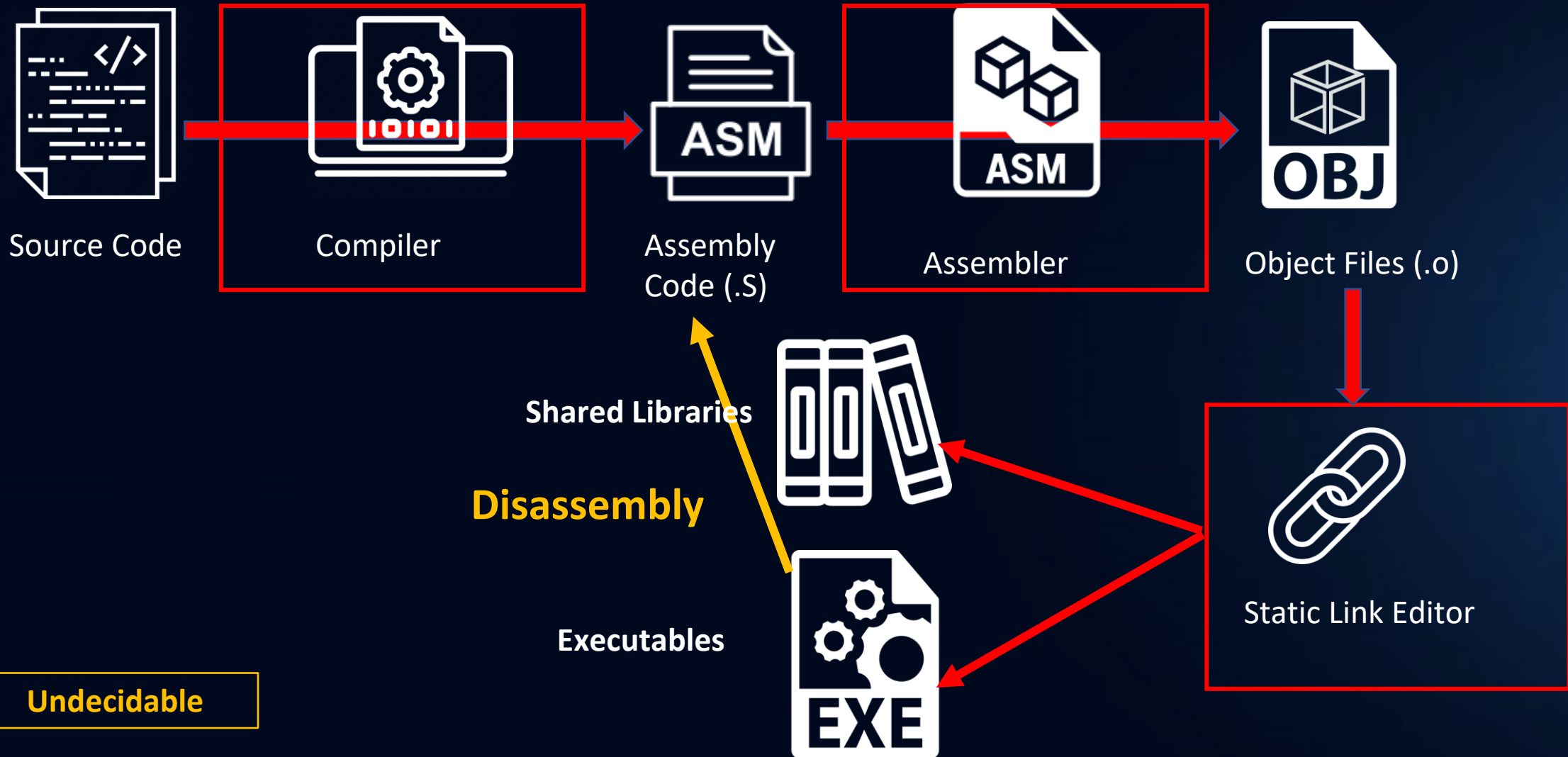
Loading Dynamic Executables : Introduction to ELF Libification

"Generalize to simplify" : A.
Grothendieck

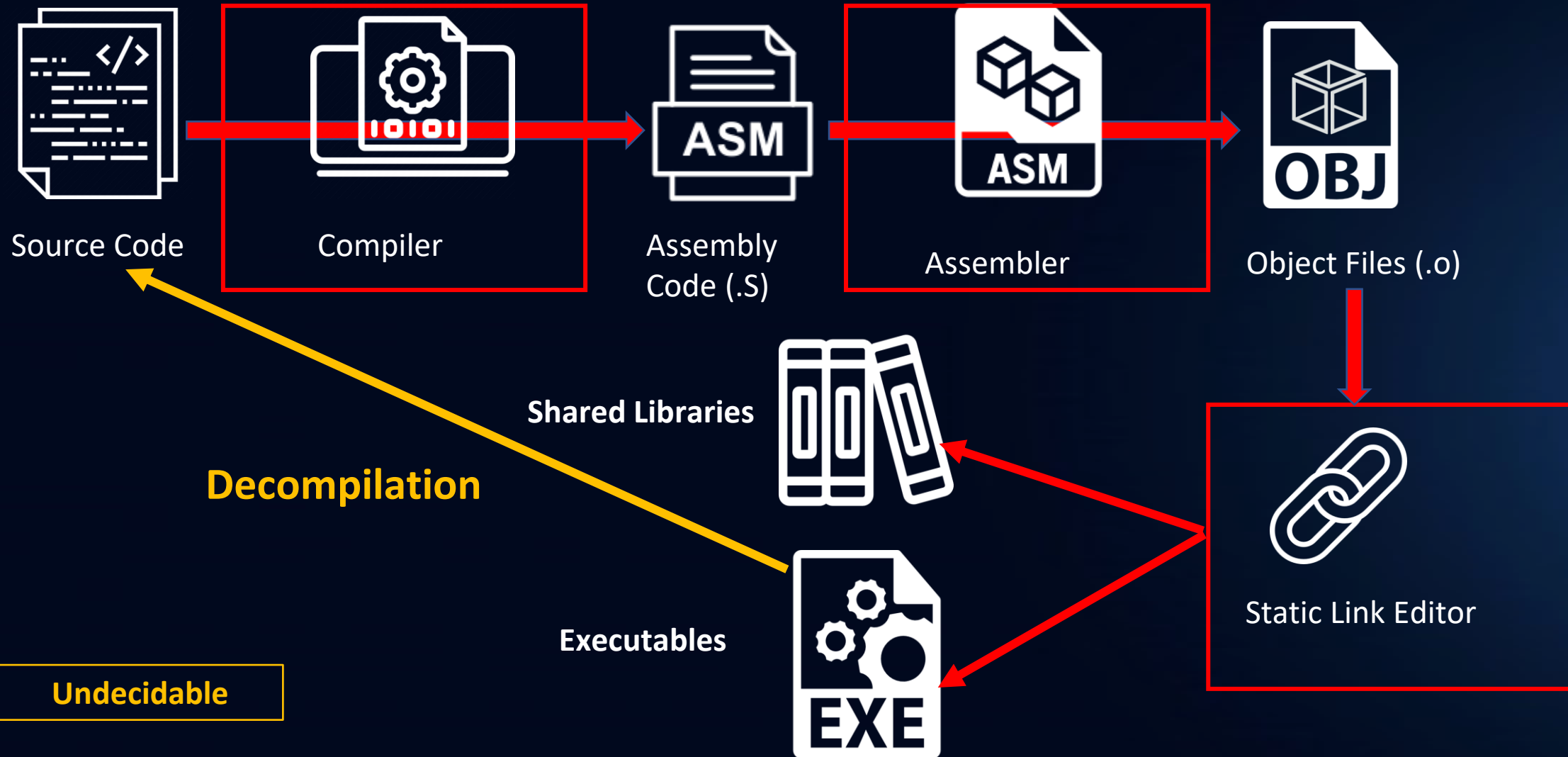
Reverse Engineering : Compilation Process



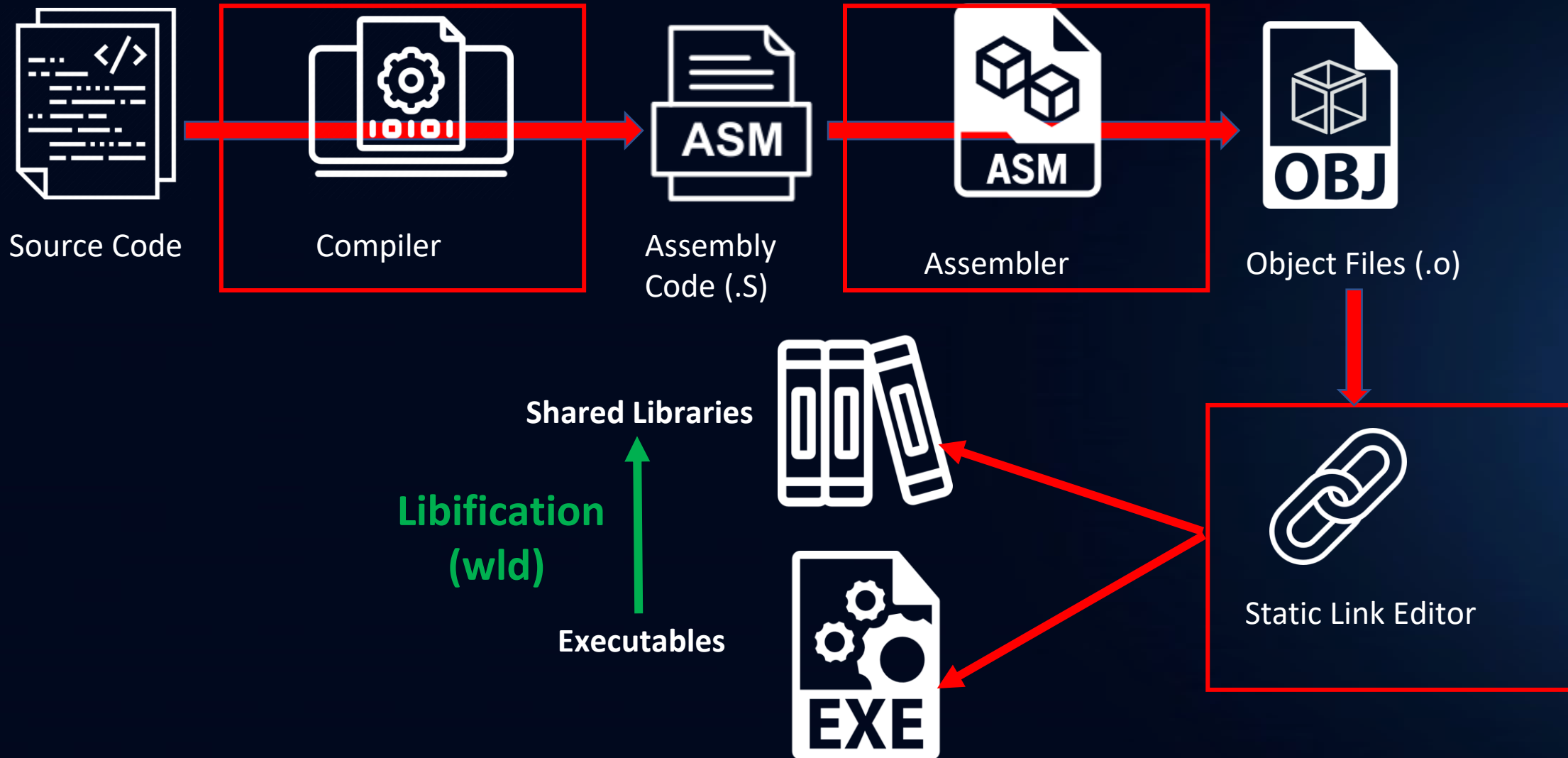
Reverse Engineering : Disassembly



Reverse Engineering : Decompilation



Reverse Engineering : Libification



Executables vs Shared Libraries : How Do They Differ ?

Same headers, same segments, same sections. They mostly differ through their **metadata (various ELF headers)**

Tool Interface Standard (TIS) Executable and Linking Format (ELF) Specification Version 1.2

TIS Committee
May 1995

The work to be done:

Modify the various ELF headers to
turn an Executable into a Shared
Library

ELF Header

Some object file control structures can grow, because the ELF header contains their actual sizes. If the object file format changes, a program may encounter control structures that are larger or smaller than expected. Programs might therefore ignore "extra" information. The treatment of "missing" information depends on context and will be specified when and if extensions are defined.

Figure 1-3. ELF Header

```
#define EI_NIDENT      16

typedef struct {
    unsigned char  e_ident[EI_NIDENT];
    Elf32_Half    e_type;
    Elf32_Half    e_machine;
    Elf32_Word    e_version;
    Elf32_Addr    e_entry;
    Elf32_Off     e_phoff;
    Elf32_Off     e_shoff;
    Elf32_Word    e_flags;
    Elf32_Half    e_ehsize;
    Elf32_Half    e_phsize;
    Elf32_Half    e_phnum;
    Elf32_Half    e_shentsize;
    Elf32_Half    e_shnum;
    Elf32_Half    e_shstrndx;
} Elf32_Ehdr;
```

e_ident The initial bytes mark the file as an object file and provide machine-independent data with which to decode and interpret the file's contents. Complete descriptions appear below, in "ELF Identification."

e_type This member identifies the object file type.

Name	Value	Meaning
ET_NONE	0	No file type
ET_REL	1	Relocatable file
ET_EXEC	2	Executable file
ET_DYN	3	Shared object file
ET_CORE	4	Core file
ET_LOPROC	0xff00	Processor-specific
ET_HIPROC	0xffff	Processor-specific

Automated Libification : the Witchcraft Linker

URL: <https://github.com/endrazine/wcc>

License: MIT/BSD-2

```
jonathan@blackbox: ~/woot
Fichier  Édition  Affichage  Rechercher  Terminal  Aide
jonathan@blackbox:~/woot$ file ls
ls: ELF 64-bit LSB pie executable, x86-64, version 1 (SYSV), dynamically
linked, interpreter /lib64/ld-linux-x86-64.so.2, BuildID[sha1]=3eca7e
3905b37d48cf0a88b576faa7b95cc3097b, for GNU/Linux 3.2.0, stripped
jonathan@blackbox:~/woot$ wld -libify ./ls
jonathan@blackbox:~/woot$ file ls
ls: ELF 64-bit LSB shared object, x86-64, version 1 (SYSV), dynamically
linked, interpreter /lib64/ld-linux-x86-64.so.2, BuildID[sha1]=3eca7e3
905b37d48cf0a88b576faa7b95cc3097b, for GNU/Linux 3.2.0, stripped
jonathan@blackbox:~/woot$
```


Validation

Test Repository:
<https://github.com/endrazine/wcc-tests>

Version	Codename	Year	GCC Version	Passed	Failed
24.04	noble	2024	12.3.0	440	0
22.04	Jammy	2022	11.2.0	435	0
20.04	focal	2020	9.3.0	435	0
18.04	bionic	2018	7.3.0	369	0
16.04	xenial	2016	5.3.1	363	0
14.04	trusty	2014	4.8.2	352	0



DOI 10.5281/zenodo.11301409

Procedural Debugging

Procedural Debugging : Libification based Debugging



ELF Executable
or Library

Load in Memory
(+ load dependencies
& solve relocations)

```
function1()
```

```
...
```

```
function...()
```

```
...
```

```
function_vulnerable()
```

```
....
```

If the ELF is dynamically linked, libify it and load via `dlopen()`.

If the ELF is statically linked, loading its segments is enough.

We simply need a userland ELF loader.

Confere userland `exec()` techniques (eg: The Grugq, phrack, 2004)

Address Space

(Without Control Flow Analysis, which is Undecidable)

In Dynamically linked ELFs, parse the link_map

- Provides a list of all ELFs in memory
- Dynamic linker exports iterators to symbols
- Use dlsym() function from the Dynamic Linker to obtain address of symbols.

function1()

...

function...()

...

function_vulnerable()

....

Library_1

...

Library_n

Address Space

Procedural Debugging : Crafting Minimal Context/Arguments

Calling convention is architecture specific.
On x86_64 GNU/Linux, arguments passing is done via registers:

RDI, RSI, RDX, RCX, R8, R9 for integers or pointers, and XMM0, XMM1, XMM2, XMM3, XMM4, XMM5, XMM6 and XMM7 for floating-point arguments.

We can craft those arguments and point RIP to `function_vulnerable()` to achieve selective procedural execution.

- **Avoid reachability problem.**

`function1()`

...

`function...()`

...

`function_vulnerable()`

....

Library_1

...

Library_n

WSH

We would like to map the ELF in memory, solve dependencies/relocations if the ELF is dynamically linked.

Then call the potentially vulnerable function with chosen arguments, with just enough context to trigger the vulnerability.

Address Space

Binary Reflection

The Witchcraft Shell implements:

- An exposition of binary functions from the LUA interpreter.

Thanks to lua, we then implement:

- Binary introspection
- Binary intercession

=> We implement binary reflection, **without source code**.



DOI 10.5281/zenodo.13902925

Procedural Debugging

- Libify ELF executables
- Make ELF executables scriptable
- Call arbitrary functions (procedural debugging)

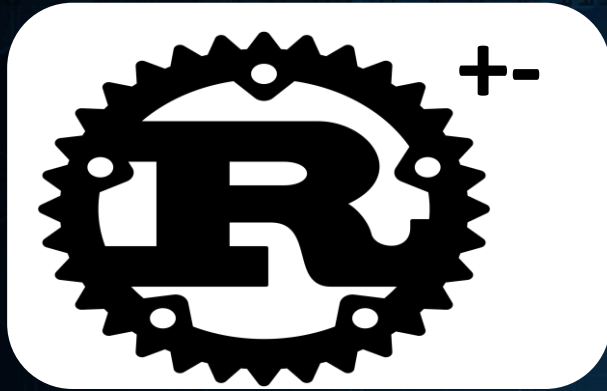
```
jonathan@blackbox: ~  
Fichier  Édition  Affichage  Rechercher  Terminal  Aide  
jonathan@blackbox:~$ wsh /usr/sbin/apache2  
ERROR: dlopen() /usr/sbin/apache2: cannot dynamically load  
position-independent executable  
** libifying /usr/sbin/apache2 to //tmp/.wsh-964913/apache  
2 (754232 bytes)  
** loading of libified binary succeeded  
> a = ap_get_server_banner()  
> print(a)  
Apache/2.4.58  
> |
```

URL: <https://github.com/endrazine/wcc>
License: MIT/BSD-2

DOI 10.5281/zenodo.13902925

Oxydation of WCC

Oxydation of WCC :



- Symbol demangling in Rust is quite specific
- Done in Rust, linked to WCC
- Available from version v0.0.10

URL: <https://github.com/endrazine/wcc>
License: MIT/BSD-2

```
jonathan@blackbox: ~/rust/rust2c
Fichier  Édition  Affichage  Rechercher  Terminal  Aide
jonathan@blackbox:~/rust/rust2c$ cat src/lib.rs
// lib.rs
#[no_mangle]
pub extern "C" fn add_numbers(a: i32, b: i32) -> i32 {
    a + b
}

#[no_mangle]
pub extern "C" fn hello_from_rust() {
    println!("Hello from Rust!");
}
jonathan@blackbox:~/rust/rust2c$
```

```
jonathan@blackbox: ~/rust/rust2c
Fichier  Édition  Affichage  Rechercher  Terminal  Aide
jonathan@blackbox:~/rust/rust2c$ wsh target/release/libmyrust.so
> hello_from_rust()
Hello from Rust!
>
```


Oxydation of WCC : R-



- Caveat : May need to copy symbols from `.symtab/.strtab` to `.dynsym/.dynstr`

URL: <https://github.com/endrazine/wcc>
License: MIT/BSD-2

Advances in Lightweight Cross-Architecture Procedural Debugging

Jonathan Brossard
CEDRIC/ISID
Conservatoire National des Arts et Métiers
Paris, France
jonathan.brossard@lecnam.net

Abstract—As embedded software is becoming more and more prevalent in mission-critical devices such as self-driving vehicles or medical equipment, Product Security Incident Response Teams (PSIRTs) face the need to debug software across multiple architectures, often different from their host desktop machines. Concomitantly, recent updates to the European legislative framework, in particular the NIS2 and Cyber Resilience Act mandate that critical vulnerabilities be reported on a daily basis. To help bridge the gap between those upcoming regulations and the reality of the industry, this article focuses on facilitating cross-architecture vulnerability validation via partial proofs of vulnerabilities by leveraging binary libification and lightweight procedural debugging.

Index Terms—Libification, Reverse Engineering, JIT binary translation, NIS2, CRA, lightweight debugging

I. INTRODUCTION

Recent enhancements to the European legal framework governing Cloud service security require that Product Security Incident Response Teams (PSIRTs) operating in Europe report newly discovered vulnerabilities affecting their products and services to their respective national authorities within 24 hours. Specifically, Directive (EU) 2022/2555, known as NIS2 [1], mandates the assessment and reporting of security vulnerabilities within this timeframe.

In 2024, the NIST publicly disclosed over 40,000 vulnerabilities in the form of CVEs [3], a significant increase from approximately 29,000 in 2023. This surge leaves PSIRT teams facing the formidable challenge of assessing their services' exposure to more than 100 newly reported vulnerabilities each day. While tools like Software Bills of Materials (SBOMs) have emerged to ease this burden by automating the identification of potential vulnerabilities [21]—based solely on the versions of binaries present in containers—recent studies [22] have raised serious concerns about the reliability and reproducibility of SBOMs. As a result, PSIRT teams are often required to manually validate vulnerabilities flagged through SBOM monitoring tools [8] via extensive debugging efforts. Critically, a specific version of a software component may or may not be affected by a vulnerability, and its CVSS [4] score can vary significantly depending on compilation options [13] [14], details that are currently not captured by major SBOM standards [23] [5] [24] [28].

In this article, we devise new methodologies to quickly validate the existence of vulnerabilities in compiled binaries,

without access to their source code, since shipping compiled programs is common practice in the industry [16]. Focusing primarily on embedded devices, we will detail how binary libification and procedural debugging may be leveraged to produce partial proofs of vulnerabilities cross-architecture. Because market research over the past decade indicate an overwhelming use of GNU/Linux as a base OS in embedded devices, with a market penetration ranging from 40% to 60% depending on industry verticals [19] [17] [18], we shall focus primarily on debugging GNU/Linux, but will generalize our methodologies to other operating systems where applicable.

Our contribution to cross-architecture libification and procedural debugging is implemented into the Witchcraft Compiler Collection, a reverse engineering framework published under a permissive dual MIT/BSD license [32]. Our implementation of the Witchcraft tracer, an extension of the venerable *ltrace* [20] debugging tool is published under the GPL-2 open-source license.

II. PREVIOUS WORK

In this section, we offer a brief summary of our recent work in the field of reverse engineering ELF binaries, with the aim to provide foundational knowledge for the subsequent sections, where we will aim at generalizing those findings to achieve cross-architecture debugging.

A. Overview of binary Libification

In “Introduction to Procedural Debugging through Binary Libification” [30], we introduced the concept of *Binary libification*, which refers to the process of converting a dynamically linked ELF [11] executable into a shared library. This allows vulnerable functions to be called directly, which circumvents the traditional reachability problem [26] over the callgraph of the binary, which is undecidable. This mechanism provides flexibility in inspecting and interacting with memory corruption vulnerabilities by simplifying how functions are invoked and observed.

B. Overview of procedural debugging

This paper also introduces *Procedural debugging* [30], a technique used to analyze vulnerabilities by executing specific program functions without having to write entire exploits, focusing on intra-procedural triggers generation. This method

Oxydation of WCC : R+-

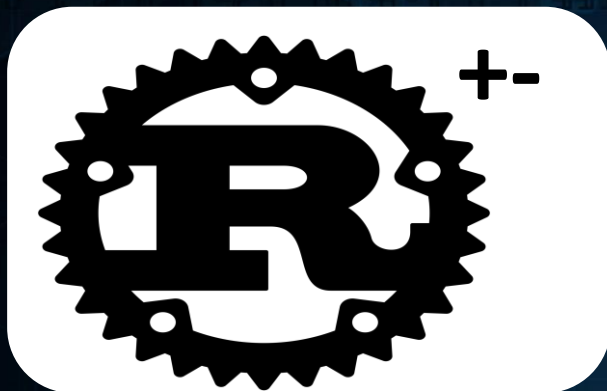


- Caveat : May need to copy symbols from .symtab/.strtab to .dynsym/.dynstr

URL: <https://github.com/endrazine/wcc>
License: MIT/BSD-2

```
jonathan@blackbox: ~/rust/openssl
Fichier  Édition  Affichage  Rechercher  Terminal  Aide
jonathan@blackbox:~/rust/openssl$ file ./target/release/openssl-test
./target/release/openssl-test: ELF 64-bit LSB pie executable, x86-64, version 1
(SYSV), dynamically linked, interpreter /lib64/ld-linux-x86-64.so.2, BuildID[sha
1]=2930591bc66720cc693193be0b3a93923bf18e18, for GNU/Linux 3.2.0, not stripped
jonathan@blackbox:~/rust/openssl$ ./target/release/openssl-test
OpenSSL version: OpenSSL 3.0.10 1 Aug 2023
OpenSSL version number: 300000a0
jonathan@blackbox:~/rust/openssl$ nm -D ./target/release/openssl-test|grep " _Z\
| _R"|wc -l
0
jonathan@blackbox:~/rust/openssl$ wunstrip ./target/release/openssl-test -o /tmp
/openssl-test2 -r
-- saved unstripped file as /tmp/openssl-test2
jonathan@blackbox:~/rust/openssl$ nm -D /tmp/openssl-test2|grep " _Z\| _R"|wc -l
252
jonathan@blackbox:~/rust/openssl$
```


Oxydation of WCC : R+-



- Caveat : May need to copy symbols from .symtab/.strtab to .dynsym/.dynstr

URL: <https://github.com/endrazine/wcc>
License: MIT/BSD-2

```
jonathan@blackbox: ~/rust/openssl
Fichier  Édition  Affichage  Rechercher  Terminal  Aide
jonathan@blackbox:~/rust/openssl$ wsh /tmp/openssl-test2
ERROR: dlopen() /tmp/openssl-test2: cannot dynamically load position-independent
executable
** libifying /tmp/openssl-test2 to //tmp/.wsh-415634/openssl-test2 (475696 byte
s)
** loading of libified binary succeeded
WARNING: Wsh internal lua initialization (Synthax Error): Wsh internal lua buffe
r:2690: <name> expected near '('
> libs()
/usr/local/openssl-3.0.10/lib/libcrypto.so.3
/usr/local/openssl-3.0.10/lib/libssl.so.3
/lib/x86_64-linux-gnu/libc.so.6
/lib/x86_64-linux-gnu/libgcc_s.so.1
//tmp/.wsh-415634/openssl-test2
/lib/x86_64-linux-gnu/libgtk3-nocsd.so.0
/lib64/ld-linux-x86-64.so.2
linux-vdso.so.1
linux-vdso.so.1

-- Total: 9 libraries
> |
```

Oxydation of WCC : R+-

wsh: invalid lua syntax from mangled symbols #24

Open



sp3d opened on Jun 2, 2017

Running wsh on binaries with non-alphanumeric symbols (e.g. the mangled ones produced by rustc) results in output like

```
ERROR: Wsh internal lua initialization (Synthax Error): Wsh internal lua buffer:2070: '(' expected near '$'
```

which refers to this line in the generated lua:

```
function _ZN4core3num68_SLT$impl$u20$core..convert..TryFromSLT$i16$GT$$u20$for$u20$u128$GT$8try_from17h890e
```

It might be nice to support demangling, though that may complicate parsing (angle brackets, parentheses, and colons would be valid in symbol names); in any event it would be nice for these symbols to not prevent wsh from running.

Create sub-issue



endrazine 2 days ago

Owner

Hi sp3d,

I hope you're doing well.
I just merged support for Rust symbols.

Thanks for your patience,

j-



Partial Proofs of Vulnerabilities (in Rust bindings)

RUSTSEC-2025-0022

Use-After-Free in `Md::fetch` and `Cipher::fetch`

Reported April 4, 2025

Issued April 4, 2025

Package [openssl](#) ([crates.io](#))

Type Vulnerability

Categories [memory-corruption](#)

References <https://github.com/sfackler/rust-openssl/pull/2390>

Patched `>=0.10.72`

Unaffected `<0.10.39`

Affected Functions

Version

`openssl::cipher::Cipher::fetch` `>=0.10.39, <0.10.72`

`openssl::md::Md::fetch` `>=0.10.39, <0.10.72`

Description

When a `Some(...)` value was passed to the `properties` argument of either of these functions, a use-after-free would result.

In practice this would nearly always result in OpenSSL treating the properties as an empty string (due to `CString::drop`'s behavior).

The maintainers thank [quitbug](#) for reporting this vulnerability to us.

RUSTSEC-2024-0357

`MemBio::get_buf` has undefined behavior with empty buffers

Reported July 21, 2024

Issued July 21, 2024

Package [openssl](#) ([crates.io](#))

Type Vulnerability

References <https://github.com/sfackler/rust-openssl/pull/2266>

Patched `>=0.10.66`

Affected Functions

Version

Affected Functions	Version
<code>openssl::bio::MemBio::get_buf</code>	<code><0.10.66, >=0.8.0</code>

Description

Previously, `MemBio::get_buf` called `slice::from_raw_parts` with a null-pointer, which violates the functions invariants, leading to undefined behavior. In debug builds this would produce an assertion failure. This is now fixed.

Advisory available under [CC0-1.0](#) license.



Automatic Arguments Inference

Automatic Arguments Inference

Interactive function arguments
discovery

```

jonathan@blackbox: ~
Fichier  Édition  Affichage  Rechercher  Terminal  Aide
jonathan@blackbox:~$ wsh /lib/x86_64-linux-gnu/libc.so.6
> memcpy(0x41414141, 0x42424242, 0x43434343)

[SIGSEGV]      Read      0x42424242      (address not mapped to object)
0x7d9d169988cd  /lib/x86_64-linux-gnu/libc.so.6(+0x1988cd)
0x4201b1e6      wsh()
0x4203a877      wsh()
ERROR: State not recoverable (131)
> memcpy(0x41414141, "bbbbbbbb", 0x43434343)

[SIGSEGV]      Read      0x8fbe704b      (address not mapped to object)
0x7d9d16998a46  /lib/x86_64-linux-gnu/libc.so.6(+0x198a46)
0x4201b1e6      wsh()
0x4203a877      wsh()
ERROR: State not recoverable (131)
> memcpy(0x41414141, "bbbbbbbb", 1)

[SIGSEGV]      Write     0x41414141      (address not mapped to object)
0x7d9d16998923  /lib/x86_64-linux-gnu/libc.so.6(+0x198923)
0x4201b1e6      wsh()
0x4203a877      wsh()
ERROR: State not recoverable (131)
> memcpy("aaaaaaaa", "bbbbbbbb", 1)
>
> prototypes()

[*] Prototypes: (from 2 tag informations)
/lib/x86_64-linux-gnu/libc.so.6 __memmove_evex_unaligned_erms  argument1  _output_ptr  0
/lib/x86_64-linux-gnu/libc.so.6 __memmove_evex_unaligned_erms  argument2  _input_ptr   0
>

```

Automatic Arguments Inference

Automation

```
jonathan@blackbox: ~
Fichier  Edition  Affichage  Recherche  Terminal  Aide
> testfunc(memcpy)

[SIGSEGV]      Read    0x42424242      (address not mapped to object)
0x7dc8eaf988cd  /lib/x86_64-linux-gnu/libc.so.6(+0x1988cd)
0x4201b1e6     wsh()
0x4203a877     wsh()
ERROR: State not recoverable (131)

[SIGSEGV]      Read    0xc5c5c5c67e0    (address not mapped to object)
0x7dc8eaf98a46  /lib/x86_64-linux-gnu/libc.so.6(+0x198a46)
0x4201b1e6     wsh()
0x4203a877     wsh()
ERROR: State not recoverable (131)

[SIGSEGV]      Read    0xfa4f05f0      (address not mapped to object)
0x7dc8eaf98a46  /lib/x86_64-linux-gnu/libc.so.6(+0x198a46)
0x4201b1e6     wsh()
0x4203a877     wsh()
ERROR: State not recoverable (131)

[SIGSEGV]      Read    0xd1d1d1d0fe0    (address not mapped to object)
0x7dc8eaf98a46  /lib/x86_64-linux-gnu/libc.so.6(+0x198a46)
0x4201b1e6     wsh()
0x4203a877     wsh()
ERROR: State not recoverable (131)

[SIGSEGV]      Read    0xfa4f05f0      (address not mapped to object)
0x7dc8eaf98a46  /lib/x86_64-linux-gnu/libc.so.6(+0x198a46)
0x4201b1e6     wsh()
0x4203a877     wsh()
ERROR: State not recoverable (131)

[SIGSEGV]      Write   0x200          (address not mapped to object)
0x7dc8eaf98c43  /lib/x86_64-linux-gnu/libc.so.6(+0x198c43)
0x4201b1e6     wsh()
0x4203a877     wsh()
ERROR: State not recoverable (131)
[*] all tests done (5 calls)
> prototypes()

[*] Prototypes: (from 2 tag informations)
/lib/x86_64-linux-gnu/libc.so.6 __memmove_evex_unaligned_erms  argument1  _output_ptr  0
/lib/x86_64-linux-gnu/libc.so.6 __memmove_evex_unaligned_erms  argument2  _input_ptr   0
> |
```

Context: Replaying Itrace



```
jonathan@blackbox: ~
Fichier  Édition  Affichage  Rechercher  Terminal  Aide
jonathan@blackbox:~$ wltrace -W /bin/cat /etc/issue.net 2>/tmp/demo.sh
Ubuntu 24.04.2 LTS
jonathan@blackbox:~$ chmod +x /tmp/demo.sh
jonathan@blackbox:~$ /tmp/demo.sh
ERROR: dlopen() /bin/cat: cannot dynamically load position-independent executable
** libifying /bin/cat to //tmp/.wsh-48767/cat (39384 bytes)
** loading of libified binary succeeded
ERROR: Bad address (14)
ERROR: Bad address (14)
Ubuntu 24.04.2 LTS
jonathan@blackbox:~$ cat /tmp/demo.sh |grep -v "^\\-\\-"
#!/usr/bin/wsh /bin/cat

ret_1 = strrchr("/bin/cat", '/') ; -- "/cat"
ret_2 = setlocale(LC_ALL, "") ; -- "C"
ret_3 = bindtextdomain("coreutils", "/usr/share/locale") ; -- "/usr/share/locale"
ret_4 = textdomain("coreutils") ; -- "coreutils"
ret_7 = fstat(1, var_0x7ffc57f9be90, var_0x772d15e03240, 0) ; -- 0
ret_8 = getpagesize() ; -- 4096
ret_9 = open("/etc/issue.net", 0, 01777) ; -- 3
ret_10 = fstat(3, var_0x7ffc57f9be90, 0, var_0x772d15d1b175) ; -- 0
ret_11 = posix_fadvise(3, 0, 0, 2) ; -- 0
ret_12 = aligned_alloc(4096, var_0x20000, 0, var_0x772d15d1b52e) ; -- var_0x772d1600e000
ret_13 = read(3, "Ubuntu 24.04.2 LTS\n", 131072) ; -- 19
ret_14 = write(1, "Ubuntu 24.04.2 LTS\n", 19) ; -- 19
ret_15 = read(3, "", 131072) ; -- 0
ret_17 = close(3) ; -- 0
ret_22 = fflush(var_0x772d15e045c0) ; -- 0
ret_28 = fflush(var_0x772d15e044e0) ; -- 0
ret_30 = exit(0)
jonathan@blackbox:~$
```


Industry Adoption

The Framework is already available in major Linux distributions

Available in the repositories from:

- Debian
- Ubuntu
- Arch Linux
- Kali Linux

```
root@blackbox:~# apt install wcc
```

URL: <https://github.com/endrazine/wcc>
License: MIT/BSD-2



Applications : Generic Code re-use outside Cybersecurity

Code reuse:

- Source code lost/no longer compiles
- Legacy applications
- Cross platform execution



Cross Architecture Procedural Debugging (JIT)

By cross-compiling WSH to other architectures and running it within qemu-user, we benefit from JIT translation to the host architecture (eg: Aarch64 to x86_64) without a Virtual Machine, and in a single process (debugger + debuggee + JIT translator)

```
function1()
...
function...()
...
function_vulnerable()
....
```

Library_1

Library_n

WSH

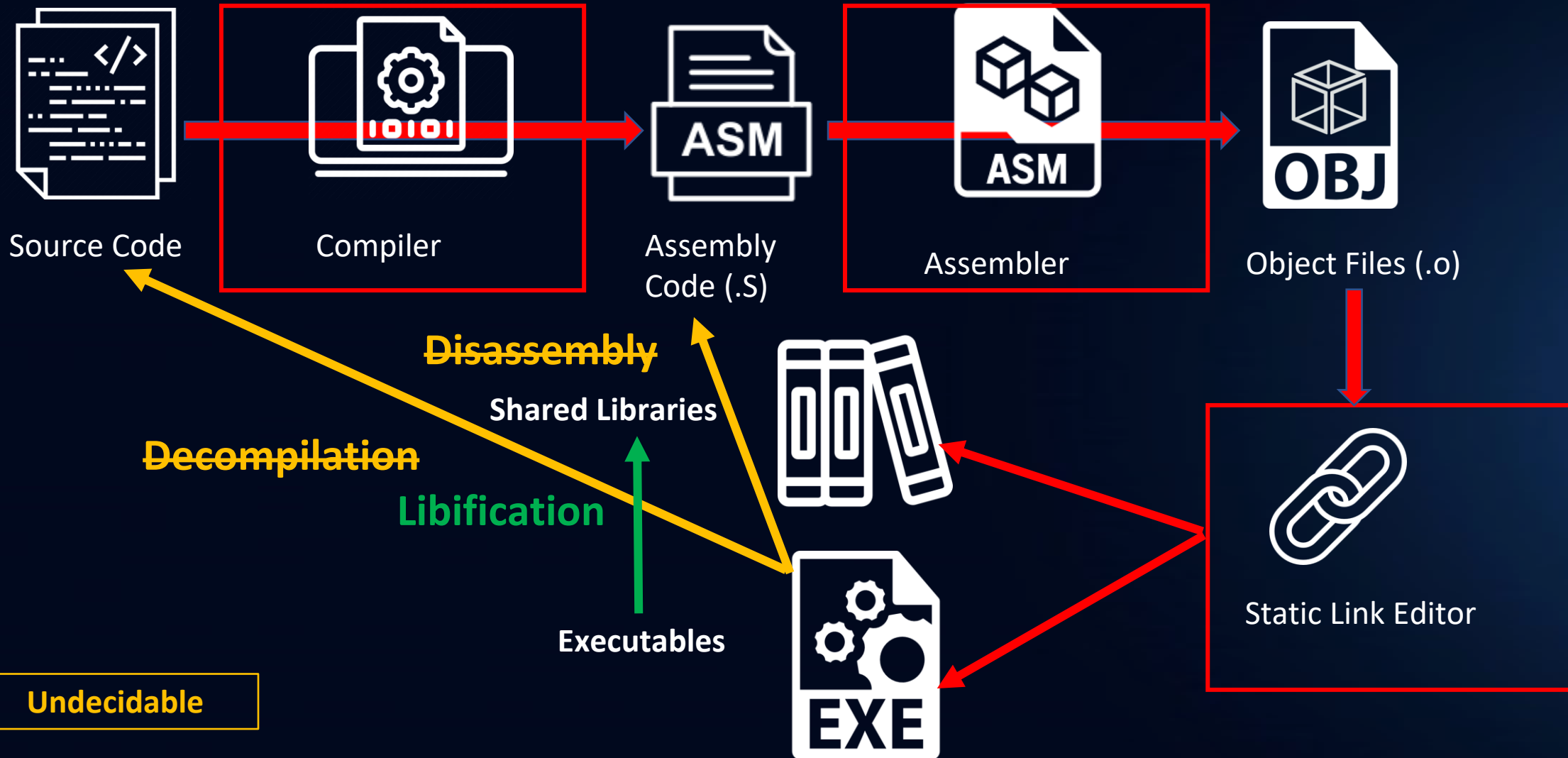
Qemu-user (JIT)

