Unleashing 🦀 The Ferris Within

EuroRust

October 2024

@ciura_victor
@ciura_victor@hachyderm.io
@ciuravictor.bsky.social

Victor Ciura
Principal Engineer
Rust Tooling @ Microsoft

Abstract

"Let's rewrite it in Rust" is no longer a party joke. It's happening!

Let me share a couple of stories of learning, appreciating and rewriting stuff in Rust. How we came to love 🦀 Ferris: cargo cult or real need?

What is it like to come to Rust from two very different directions: C++ and C#? What are the gaps, the needs, the gems and the tools you should know about? Here's a real journey and the various experiments leading up towards the success stories at Microsoft.

What have we learned and can bring back to day-to-day C++?
Want to compare notes? Let's chat.

About me

Advanced Installer