DOI 10.5281/zenodo.13902925

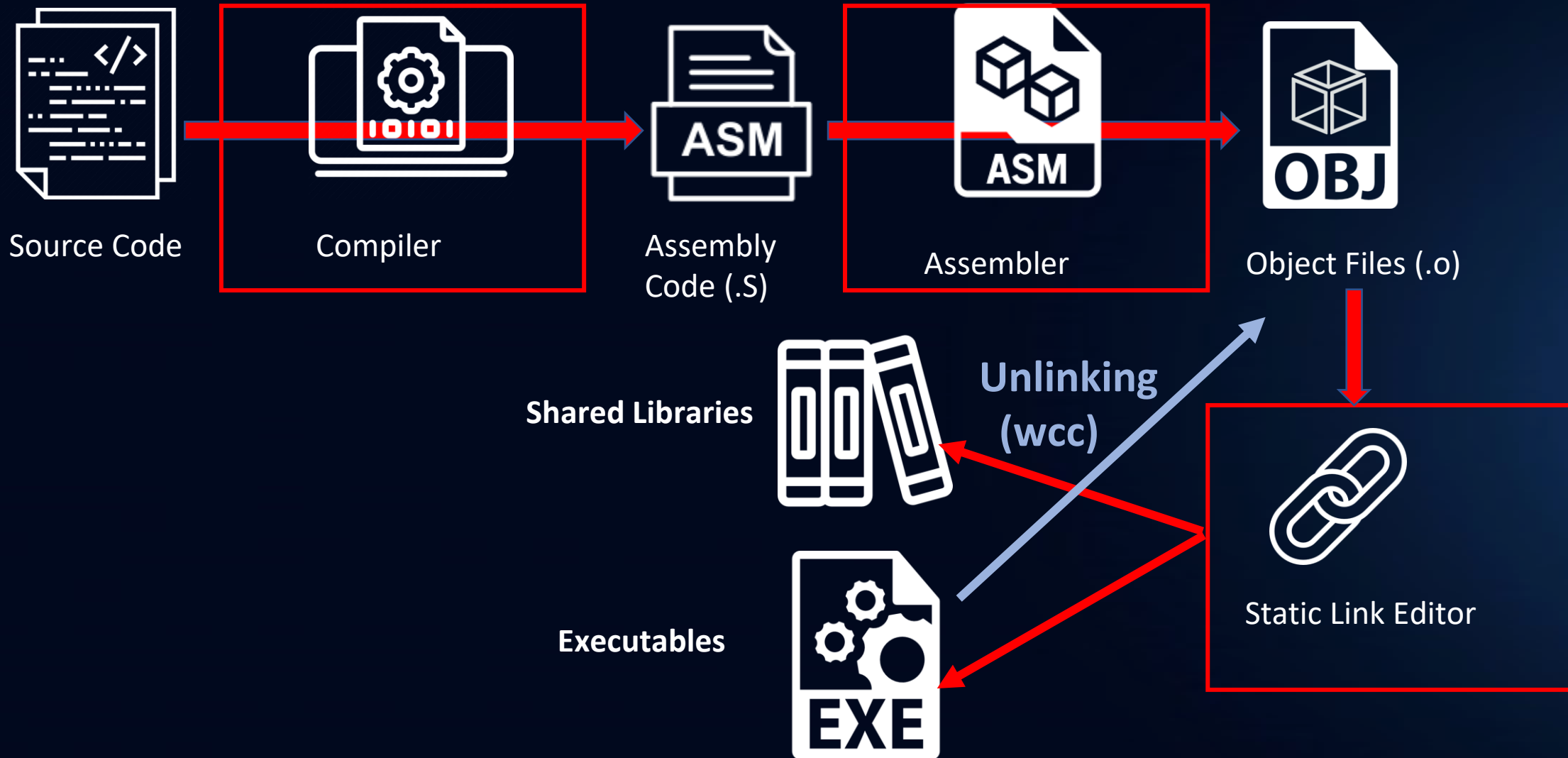
Address Space

Conclusion & Future Work

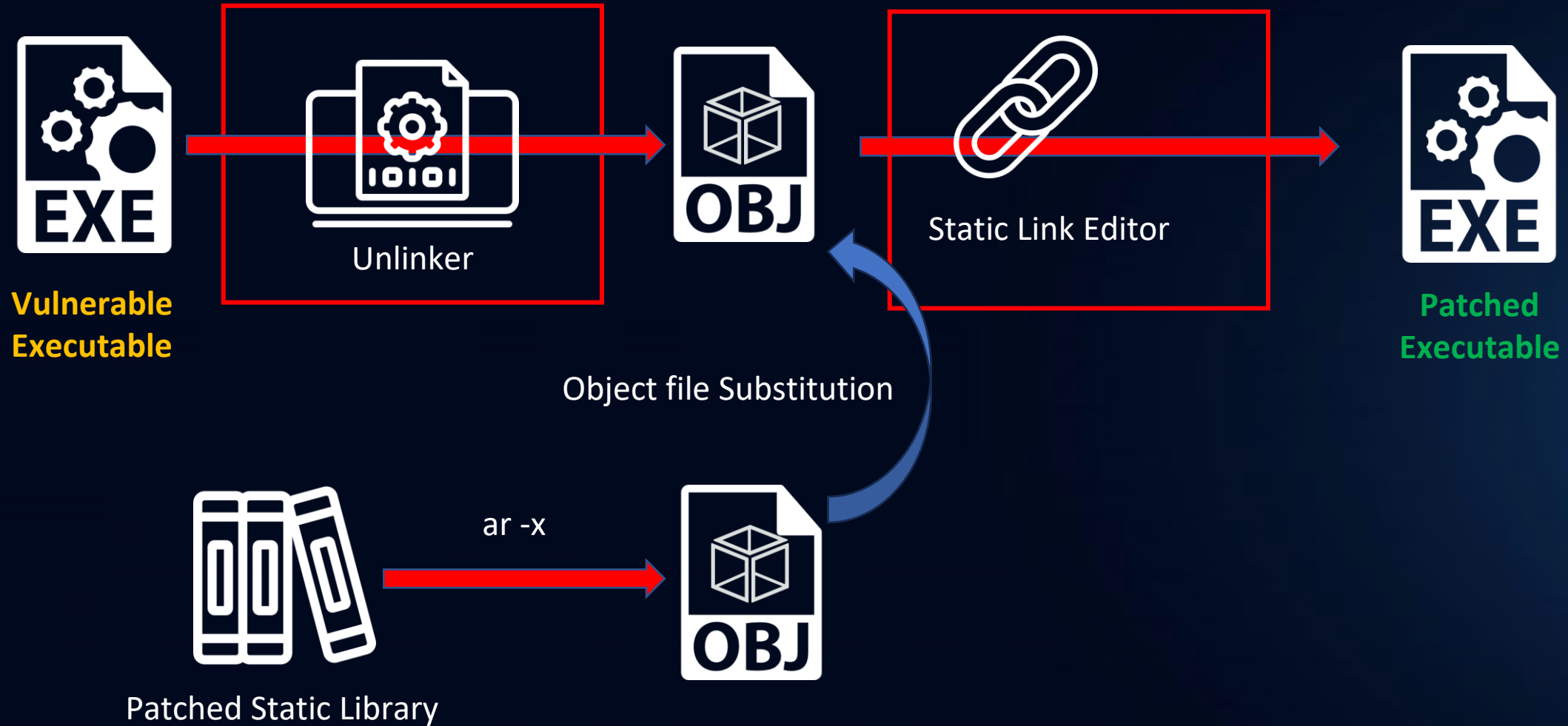
Witchcraft Linker : Libification



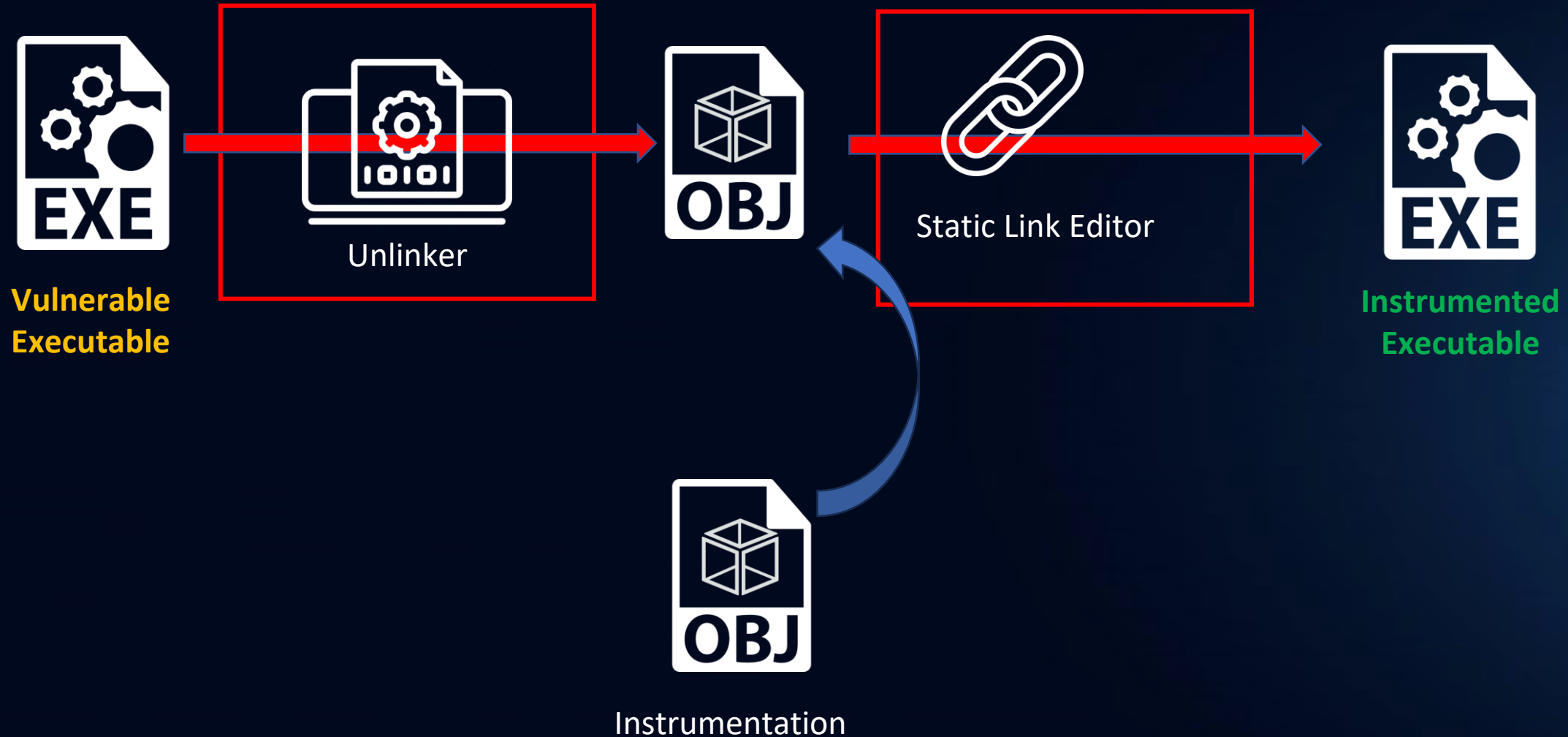
Witchcraft Compiler: Unlinking



Witchcraft Compiler: Unlinking + Relinking = Assured Patching



Witchcraft Compiler: Unlinking + Instrumentation = Fuzzing





le cnam

THANK YOU FOR YOUR ATTENTION

Jonathan Brossard
jonathan.brossard@lecnam.net



Nicolas Kateb

- **Rust pour le trading haute fréquence**



Senior Trader
XBTO



XBTO - Rust Paris 2025

High Frequency Trading with Rust

2025-06-25

HFT desk



June 2025

Business challenges

Why Rust

How

Results



What is high frequency trading?

- A set of complex trading algorithms that places large amounts of orders on multiple markets in fractions of seconds
- Represents between 40% and 70% of volume traded globally across different markets (equities, bonds, cryptos, commodities, FX. etc.)
- Cross team expertise required in trading, data analytics, software development and infrastructure
- Velocity is key, with profitable algorithms half life of a few months before other participants catch up
- Complex systems with many interactions
- Bursty, irregular and unpredictable data spikes
- Latency is key (magnitude of microsecond)

What was our business problem ?

Pre-existing trading system not suited for HFT

- C# tech stack
 - Non-deterministic garbage collection, triggered on activity spikes
 - Reliance on general, non-owned libraries for the critical path, not suited for high-performance contexts
- Microservice architecture, communication through NATS
 - Overall high latency, magnitude of several ms
 - Cannot gracefully absorb sudden spikes of market activity (from 100k/s, to 10M/s market events in a second), scaling is slow
 - Complexity, low development velocity
- Heavy dependency on SQL databases
 - Not suited for time-series oriented data with high insertion throughput
- Not leveraging hardware
 - Every service is containerized, virtualized network

Why use Rust ?

- Good mix between performance (critical for processing millions of events per second) and stability (most of the technical bugs prevented before compilation)
- On par with C++ performance without the compilation and maintenance nightmare
- Can leverage hardware optimization with specific compilation flags
- Resources efficient (RAM, CPU)
- No garbage collection occurring at worst times
- Functional features allow for traders / quants to actively participate in strategy development and to the code base
- Faster development cycles lead to shorter time to market

How we did use it?

- Monolith approach for the main trading engine
- Ring buffer and lock free data structures for critical features - market data distribution, orders events processing, state aware components for risk management
- Hardware timestamping to the nanosecond for precise benchmarking
- Latency tuning with connectivity tweaks
- Custom asynchronous runtime for events processing synchronously without the jitterness of async
- Dedicated decimal crate for number parsing without losing the precision
- Event based and state machine implementation for order management and strategies development

What were the results?

- PoC approach with a small team (3 FTEs)
- Internal latency divided by almost 1000 ! scale of a few microseconds (q99)
- Velocity increased dramatically with both faster compilation times and a quick development process
- Reduction of resources consumption, even with bare metal setup ! 1 server could support 20 times more coverage than the microservices approach
- Benchmark measurement and precision to the nanoseconds allow pinpointing bottlenecks way faster
- Rust itself was not alone.. But it makes a strong case to design a new architecture tailored for business needs
 - Switch to time series database for events logging
 - Monolithic approach and lock free queues
- And it makes for a smooth transition
 - Compiler checks prevent a lot of technical bugs
 - Deployment and testing is a breath
 - Performance is impressive

Disclaimer

This Confidential Information Presentation (the “CIP”) is intended solely for the use of prospective counterparties in determining whether or not to pursue any opportunities with XBTO Crypto Opportunities Fund SAC Ltd., XBTO Global Ltd., XBTO International Ltd., XBTO Trading LLC, XBTO Strategies Ltd, Stablehouse Ltd or any of its group or related companies (together, the “Company”).

XBTO International Ltd and Stablehouse Ltd are licensed to conduct digital asset business by the Bermuda Monetary Authority.

The CIP is of a proprietary and confidential nature and is only being furnished to those parties who have agreed to be bound by the terms and conditions of a Confidentiality Agreement. The information (“Information”) contained herein may not be reproduced or redistributed in whole or in part, in any format, without the express written approval of the Company.

By accepting this CIP, the recipient agrees that it will, and it will cause its directors, officers, employees, consultants, agents and representatives to, use the CIP and all of the information contained therein only to evaluate any opportunities with the Company and for no other purpose and shall return this CIP together with any copies to the Company upon request. This CIP contains confidential material which is non-public information concerning the Company. Receipt of this CIP constitutes your acknowledgment that you will maintain the information contained herein in strict confidence. This CIP does not constitute an offer to sell or the solicitation of an offer to buy any securities or assets of the Company, nor should this CIP be construed to indicate that the business of the Company remains unchanged since the date of this CIP. The Company has no obligations to amend or update this CIP at any time.

All information contained herein has been provided by the Company or other sources that the Company deems reliable; however, the Company has not independently verified certain of the information, including the financial estimates and projections, contained herein.

This CIP includes certain statements, estimates, and projections provided by the Company with respect to its anticipated future performance. Such statements, estimates, and projections reflect various assumptions concerning anticipated results, which assumptions may or may not prove to be correct. Such statements, projections and estimates are not necessarily indicative of current value or future performance, which may be significantly more or less favorable than as reflected herein. No representations or warranties of any kind are intended or should be inferred with respect to any economic return from the Information.

The products, product names, logos, brands, and their trademarks featured or displayed within the CIP are the property of their respective trademark owners. Unless indicated otherwise, the trademark owners are not affiliated with, nor do they sponsor or endorse, the Company nor the Company's securities, products or services.

This CIP does not purport to contain all of the information that may be required to evaluate any transaction, and any recipient hereof should conduct its own independent analysis of the Company and the data contained or referred to herein. The Company has no obligations to provide additional information or to correct or update any of the information set forth in this CIP. The Company makes no representation or warranty, whether express or implied, regarding this CIP, including without limitation the accuracy, completeness, or suitability of the information contained in this CIP, or made available, orally or in writing, in connection with any investigation of the Company, and nothing contained herein is, or shall be relied upon as, a promise or representation, whether as to the past or the future. The recipient should rely solely on the representations and warranties made to it by the Company in the executed definitive agreement regarding the transaction (if any). No party has any obligation (whether express or implied) to offer, accept or negotiate any agreement.

Nothing herein is intended to in any way modify, amend or supersede any of the terms and conditions set forth in any Confidentiality Agreement made between the parties, which remains in full force and effect in accordance with its terms. If you have any questions or need additional information, please contact the Company.

Any investment opportunity herein is not intended for distribution to, or use by, any person or entity in any jurisdiction or country where such distribution or use would be contrary to local law or regulation. Any investment opportunity herein is available only for Non-US Accredited Investors. It is the responsibility of each person outside of the United States to comply with any applicable laws and regulations of their jurisdiction before accessing or using this investment.

Past performance, if any, does not guarantee or indicate future results or returns. The value of investments can fluctuate, and investors may lose all or more than their original investment. Investment outcomes are subject to various market, economic, and other risks that may cause future returns to differ from historical performance.

This material is provided for educational purposes only and does not constitute an offer to buy or sell securities, products, or services in any jurisdiction including without limitation, within Latin America, North America or where such an offer is illegal or to any person for whom this offer is illegal. If any securities, products, and services are mentioned or inferred in this CIP, they may not have been registered with the securities regulator in Brazil, or any other securities regulator in any Latin American country or North America. None of the information discussed herein should be provided to the public in Brazil, any other Latin American Country or North America.

If you have a complaint or concern about any aspects of our products or services provided in the Information, please contact our Customer Support at support@stablehouse.com. Please provide clear details of your nature of complaint and we will promptly acknowledge receipt and investigate as necessary.

RUST PARIS 2025



Pause

Reprise à 11h25



Reyyan Tekin

- Rust et WebAssembly, levier de frugalité et de robustesse pour les traitements de données



Engineering Delivery Manager
Thales



Abstract geometric lines in the top left corner, consisting of several overlapping, irregular polygons and lines in a light beige color.

RUST ET WEBASSEMBLY, LEVIER DE FRUGALITÉ ET DE ROBUSTESSE POUR LES TRAITEMENTS DE DONNÉES

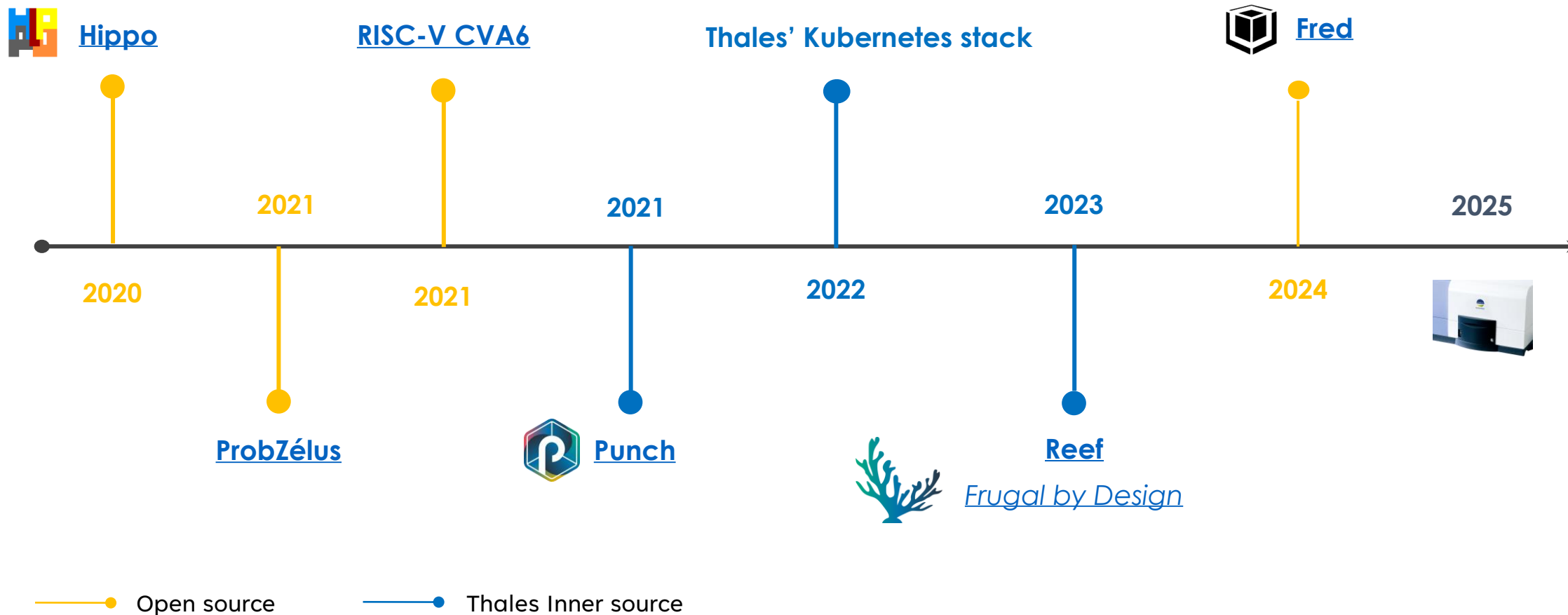
Rust Paris

25 juin 2025



Reyyan TEKIN, Thales

BUILDING TOGETHER, INNOVATING FURTHER





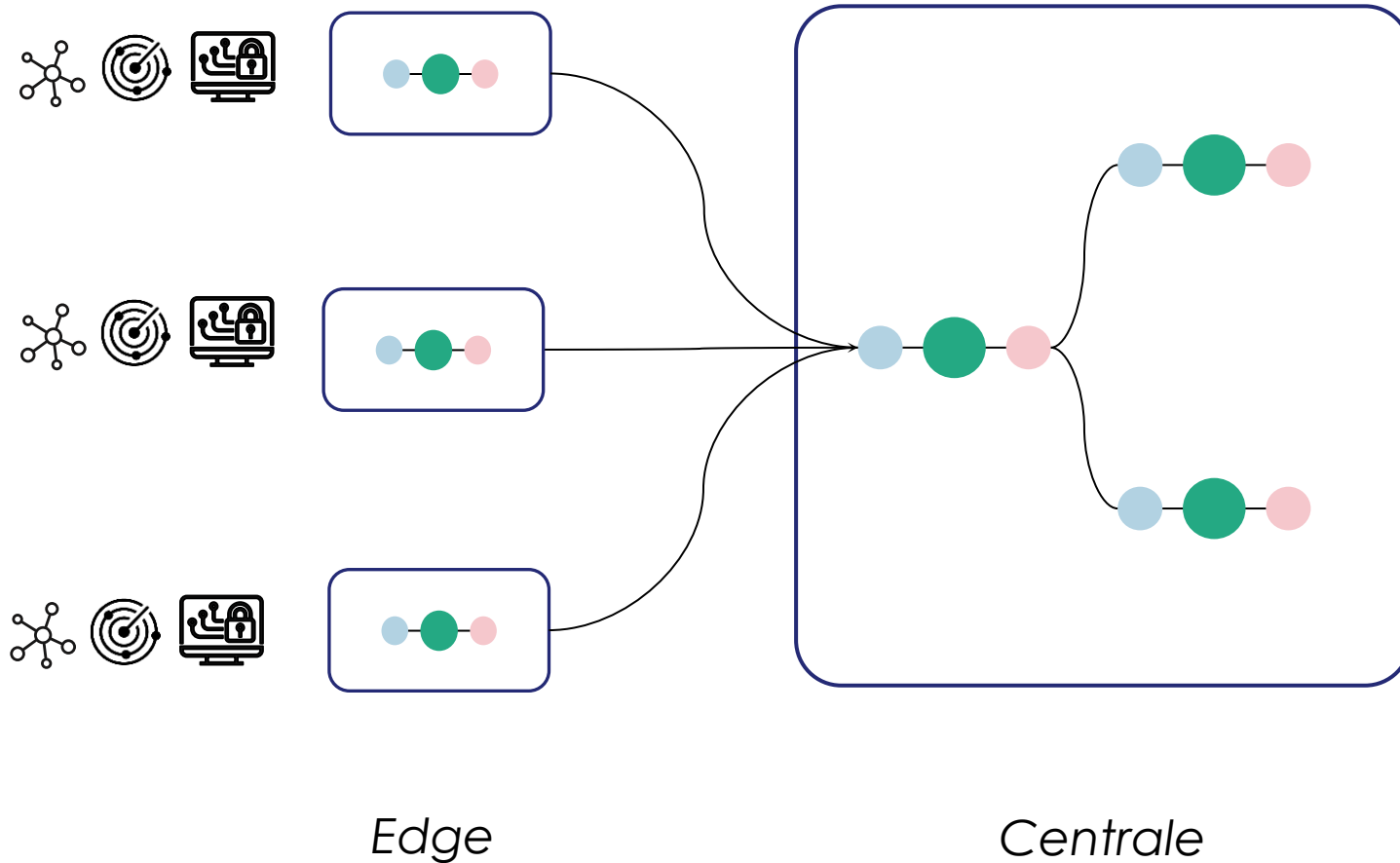
SAUVER LES OISEAUX AVEC RUST ET WASM

Une approche frugale aux
traitements de données

CONCILIER PERFORMANCE, SÉCURITÉ ET LÉGÈRETÉ

- Sobriété numérique
 - Réduire l'empreinte énergétique des traitements de données.
- Edge computing
 - Besoin de traitements locaux, rapides et autonomes.
- Sécurité des données
 - Protection des données sensibles dans des environnements distribués.
- Contraintes métiers
 - Ressources limitées, connectivité intermittente, exigences temps réel.

TRAITEMENTS DE DONNÉES CRITIQUES



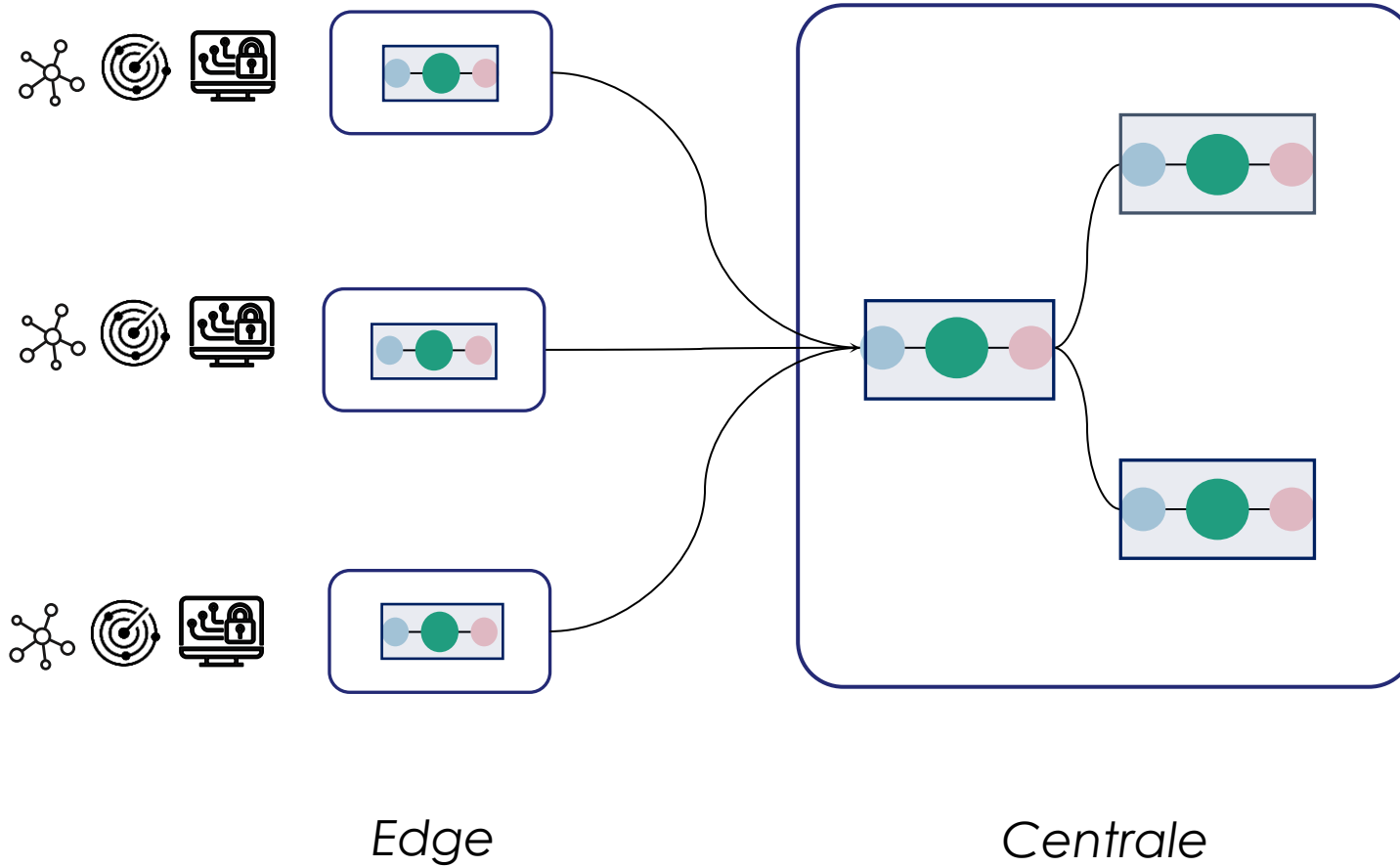
Cas d'usage

- Cybersécurité
- Capteurs critiques
- Systèmes satellite
- Smart industries

Fonctions (FaaS)

- Formatage
- Enrichissement
- Identification
- Détection

CHALLENGES



Portabilité

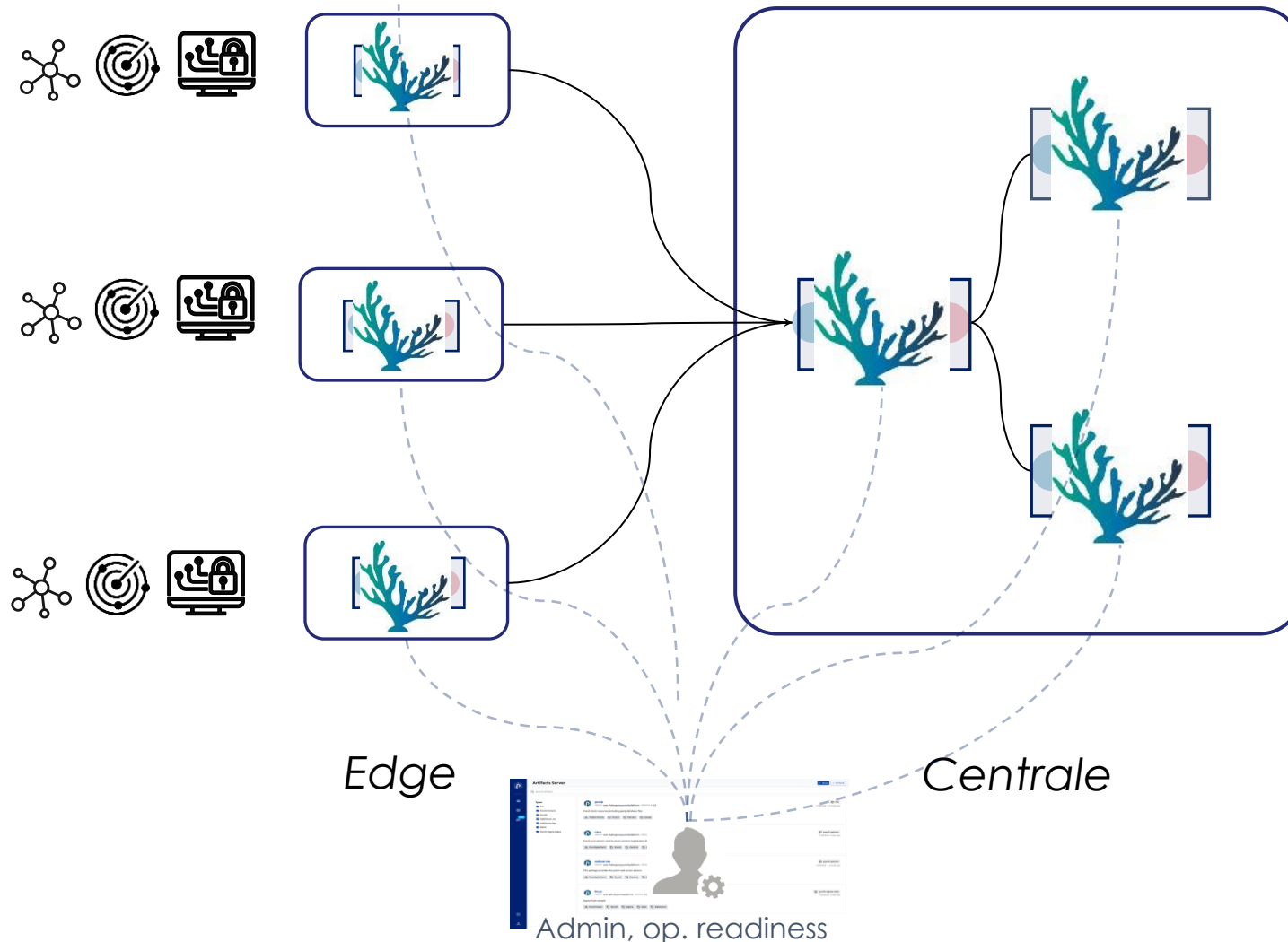


Sûreté &
sécurité



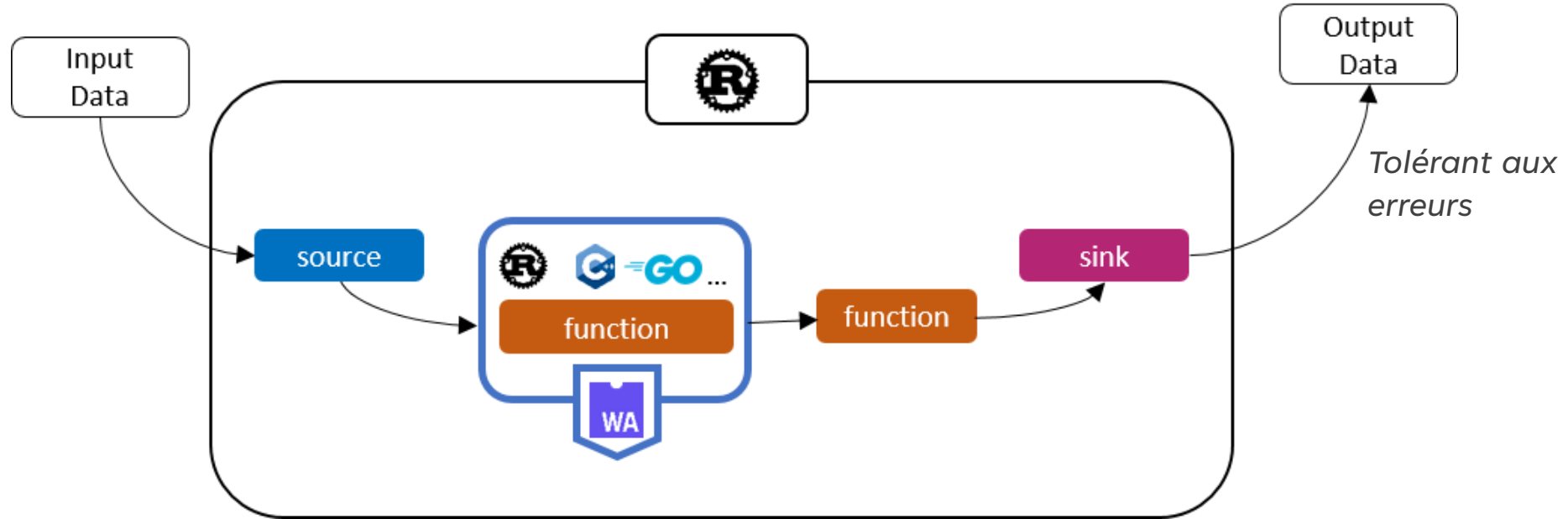
Faible
empreinte

REEF : RESOURCES EFFICIENT FUNCTIONS



- Codé en Rust
- 25+ connecteurs : encryption, Tensorflow Lite, Kafka, ...
- SDK pour les connecteurs customs Wasm
- Même composant en edge et en centrale

REEF : COMBINER RUST ET WASM POUR UNE ARCHITECTURE ROBUSTE

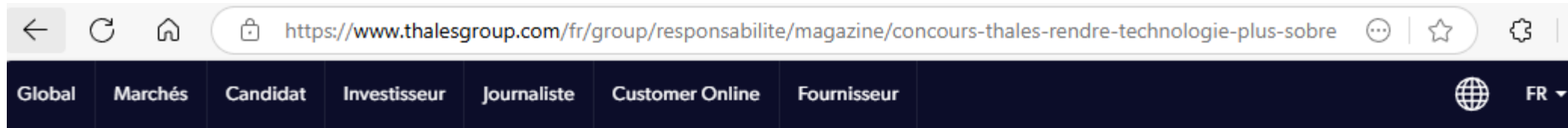


Rust : Sûreté, hautes performances



WebAssembly (Wasm) : ultra léger, sécurée, multi-langage, multi-plateforme

REEF, STRATÉGIQUE



“Reef représente un système de fonction de traitement de données à faible empreinte et haute performance pour des applications modernes d’analyse de données distribuées, avec des exigences strictes en matière de sûreté et de sécurité.”
Olivier Flous



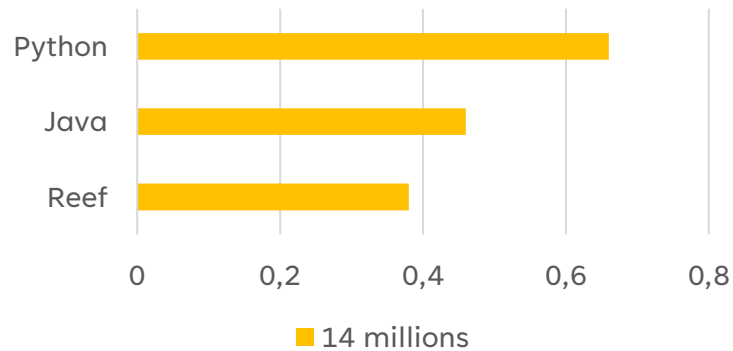
Thales “Frugal by Design” Contest



Thales Innovation Award

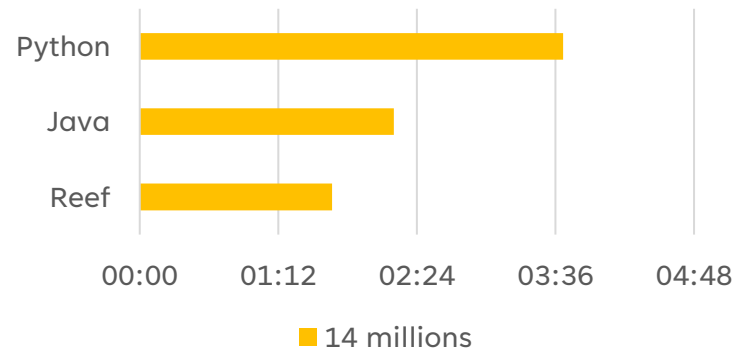
DES PREMIERS BENCHMARKS PROMETTEURS

Consommation d'énergie (Wh)



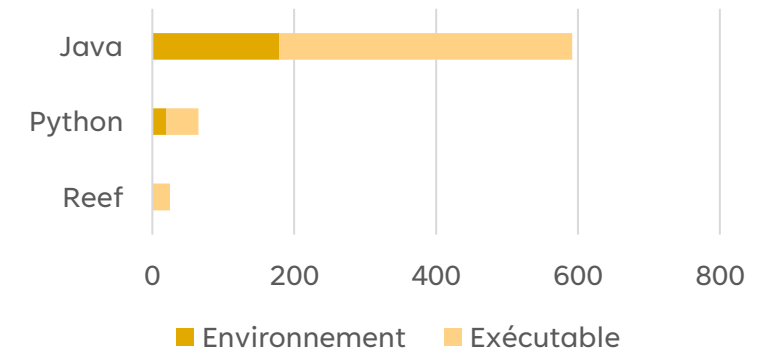
73% plus économe que FaaS Python
18% plus économe que FaaS Java

Temps d'exécution (min)

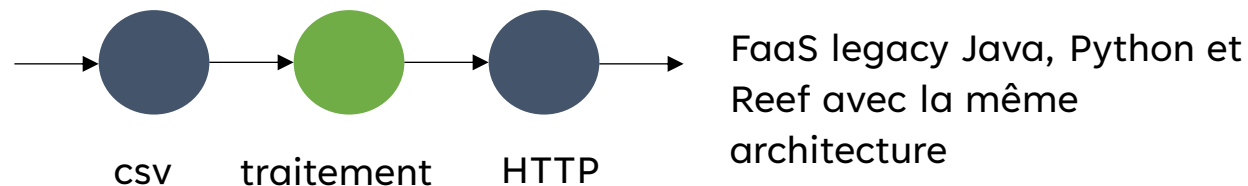


120% plus rapide que FaaS Python
32% plus rapide que FaaS Java

Poids de l'application (Mo)



x23 plus léger que FaaS Java
x2,6 plus léger que FaaS Python



RETOUR D'EXPÉRIENCE

Ce qui a bien fonctionné

- Déploiement fluide sur architectures hétérogènes (x86, ARM, RISC-V).
- Faible empreinte mémoire et CPU.
- On embarque uniquement l'essentiel grâce à Wasm.
- Mises à jour dynamiques et sécurisées des fonctions.

[Rust, WebAssembly, TensorFlow, and RISC-V: a powerful cocktail for edge AI.](#) D. Tombroff, punchplatform.com

Difficultés rencontrées

- Debug Wasm encore limité (code et compilation)
- Chaîne de cross-compilation complexe à maintenir.
- Certaines bibliothèques Rust non compatibles ou partiellement supportées en Wasm.
- Courbe d'apprentissage Rust pour les équipes non initiées.

[GitHub - ThalesGroup/rust-coding-dojo: Gather ressources to practice Rust language](#)



VOYONS UN CAS D'USAGE

ÉTUDE DES OISEAUX AUTOUR DES ÉOLIENNES

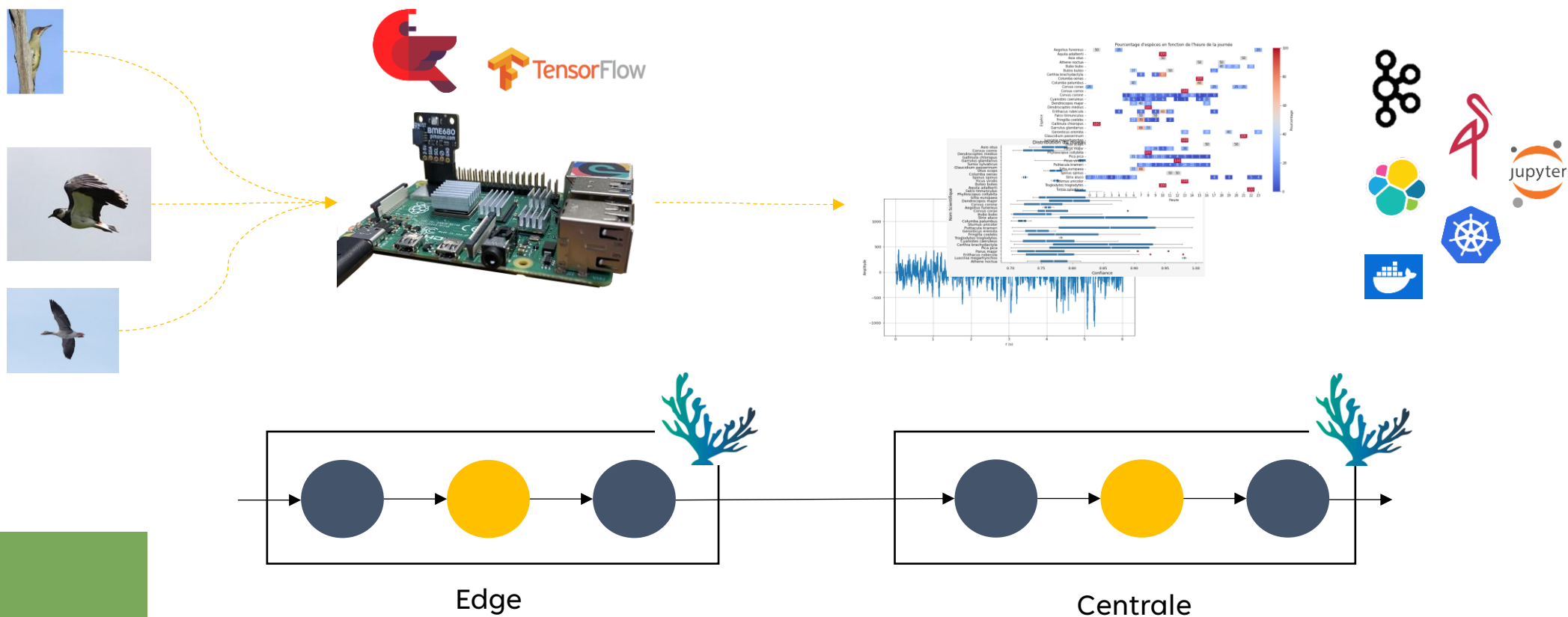


Éoliennes Offshore

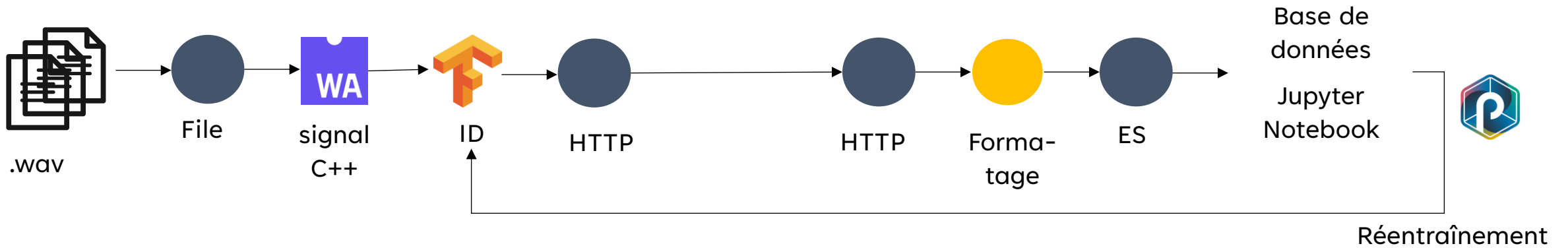
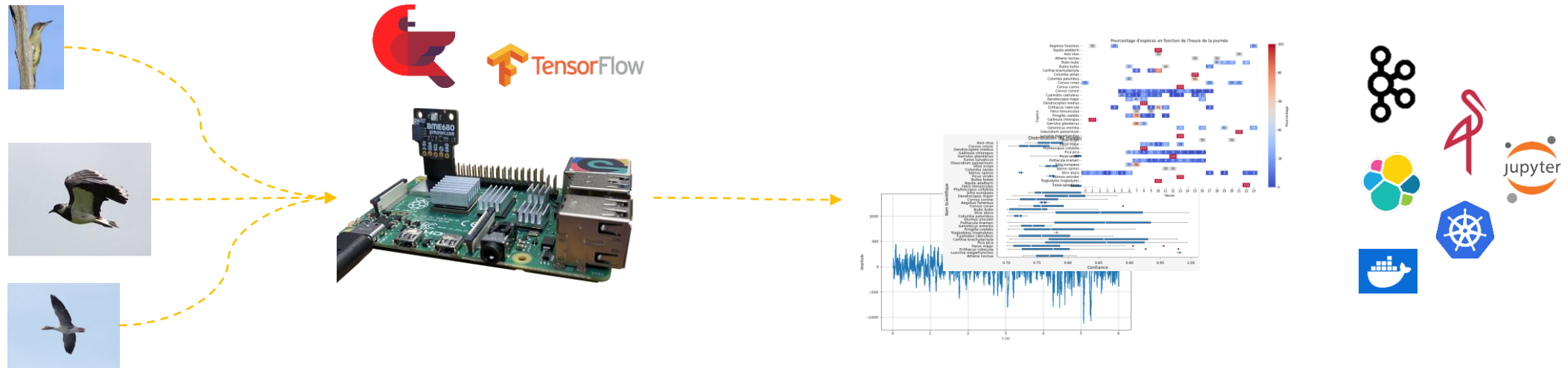


Experts à une localisation spécifique, à un moment précis

BEAK : BIODIVERSITY AND ENVIRONMENTAL ANALYTICS FOR BETTER KNOWLEDGE



BEAK : ARCHITECTURE



GESTION DE FLOTTE



Dashboard

Artifacts

Configurations

Lineage

Sigma Rules

Alerts

Devices

Settings

Profile

Devices

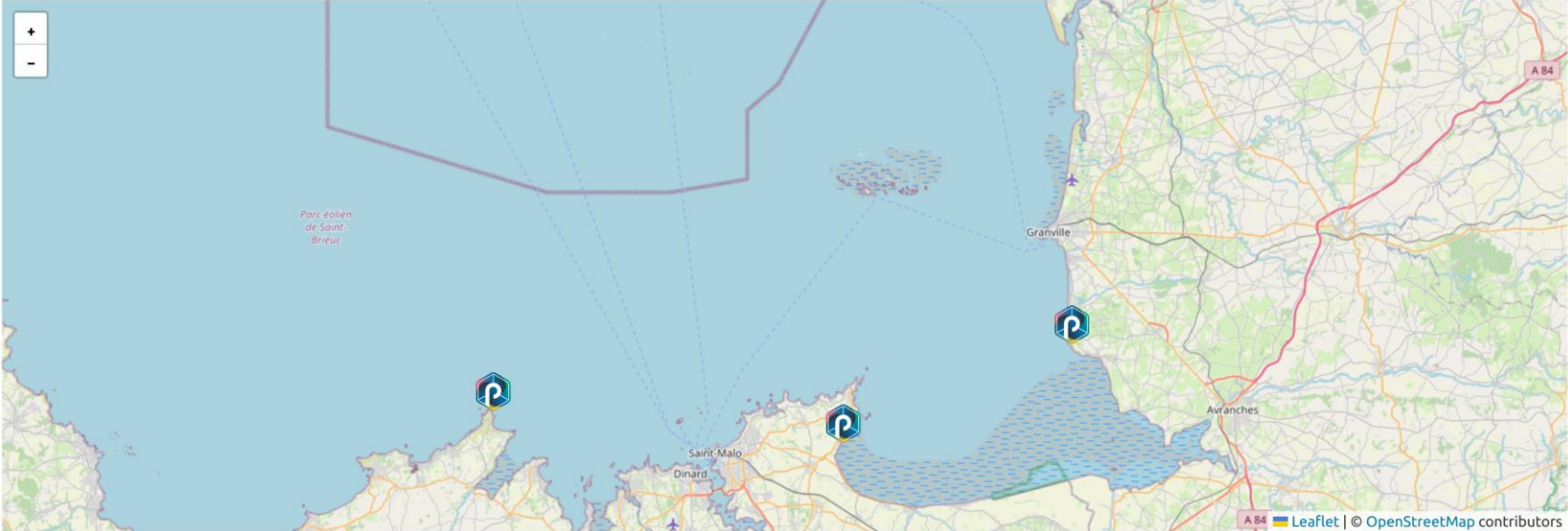
Delete

Refresh

5s

Enable Activity

Enable Map



Search Alert

Options

Name ↑↓	Device ID ↑↓	Device type ↑↓	Synchronisation status ↑↓	Health status ↑↓	
bird-identification-ltr	zone-europe-west1-bird-capfrehel-1	RLINE	sync	ACTIVE	
bird-identification-ltr	zone-europe-west1-bird-cancale	RLINE	sync	ACTIVE	
bird-identification-ltr	zone-europe-west1-bird-capfrehel	RLINE	sync	ACTIVE	
bird-identification-ltr	zone-europe-west1-bird-champeaux	RLINE	sync	INACTIVE	

TRAITEMENT DE DONNÉES EN EDGE



Dashboard

Artifacts

Configurations

Lineage

Sigma Rules

Alerts

Devices

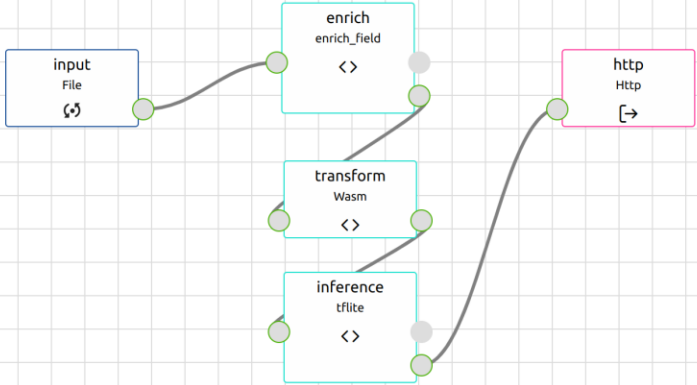
Settings

Profile

bird-identification-ltr



5s



```
graph LR; input[input File] --> enrich[enrich enrich_field]; enrich --> transform[transform Wasm]; transform --> inference[inference tflite]; inference --> http[http Http];
```

ConfigurationDependencies

Name

bird-identification-ltr

Image

ghcr.io/punchplatform/rline:0.1-dev

Engine

Rline

Kind

StreamPunchlineBatchPunchline

DÉPLOYER ET METTRE À JOUR LES FONCTIONS



Dashboard

Artifacts

Configurations

Lineage

Sigma Rules

Alerts

Devices

Settings

Profile

Artifacts Server

+ New

Delete

Refresh

Search artifacts

Types

☒ File

☒ Punch Parsers

☒ Model

☒ Additional Jar

☒ Additional Pex

☒ Patch

☒ Punch Sigma Rules

☒ Wasm

Groups

☒ io.model

☒ com.github.punchplatform

☒ com.github.rline

**vehicle-predictor**
GROUP: io.model VERSION: 1.0

model

Published : 17 days ago

Vehicle predictor model pandas(2.0.0), sklearn(1.1.1)

**starter-sigma-rules-wasm-compiled**
GROUP: com.github.punchplatform VERSION: 0.0.0-dev

wasm

Published : 9 days ago

This artifact is a wasm module generated from rust code

Punch

Rust

Wasm

Generated

**ip-anonymization**
GROUP: com.github.rline VERSION: 0.1.0

wasm

Published : 15 days ago

This package provides ip anonymization functions.

Punchplatform

Rust

WebAssembly

Cybersecurity

**melspectrogram**
GROUP: com.github.rline VERSION: 1.0.0

wasm

Published : 10 days ago

Signal processing: melspectrogram (C++)

Punchplatform

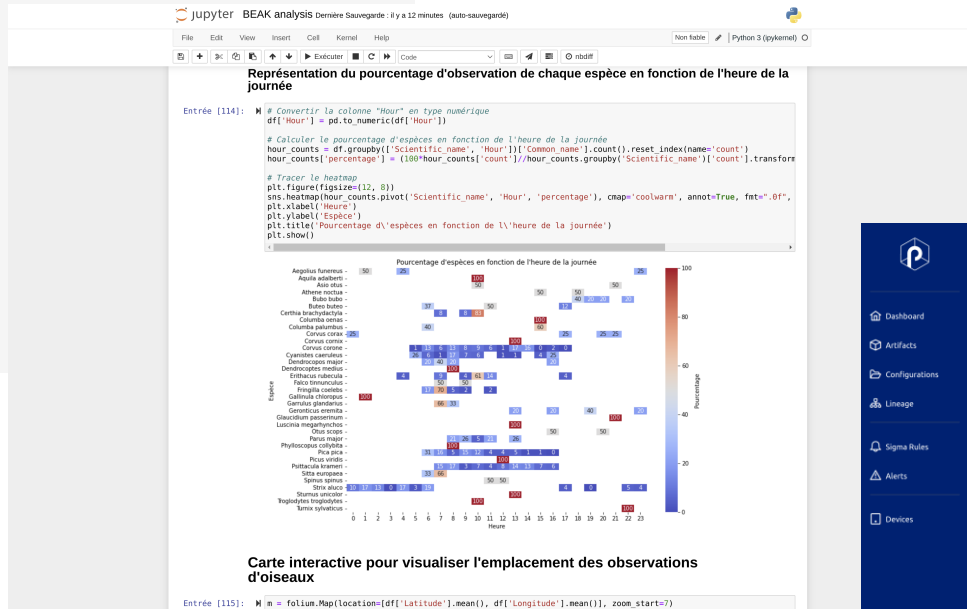
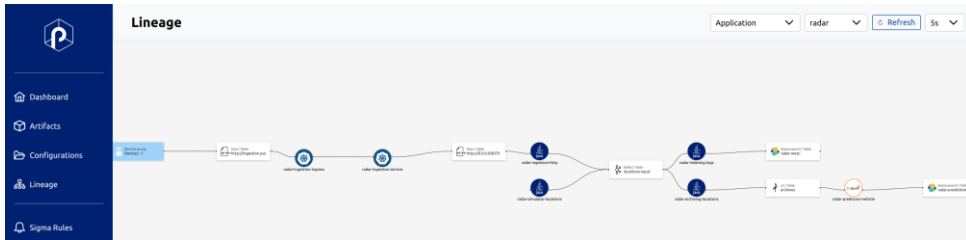
Rust

WebAssembly

IoT

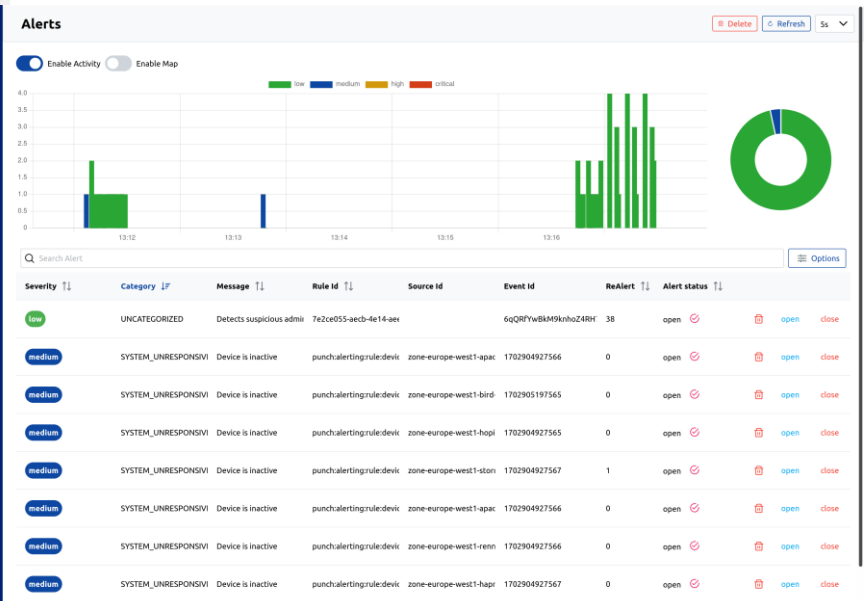
Signal processing

TRAITEMENT DE DONNÉES EN CENTRALE



Pipeline d'ingestion

Jupyter Notebook



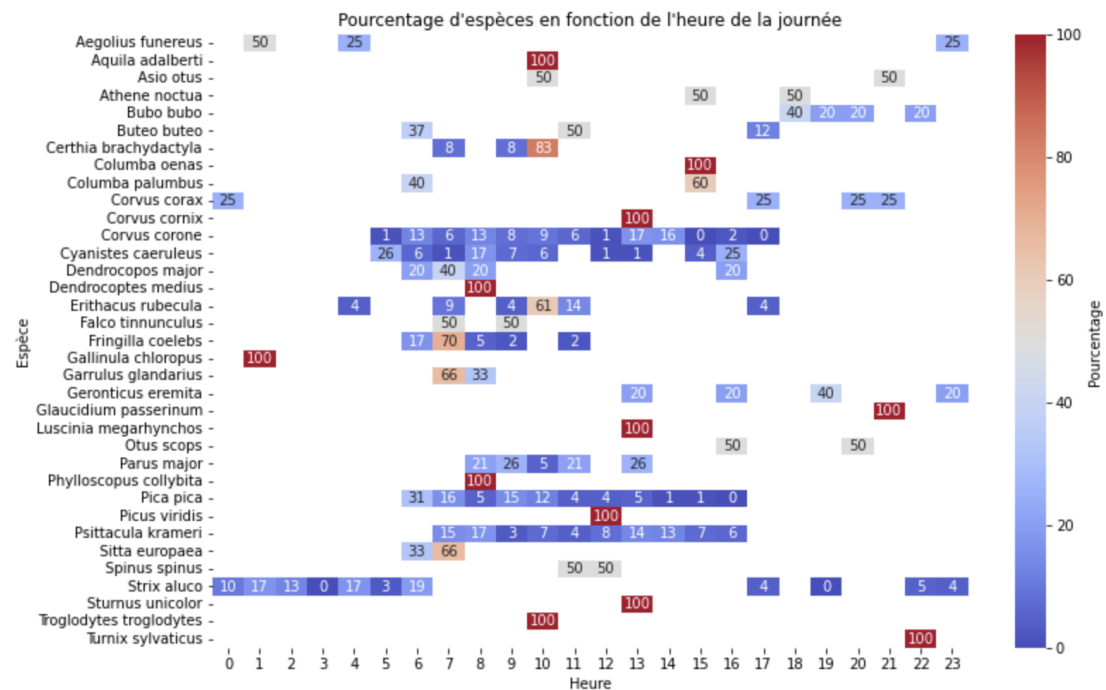
Règles d'alerte

Représentation du pourcentage d'observation de chaque espèce en fonction de l'heure de la journée

```
Entrée [114]: # Convertir la colonne "Hour" en type numérique
df['Hour'] = pd.to_numeric(df['Hour'])

# Calculer le pourcentage d'espèces en fonction de l'heure de la journée
hour_counts = df.groupby(['Scientific_name', 'Hour'])['Common_name'].count().reset_index(name='count')
hour_counts['percentage'] = (100*hour_counts['count']/hour_counts.groupby('Scientific_name')['count'].transform('sum'))

# Tracer le heatmap
plt.figure(figsize=(12, 8))
sns.heatmap(hour_counts.pivot('Scientific_name', 'Hour', 'percentage'), cmap='coolwarm', annot=True, fmt=".0f",
            plt.xlabel('Heure')
            plt.ylabel('Espèce')
            plt.title('Pourcentage d\'espèces en fonction de l\'heure de la journée')
            plt.show()
```



Carte interactive pour visualiser l'emplacement des observations d'oiseaux

DANS LA CONTINUITÉ



- Renforcement de la collaboration avec Centrale Nantes
- Module de cours dispensé en mai
- Premiers résultats en cours sur l'IA miniature (ANR OWL)

DANS LA CONTINUITÉ

Reef fait partie de Frugal IT



THAles Low Carbon : capteurs et tableaux de bords

Fred : on-going Thales Open Source Assistant-architecte cloud multi-agent LLM

The screenshot shows the Frugal IT interface. The sidebar on the left contains a search bar, a dropdown menu with 'rift', and a list of navigation options: Inspect, Explain, Facts, Audit, Chat, Optimize, and Profile. The main content area on the right displays a text block with the following content:

numériques. Dans cette perspective, le référentiel général de l'écoconception des services numériques vise à établir une base de connaissances et recommandations commune permettant d'accompagner les démarches volontaires d'écoconception. Outil d'action et de sensibilisation, il poursuit quatre objectifs principaux :

1. Concevoir des services numériques plus durables permettant d'allonger la durée de vie des

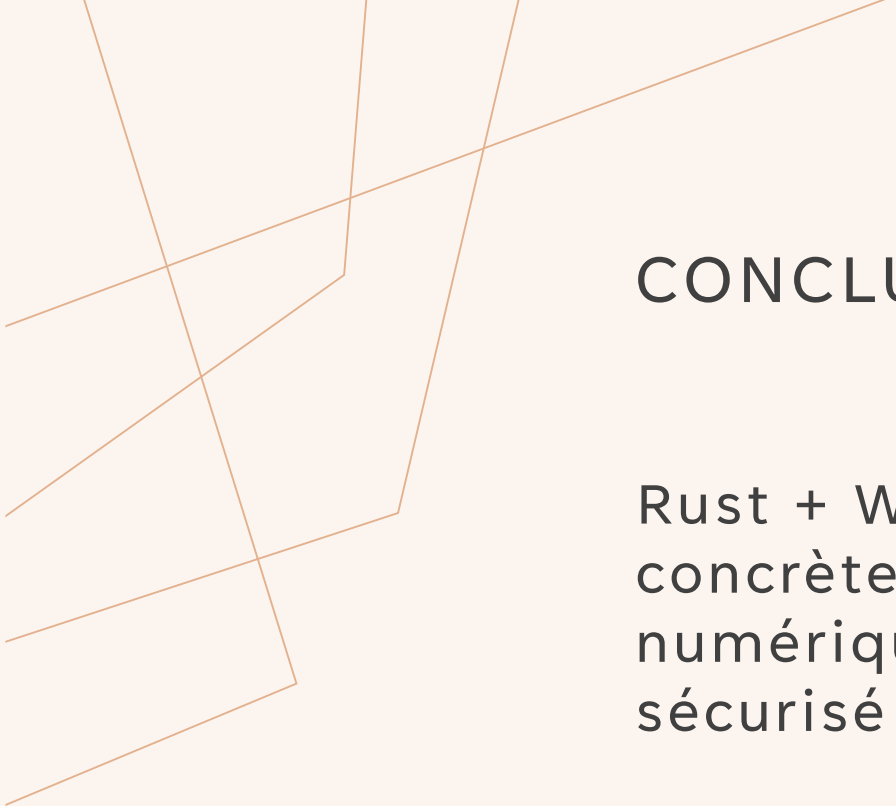
The main eco-design principles for digital services, as outlined in the "Référentiel Général d'Ecoconception des Services Numériques," include:

1. **Designing Sustainable Digital Services:** This involves extending the lifespan of digital devices by ensuring services are compatible with older devices, optimizing for various usage contexts, and using open-source solutions to prolong service life and device usage.
2. **Promoting Environmental Sobriety:** This principle focuses on reducing the negative impacts of the attention economy by limiting features that encourage excessive use, giving users control over their usage, and minimizing data collection for advertising profiling.
3. **Reducing Resource Usage:** This involves optimizing data traffic and infrastructure use, minimizing multimedia content size, reducing server requests, and using efficient hosting solutions to lower the environmental footprint of digital services.
4. **Increasing Transparency:** Encouraging the publication of reliable environmental indicators to promote the development of sustainable solutions and inform users about the environmental impact of their digital usage.

These principles aim to reduce the environmental impact of digital services throughout their lifecycle, from design to usage, by focusing on sustainability, efficiency, and transparency.

At the bottom of the interface, there is a chat input field with the placeholder text "Enter your question here..." and a "Light" toggle switch.

Fred design and architecture. T. Jouannic, D. Tombroff, L. Gerardi and R. Tekin, fredk8.dev



CONCLUSION

Rust + Wasm : une voie
concrète vers un
numérique frugal et
sécurisé

Sécurité : Rust élimine les erreurs
mémoire, Wasm isole les fonctions.

Frugalité : faible empreinte CPU/RAM,
idéal pour l'edge.

Portabilité : même code sur ARM, x86,
RISC-V, cloud ou embarqué.

Ce qu'il faut maîtriser
pour réussir

Debugging Wasm.

Chaîne de **cross-compilation** complexe.

Écosystème Rust/Wasm encore jeune
pour certains usages (IA par exemple).

A series of thin, light brown lines forming an abstract geometric pattern on the left side of the slide. The lines intersect to create various polygons and shapes, some of which are filled with a light beige color.

MERCI

Reyyan TEKIN

Thales

Jean-Jacques Lafay



*Expert en
developement logiciel
Secretarium France*

Etienne Bossé



*Chef de plateforme
ingénierie
Secretarium France*

■ Rust & WebAssembly : Sécurité et Portabilité du Cloud aux Enclaves



Rust & WebAssembly: Sécurité et Portabilité du Cloud aux Enclaves

Étienne Bossé & Jean-Jacques Lafay, Secretarium | Juin 2025



Qui sommes nous?



Etienne Bossé
Responsable Produit
Secretarium/Klave



Jean-Jacques Lafay
Développeur Senior
Secretarium/Klave

Qui sommes nous?

Le talk entre vous et votre
pause déj !



Vous êtes ici!



Agenda

- Secretarium et sa mission
- Notre contexte technique et le rôle de Rust
- Rust & C++: Intégration de Rust dans des enclaves
- Rust & WebAssembly: Modèle en composant et écosystème
- Démo: Construction d'une application Rust qui tourne dans une enclave sécurisée.

Secretarium: Mission



Nous accompagnons nos clients sur les cas d'usage suivants :



Collaboration sur données sensible



Sécurité et confidentialité des données



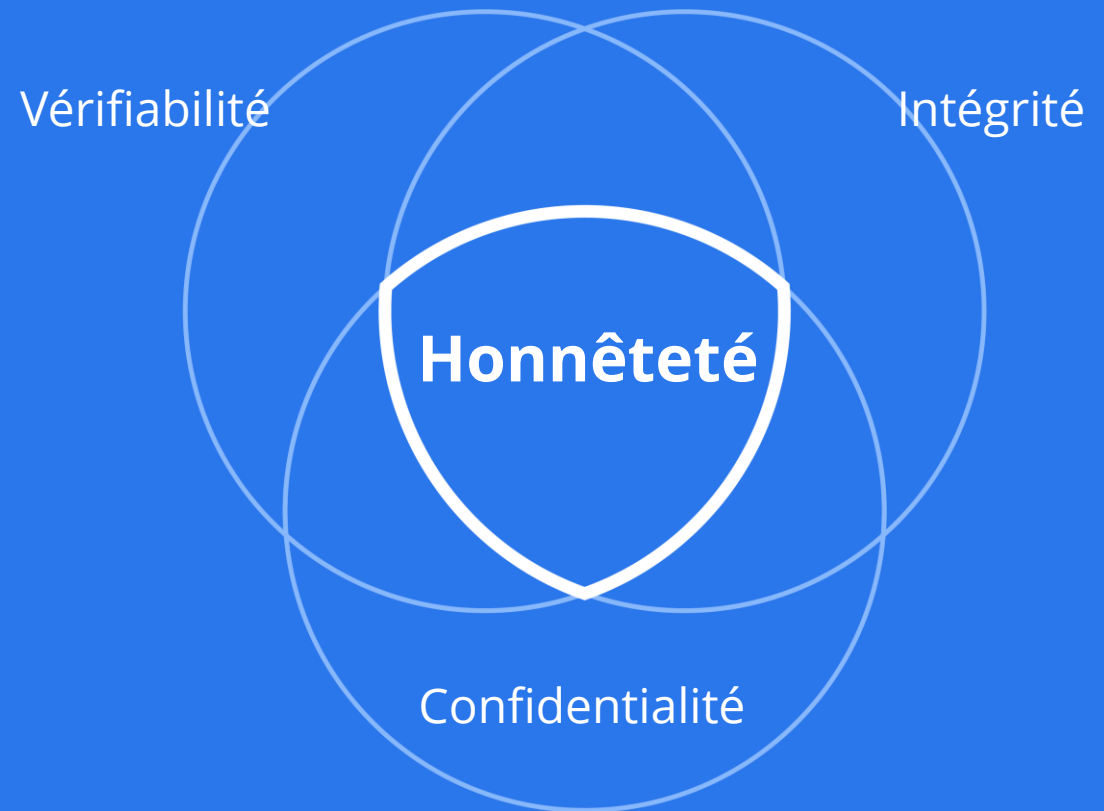
Conformité réglementaire



IA Confidentielle

Secretarium: Systeme Honnête

« Apporter de la transparence sur la manière dont les données sont protégées, gérées et utilisées. »



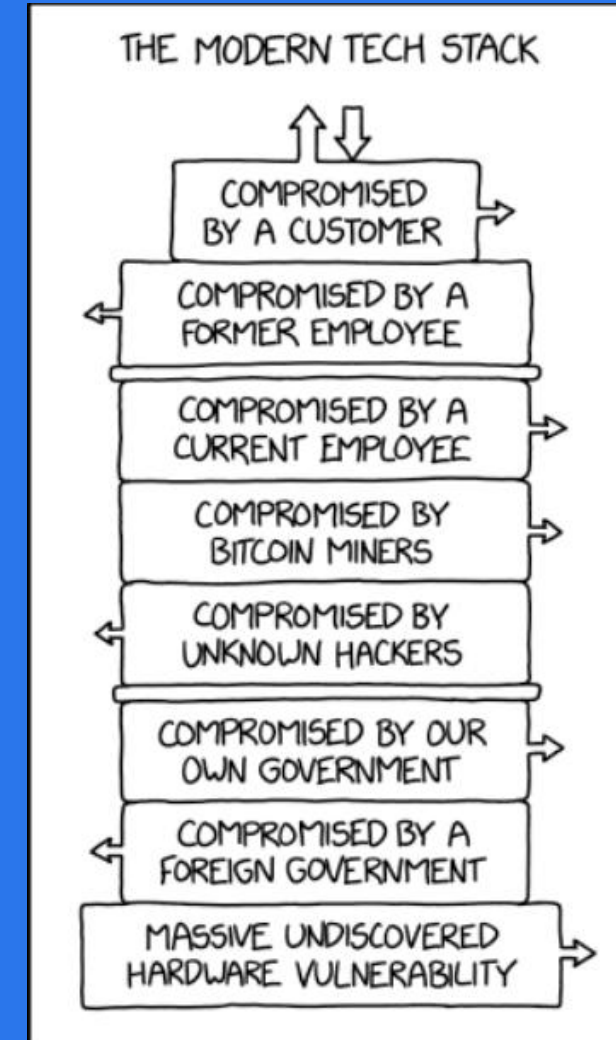
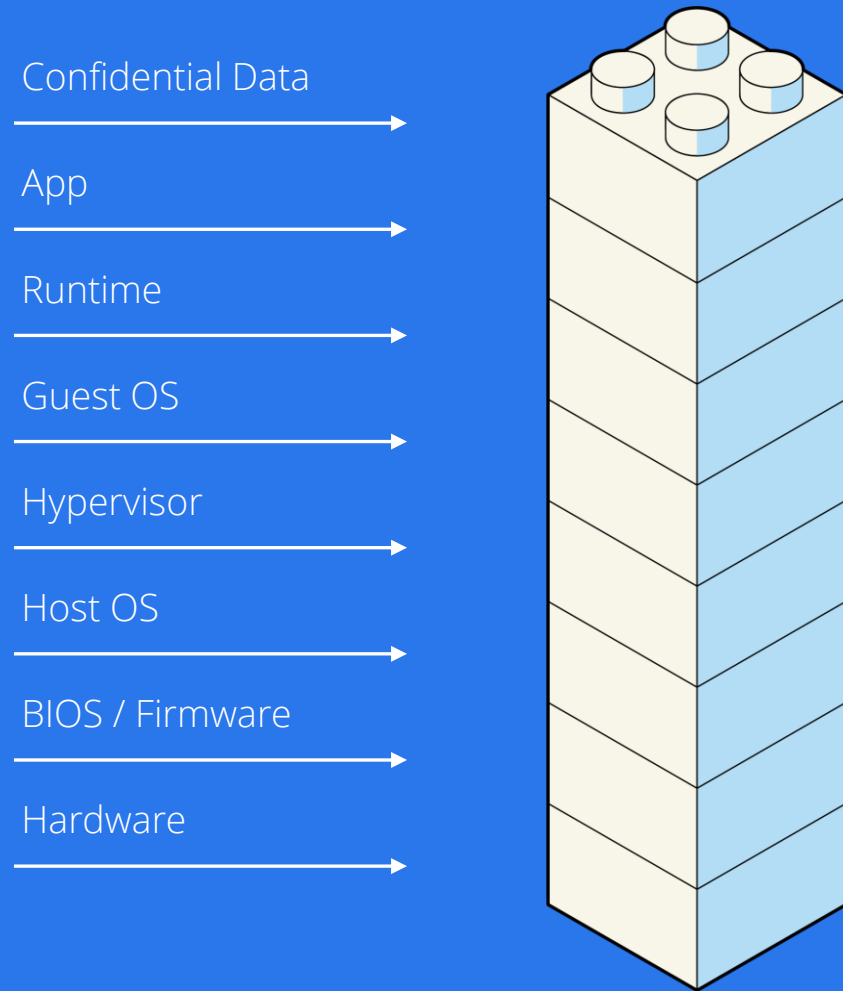
Caractéristiques d'un Système honnête



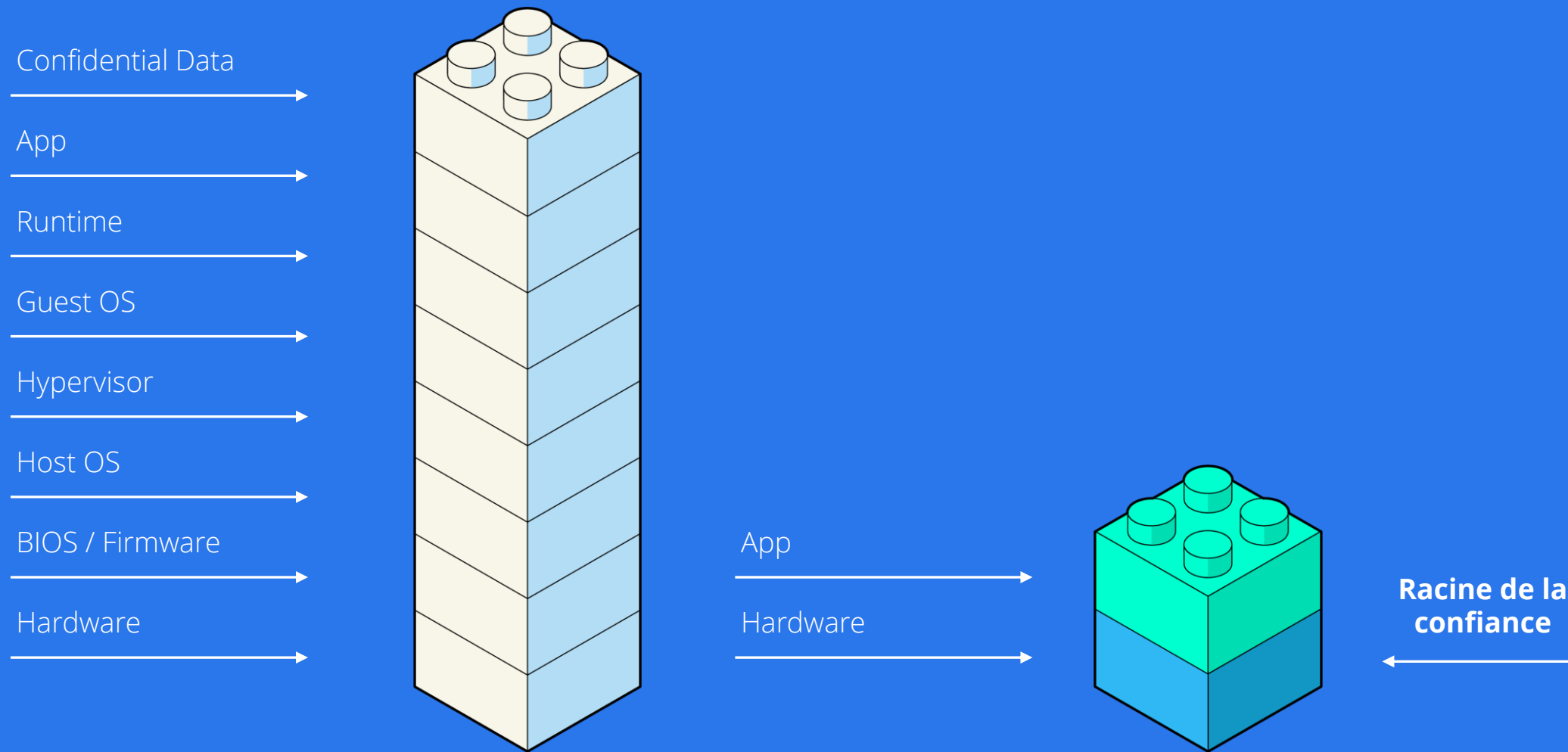
Principales caractéristiques d'un système honnête :

- 1- Garder l'**intégrité** des données et du code en tout temps
- 2- Garder la **confidentialité** des données et du code en tout temps
- 3- Garder un comportement vérifiable et **déterministe** en tout temps
- 4- Et tout ça, de manière **vérifiable** et **infalsifiable**

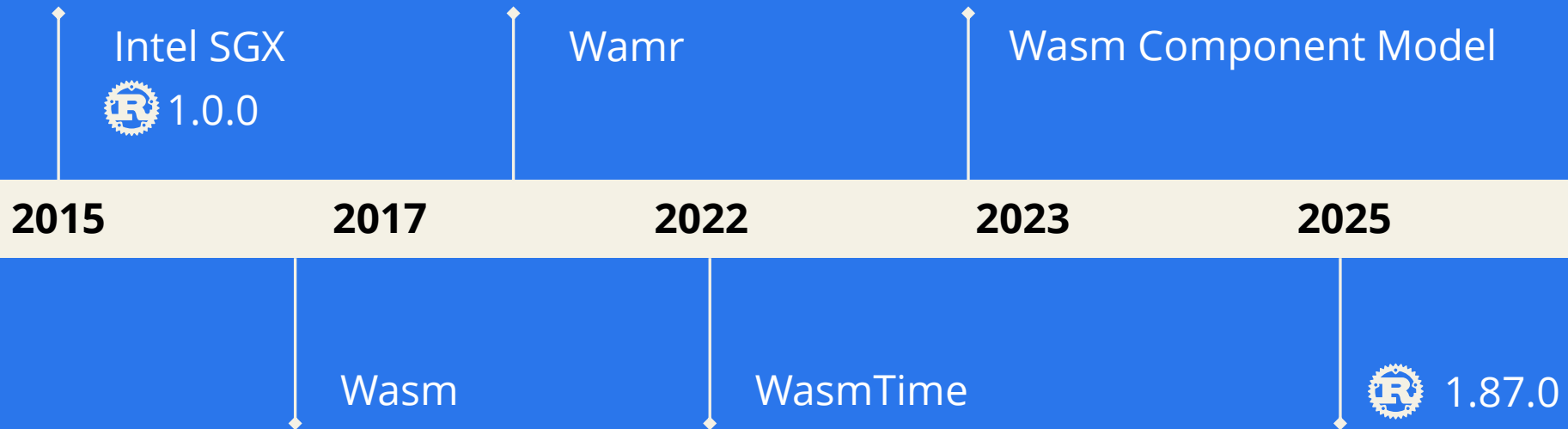
Construire un Système Honnête



Construire un Système Honnête



Construire un Système Honnête: Technologies



WASM + TEE + DLT = Honest Computing

Informatique Confidentielle et Environnements d'Exécution de Confiance

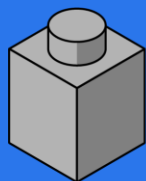


Concepts clefs:

- ✓ Sécurité Hardware
- ✓ Protège l'intégrité du code et des données en utilisation
- ✓ Apporte le concept d'enclave sécurisée

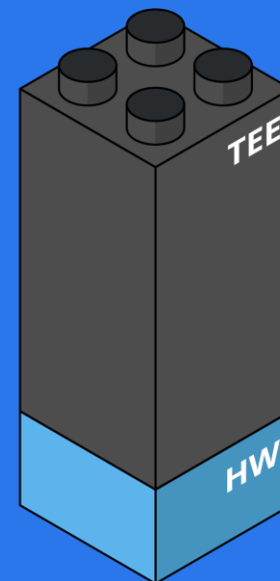
Au repos

Chiffrement au repos



En utilisation

Exécuter des traitements dans les enclaves sécurisées. Ce qui se passe dans un TEE reste dans le TEE.

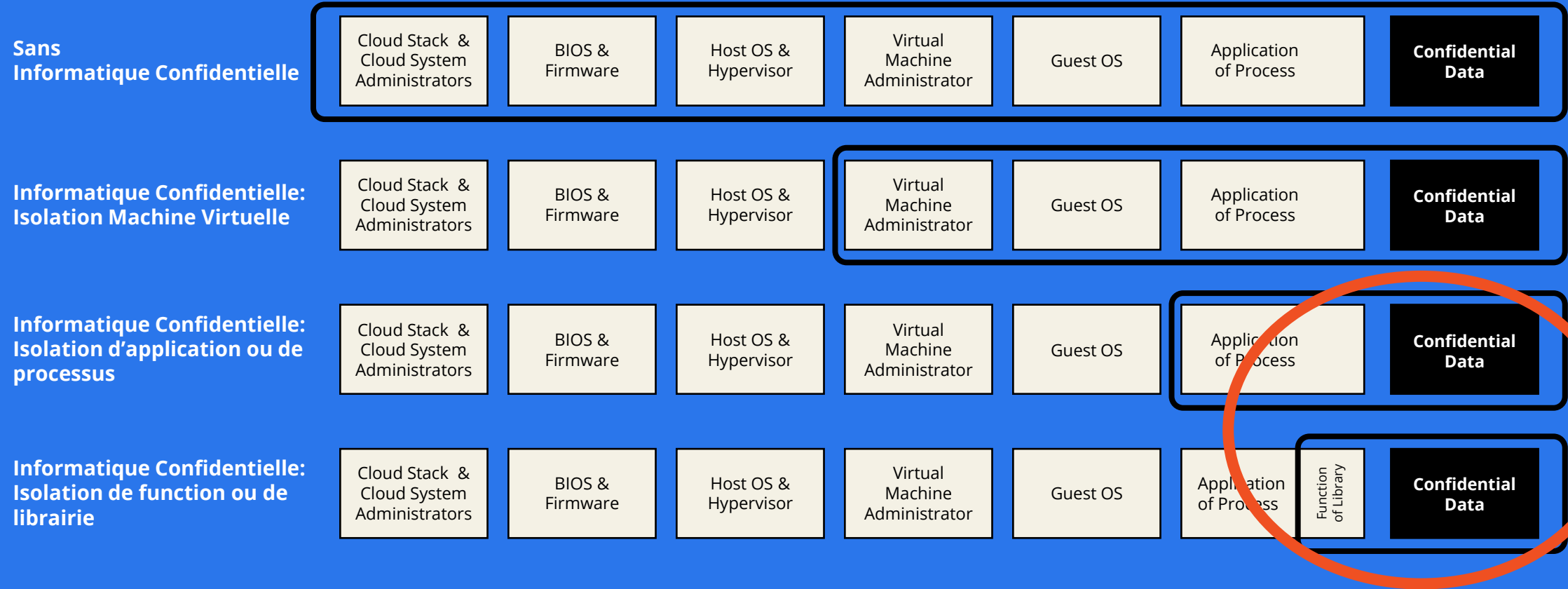


En transit

Transport sécurisé et chiffré de bout en bout

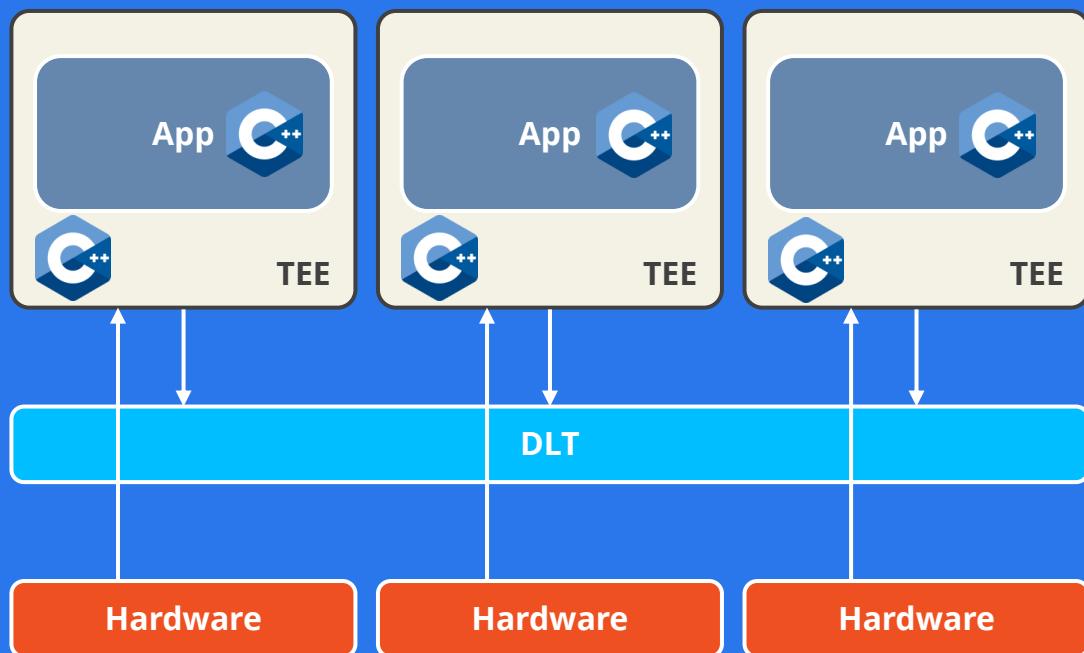


Informatique Confidentielle et Environnements d'Exécution de Confiance



Notre Architecture Originelle

Distribuée & Décentralisée

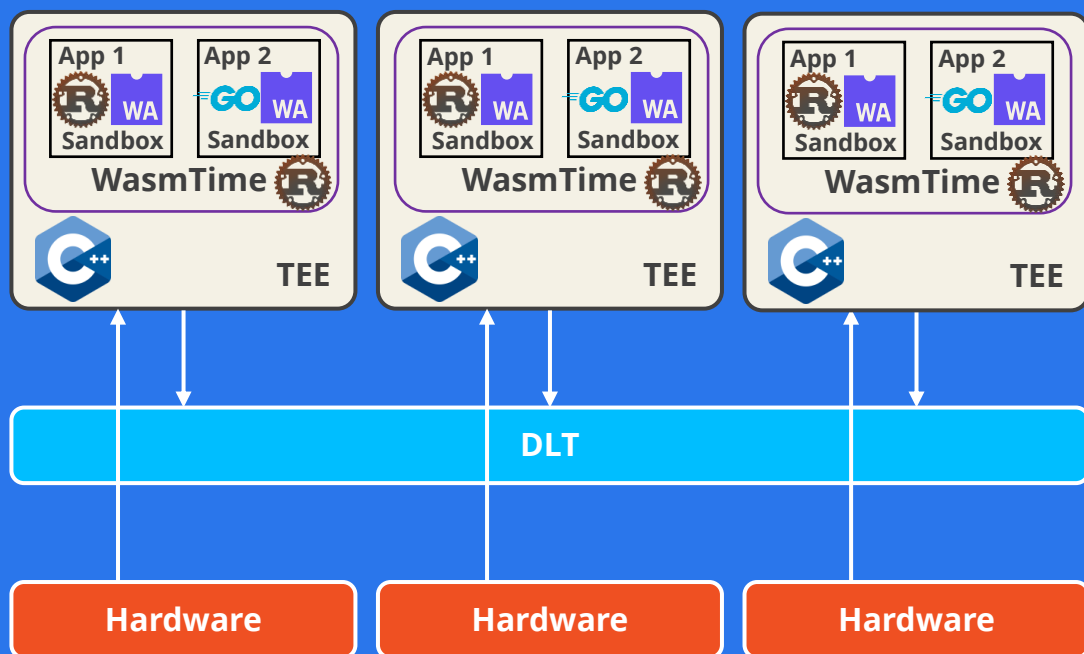


Défis à adresser:

- ✓ Complicé à construire/utiliser
- ✓ Démarrage lents
- ✓ Pas de mutualisation

Notre Architecture Actuelle

Distribuée & Décentralisée



Éléments Clefs:

- ✓ Intégration Rust & C++
- ✓ Intégration de Wasm
- ✓ Utilisation de Rust avec Wasm

Rust & C++ Intégration

Motivations:



Intégrer un environnement d'exécution Wasm



Cela va permettre de :

- ✓ Faciliter l'adoption par les devs
- ✓ Mutualiser les apps dans des enclaves
- ✓ Réduire les temps de démarrage



MVP: WAMR

- ✓ Intégration facile avec SGX
- ✓ En C++



Limitations:

- ⊗ Pas de Component Model
- ⊗ Difficile d'utiliser le mode compilé

Rust & C++ Intégration

Passage à Wasmtime:



Migrer vers WasmTime (construit en Rust).



Cela va permettre de :

- ✓ Supporter le Wasm Component Model
- ✓ Atteindre des perfs natives pour les apps
- ✓ Préparer la migration vers Rust



Les avantages de Rust:

- ✓ Intégration facile avec le C++
- ✓ Sûreté mémoire
- ✓ Plus d'UB en pratique
- ✓ Performances comparables à C++

Rust & C++ Intégration

Rust dans une enclave SGX, challenges et solutions:



Utiliser Rust dans nos enclaves, tout en conservant l'infra C/C++ existante.



Contraintes:

- ➞ Architecture et modèle de sûreté des enclaves
- ➞ Utilisation du SDK SGX (C/C++)
- ➞ Solutions limitées en Rust:
 - Fortanix (trop spécifique pour notre usage mixte)
 - Teaclave SGX SDK: Peu maintenu



Solution: Forker Teaclave -> TeaKlave

- ✓ Fourni les fonctions core et std de Rust
- ✓ Compatible avec les versions récentes du SDK SGX

Rust & C++ Intégration (dans une enclave)

TeaKlave, le pont entre les deux mondes :



Intégrer Wasmtime avec TeaKlave dans l'enclave.



Intégration de Wasmtime avec TeaKlave:

- ✓ Utilisation d'une target spécifique pour Wasmtime
- ✓ Implémentation des fonctions utilisées par le code Rust via la target en utilisant le SDK SGX
 - Allocation de mémoire & protection de pages
 - Gestion des traps Wasm via signaux (setjmp/longjmp)
- ✓ Extension de la Wasmtime C-API pour supporter le Component Model.

Rust & C++ Intégration (dans une enclave)

Retour d'expérience:



Ce qu'on a accompli:



- ✓ Enclaves hybrides C/C++ & Rust grâce à notre fork Teaklave
- ✓ Intégration de Wasmtime via le support platform-custom
- ✓ Extension de la C-API pour exposer le Component Model



Prochaines étapes:

- ➞ Finaliser le support Component Model
- ➞ Améliorer la compatibilité WASI
- ➞ Réduire les frictions CI (Rust nightly, Docker multi-user)

Rust + WebAssembly pour les Apps

Pourquoi Rust ?

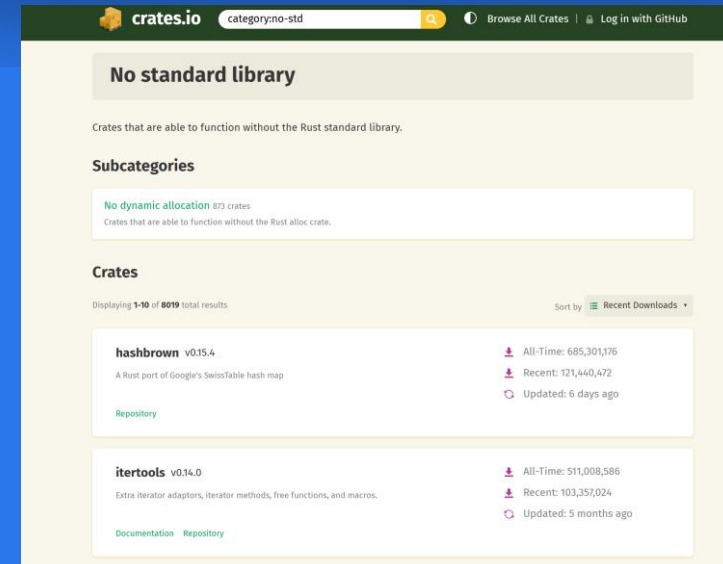


Les avantages de Rust pour les Apps:

- ✓ Compilation native vers WebAssembly
- ✓ Performance du code généré
- ✓ Écosystème mature & tooling solide
- ✓ Sécurité renforcée : sandbox Wasm + garanties Rust



```
cargo build --target wasm32-unknown-unknown --release
```



Rust + WebAssembly pour les Apps

Wasm Component Model & WIT



Pourquoi utiliser le Component Model ?

- ✓ Dépasse les limites des modules Wasm classiques : fini les fonctions à types primitifs uniquement
- ✓ Expose des interfaces riches (chaînes, listes, structures...) via WIT
- ✓ Interopérabilité entre langages : Rust, C, Go... se comprennent
- ✓ Isolation mémoire renforcée : plus d'export de mémoire brute → communication uniquement via interfaces

Rust + WebAssembly pour les Apps

Wasm Component Model & WIT

```
package klave:sdk;  
  
interface sdk {  
    // Notifications  
    notify: func(msg: string);  
  
    // Ledger  
    read-ledger: func(table: string, key: list<u8>) -> result<list<u8>, string>;  
  
    // ...  
}
```

```
package component:rust-example;  
  
/// An example world for the component to target.  
world rust-example {  
    import klave:sdk/sdk;  
    export greet: func(user: string);  
}
```



```
#[allow(warnings)]  
mod bindings;  
  
use bindings::Guest;  
use klave;  
struct Component;  
  
impl Guest for Component {  
    fn greet_user(user: String) {  
        let Ok(city) = sdk::read_ledger("user_city", &user) else {  
            sdk::notify(&format!("failed to read from ledger: '{}'", &user));  
            return;  
        };  
        let msg = if city.is_empty() {  
            format!("the key '{}' was not found in table my_table", user)  
        } else {  
            format!(  
                "Hello {} from city {}",  
                user,  
                String::from_utf8(city).unwrap()  
            )  
        };  
        sdk::notify(&msg);  
    }  
}  
  
bindings::export!(Component with_types_in bindings);
```

Rust + WebAssembly pour les Apps

Retour d'expérience:

 Rust n'est pas juste compatible avec WebAssembly, il semble avoir été conçu pour lui.



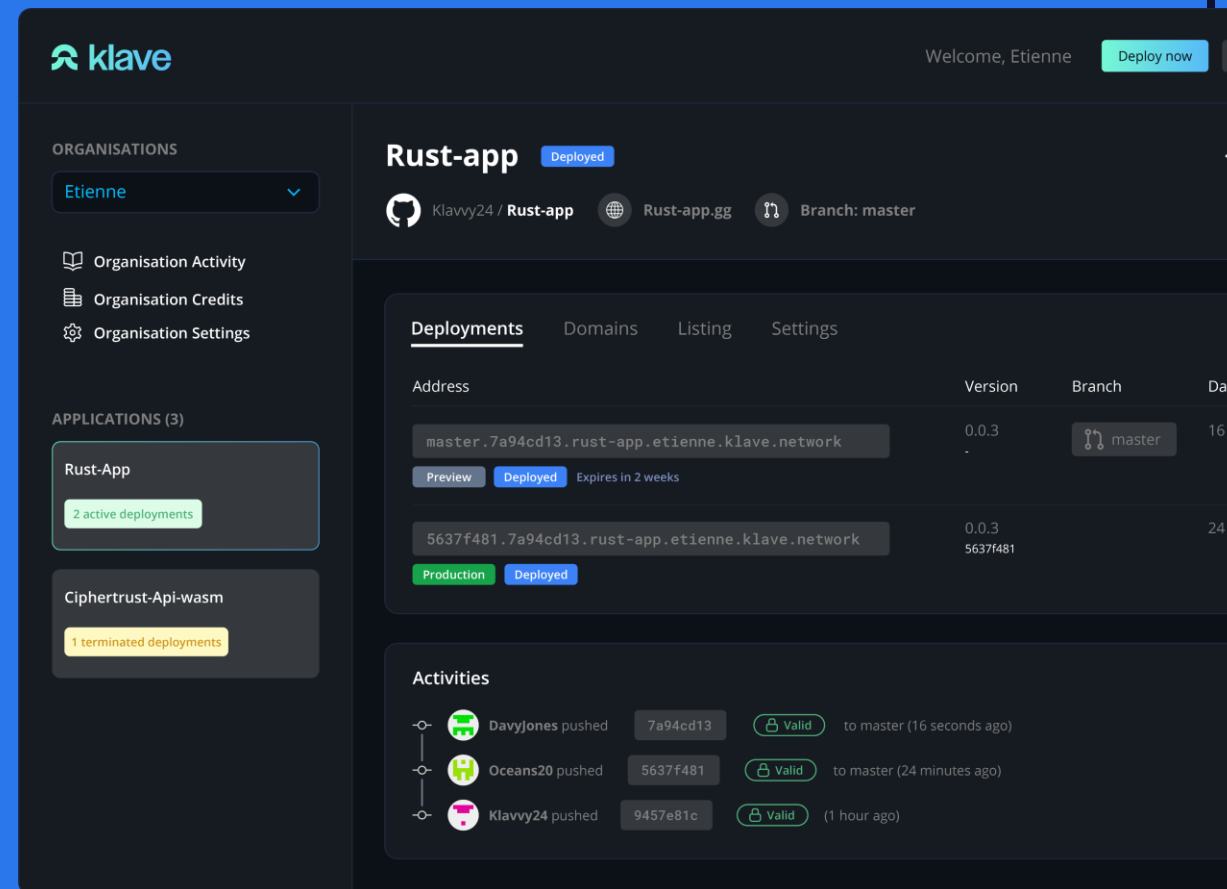
En revanche, il reste des défis à adresser:

- ➞ L'utilisation de librairies Rust conduit vite à de gros fichiers Wasm
- ➞ Sans Component Model la gestion de la mémoire est plus délicate
- ➞ Les builds Rust vers Wasm sont difficiles à reproduire (attestation)

Démo

Démo sur la plateforme Klave:

- ✓ Création d'un composant écrit en Rust
- ✓ Déploiement sécurisé dans une enclave (TEE)
- ✓ Interaction avec le composant via son interface



The screenshot displays the Klave dashboard for a user named Etienne. The interface is divided into several sections:

- Header:** Klave logo, user name "Welcome, Etienne", and a "Deploy now" button.
- Organisations:** A dropdown menu showing "Etienne".
- Applications (3):** A list of applications with their deployment status:
 - Rust-App:** 2 active deployments (green button).
 - Ciphertrust-Api-wasm:** 1 terminated deployment (yellow button).
- Deployments:** A table showing deployment details for the Rust-app.

Address	Version	Branch	Deployment Type
master.7a94cd13.rust-app.etienne.klave.network	0.0.3	master	Preview (Deployed, Expires in 2 weeks)
5637f481.7a94cd13.rust-app.etienne.klave.network	0.0.3	5637f481	Production (Deployed)
- Activities:** A list of recent events:
 - DavyJones pushed 7a94cd13 (Valid) to master (16 seconds ago)
 - Oceans20 pushed 5637f481 (Valid) to master (24 minutes ago)
 - Klavvy24 pushed 9457e81c (Valid) (1 hour ago)

Conclusion et Q&A

Rust est au ❤️ de notre architecture

- ✓ Sécurité mémoire et isolation garanties
- ✓ Interopérable avec notre base C++ via Teaklave
- ✓ Déploiement WebAssembly dans des enclaves SGX
- ✓ Component Model + interop multi-langage activés

Q&A

RUST PARIS 2025



Systematic

Paris Region Deep Tech Ecosystem

Cocktail Déjeunatoire

Reprise à 14h



OCaml **PRO**

THALES
Building a future we can all trust

AdaCore

CÉNOTÉLIE

 **zetta**
scale



XBTO

RUST PARIS 2025



Arthur Carcano

Ingénieur R&D Rust |

Docteur en mathématiques et informatique

Ocamlpro



Jean-François Bortolotti

- **Rust dans l'automobile :
efficacité, sécurité, sûreté, et
attractivité**



Software Expert Leader
Ampere (Renault Group)





AMPERE

Renault
Group

RUST IN CARS

SOFTWARE-DEFINED EFFICIENCY AT SCALE

RUST PARIS 2025

FRÉDÉRIC AMEYE/JEAN-FRANCOIS BORTOLOTTI – AMPERE (RENAULT GROUP)

Who am I ?

Jean-François Bortolotti

Expert Leader Embedded Software – Ampere (Renault Group)

Passionate about technology. Started coding in 1984 when computer Ram was in kilobytes 😊.

32 years of experience in Software: Mobile Telecommunications and Automotive industry.

Cross domains expertise: Hardware and Software, Embedded and Cloud Software.

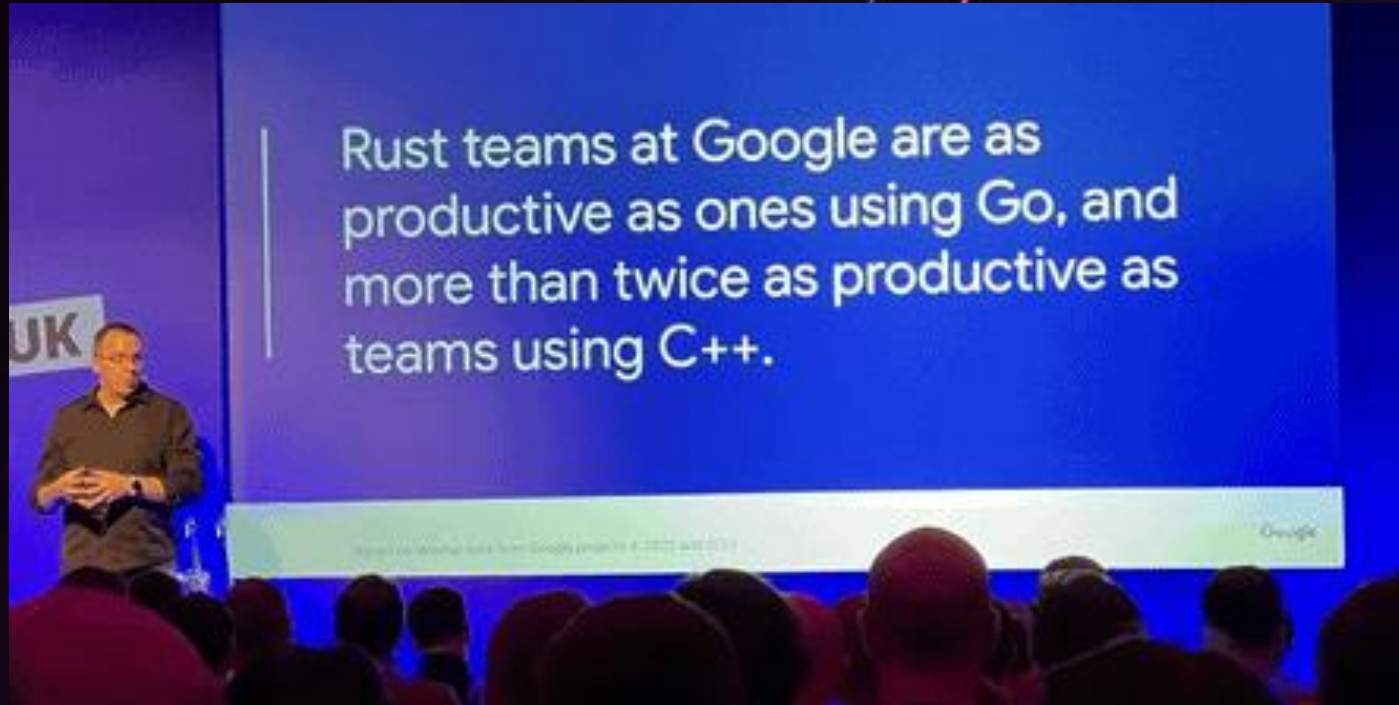
Focus, Data driven, Hands On and Efficient leader.

Worked at IBM, Philips, Motorola, Google, Intel, Renault/Ampere, ...

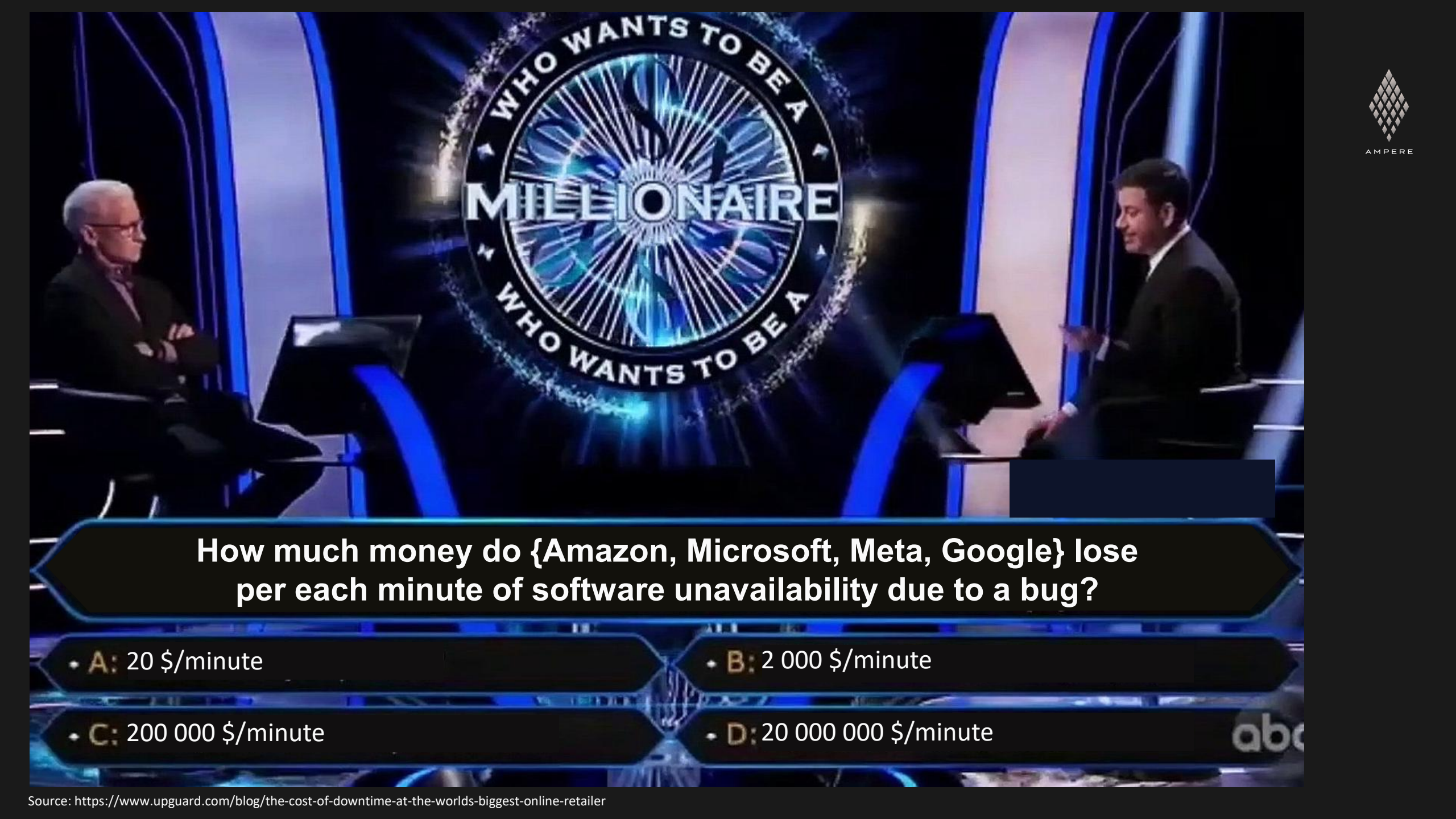
Note: Frédéric AMEYE is the main author of this presentation.

Fact: many critical infrastructures in cloud/tech use Rust.

Cloudflare/AWS Firecracker, Amazon Prime Video, Microsoft Hyperlight, Microsoft Pluton, Google Android drivers, Cloudflare Pingora,...



Is it just to follow some hype?



How much money do {Amazon, Microsoft, Meta, Google} lose per each minute of software unavailability due to a bug?

• A: 20 \$/minute

• B: 2 000 \$/minute

• C: 200 000 \$/minute

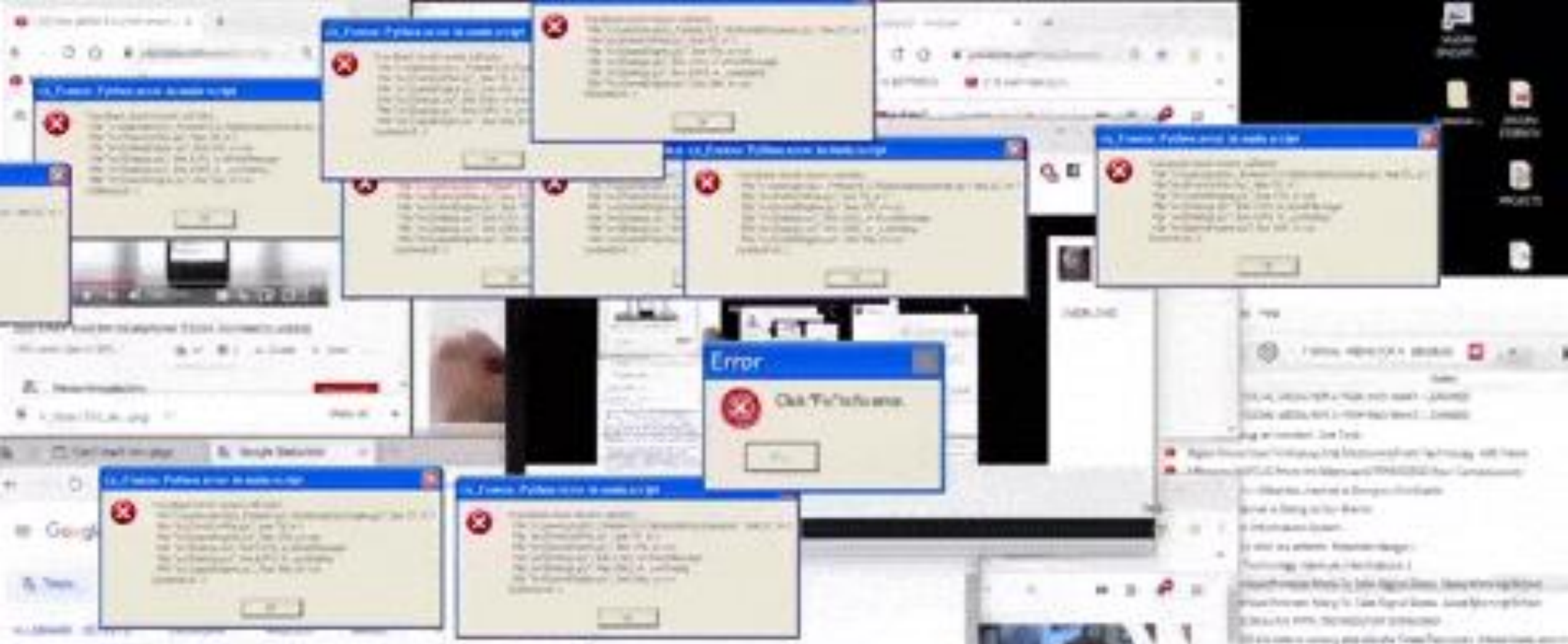
• D: 20 000 000 \$/minute

abc

“All SW shall be mission-critical”

is the basic idea behind the Rust
programming language

without any compromise:
free, performant, large OSS community,
efficient toolchains, flexible language



Allen 2018 wurden 1,3 Billionen digitale Bilder geschossen. Täglich werden 2,5 Terillionen Bytes Daten erzeugt. Permanent werden diese Daten gesammelt, gespeichert und analysiert. Das menschliche Gehirn kann diese Datenmenge kaum mehr fassen und sie nur langsam verarbeiten. Künstliche Intelligenz und Algorithmen ermöglichen die Bewältigung von Big Data. Doch was geschieht wenn wir keine direkten Einfluss mehr darauf haben? Social Media ist eine Möglichkeit, die Rückgewinnung unserer digitalen Spuren als eine Art zu sehen.

Exemple 2018, 1,3 billions de photos numériques ont été prises. Chaque jour, 2,5 billions d'octets de données sont générés. Ces données sont constamment collectées, stockées et analysées. Le cerveau humain peut à peine comprendre cette quantité de données et en conclure une signification. L'intelligence artificielle et les algorithmes permettent de gérer les mégadonnées. Mais qu'arrive-t-il si nous n'avons aucune influence directe? Le système social est un moyen de récupérer les données.



Here's This Morning's Daily Morning

The video 'This Morning' is a daily news and entertainment program. It features a host and several guests, including celebrities and experts. The program is broadcast on the BBC One channel. The video player shows the first few minutes of the program, with the host introducing the day's topics. The video is titled 'This Morning' and is part of a series of daily uploads.

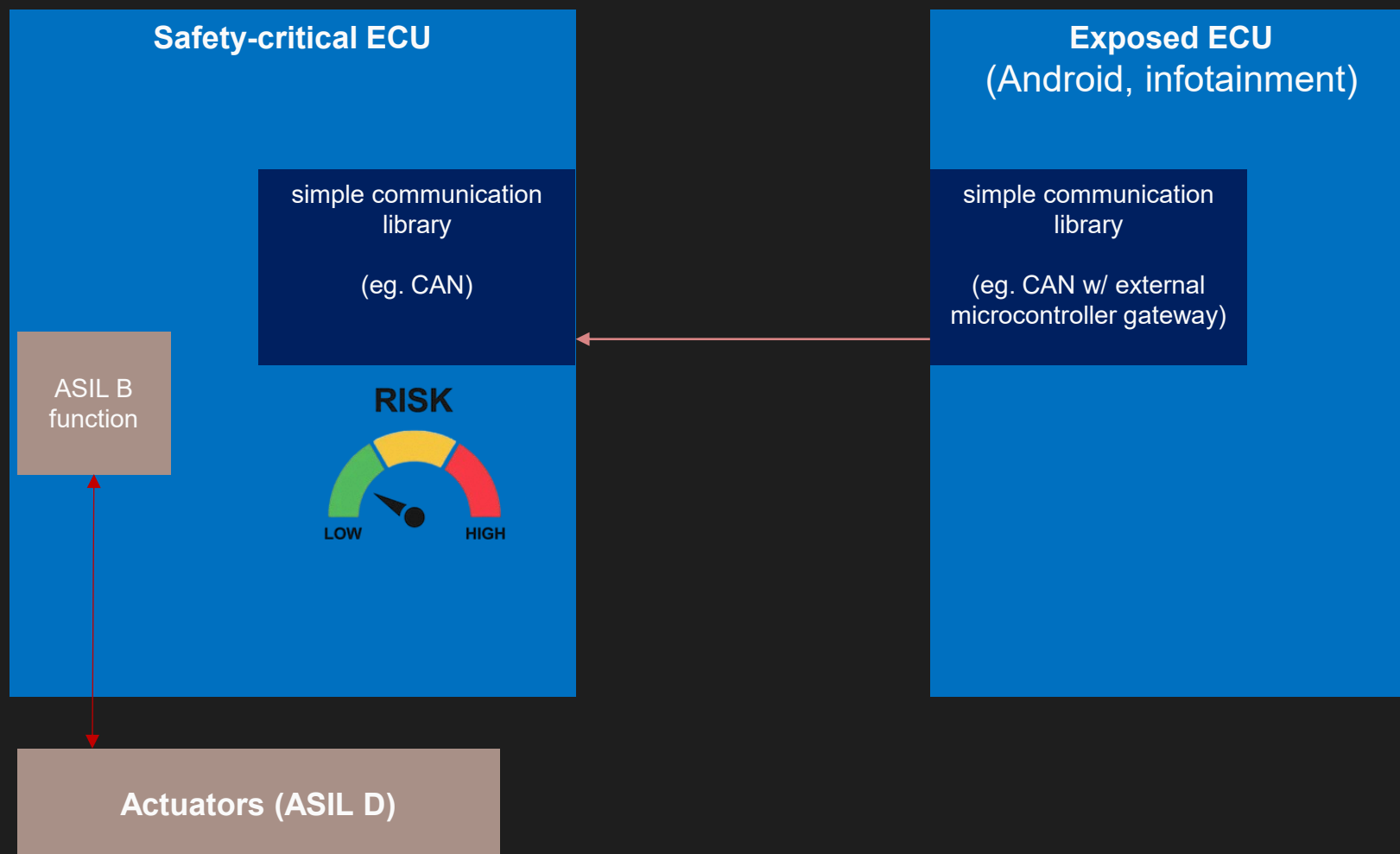
Programming language: an irrelevant implementation detail in the era of LLMs?



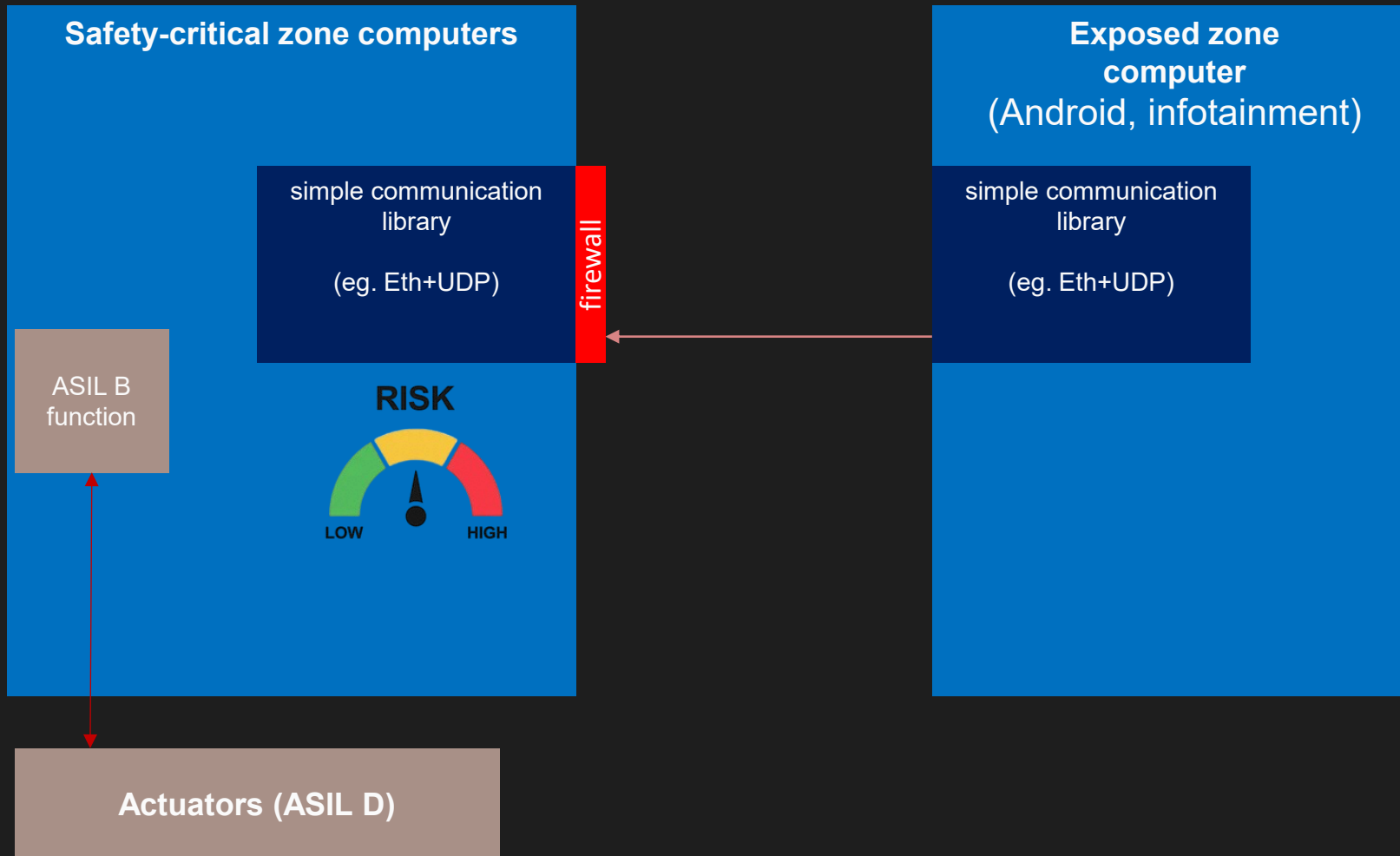
The reality is:
It has an impact on

system architecture, software
architecture, reliability, ease of data
exchange, quality, maintainability,
Productivity, Cost & Time-to-Market

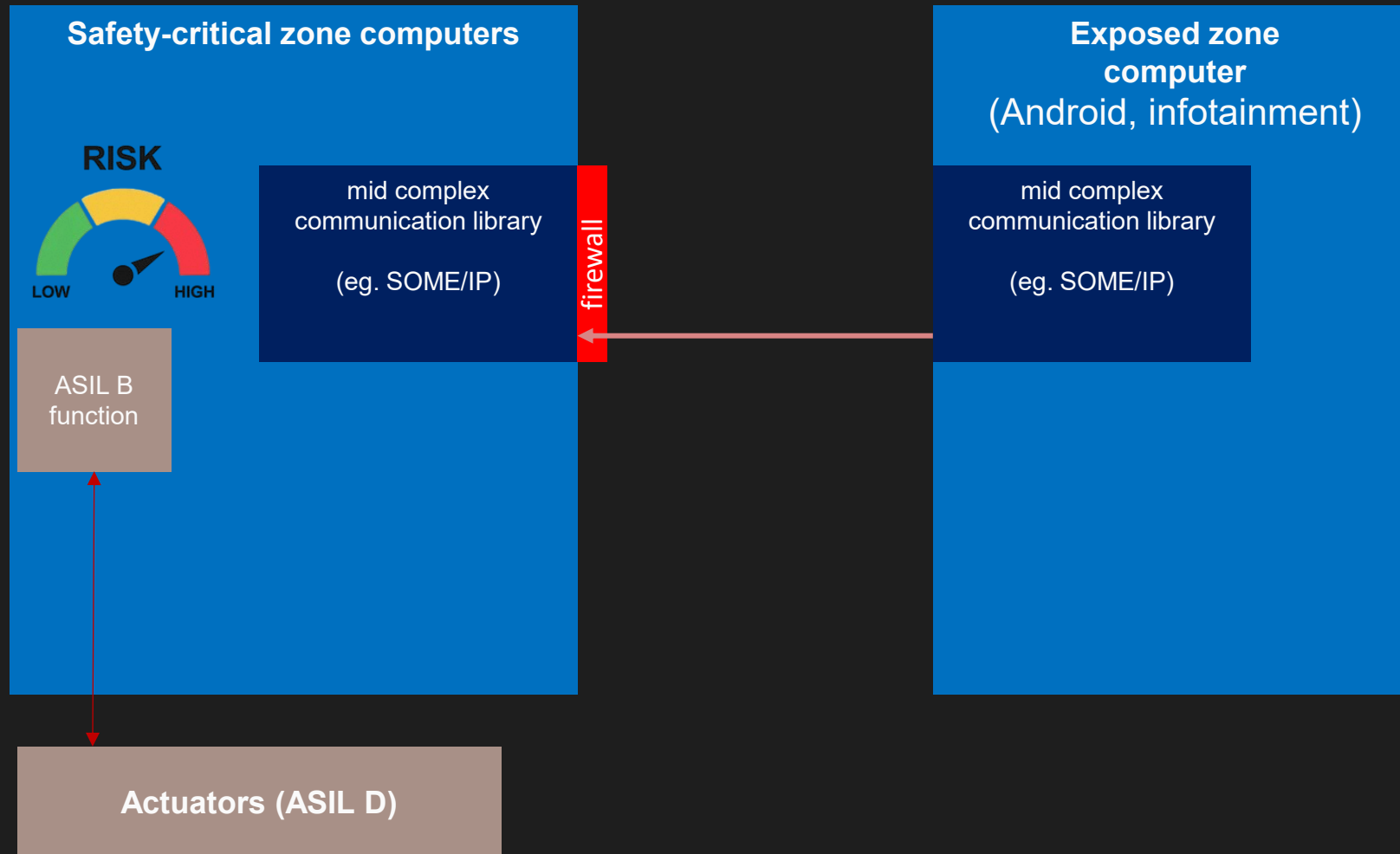
THE TIPPING POINT OF SDV: SECURITY VS. OPEN DATA FLOWS



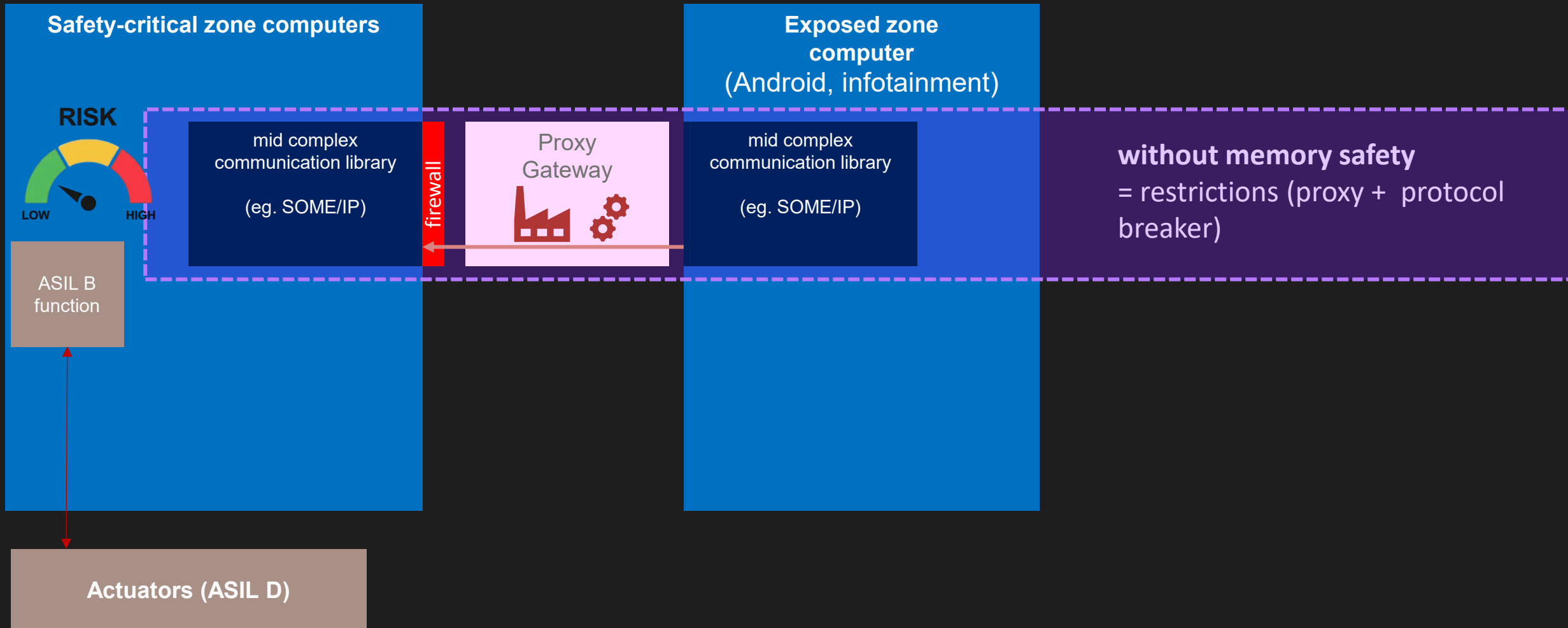
THE TIPPING POINT OF SDV: SECURITY VS. OPEN DATA FLOWS



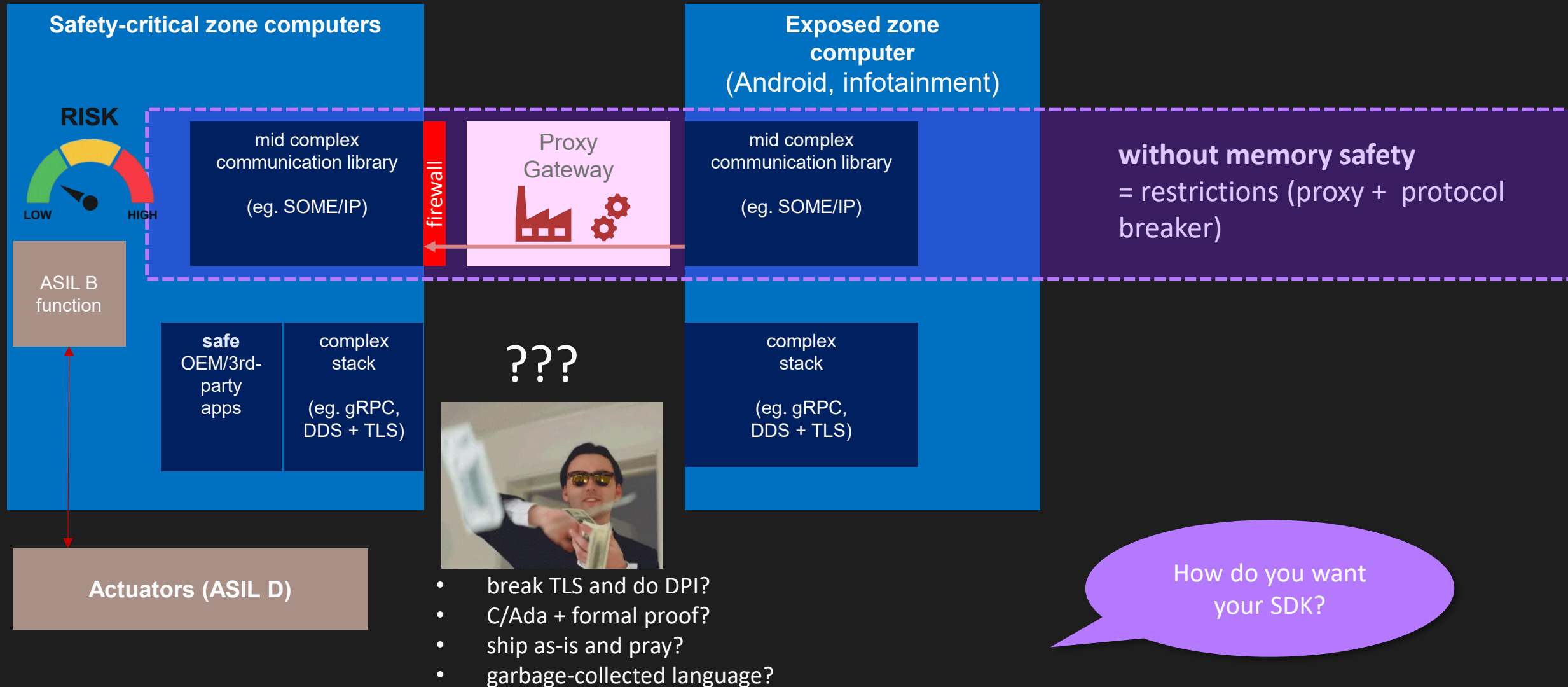
THE TIPPING POINT OF SDV: SECURITY VS. OPEN DATA FLOWS



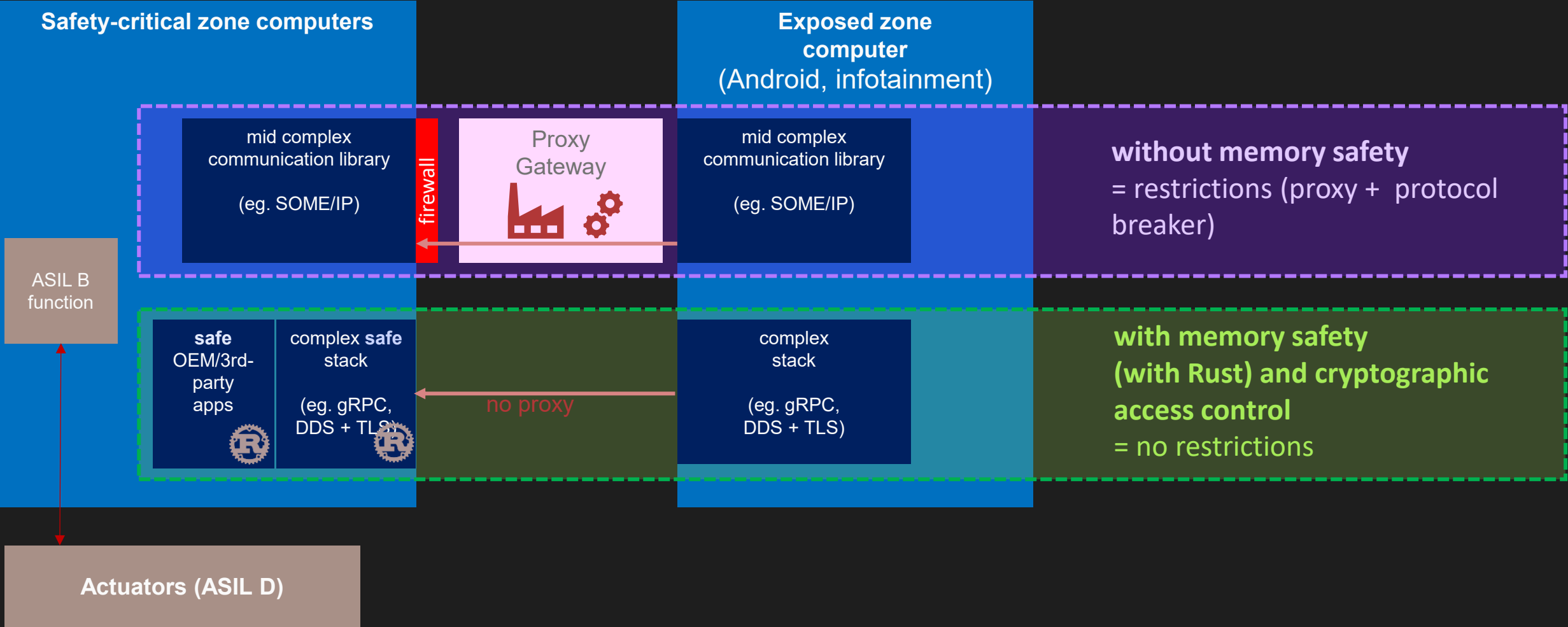
THE TIPPING POINT OF SDV: SECURITY VS. OPEN DATA FLOWS



THE TIPPING POINT OF SDV: SECURITY VS. OPEN DATA FLOWS



THE TIPPING POINT OF SDV: SECURITY VS. OPEN DATA FLOWS



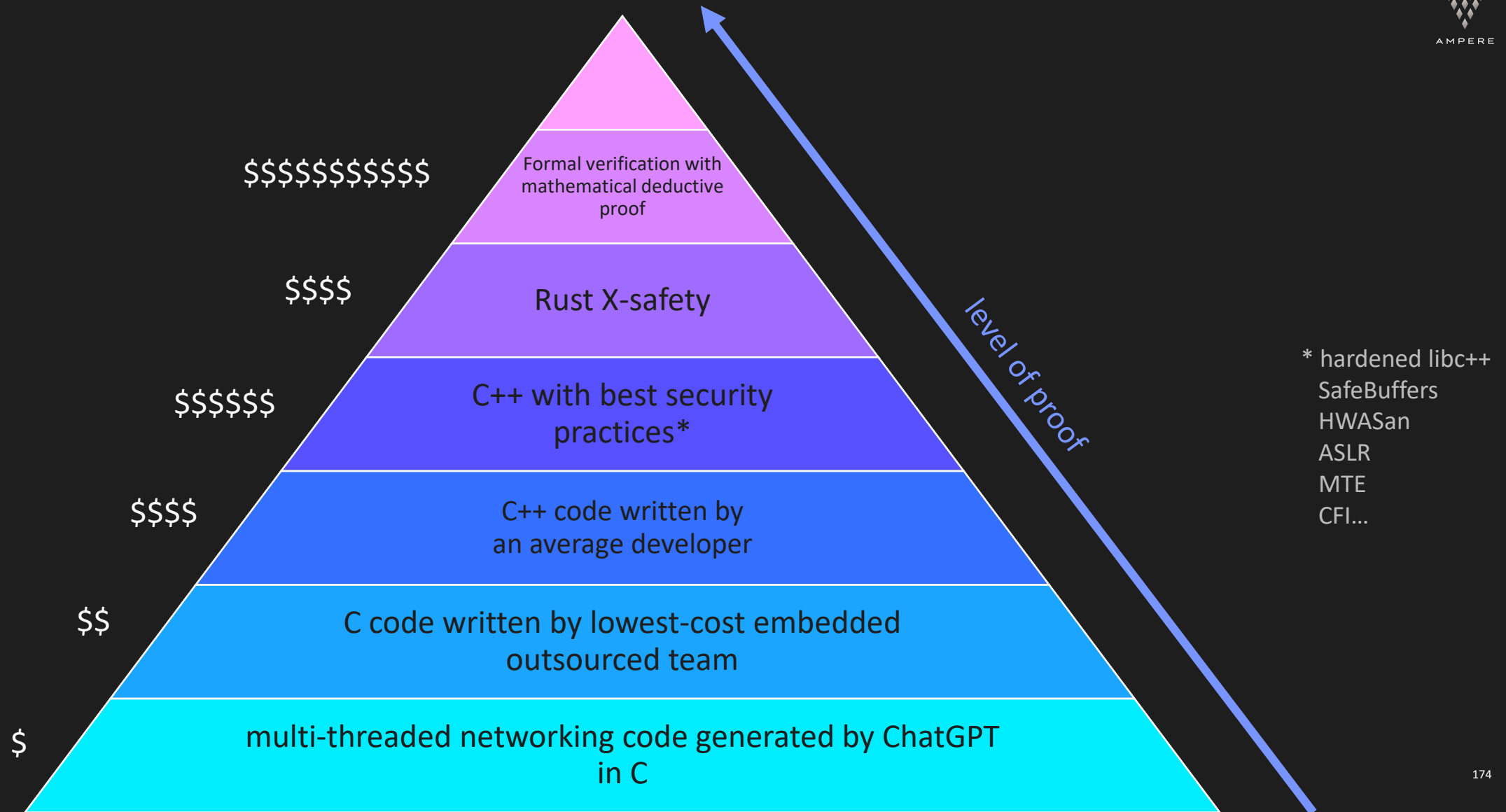
THE TIPPING POINT OF SDV: SECURITY VS. OPEN DATA FLOWS



	BEFORE SDV	TRADITIONAL SDV (EG. OUR COMPETITORS 😊)	AMPERE SDV
Access control management (eg. unauthorized service access)	+ Firewall + Simple Gateway + Ethernet physical port matching	+ Firewall + Complex Gateway + Ethernet physical port matching + TLS/MACSEC	+ Firewall + Zonal cryptographic authentication
Mechanisms to limit attack surface on protocols/payloads (eg. memory-safety)	Nothing (« simple protocols ») -or- Protocol breakers (for the best of us) -or- Protocol changes (for the ones who tried)	Nothing (« crossing fingers » or « fast bug fixes ») -or- Host packet inspection (too late and non-exhaustive)	+ Rust + Zonal memory safety rules + App sandboxing
Cyber-Effectiveness	😞	😞	😊
EE/SW-arch flexibility	😞	😞	😊
Data availability	Rigid	Rigid	Flex
Total cost impact	Small	Mid	Small



A SCALE ON THE LEVEL OF PROOF (SECURITY & RELIABILITY)



It all started from there:
Reliability and Security.



At scale | technically
| organisationally



A **good C/C++ developer** needs 10+ years of experience and will still make memory-management mistakes, leading to functional bugs and severe security issues.



A **newbie Rust developer** will be caught by the compiler before making the same mistakes.

That's not just memory safety.

Dependency management (Cargo)

Fearless concurrency

Compiler-assisted refactoring

Docs & testing frameworks

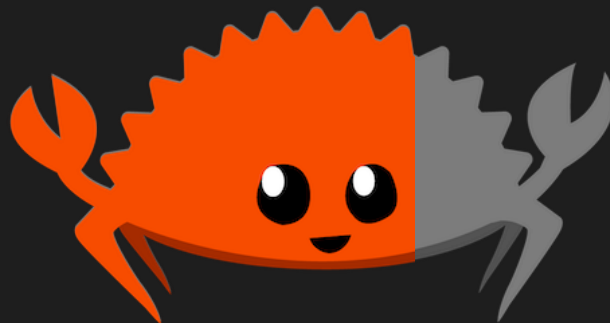
Zero-cost abstractions

Type-safety & error-handling

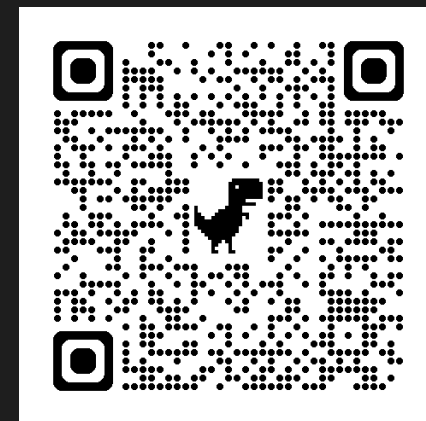
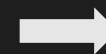


productivity++

THE BEGINNING



Now, **more than 70%** of the non-safety-critical in-house components are built using Rust, and the OS/SDK we are developing with Google is Rust-based and provides **Rust-by-default APIs**



This gave **incentive** to propagate Rust slowly, with now **>200 engineers** onboarded.

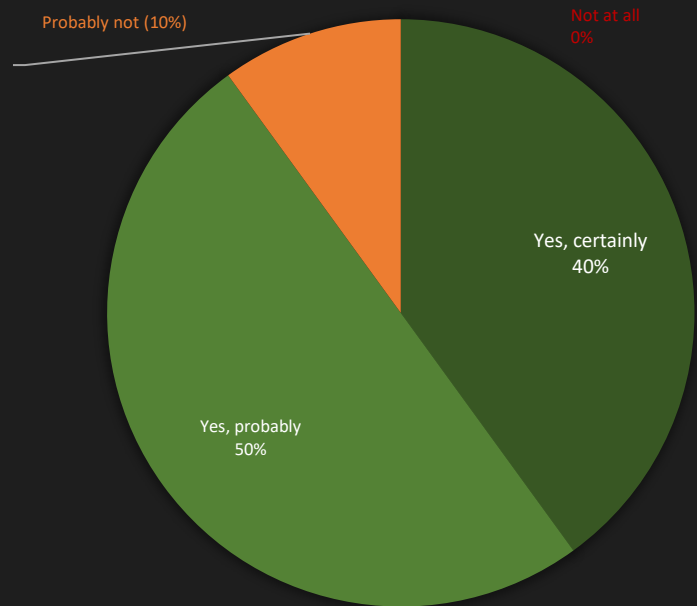
Hiring and training?



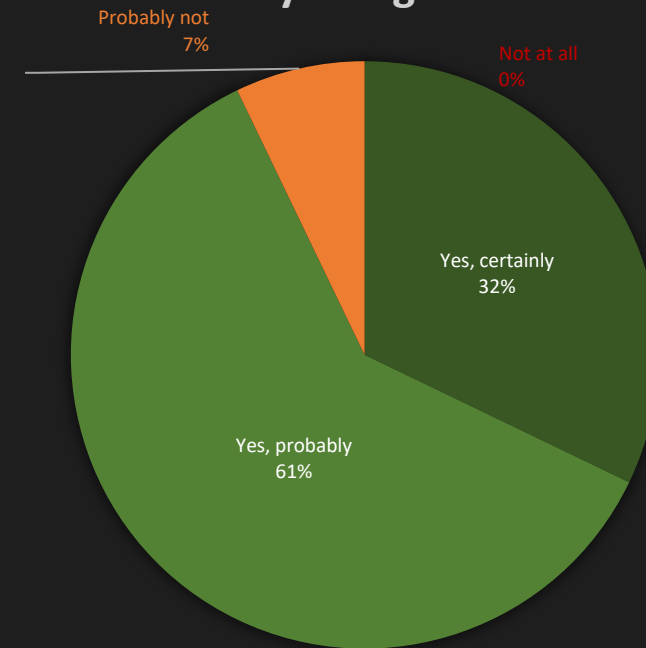
AFTER TRAINING 200+ PEOPLE: THEY ARE HAPPY



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



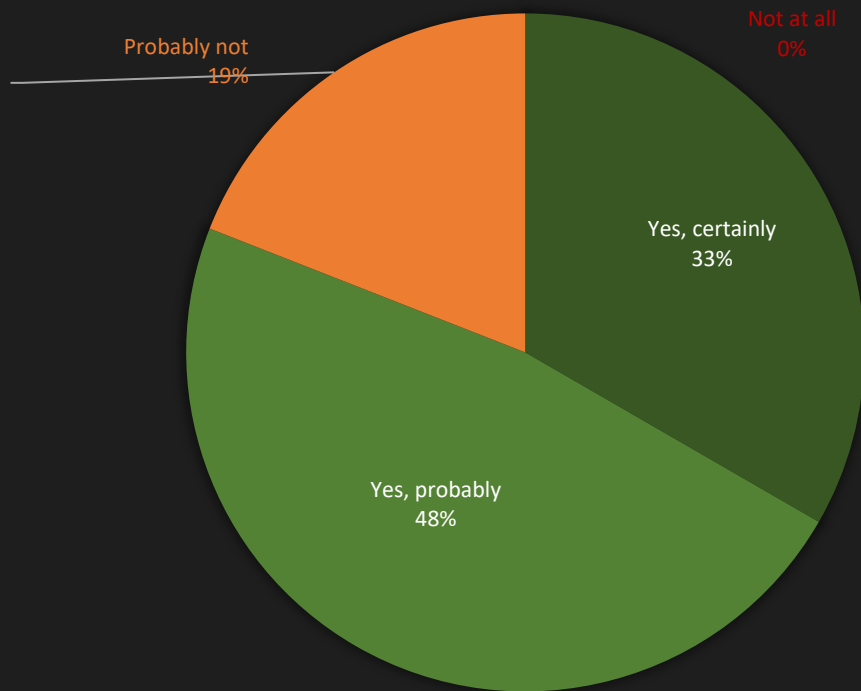
Do you think vehicle safety/security is improved by using Rust?



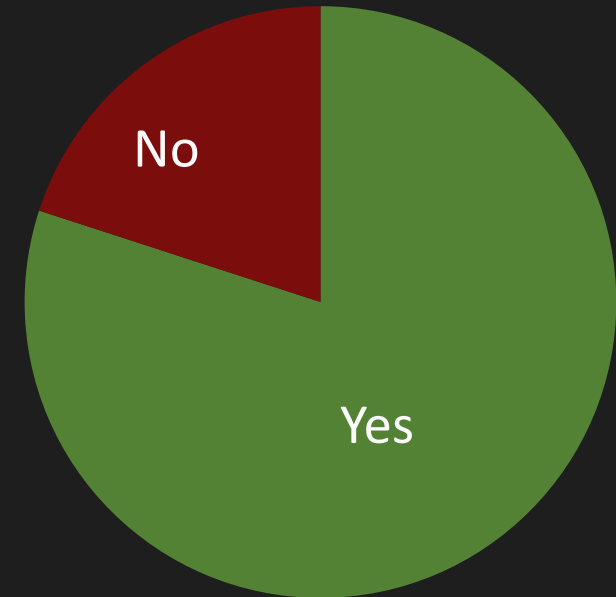
AFTER TRAINING 200+ PEOPLE: THEY ARE HAPPY



Do you think Rust is an asset to your career?



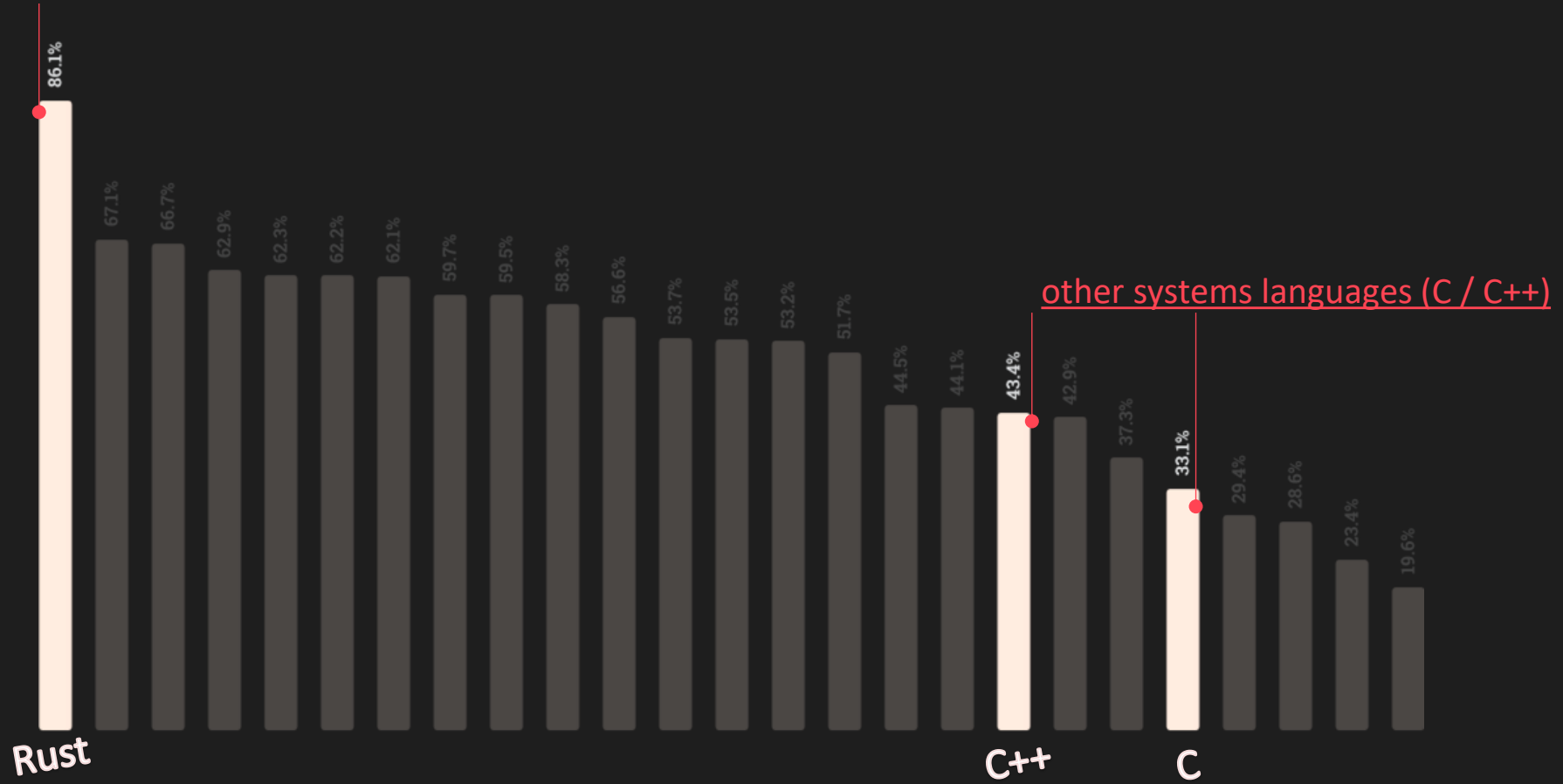
If you had to create a new SW component now, would you choose Rust?



IT'S ALSO AN ATTRACTIVE TECHNOLOGY (VS. TALENT POOL)



Rust = 1st position for the 10th year



Accelerating & Committing



Goal 1 : involve more the overall automotive ecosystem

Goal 2 : propagate Rust to safety-critical code

In short: Rust will become a mandatory requirement.

Incl, not limited to: bootloaders, shared HSMs, shared hypervisor drivers/code, exposed network/BT/USB drivers, general code exposed to untrusted data (SWupdate, cloud, connected ADAS).

AND THAT HAS BEEN OUR
EXACT STRATEGY SINCE 2023
(...EARLY POCS IN 2022)

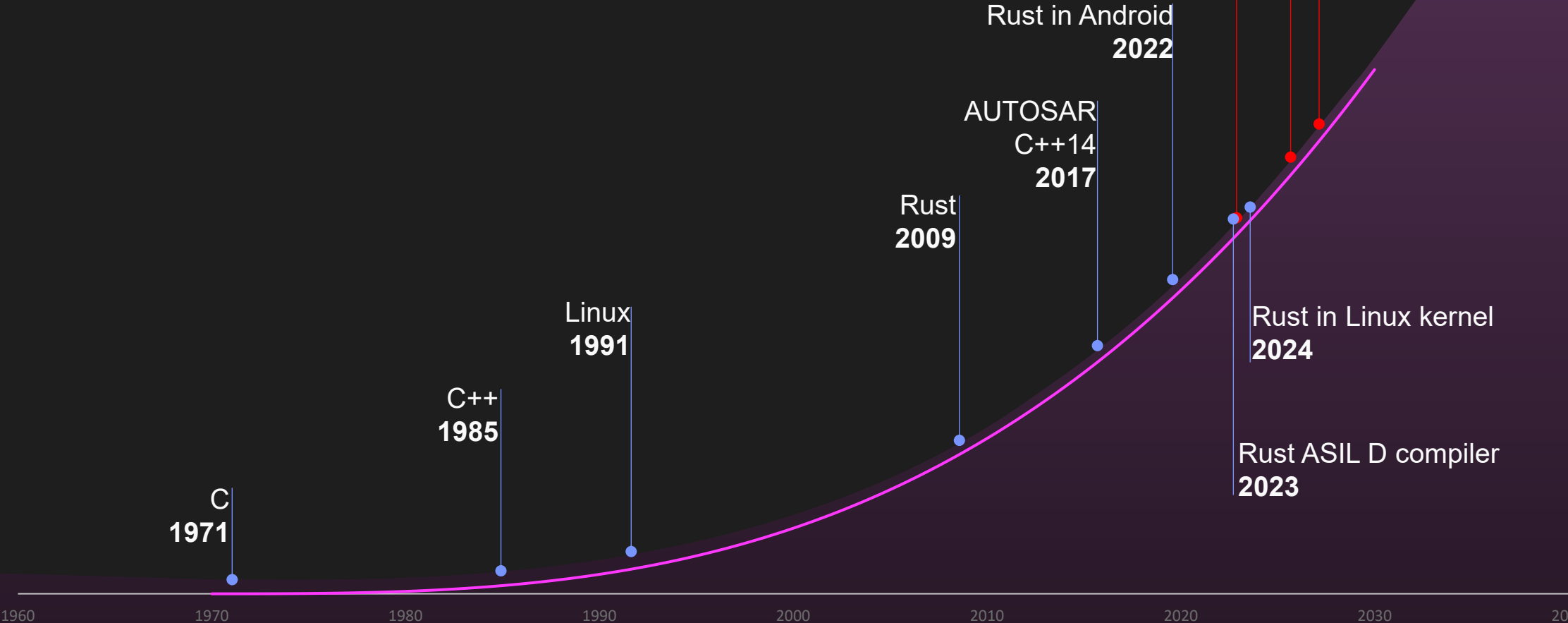


AMPERE

target: Rust in Ampere SDV
(incl Safety-Critical)
SoP 2028-2031

Rust in Ampere SDV
& Google SDV (QM only)
SoP 2026

Official Rust strategy for
security (QM) at Ampere
2023



Summary



- **At Ampere/Renault, we are convinced Rust is a game changer for Software in the Automotive industry**
- **We are well engaged on a Rust based Sw Strategy**
- **We are pushing the Automotive industry to embrace it**



AMPERE

Thank you

jean-francois.bortolotti@ampere.cars



AMPERE.CARS

Hugo Mercier



Software Architect
IoT.bzh



- **Rust dans l'embarqué : retours sur le développement d'une borne de recharge pour véhicules électriques**



Embedded Rust: case of an EV charging station

IoT.bzh at a glance

**Location: Lorient,
Brittany, France**



Our product

redpesk®: SaaS platform (or On Prem) Linux for industrial IoT (auto, mil-aero, energy...)



+30 years of embedded SW expertise

WIND RIVER

Vannes, France 1990
1st Wind River european R&D center

Open Source contributions



OS open source, Samsung TVs
Intel Vannes (2011-2015)



Some customers



An EV charger in Rust

2025-06-25, Rust Paris

Tux-EVSE, an open-source EV charger

- Work in collaboration with an automotive supplier
- Outline
 - EV Charging 101
 - Software architecture
 - Why Rust ?
 - FFI is hard
 - Dependency hell
 - Conclusion

“Basic” Charging

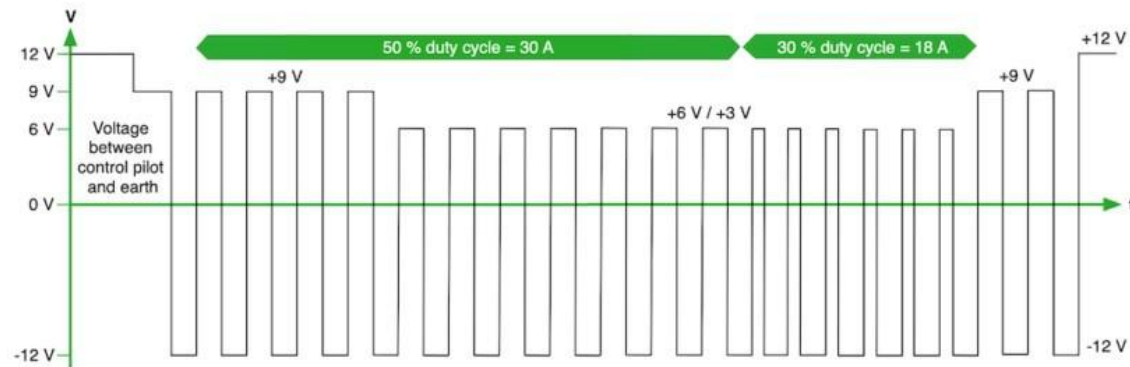
NFC



IEC 61851 (PWM)



Car → charger: voltage levels
Charger → Car: PWM duty cycle

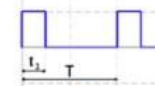


Defined by IEC 61851-1 Annex A

Charging station controls PWM „duty cycle” (in %) by turning switch on and off at 1 kHz
→ Duty cycle = proportion between on and off periods
The longer the switch is turned on, the higher the available amperage

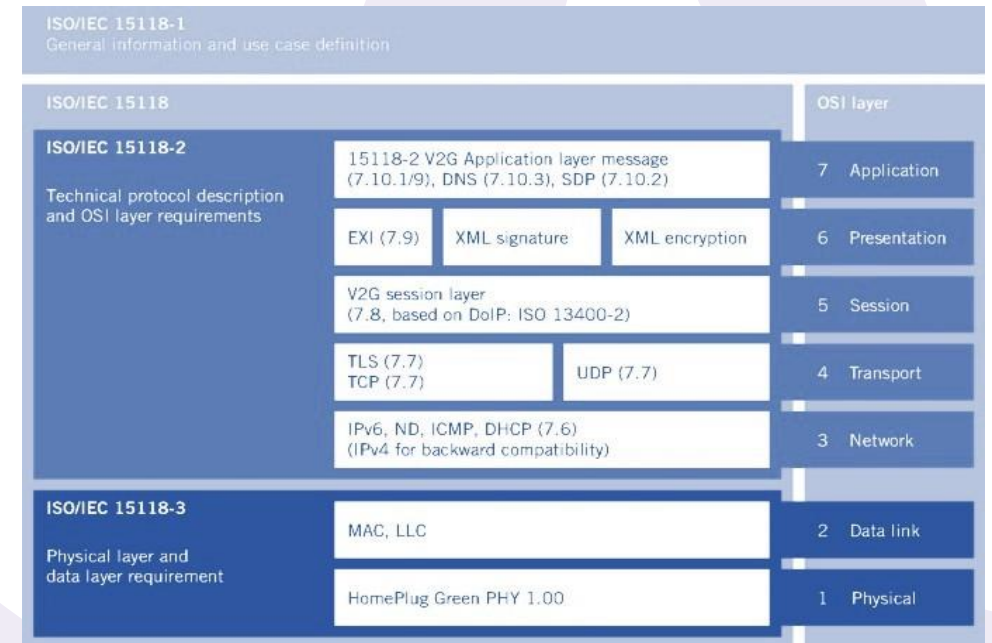
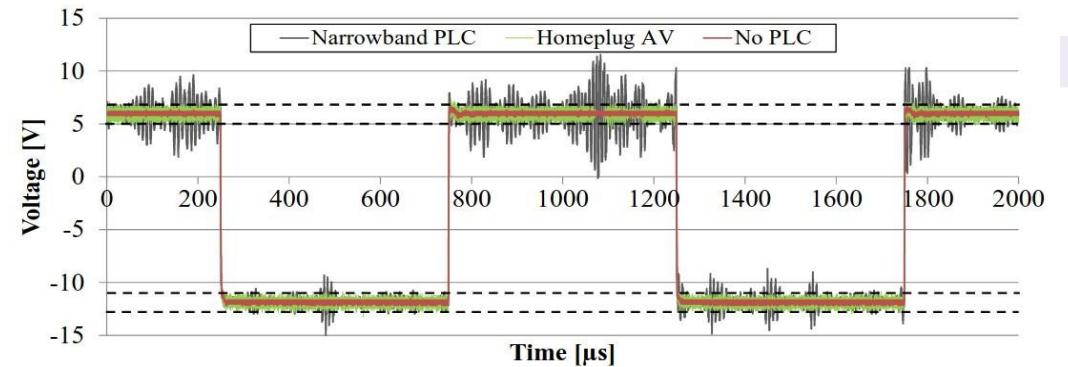
EV controls voltage level

PWM	Current
96 %	80 A
95 %	77,5 A
90 %	65 A
85 %	51 A
80 %	48 A
70 %	42 A
60 %	36 A
50 %	30 A
40 %	24 A
30 %	18 A
25 %	15 A
16 %	9,6 A
10 %	6 A

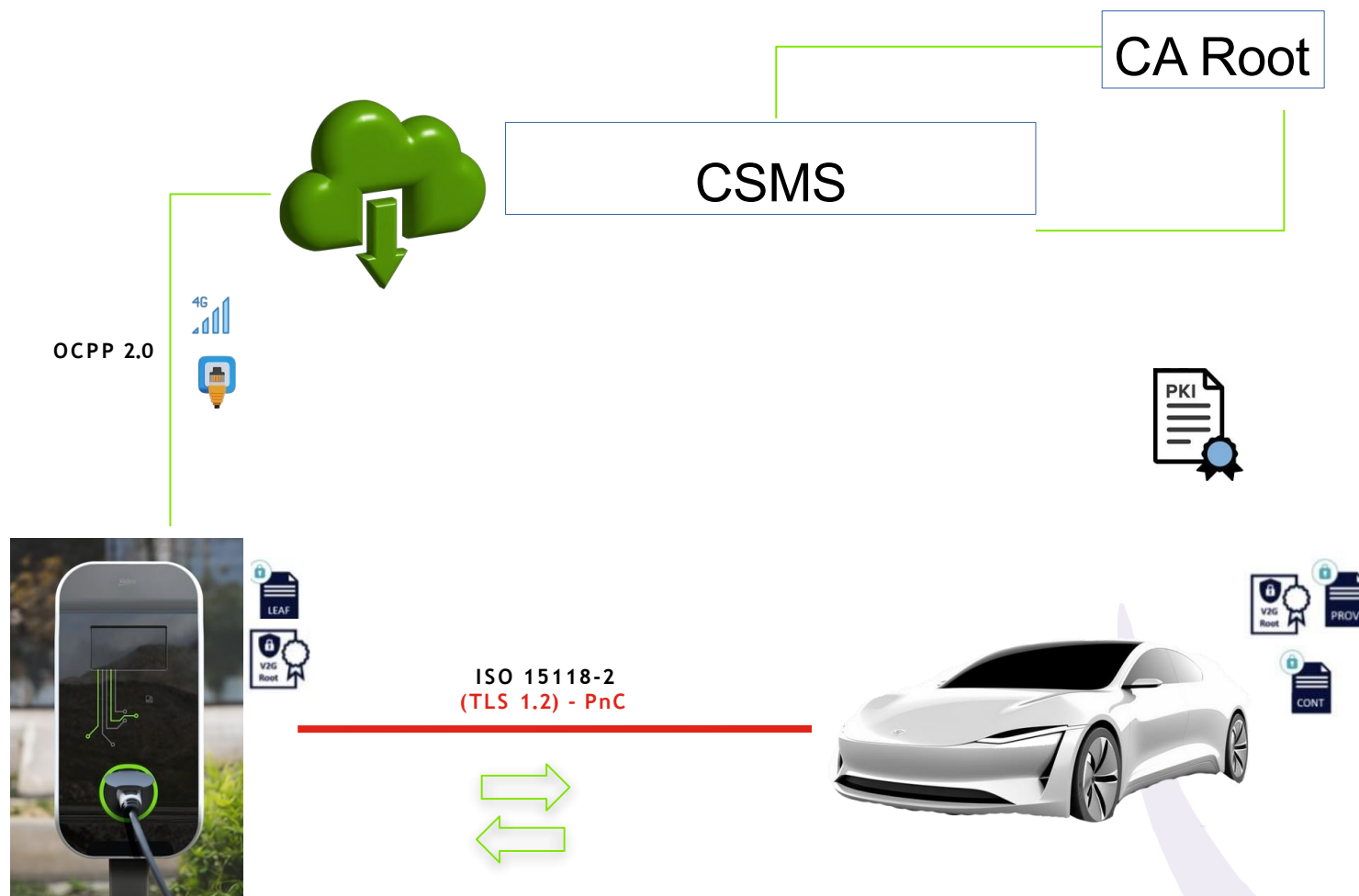


ISO-15118-2x

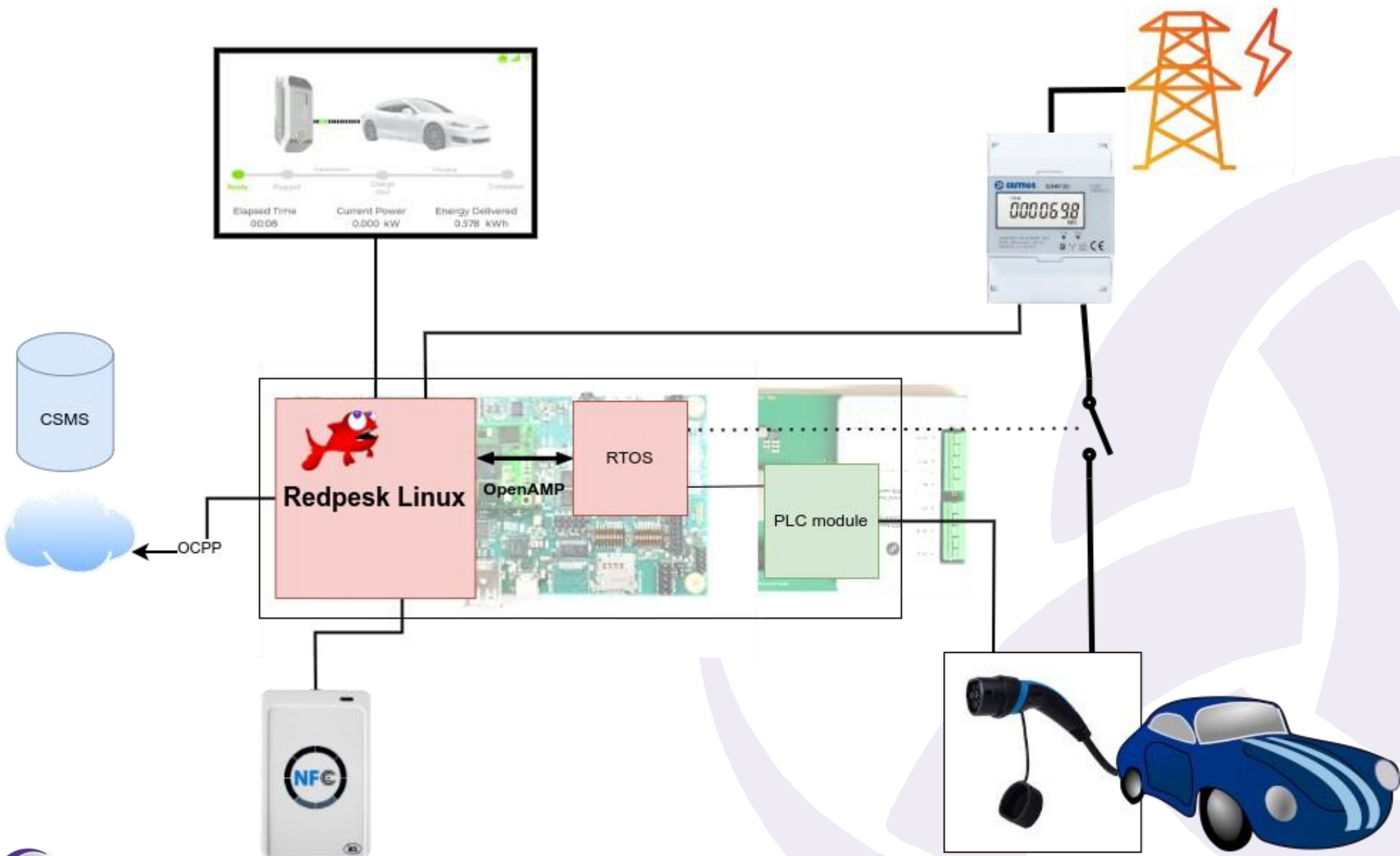
- Full network stack on 2 wires !
 - Ethernet link over PLC over PWM
 - TCP / IPv6 / TLS
 - Binary XML (EXI)



ISO-15118-2x



- TCP/TLS
- Authentication by RFID card
- Or “Plug and Charge”: online verification of signed contract through a trust chain (PKI)
- “Vehicule to Grid”: ability to discharge

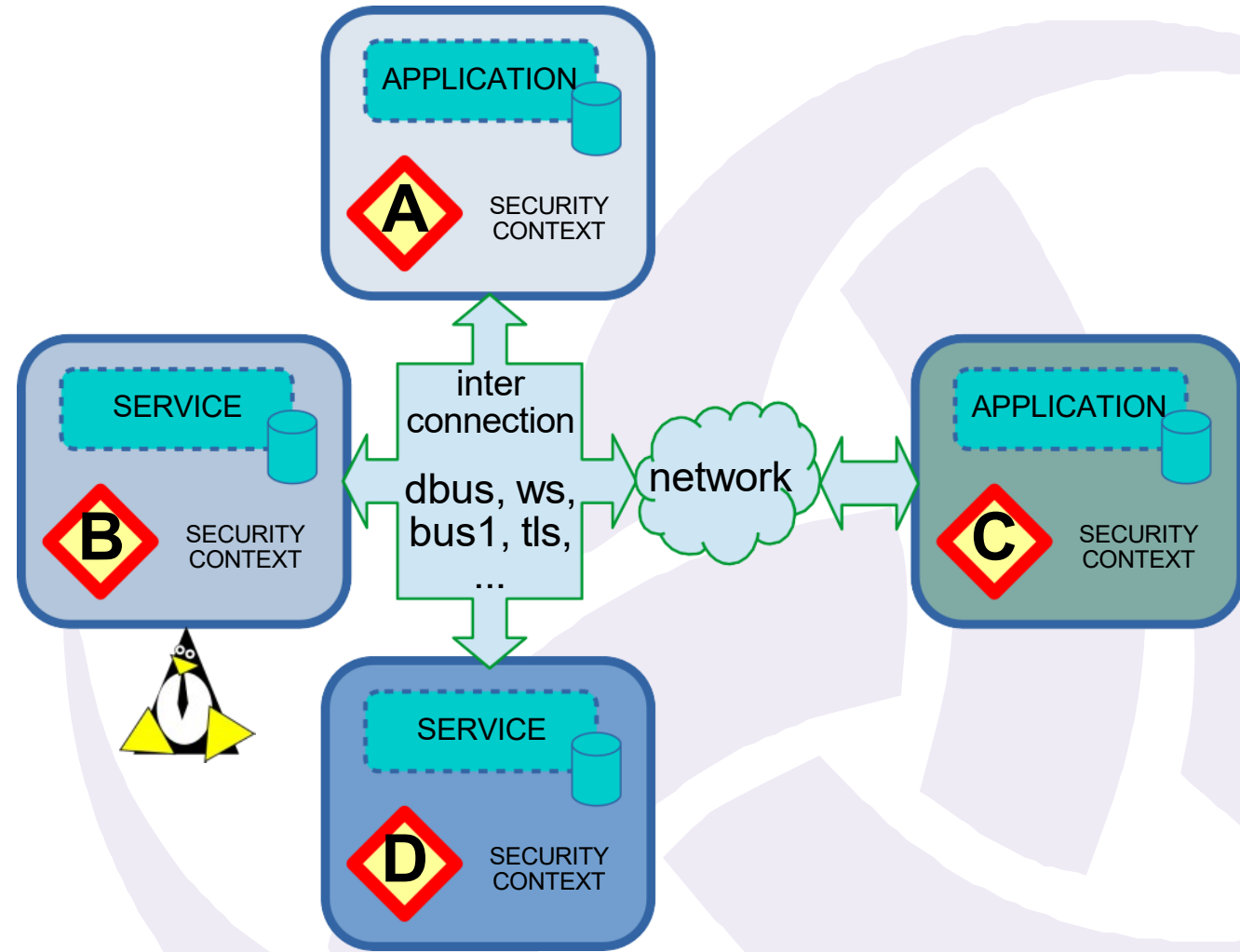


Security issues

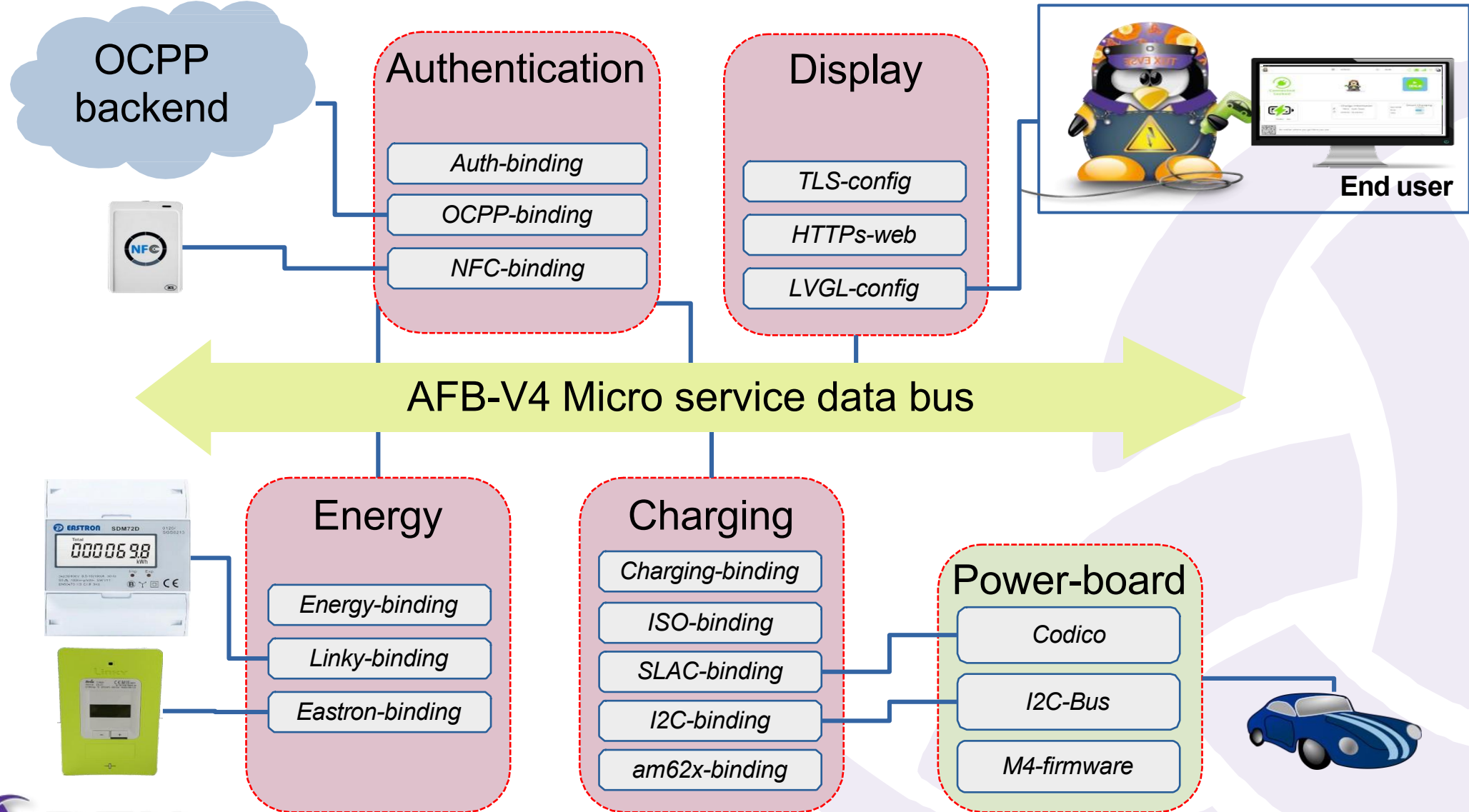
- Large surface of attack
 - Internet: payment, authentication, etc.
 - PLC: vehicle
 - Complexity of protocols (ISO-15118, OCPP, PKI)
- Threats
 - Free riding
 - Compromission of station's or car's network
 - Electrical risk

AFB: a micro-service framework for security

- Coming from Tizen (Samsung TV)
 - Reused in Automotive Grade Linux
- Micro-services communicate through RPC (api, verbs, events)
- Security isolation with network transparency
- Linux Security Module: SMACK
 - “embedded” SELinux
- Permission database: cynagora



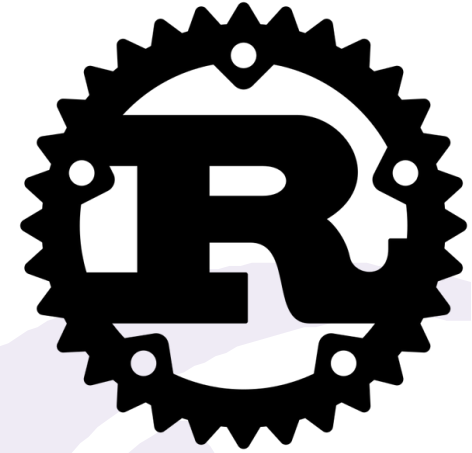
Micro services



Tux-EVSE

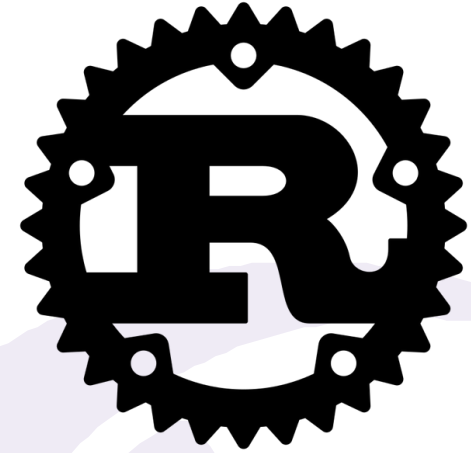
- Open source components to build a charger (EVSE)
- <https://github.com/tux-evse>
- An ISO-15118-2x simulator
- <https://github.com/tux-evse/iso15118-simulator-rs>

Why Rust ?



- Memory safety without performance sacrifice
 - More generally no Undefined Behaviour (e.g. integer overflows)
- Powerful type system
 - Option, Result, enums
 - Illegal states hard to represent
- Extensibility
 - Traits and macros
- Important ecosystem, from different fields

Why Rust ?



- Cyber Resilience Act
 - Rust “blessed” by major actors
 - Clients are coming to it
- In the long run, less expensive than C(++)

Migrate everything to Rust ?

RUSTIFY ALL THE THINGS!

- Large existing codebase written in C
- We need an interface to it in Rust then !
- Or “Learn Rust the hard FFI way”
- Core part in C with Rust wrapping
 - <https://github.com/redpesk-common/afb-librust>
 - FFI with “unsafe” parts
- Unsafe means “Ok compiler, I know what I am doing”



How ?

- First: bindgen !
- Auto (re)generate Rust interface from C headers
- Embedded in build.rs scripts

```
fn main() {  
    /* ... */  
    let _capi_map = bindgen::Builder::default()  
        // main entry point for wrapper  
        .header("capi/_libafb-map.c")  
        .raw_line(header)  
        // default wrapper config  
        .parse_callbacks(Box::new(bindgen::CargoCallbacks::new()))  
        .derive_debug(false)  
        .layout_tests(false)  
        .allowlist_function("errno")  
        .allowlist_function("strerror_r")  
        .allowlist_function("__errno_location")  
        .allowlist_function("afb_*")  
        .allowlist_type("afb_syslog_*")  
        .allowlist_type("afb_epoll_*")  
        .allowlist_type("afb_req_subcall_flags")  
        .allowlist_var("afbBinding_*")  
        .blocklist_item("json_object")  
        // generate libafb wrapper  
        .generate()  
        .expect("Unable to generate libafb");  
    /* ... */  
}
```


How ?

- Expose a safe API, but with “unsafe” in its implementation
- We have a C asynchronous event framework (AFB)
- How to register Rust functions as C callbacks with context?

```
int afb_api_add_verb(  
    struct afb_api_v4 *apiv4,  
    const char *verb,  
    void (*callback)(struct afb_req_v4 *req, unsigned  
nparams, struct afb_data * const params[]),  
    void *userdata  
);
```

Registration of “verbs”

```
struct UserData {  
    /* ... */  
};  
  
let verb = AfbVerb::new("myverb")  
    .set_callback(myverb_callback)  
    .set_context(UserData { /* ... */ })  
  
fn myverb_callback(request: &AfbRequest, _args: &AfbRqtData, ctx: &AfbCtxData)  
-> Result<(), AfbError> {  
    let userdata = ctx.get_ref::<UserData>()?;  
    /* ... */  
}
```

Cast void* to UserData

Implementation - context

```
pub struct AfbCtxData {
    raw: *mut std::ffi::c_void,
    typeid: Typeld
}

impl AfbCtxData {
    pub fn new<T>(ctx: T) -> Self
    where
        T: 'static,
    {
        let typeid = Typeld::of::<T>();
        Self {
            raw: Box::into_raw(Box::new(ctx)) as *mut std::ffi::c_void,
            typeid
        }
    }
    /* ... */
    pub fn get_ref<T>(&self) -> Result<&T, AfbError>
    where
        T: 'static,
    {
        self.check_type(Typeld::of::<T>())?;
        let value: &mut T = unsafe { &mut *(self.raw as *mut T) };
        Ok(value)
    }
}
```

“Leak” memory (safe !)

Convert back to initial
pointer type

Implementation – callback “trampoline”

```
impl AfbVerb {  
    /* ... */  
    pub fn register(&self, apiv4: cglue::afb_api_t) -> i32 {  
        unsafe {  
            cglue::afb_api_add_verb(  
                apiv4,  
                verb_name.as_ptr(),  
                Some(api_verbs_cb),  
                self as *const _ as *mut std::ffi::c_void  
            )  
        }  
    }  
}  
  
pub extern "C" fn api_verbs_cb(/* ... */ ctx: *mut std::ffi::c_void)  
{  
    /* ... */  
    let verb_ref = unsafe { &mut *(ctx as *mut AfbVerb) };  
    /* ... */  
    let result = (verb_ref.callback)(/* ... */ &verb_ref.context);  
}
```

Pass a (Rust) extern “C”
function as callback
With “self” as (void*)
userdata

This will be called by the
underlying C lib. And will
then call the actual Rust
function

Ergonomic Rust with FFI is hard

- Wait, how is AfbCtxData different from a &dyn Any ?
 - AfbCtxData::get_ref<T> ~= Any::downcast_ref<T>
- Wait, “function with context” = closure !

```
// Usage
let ctx = UserData { /* ... */ };
let verb = AfbVerb::new("myverb")
    .set_callback(move |req, args| { ctx /* ... */ } );

// Implementation
// We need 'static, because the closure may outlive the call
fn AfbVerb::set_callback<F: Fn(&AfbRequest, &AFbRqtData) + 'static>(callback: F)
{ /* .. */ }
```

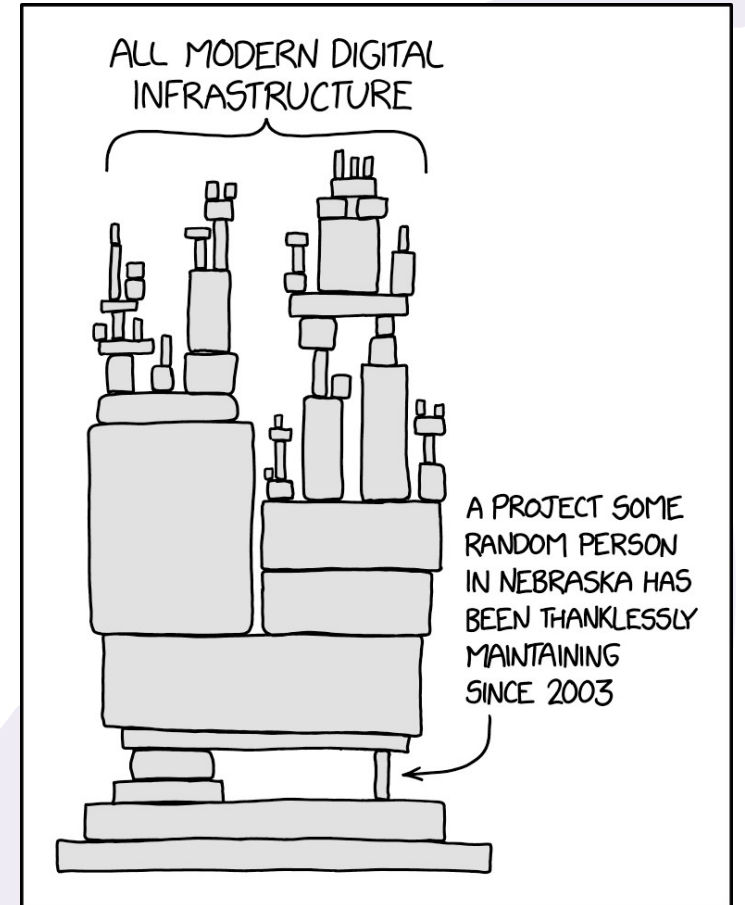
Ergonomic Rust with FFI is hard

- We also have async calls
- Wait, could this be exposed as async Rust ?
 - async / await / Future / Stream / etc.

```
void
afb_api_call(
    struct afb_api_v4 *apiv4,
    const char *verbname,
    unsigned nparams,
    struct afb_data * const params[],
    void (*callback)(
        void *closure,
        int status,
        unsigned nreplies,
        struct afb_data * const replies[],
        struct afb_api_v4 *api),
    void *closure);
```


Dependency hell?

- Cargo: great tool
- But: Easy to introduce lots of dependencies
- Risk of “leftpad”-like issues
- Audits happen in embedded (security, good practices, etc.)



xkcd.com/2347

Dependency hell?

- Problem not specific to Rust
- If you don't trust the dependency developers
 - Trust your distribution's packagers ?
- /!\ Rust has no stable ABI
 - No “Rust” shared libs, only “extern C” shared libs
- Need a way to rebuild everything in case of a (security) update

Cryptography needs

- ISO-15118-2x
 - TLS 1.2 and 1.3
 - ECDSA signature
 - EC secp256r1
 - SHA256
- Rust crypto ecosystem
 - Rustls: too high level
 - OpenSSL (FFI): yes, but we already use GnuTLS in AFB
 - Aws-lc-rs: maybe, but not as known as OpenSSL or GnuTLS

Cryptography: what we did

- Our own small Rust unsafe wrapper over GnuTLS
- Bindgen generated code + glue code (~1000 LoC)

Conclusion

- After 2 years and an important project using it
- Safe Rust is clean and powerful
 - Even according to our clients
- Writing a safe Rust API on a C API through FFI is hard(er)
 - Especially for an “ergonomic” API
 - Ergonomy = less surprise, more predictability
- Don't overlook dependencies
- Ongoing internal adoption and skill ramp-up

Romain Fouquet



Ingénieur de recherche
Inria Saclay

Inria

- **Ariel OS: a Rust embedded OS standing on the shoulders of giants**



Ariel OS: a Rust embedded OS standing on the shoulders of giants

Ariel OS: Who Are We?

Romain Fouquet/@ROMemories

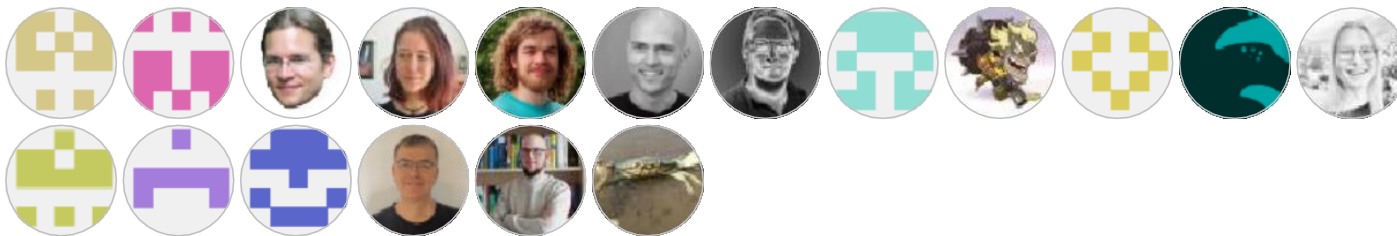


Project founders

Emmanuel Baccelli, Kaspar Schleiser



Ariel OS contributors



Part One: Rust for Embedded

Constrained Embedded —Target Devices

Little flash memory:
hundreds of kibibytes

Little RAM:
dozens/hundreds of kibibytes
`=> no alloc`

Low energy consumption:
multi-month battery life

Networking capabilities:
Wi-Fi, USB, Ethernet,
LTE-M, LoRa

Constrained Embedded —Target Devices

Little flash memory:
hundreds of kibibytes

Little RAM:
dozens/hundreds of kibibytes
`=> no alloc`

Low energy consumption:
multi-month battery life

Networking capabilities:
Wi-Fi, USB, Ethernet,
LTE-M, LoRa

Constrained Embedded —Target Devices

Little flash memory:
hundreds of kibibytes

Little RAM:
dozens/hundreds of kibibytes
`=> no alloc`

Low energy consumption:
multi-month battery life

Networking capabilities:
Wi-Fi, USB, Ethernet,
LTE-M, LoRa

Constrained Embedded —Target Devices

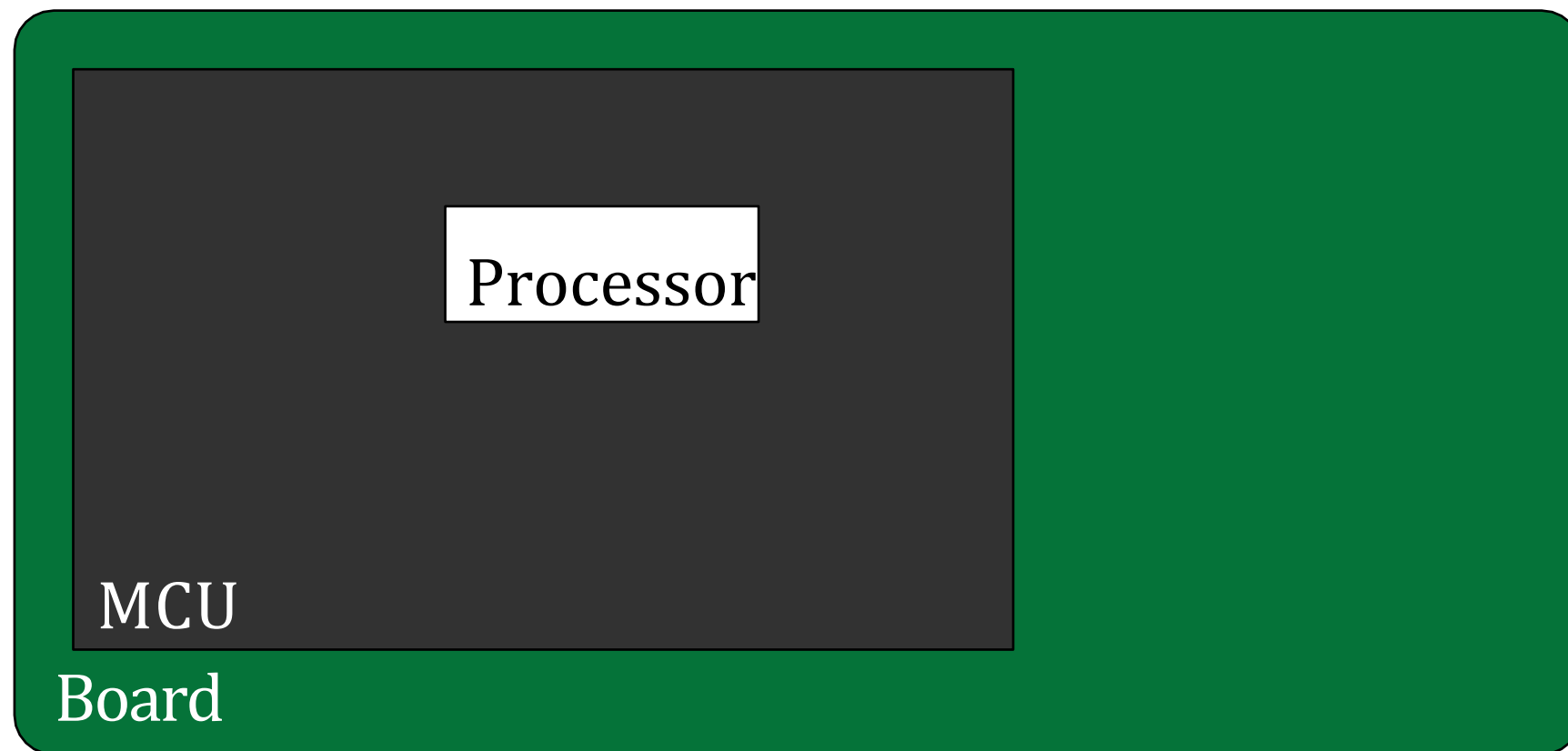
Little flash memory:
hundreds of kibibytes

Little RAM:
dozens/hundreds of kibibytes
`=> no alloc`

Low energy consumption:
multi-month battery life

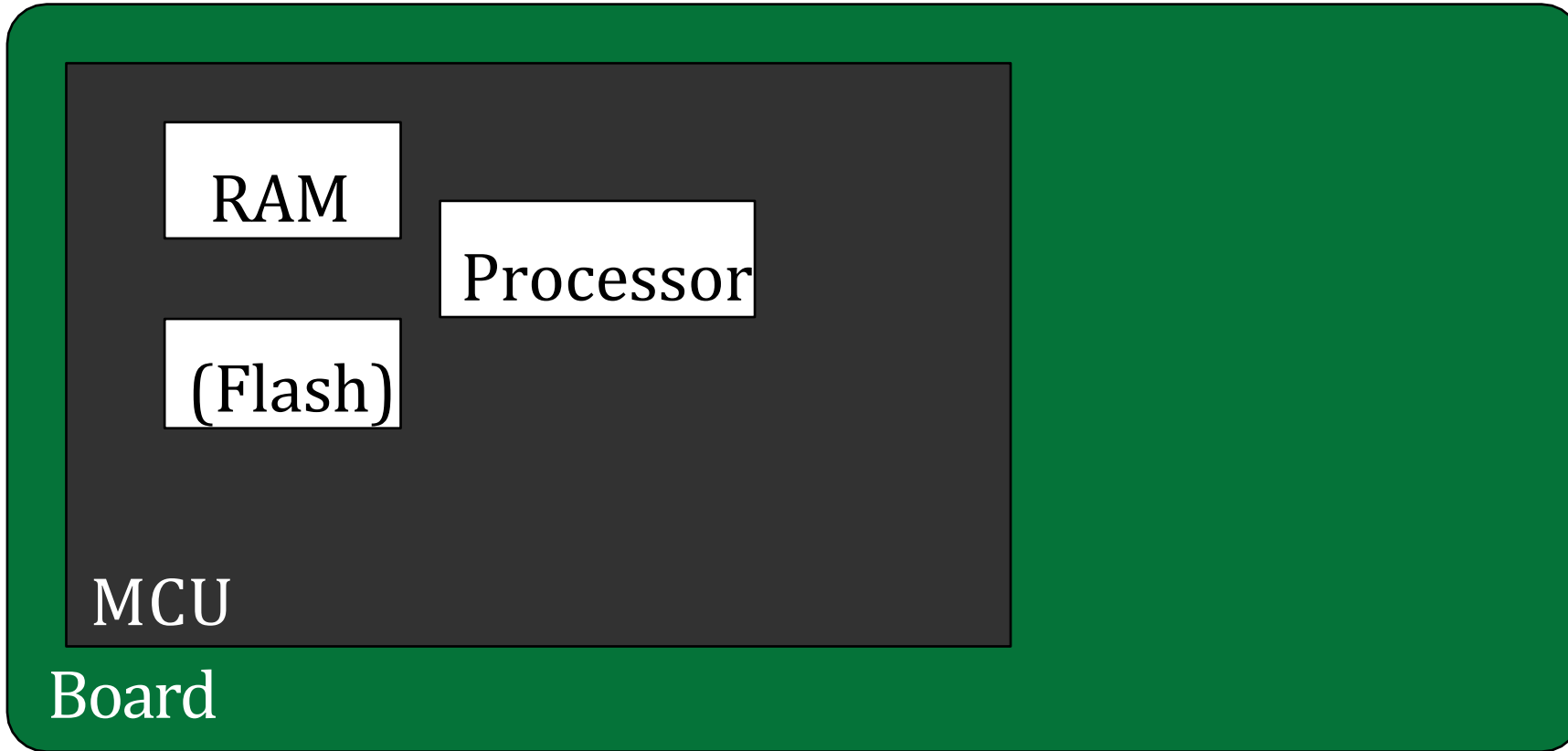
Networking capabilities:
Wi-Fi, USB, Ethernet,
LTE-M, LoRa

Microcontrollers and Boards



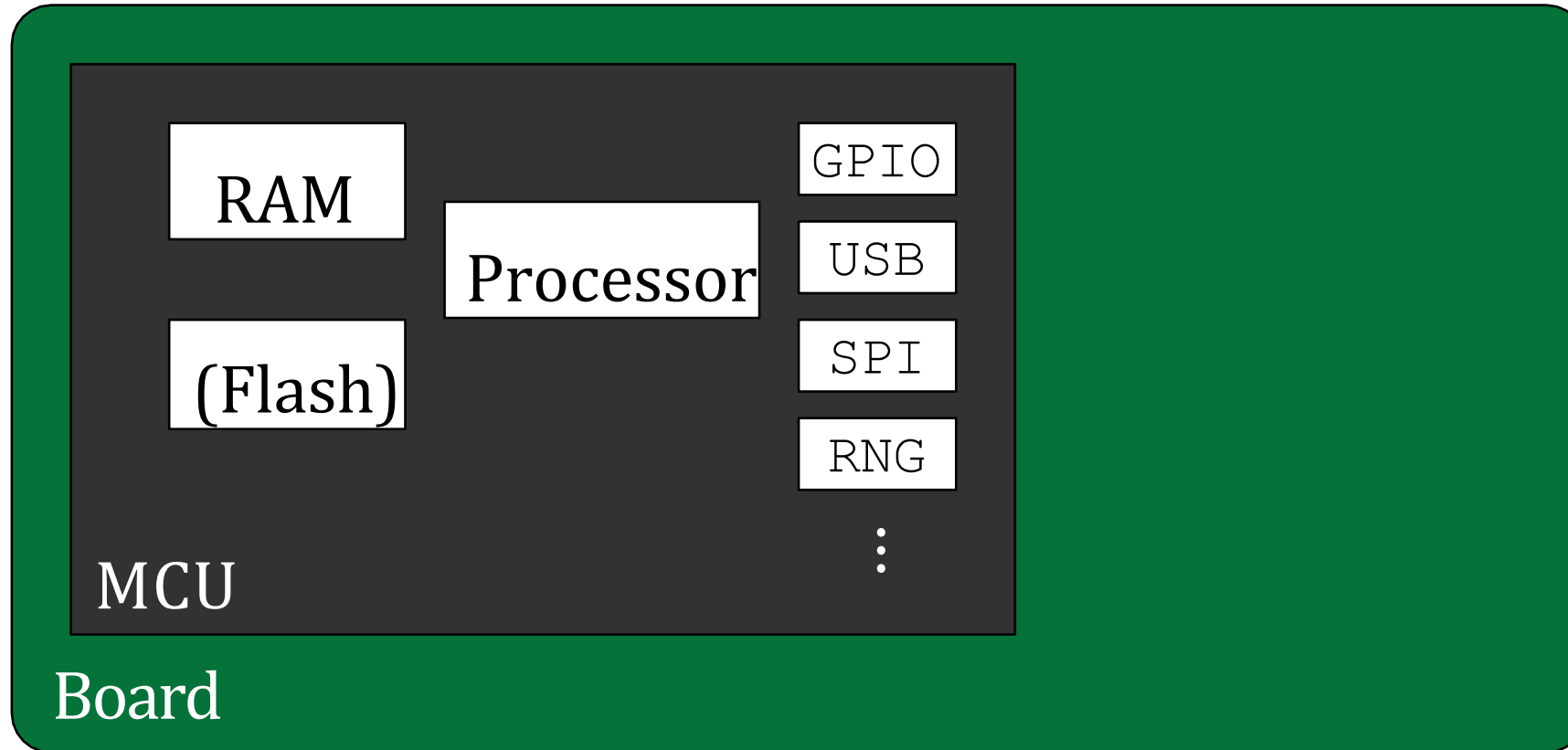
MCU: Micro-Controller Unit

Microcontrollers and Boards



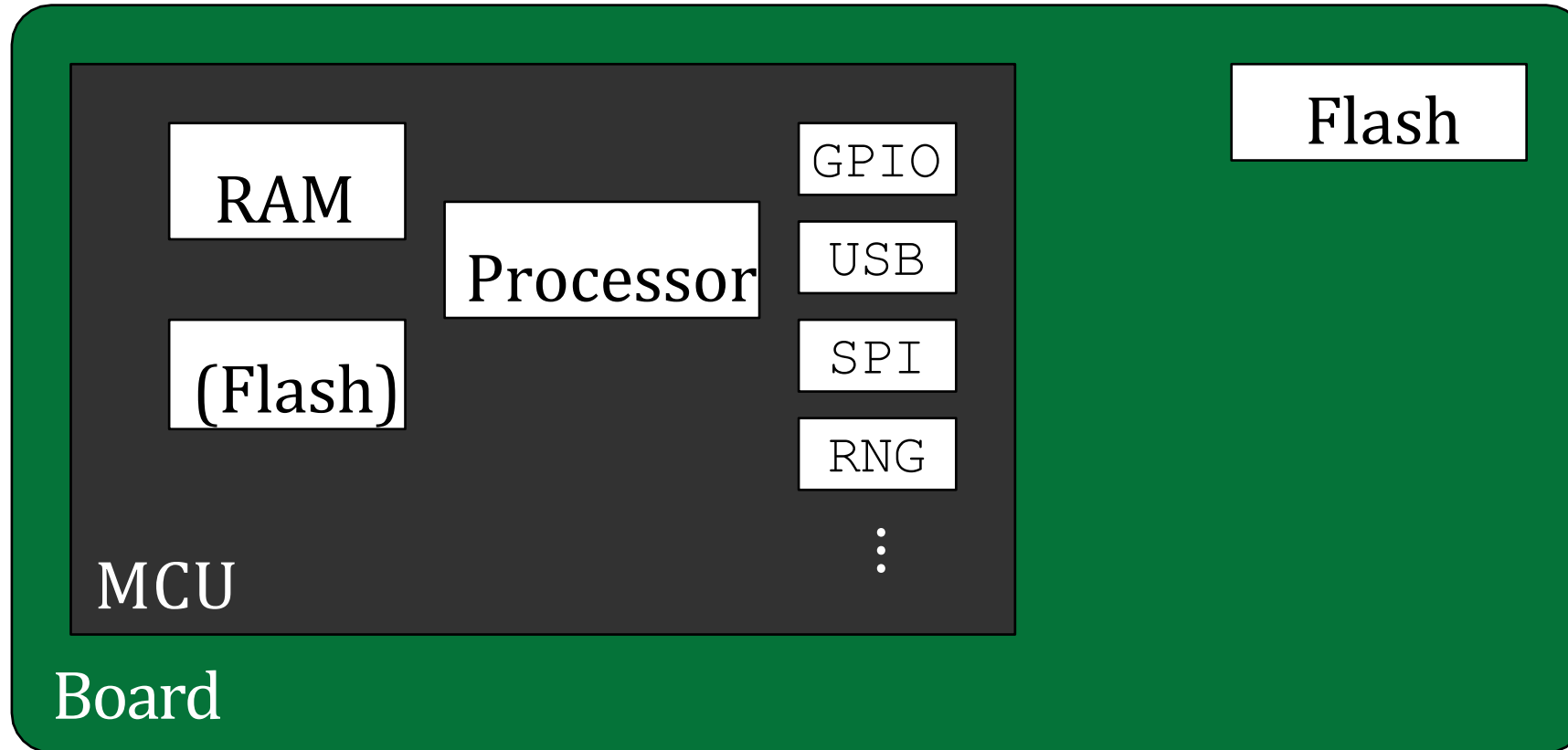
MCU: Micro-Controller Unit

Microcontrollers and Boards



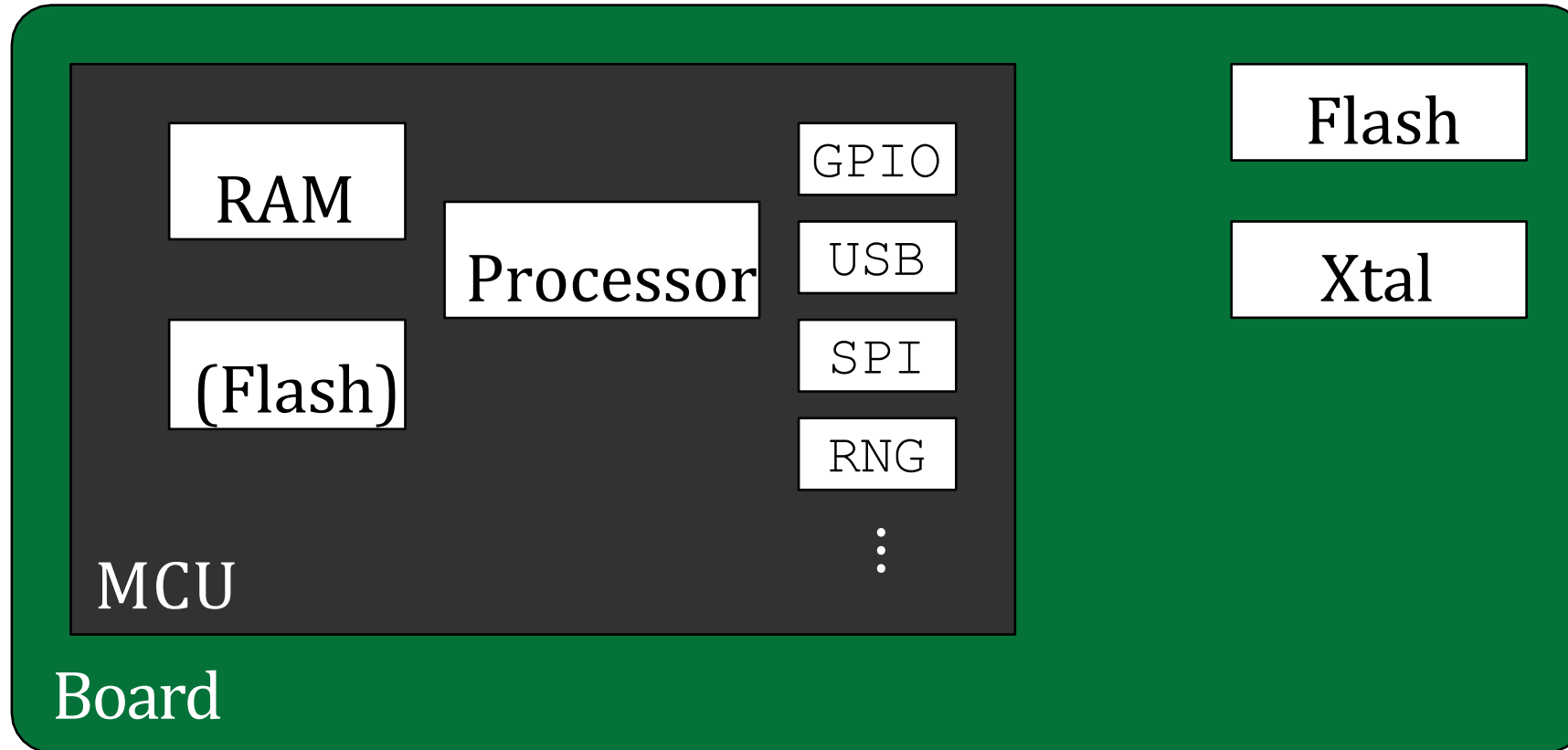
MCU: Micro-Controller Unit

Microcontrollers and Boards



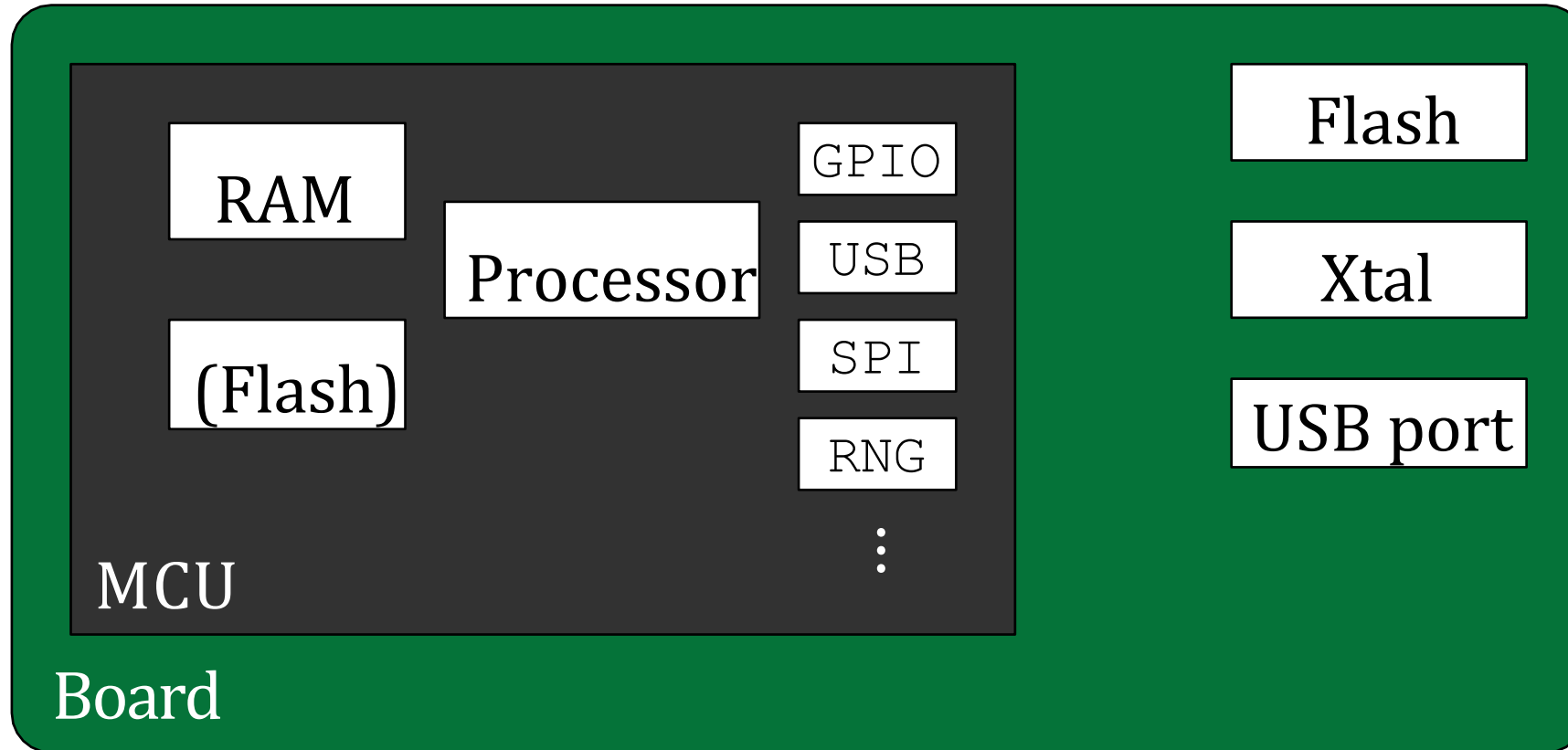
MCU: Micro-Controller Unit

Microcontrollers and Boards



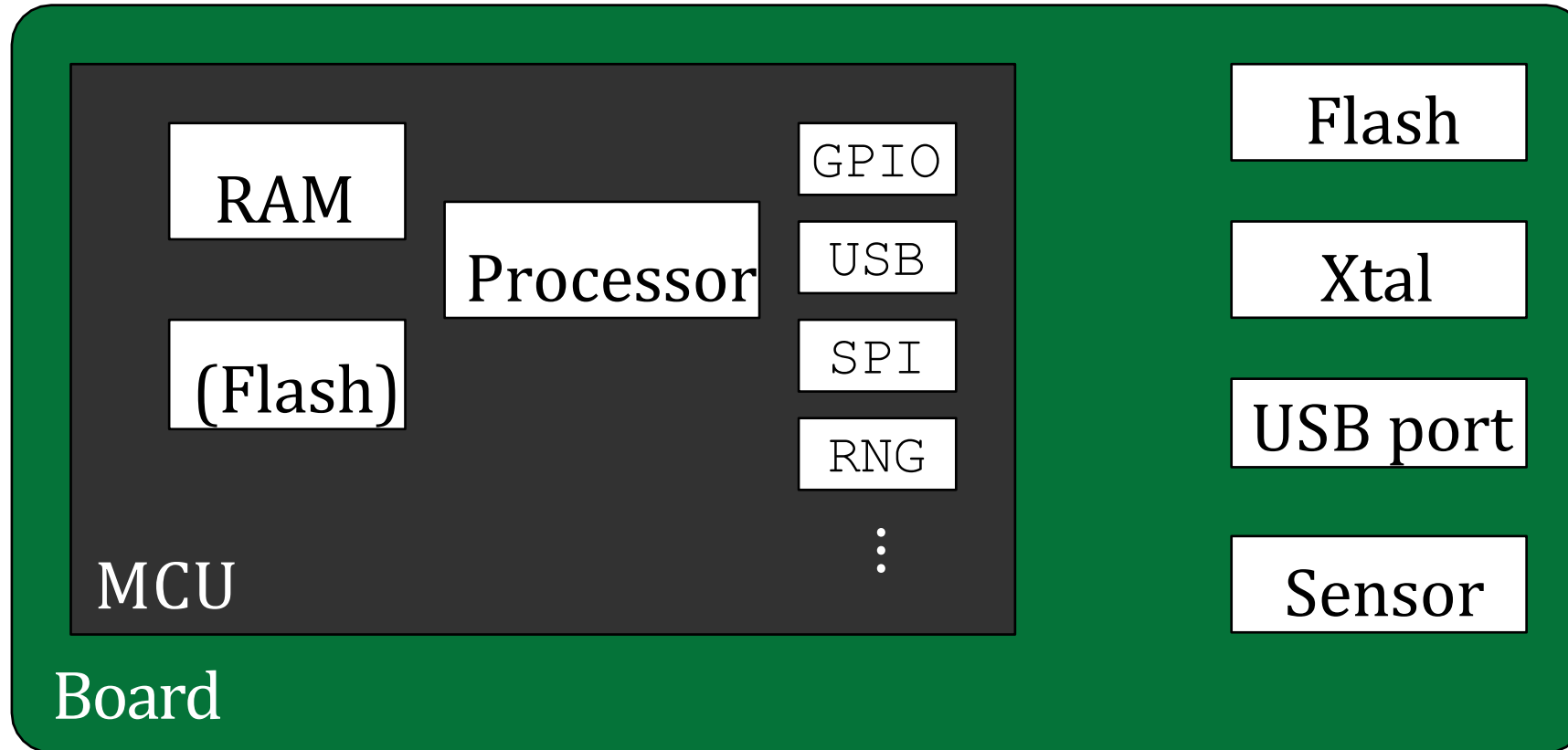
MCU: Micro-Controller Unit

Microcontrollers and Boards



MCU: Micro-Controller Unit

Microcontrollers and Boards



MCU: Micro-Controller Unit

How the Embedded Ecosystem Leverages Rust

1. Memory-Mapped Peripherals and Abstraction Layers
2. Modeling Memory-Mapped Peripherals w/ Ownership
3. Ecosystem Traits for Driver Interoperability
4. Concurrency with Async —Energy Efficiency

Memory-Mapped Peripherals and Abstraction Layers

Hardware Abstraction Library (HAL)

e.g., `embassy-nrf`

```
let mut led = Output::new(P0_13, Level::High);  
led.set_low();
```

Peripheral Access Crate (PAC), e.g., `nrf-pac`

```
// + GPIO configuration  
nrf_pac::P0.outclr().write(|w| w.set_pin(13, true));
```

Hardware (nRF52840)

```
// + GPIO configuration  
unsafe {  
    let gpio_p0_outclr = 0x5000_050c as *mut u32;  
    *gpio_p0_outclr = 1 << 13; // Set P0.13 low  
}
```

GPIO: General-Purpose Input/Output

Modeling Memory-Mapped Peripherals w/ Ownership

embassy-nrf/src/nvmc.rs

```
pub struct Nvmc<'d> {
    _p: Peri<'d, NVMC>,
}

impl<'d> Nvmc<'d> {
    pub fn new(_p: Peri<'d, NVMC>) -> Self {
        Self { _p }
    }
    // [...]
}
```

NVMC: Non-Volatile Memory Controller

ZST: Zero-Sized Type, e.g., `struct NVMC;`

main.rs

```
let config = Config::default();
let p = embassy_nrf::init(config);

let nvmc = Nvmc::new(p.NVMC);
```

Owning the peripheral ZST
singleton (`NVMC`) proves exclusive
access as it is `!Clone`.

Ecosystem Traits for Driver Interoperability

`embedded_hal::digital::OutputPin`

```
pub trait OutputPin: ErrorType {  
    // Required methods  
    fn set_low(&mut self) -> Result<(), Self::Error>;  
    fn set_high(&mut self) -> Result<(), Self::Error>;  
  
    // Provided method  
    fn set_state(&mut self, state: PinState) -> Result<(), Self::Error> { ... }  
}
```

- Implemented on GPIO output drivers, e.g., `Output`
- Used by HAL-agnostic driver implementations

Concurrency with Async —Energy Efficiency

main.rs

```
button.wait_for_falling_edge().await;
```

embassy-executor/src/arch/cortex_m.rs

```
impl Executor {  
    pub fn run(/* [...] */) -> ! {  
        // [...]  
        loop { unsafe {  
            self.inner.poll();  
            asm!("wfe");  
        } }  
    }  
}
```

WFE: Wait For Event

1. Interrupt triggers
2. Interrupt handler gets called
3. Interrupt handler calls registered waker's `.wake()`
4. Async executor calls `Future::poll()` on the future
5. Future returns `Poll::Pending` or `Poll::Ready(_)`
6. Executor puts the processor to sleep again (`wfe/wfi`)

Part Two: Introducing Ariel OS

Embassy and Application Portability —RNG example

~~stm32l4~~

```
bind_interrupts!(struct Irqs { RNG =>
    rng::InterruptHandler<peripherals::RNG>;
});

let config = /* board-specific clk config */; let p
= embassy_stm32::init(config);

let mut buf = [0u8; 16];

let mut rng = Rng::new(p.RNG, Irqs);
rng.fill_bytes(&mut buf);
```

rp235x

```
bind_interrupts!(struct Irqs {
    TRNG_IRQ =>
        trng::InterruptHandler<peripherals::TRNG>;
});

let config = /* board-specific clk config */;
= embassy_rp::init(config);

let mut buf = [0u8; 16];

let c = RngConfig::default();
let mut rng = Trng::new(p.TRNG, Irqs, c);
rng.blocking_fill_bytes(&mut buf);
```

Embassy and Application Portability —RNG example

~~stm32u0~~

```
bind_interrupts!(struct Irqs {  
    RNG_Cryp =>  
        rng::InterruptHandler<peripherals::RNG>;  
});
```

```
let config = /* board-specific clk config */;  
let p = embassy_stm32::init(config);
```

```
let mut buf = [0u8; 16];
```

```
let mut rng = Rng::new(p.RNG, Irqs);  
rng.fill_bytes(&mut buf);
```

stm32l4

```
bind_interrupts!(struct Irqs {  
    RNG =>  
        rng::InterruptHandler<peripherals::RNG>;  
});
```

```
let config = /* board-specific clk config */;  
let p = embassy_stm32::init(config);
```

```
let mut buf = [0u8; 16];
```

```
let mut rng = Rng::new(p.RNG, Irqs);  
rng.fill_bytes(&mut buf);
```

Ariel OS: an Embedded OS for IoT —Features

Application portability:
ESP32, nRF, RPi RP, STM32

Modularity & ecosystem
integration

Preemptive scheduler
with multicore support

Networking capabilities

Ariel OS: an Embedded OS for IoT —Features

Application portability:
ESP32, nRF, RPi RP, STM32

Modularity & ecosystem
integration

Preemptive scheduler
with multicore support

Networking capabilities

Ariel OS: an Embedded OS for IoT —Features

Application portability:
ESP32, nRF, RPi RP, STM32

Modularity & ecosystem
integration

Preemptive scheduler
with multicore support

Networking capabilities

Ariel OS: an Embedded OS for IoT —Features

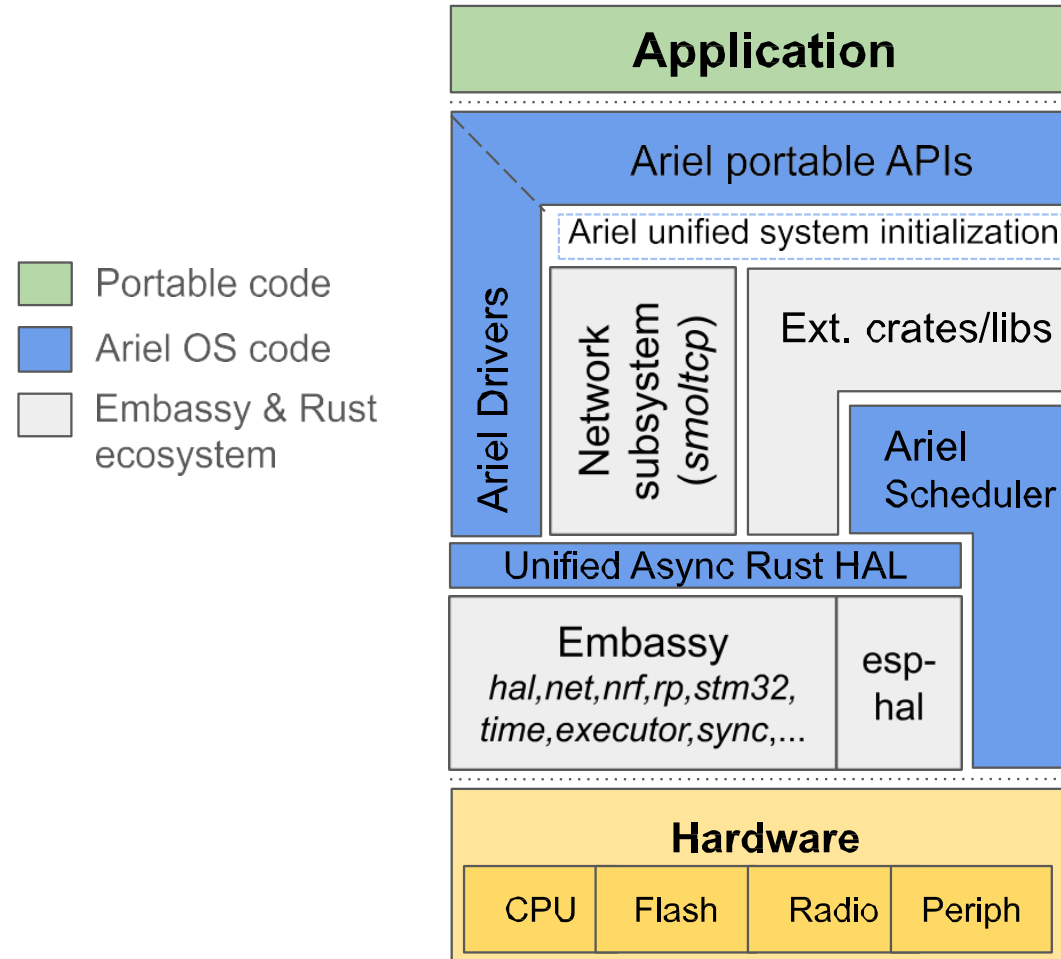
Application portability:
ESP32, nRF, RPi RP, STM32

Modularity & ecosystem
integration

Preemptive scheduler
with multicore support

Networking capabilities

Position in the Ecosystem



A Hello World Application

src/main.rs

```
#![no_main]
#![no_std]

use ariel_os::debug::{
    exit, log::info, ExitCode,
};

#[ Ariel_Os::task(autostart) ]
async fn main() {
    info!("Hello world!");
    exit(ExitCode::SUCCESS);
}
```

Cargo.toml

```
[package]
name = "ariel-os-hello"
version = "0.1.0"
edition = "2024"

[dependencies.ariel-os]
path = "build/imports/ariel-os/src/ariel-os"

[dependencies.ariel-os-boards]
path = "build/imports/ariel-os/src/ariel-os-boards"
```

The Build System: laze

Project layout

```
.
├── src
│   └── main.rs
├── Cargo.lock
├── Cargo.toml
└── laze-project.yml
```

laze-project.yml

```
imports:
  - git:
      url: https://github.com/ariel-os/ariel-os
      tag: v0.2.0
      dldir: ariel-os

apps:
  - name: ariel-os-hello
```

Compiling & flashing:

```
laze build -b nrf52840dk run
```

Using Digital Outputs

src/main.rs

```
#[ariel_os::task(autostart, peripherals)]
async fn blinky(p: LedPeripherals) {
    let mut led = Output::new(p.led, Level::Low);

    loop {
        led.toggle();
        Timer::after_millis(500).await;
    }
}

#[cfg(context = "nrf52840dk")]
ariel_os::hal::define_peripherals!(LedPeripherals {
    led: P0_13
});
```

Cargo.toml

```
[dependencies.ariel-os]
# [...]
features = ["time"]
```


Using an RNG

src/main.rs

```
use rand::Rng as _;

#[ariel_os::task(autostart)]
async fn main() {
    let mut rng = ariel_os::random::fast_rng();
    let value = rng.gen_range(1..=6);
    info!("Dice roll: {}", value);
}
```

Cargo.toml

```
[dependencies.ariel-os]
# [...]
features = ["random"]
```

Setting up Networking

src/main.rs

```
#[ariel_os::task(autostart)]
async fn main() {
    let stack = net::network_stack().await.unwrap();

    let mut socket = UdpSocket::new(
        stack,
        // ...buffers
    );
}
```

laze-project.yml

```
apps:
  - name: <name>
    selects:
      - network
```

See the `http-client`, `http-server`, `tcp-echo`, `udp-echo` examples.

Selecting the Network Link

Defaults to Wi-Fi:

```
CONFIG_WIFI_NETWORK=<ssid> CONFIG_WIFI_PASSWORD=<pwd> laze build -b rpi-pico-w
```

Equivalent to:

```
[CONFIG_WIFI_*] laze build --select wifi-cyw43 -b rpi-pico-w
```

Selecting Ethernet-over-USB instead:

```
laze build --select usb-ethernet --disable network-config-dhcp -b rpi-pico-w
```

Future Plans

Sensor abstraction: get all your
async readings at once!

Out-of-tree board definitions:
bring your own board!

Low-power management:
run on battery for months!

OTA updates: update your
deployed devices securely!

Future Plans

Sensor abstraction: get all your
async readings at once!

Out-of-tree board definitions:
bring your own board!

Low-power management:
run on battery for months!

OTA updates: update your
deployed devices securely!

Future Plans

Sensor abstraction: get all your
async readings at once!

Out-of-tree board definitions:
bring your own board!

Low-power management:
run on battery for months!

OTA updates: update your
deployed devices securely!

Future Plans

Sensor abstraction: get all your
async readings at once!

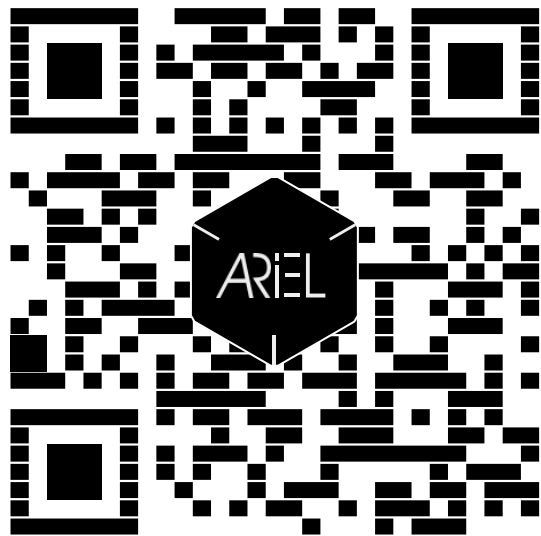
Out-of-tree board definitions:
bring your own board!

Low-power management:
run on battery for months!

OTA updates: update your
deployed devices securely!

Thanks!

Come and try it out!



<https://ariel-os.org>

Multithreading with Multicore Support

src/main.rs

```
#[ariel_os::thread(autostart, priority = 2)]
fn thread0() {
    info!("Hello from thread0!");
}

#[ariel_os::thread(autostart)]
fn thread1() {
    info!("Hello from thread1!");
}
```

laze-project.yml

```
apps:
  - name: threading-multicore
    selects:
      - sw/threading
      - multi-core
```

Abstraction without Limitation

ariel-os-embassy/src/gpio.rs

```
impl OutputBuilder {
    pub fn speed(self, speed: Speed<HalSpeed>) -> Self {
        const { assert!(
            hal::gpio::output::SPEED_CONFIGURABLE,
            "This HAL does not support setting the speed of GPIO outputs."
        ); }
        Self { speed, ..self }
    }
}
```

ariel-os-nrf/src/gpio.rs

```
pub mod output {
    pub const SPEED_CONFIGURABLE: bool = false;
}
```

Ecosystem Traits for Driver Interoperability

Digital IO

- `embedded_hal::digital::{InputPin, OutputPin}`
- `embedded_hal_async::digital::Wait`, **e.g.**, `Wait::wait_for_low()`
- `embedded_hal::pwm::SetDutyCycle`

Communication Buses

- `{embedded_hal, embedded_hal_async}::i2c::I2c`
- `{embedded_hal, embedded_hal_async}::spi::{
 SpiBus, SpiDevice
}`

RUST PARIS 2025



Pause

Reprise à 16h



RUST PARIS 2025

Florian Bennetot

DevOps Engineer

Parsec



- Une démarche de certification industrielle avec Ferrocene et TrustInSoft Analyzer

Aïssata Maïga



Ingénieure logiciel
Ferrous Systems



Sasha Pourcelot



Ingénieure Rust & Ocaml
TrustInSoft



Une démarche de certification industrielle avec Ferrocene et TrustInSoft Analyzer

Sasha Pourcelot & Aïssata Maïga

Qui sommes-nous ?

Sasha :

- Équipe Rust de TrustInSoft
- Membre du Safety Critical Rust Consortium

Aïssata :

- Codebase team de Ferrous Systems : PR et known problems
- CriticalUp : toolchain manager (un "rustup sécurisé")

Objectif : définir comment Rust peut être utilisé dans les systèmes critiques

Membres : Woven, OxidOS, Ferrous Systems, Adacore, Arm, ...

La mission de TrustInSoft

Méthodes formelles pour la certification

Objectif : prouver l'absence d'Undefined Behavior

unsafe → aucune garantie de memory safety

panic! ou overflow

Foreign Function Interface (FFI) : aucune garantie que le code C/C++ appelé ne comporte pas d'UB

→ Toujours possible d'écrire des bugs en Rust

TrustInSoft Analyzer

Initialement : analyseur statique de code C

Objectif : détection exhaustive d'UB

Support de Rust : détection d'UB, détection de panic!

Foreign Function Interface (FFI) avec C/C++ et détection des UBs associés

TrustInSoft Analyzer Root Cause Investigator

The screenshot displays the TrustInSoft Analyzer Root Cause Investigator interface, showing a Rust code file named `verify_something` with a highlighted error: **Division by zero**.

Overview Panel:

- Analysis finished after <1sec
- 22 potential undefined behaviors have been found.
- 1 file does not contain functions used for the analysis.
- Coverage: 0% of functions; 100% of statements.
- The entry point function successfully returns.
- Architecture: gcc - X86-32bits mode (little endian).

Inspector Panel:

- Term: `acc`
- Show Defs, Occurrences, Track value
- Show Defs, Show Defs[...]
- Clear tracking

Call tree Panel:

- driver_generalized
- `verify_something`

Main Code Editor:

```
uint32_t verify_something(u32 const x)
{
    uint32_t __retres;
    lib::Keys keys;
    usize key_index;
    u32 acc;
    keys.public._[0] = 3057U;
    keys.public._[1] = 6115U;
    keys.public._[2] = 1753U;
    keys.private._[0] = 812U;
    keys.private._[1] = 2439U;
    keys.private._[2] = 303U;
    key_index = 0U;
    acc = 0U;
    while (1) {
        if (key_index < 3U) {
            u32 tmp;
            {
                tmp = lib::<lib::Keys>::decrypt_element(& keys, x, key_index);
                acc += tmp;
            }
            key_index ++;
        }
        else {
            break;
        }
    }
    __retres = 1U / acc;
    return __retres;
}
```

Error Details:

- Division by zero**
- Predicate:** `| assert Value: division_by_zero: acc ≠ 0;`
- Inspect acc, Generate report

Right Panel (Code Editor):

```
extern "C" {
    fn do_decrypt(value: u32, _public: u32, _private: u32) -> u32;
}

struct Keys {
    private: [u32; 3],
    public: [u32; 3],
}

impl Keys {
    fn decrypt_element(&self, value: u32, key_index: usize) -> u32 {
        // The real computation is done in a C routine, to have a Rust -> C
        // call.
        unsafe { do_decrypt(value, self.public[key_index], self.private[key_index]) }
    }
}

// Use the no_mangle attribute so that we can call this function from C easily.
#[no_mangle]
pub fn verify_something(x: u32) -> u32 {
    let keys = Keys {
        public: [3057, 6115, 1753],
        private: [812, 2439, 303],
    };
    let mut key_index = 0;
    let mut acc = 0u32;

    while key_index < 3 {
        // Decrypt the current key element.
        acc = acc + keys.decrypt_element(x, key_index);

        // Treat next key element.
        key_index += 1;
    }

    // Do the final verification.
    // According to the chinese remainder theorem, there is one unique number x
    // between 0 and 3057 * 6115 * 1753 = 32_769_801_915 such that acc will be
    // 0. Since 32_769_801_915 is bigger than u32::MAX (4_294_967_295), there is
    // a chance that this function is completely safe.
    1 / acc
}
```

Bottom Panel:

- Run, Slevel: 10, Split strategy: None, Merge after loop: Enabled, Unfocus callstack
- Values, Potential UBs
- Track a new term, Values: All, Per Callstack, Per Call, Per State, Group by: Term, Emission rank, Format: Integers, Floats, Add a predicate
- Flags, Expression, Before, After, Is e..., Search
- acc local var. (u32) € [0..10922] € [1..10922]

TrustInSoft Analyzer report

minimal-rust-demo	100%	96%	96%
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     // is a chance that this function is completely safe.
41     1 / acc
42 }
43
```

✗ 1 alarm has been found. ▼

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

```
20 pub fn verify_something(x: u32) -> u32 {
21     let keys = Keys {
25         let mut key_index = 0;
26         let mut acc = 0u32;
27
28     while key_index < 3 {
35
36         // Do the final verification.
37         // According to the chinese remainder theorem, there is one unique number x
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 is
40         // a chance that this function is completely safe.
41     1 / acc
42 }
43
```

Predicate

$acc \neq 0$

Values and expressions

From analysis generalized

$acc \in [0..10922]$

Callstack

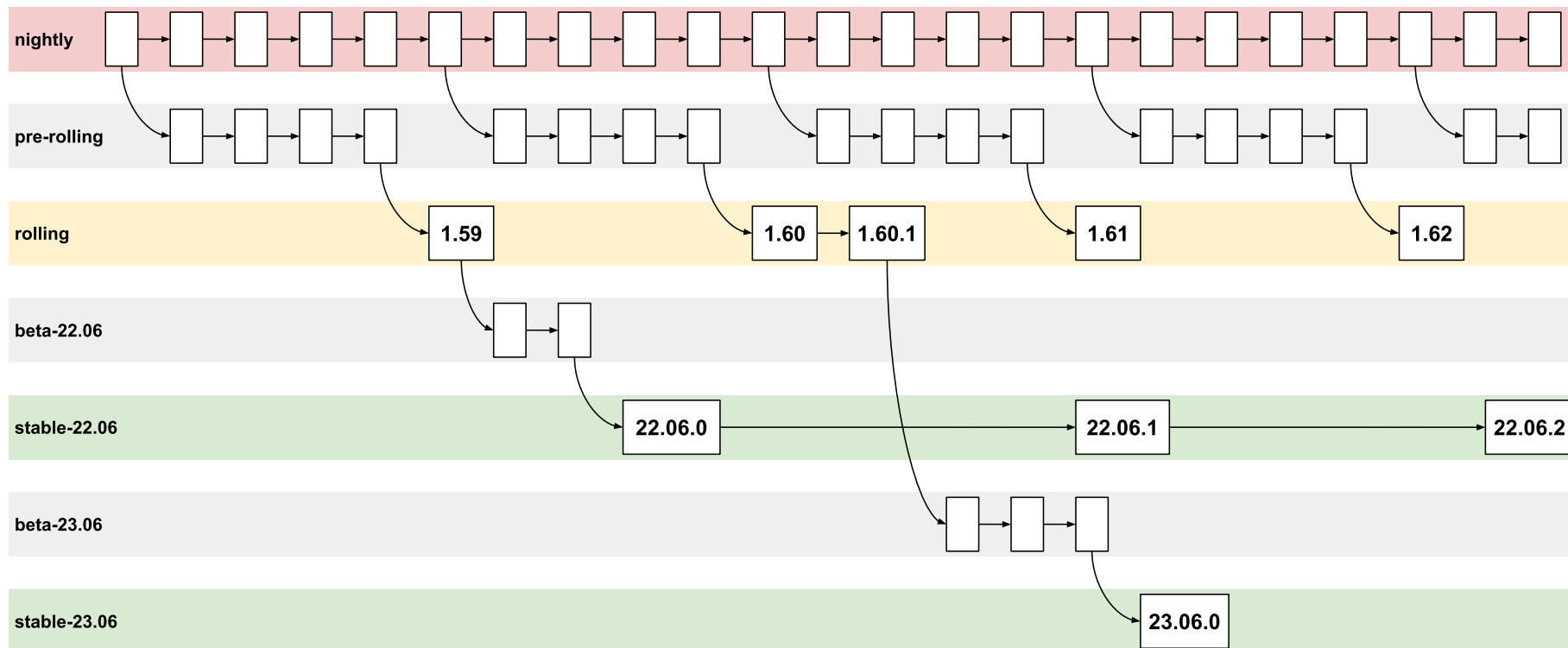
Ferrocene Language Specification

- Initialement écrite par Ferrous Systems
- Annotation des tests pour répondre aux specifications
- Adoptée par les devs du compilateur !
- Utilisée par TrustInSoft pour le développement du plugin Rust

Traceability matrix report Go back to the documentation index					
Page	Specification sections				Command line opt
	Linked #	Informational #	Total #	Completion	Linked # Informa
FLS: Associated Items	1	0	1	100.00%	
FLS: Attributes	56	1	57	100.00%	
FLS: Concurrency	4	0	4	100.00%	
FLS: Entities and Resolution	24	0	24	100.00%	
FLS: Exceptions and Errors	3	0	3	100.00%	
FLS: Expressions	59	0	59	100.00%	
FLS: FFI	5	0	5	100.00%	
FLS: Functions	1	0	1	100.00%	
FLS: General	0	8	8	100.00%	
FLS: Generics	5	0	5	100.00%	
FLS: Glossary	0	778	778	100.00%	
FLS: Implementations	3	0	3	100.00%	
FLS: Inline Assembly	9	1	10	100.00%	
FLS: Items	1	0	1	100.00%	
FLS: Lexical Elements	24	1	25	100.00%	
FLS: Licenses	0	4	4	100.00%	
FLS: List of undefined behavior	0	1	1	100.00%	

Ferrocene

1. Downstream de rustc
2. Au plus près du compilateur, avec release de Ferrocene toutes les 2 releases de Rust

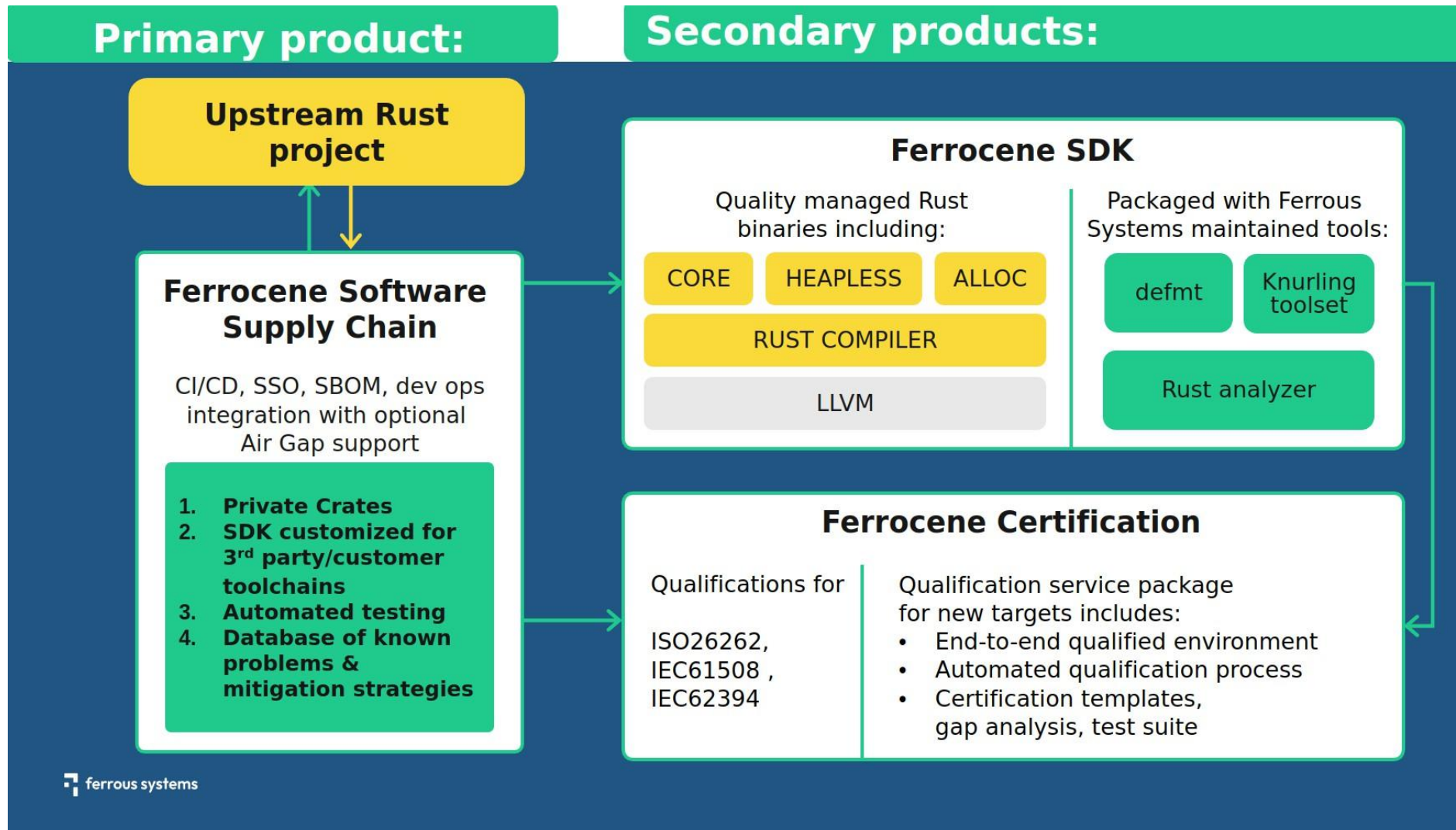


Ferrocene

1. SDK pour différentes versions de Rust, customisé pour les environnements critiques ou air-gapped
2. Construit pour les Software factories
3. Certifié par TÜV
4. Signature de chaque version
5. Fixing the bugs upstream



« Ferrocene II : visualisation du produit »



Known Problems?

- Identification
- Versions affectées
- Description
- Workaround ou "solution"

Search...



Overview

Versions and problems:

Ferrocene 23.06

Ferrocene 24.05

Ferrocene 24.08

Ferrocene 24.11

Ferrocene 25.02

Ferrocene 25.05

Ferrocene nightly

Incorrectly typed external declarations may break other external items

Problem ID:

KP-R46188

Tag:

wrong-compiler-code

References:

- <https://github.com/rust-lang/rust/issues/46188>

Affected branches:

- Ferrocene nightly
- Ferrocene 23.06
- Ferrocene 24.05
- Ferrocene 24.08
- Ferrocene 24.11
- Ferrocene 25.02
- Ferrocene 25.05

Description

Declaring an external static with a name that is already in use by another external static may impact expressions referring to the existing declared static.

Declaring an external function with a name that is already in use by another external function with a different signature may impact calling expressions to the existing external function.

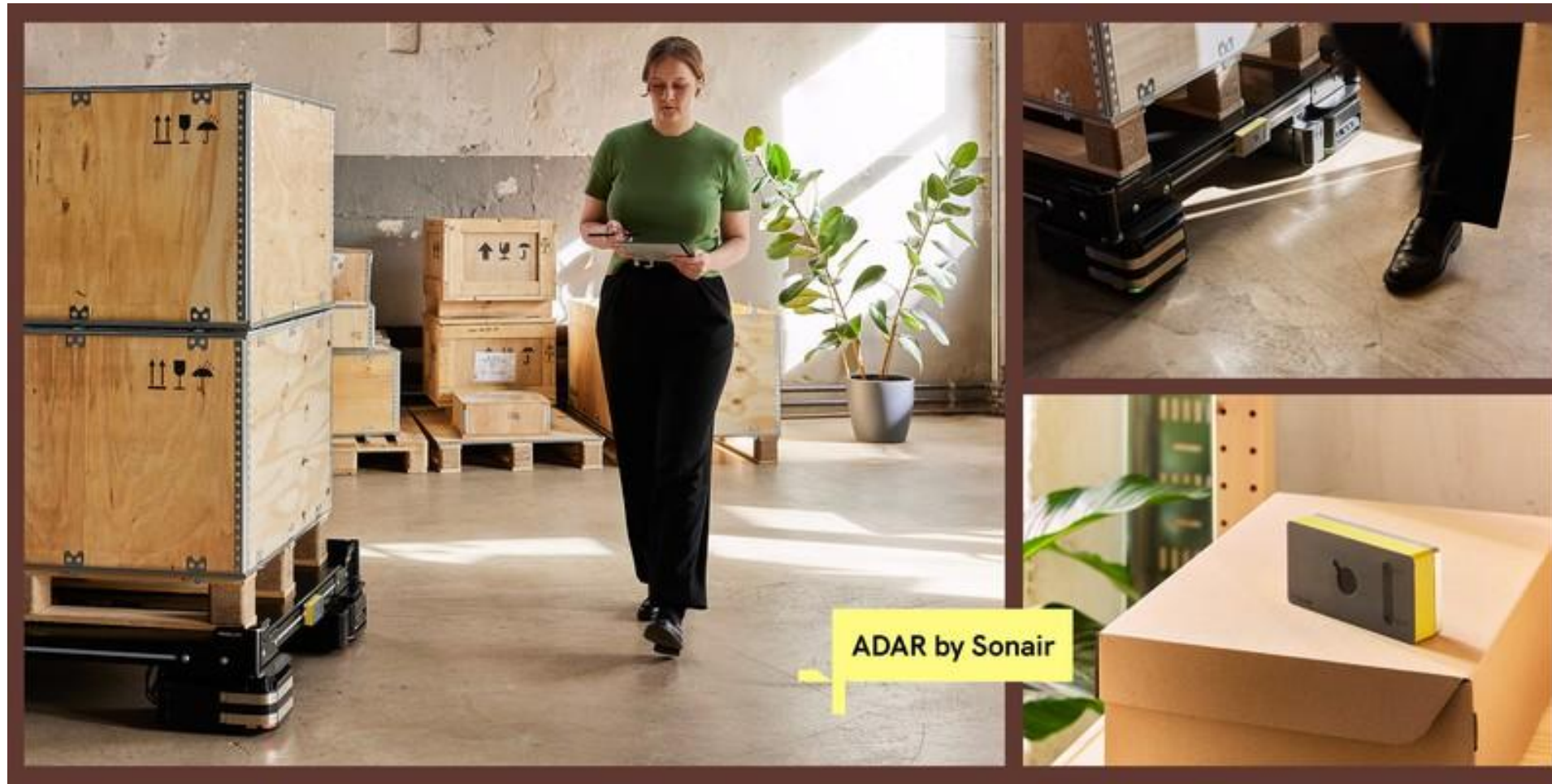
```
#[no_mangle]
pub unsafe fn a() -> *const u8 {
    extern { pub fn malloc(_: usize) -> &'static (); }
    malloc(usize::MAX) as *const _ as *const _
}
#[no_mangle]
pub unsafe fn b() -> *const u8 {
    extern { pub fn malloc(_: usize) -> *const u8; }
    let p = malloc(usize::MAX);
    if p.is_null() { panic!(); }
    p
}
```

The declaration of `malloc` in function `a` incorrectly applies the non-nullability of its return type to the declaration of `malloc` in function `b`, causing the `if` expression to be removed from the generated code.

Workaround

Do not declare external functions and statics with a preexisting name more than once in a program.

Sonair : enlever le « oups » de l'interaction humains-robots



Ferrocene et TrustInSoft Analyzer

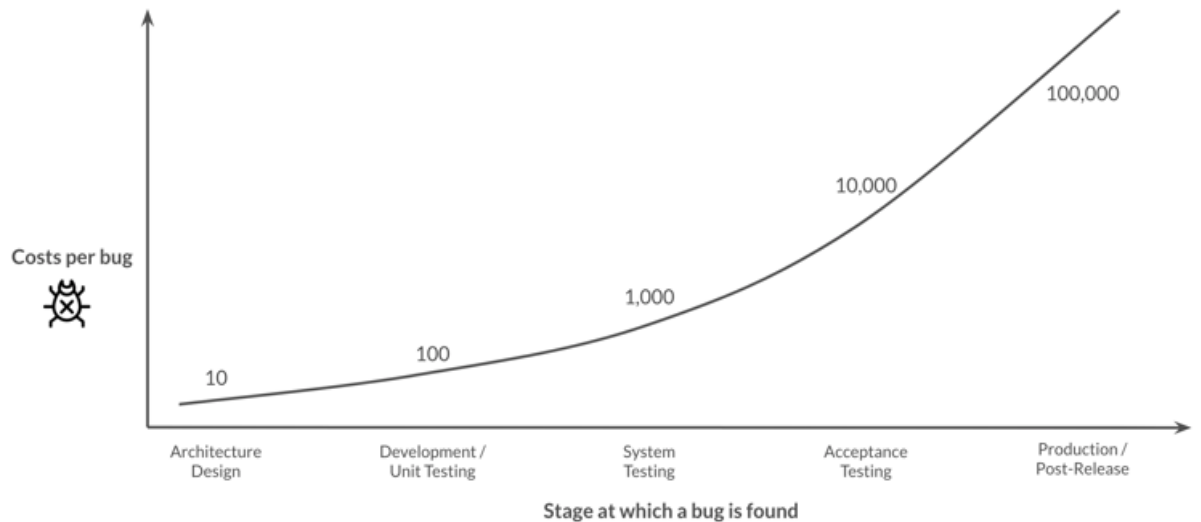
Un partenariat logique :

- Deux étapes dans une chaîne de certification

Deux outils, deux utilités :

- Ferrocene : compilateur qualifié pour systèmes critiques
- TrustInSoft Analyzer : vérifier le code

(You can't go wrong)



<https://www.code-intelligence.com/blog/rule-of-ten>

Merci!

Contact :

sasha.pourcelot@trust-in-soft.com

aissata.maiga@ferrous-systems.com

Guillaume Claret



Fondateur

Ingénieur en méthodes formelles

Formal Land



Formal Land

■ Vérification Formelle de Rust pour Ethereum

coq-of-rust



Formal verification for Rust in Rocq

Paris 2025-06-25

FORMAL LAND

- Formal verification company
- Web3, cryptography

<https://formal.land/>

Paris, France



FORMAL VERIFICATION

The idea of making sure a program is 100 % correct

EXAMPLES

- Backward compatibility between versions
- Absence of runtime errors
- Business rules

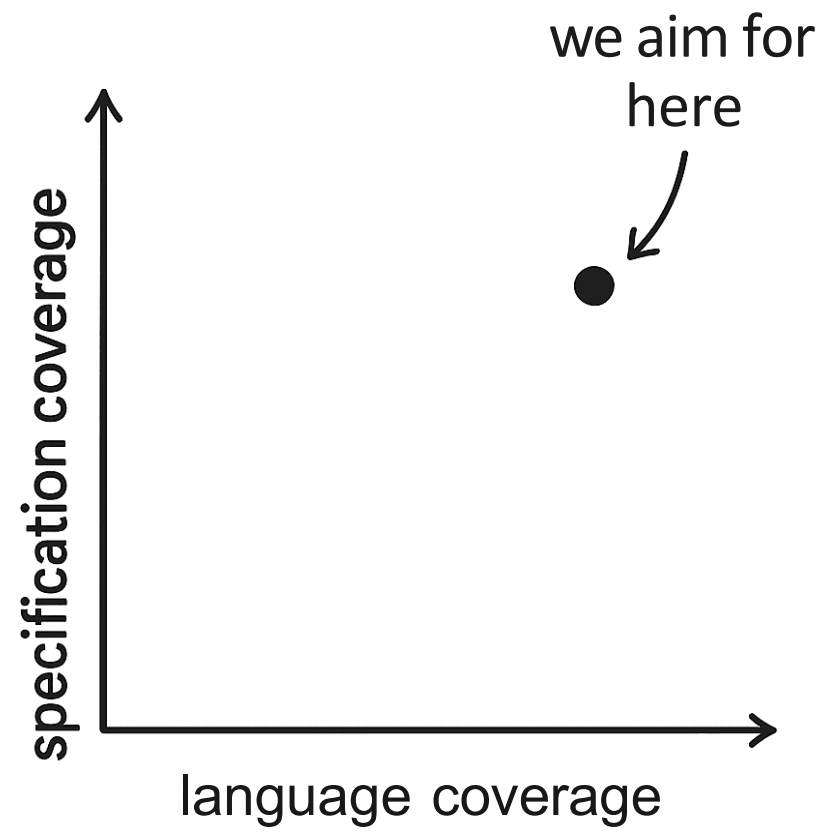
TECHNIQUE

- A mathematical analysis of the code
- Verified in a proof language

COQ-OF-RUST

- Formal verification tool for Rust 
- By translation to the proof language [Rocq](#)
- Open source

github.com/formal-land/coq-of-rust



WE PROVIDE

- Application
- Training

APPLICATION



APPLICATION

Specify Revm interpreter (WIP)

- Ethereum's virtual machine
- 5,000 lines of Rust (core)

REVM

Responsible of executing smart contracts on Ethereum

\$300B in valuation

CODE EXTRACT

ADD instruction

```
pub fn add<
  WIRE: InterpreterTypes, H: ?Sized
>(context: InstructionContext<'_, H, WIRE>) {
  gas!(context.interpreter, gas::VERYLOW);
  popn_top!([op1], op2, context.interpreter);
  *op2 = op1.wrapping_add(*op2);
}
```



GOAL

Show it is equivalent to a purely functional version



END GOAL

Show all EVM implementations are equivalent

OTHER APPLICATIONS

Verify the  zkVM implementations

STRATEGY



STEPS

1. THIR
2. Raw Rocq
3. Linking
4. Functional Rocq (WIP)

1. THIR

Obtained by `rustc`

2. RAW ROCQ

Generated by `coq-of-rust`

EXAMPLE

```
pub fn unwrap_or(self, default: T) -> T {  
    match self {  
        Some(x) => x,  
        None => default,  
    }  
}
```

RAW ROCQ

```
Definition unwrap_or (T : Ty.t) (ε : list Value.t) (τ : list Ty.t) (α : list Value.t) : M :=
  let Self : Ty.t := Self T in
  match ε, τ, α with
  | [], [], [ self; default ] => ltac:(M.monadic
    (let self := M.alloc (| self |) in
     let default := M.alloc (| default |) in
     M.read (|
       M.match_operator (|
         T,
         self,
         [
           fun γ => ltac:(M.monadic
             (let γ0_0 :=
                 M.SubPointer.get_struct_tuple_field (|
                   γ, "core::option::Option::Some", 0
                 |) in
             let x := M.copy (| γ0_0 |) in x));
           fun γ => ltac:(M.monadic
             (let _ := M.is_struct_tuple (| γ, "core::option::Option::None" |) in
              default))
         ]
       |)
     |)))
  | _, _, _ => M.impossible "wrong number of arguments" end.
```

RAW ROCQ

- Can translate the `core` library
- Most of the programs we tried



INGREDIENTS

- Untyped
- Names as strings
- Reification
- Almost no generated names

FOR NOW

- Mostly data 
- We need a semantics!

VALUES

```
Inductive t : Set :=
| Bool : bool -> t
| Integer (kind : IntegerKind.t) (z : Z) : t
| UnicodeChar : Z -> t
| String : string -> t
| Tuple : list t -> t
| Array : list t -> t
| StructRecord : string -> list t -> list Ty.t -> list (string * t) -> t
| StructTuple : string -> list t -> list Ty.t -> list t -> t
| Pointer : Pointer.t t -> t
| Closure : {'(Value, M) : (Set * Set) @ list Value -> M} -> t
| Error (message : string)
| DeclaredButUndefined.
```

POINTERS

```
Module Index.  
  Inductive t : Set :=  
  | Tuple (index : Z)  
  | Array (index : Z)  
  | StructRecord (constructor field : string)  
  | StructTuple (constructor : string) (index : Z).  
End Index.  
  
Module Path.  
  Definition t : Set := list Index.t.  
End Path.  
  
Module Core.  
  Inductive t (Value : Set) : Set :=  
  | Make {Address : Set} (address : Address) (path : Path.t).  
  Arguments Make {__ __}.  
End Core.
```

MONAD

```
Module LowM.  
  Inductive t (A : Set) : Set :=  
  | Pure (value : A)  
  | CallPrimitive (primitive : Primitive.t) (k : Value.t -> t A)  
  | CallClosure  
    (ty : Ty.t) (closure : Value.t) (args : list Value.t) (k : A -> t A)  
  | CallLogicalOp  
    (op : LogicalOp.t) (lhs : Value.t) (rhs : t A) (k : A -> t A)  
  | LetAlloc (ty : Ty.t) (e : t A) (k : A -> t A)  
  | Loop (ty : Ty.t) (body : t A) (k : A -> t A)  
  | MatchTuple (tuple : Value.t) (k : list Value.t -> t A)  
  | Impossible (message : string).  
End LowM.
```

PRIMITIVES

```
Module Primitive.  
  Inductive t : Set :=  
  | StateAlloc  
    (value : Value.t)  
  | StateRead  
    (pointer : Value.t)  
  | StateWrite  
    (pointer : Value.t)  
    (value : Value.t)  
  | GetSubPointer  
    (pointer : Value.t)  
    (index : Pointer.Index.t)  
  | GetFunction  
    (path : string)  
    (generic_consts : list Value.t)  
    (generic_tys : list Ty.t)  
  | GetAssociatedFunction  
    (ty : Ty t)
```

CONTROL FLOW

Error monad on top for:

- panics
- return
- break
- continue

3. LINKING

Target:

A smaller monad with just memory handling

WE HANDLE

- Typing
- Name resolution
- Trait resolution

LINKING MONAD

```
Module Primitive.  
  Inductive t : Set -> Set :=  
  | StateAlloc {A : Set} `{Link A} (value : A) : t (Ref.Core.t A)  
  | StateRead {A : Set} `{Link A} (ref_core : Ref.Core.t A) : t A  
  | StateWrite {A : Set} `{Link A} (ref_core : Ref.Core.t A) (value : A) :  
    t unit  
  | GetSubPointer {A : Set} `{Link A} {index : Pointer.Index.t}  
    (ref_core : Ref.Core.t A) (runner : SubPointer.Runner.t A index) :  
    let _ := runner.(SubPointer.Runner.H_Sub_A) in  
    t (Ref.Core.t runner.(SubPointer.Runner.Sub_A)).  
End Primitive.
```



TO MAKE THE LINKS

1. Prove there exists a name/trait resolution + typing
2. Generale the linked version

IDIOMATIC DATA TYPES

```
Class Link (A : Set) : Set := {  
   $\Phi$  : Ty.t;  
   $\varphi$  : A -> Value.t;  
}.
```

PREDICATE

$\{ \{ e \text{ } \triangle \text{ } R, \text{ Output} \} \}$

PREDICATE

```
| CallPrimitiveStateRead {A : Set} ` {Link A}
  (ref_core : Ref.Core.t A)
  (k : Value.t -> M) :
let ref : Ref.t Pointer.Kind.Raw A :=
  {| Ref.core := ref_core |} in
  (forall (value : A),
    {{ k (φ value)  R, Output }}
  ) ->
  {{
    LowM.CallPrimitive (Primitive.StateRead (φ ref)) k 
    R, Output
  }}
```

PREDICATE

```
| CallPrimitiveGetFunction
  (name : string) consts tys
  (function : PolymorphicFunction.t)
  (k : Value.t -> M) :
  let closure :=
    Value.Closure (existsS (_, _) (function consts tys)) in
  M.IsFunction.C name function ->
  {{ k closure ▾ R, Output }} ->
  {{
    LowM.CallPrimitive
      (Primitive.GetFunction name consts tys) k ▾
    R, Output
  }}
```


PREDICATE

We write it in interactive mode.

With automation.

PREDICATE

```
Instance run_unwrap_or {T : Set} `{Link T}
  (self : option T) (default : T) :
  Run.Trait
  (unwrap_or ( $\Phi$  T)) [] [] [  $\phi$  self;  $\phi$  default ]
  T.
Proof.
  constructor.
  run_symbolic.
Defined.
```

GENERATE

An eval function of type:

```
forall {R Output : Set} `{Link R} `{Link Output} {e : M}
  (run : {{ e  R, Output }}),
LowM.t R Output.
```

We apply it!

COVERAGE

95% of the instructions of Revm

COMPLEX CASES

Trait hierarchy $\triangle$ with multiple inheritance/associated types.

COMPLEX CASES

```
pub trait InterpreterTypes {  
    type Stack: StackTrait;  
    type Memory: MemoryTrait;  
    type Bytecode: Jumps  
        + Immediates  
        + LegacyBytecode  
        + EofData  
        + EofContainer  
        + EofCodeInfo;  
    type ReturnData: ReturnData;  
    type Input: InputsTrait;  
    type SubRoutineStack: SubRoutineStack;  
    type Control: LoopControl;  
    type RuntimeFlag: RuntimeFlag;  
    type Extend;
```

COMPLEX CASES

```
Class Run
  (Self : Set) `{Link Self}
  (types : Types.t) `{Types.AreLinks types} :
  Set := { Stack_IsAssociated :
    IsTraitAssociatedType "revm_interpreter::interpreter_types::InterpreterTypes" [] []
    (Φ Self) "Stack" (Φ types.(Types.Stack));
  run_StackTrait_for_Stack : StackTrait.Run types.(Types.Stack);
  Memory_IsAssociated :
    IsTraitAssociatedType "revm_interpreter::interpreter_types::InterpreterTypes" [] []
    (Φ Self) "Memory" (Φ types.(Types.Memory));
  run_MemoryTrait_for_Memory :
    MemoryTrait.Run
      types.(Types.Memory)
      types.(Types.Memory_Synthetic)
      types.(Types.Memory_Synthetic1);
  Bytecode_IsAssociated :
    IsTraitAssociatedType
```

4. FUNCTIONAL ROCQ (WIP)

We need to automated more memory manipulations



VERSION WITH ALIASES

Similar strategy:

- Going to a simpler monad
- Mix evaluation/proof

GENERATION

- Evaluation function
- Makes the allocation on a stack
- Requires "cast" as a primitive

GENERATION

Eval function:

```
forall {R Output : Set} {Stack : Stack.t}  
  (e : LowM.t R Output)  
  (stack : Stack.to_Set Stack) :  
t (Output.t R Output * Stack.to_Set Stack).
```

PROOF

Show we can eliminate the casts

PREDICATE

{ { e  value } }

PREDICATE

We need more automation to scale

FUNCTIONAL ROCQ

```
Definition unwrap_or {A : Set}
  (value : option A) (default : A) :
  A :=
  match value with
  | Some value => value
  | None => default
end.
```

EQUIVALENCE PROOF

```
Lemma unwrap_or_eq {A : Set} `{Link A}
  (value : option A) (default : A) :
  {{
    StackM.eval_f (Stack := [])
      (run_unwrap_or value default) tt
    🌲
    (Output.Success (unwrap_or value default), tt)
  }}.
```

Proof.

```
  destruct value; repeat (
    cbn ||
    get_can_access ||
    apply Run.Pure
  ).
```

Qed.

ANOTHER EXAMPLE

```
struct Numbers {  
    a: u64,  
    b: u64,  
    c: u64,  
}  
  
fn get_a_ref(numbers: &Numbers) -> &u64 {  
    &numbers.a  
}  
  
fn get_b_mut(numbers: &mut Numbers) -> &mut u64 {  
    &mut numbers.b  
}  
  
fn duplicate(a: &u64, b: &mut u64, c: &mut u64) {  
    *b = *a;  
    *c = *a;  
}  
  
fn apply_duplicate(numbers: &mut Numbers) {  
    duplicate(get_a_ref(&numbers), get_b_mut(&mut numbers), &mut numbers.c);  
}
```

ANOTHER EXAMPLE

Shown equal to:

```
Definition apply_duplicate (numbers : Numbers.t) :=  
  { |  
    Numbers.a := numbers.(Numbers.a);  
    Numbers.b := numbers.(Numbers.a);  
    Numbers.c := numbers.(Numbers.a);  
  | }.
```

DESIGN CHOICES

Leverage type-theory

(with Rocq)

DESIGN CHOICES

AI-friendly



=> Verbose but simple

SIZE

- 7,000 lines of Rust
- A few thousand lines of Rocq

REMARK

A lot of the techniques can apply to other programming languages (Go?)

THANKS!



Formal Land

Formal verification for everyday-life applications

<https://formal.land/>

RUST PARIS 2025



Mot de la fin



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** | 13 mars 2025 | 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 2025** | Paris, France et Online | 09 et 10 octobre 2025
- **RustConf 2025** | Seattle, Washington | 2 - 5 septembre 2025
=> <https://rustconf.com/>

Nos sponsors :



Systematic célèbre ses 20 ANS au service de l'innovation & des Deep Tech

Temps fort 1 : 13h30 – 17h30 : Séquence “Technologies & Ecosystème”

Construisez votre après-midi comme vous le souhaitez :

- **8 conférences (sur inscription)**
- **RV Business écosystème annuels**, en présentiel cette année, avec notamment des acheteurs du secteur de la Défense (sur inscription),
- **Investor Day**, réservé aux Investisseurs et start-up / PME labellisées Hi-France et/ou Champion du Pôle (sur inscription),
- **Place de l'Innovation** : le top de l'innovation avec des animations interactives sur chaque stand (accès libre),
- **Village des Accompagnements**, avec les équipes de Systematic (Innovation et Europe) ainsi que les membres du Club des prestataires pour répondre à toutes vos questions sur des sujets aussi variés qu'incontournables.

Temps fort 2 : 17h30 – 18h30 : Séquence “20 ans et demain ?”

Retour sur nos 20 ans et surtout, projection vers l'avenir

Temps fort 3 : 18h30 – 19h45 : Cocktail apéritif



Nos sponsors :



Cocktail de clôture

Merci à vous !



Systematic

Paris Region Deep Tech Ecosystem

L'action du pôle est soutenue par :



Partenaires officiels :



FRANCE RÉPUBLIQUE FRANÇAISE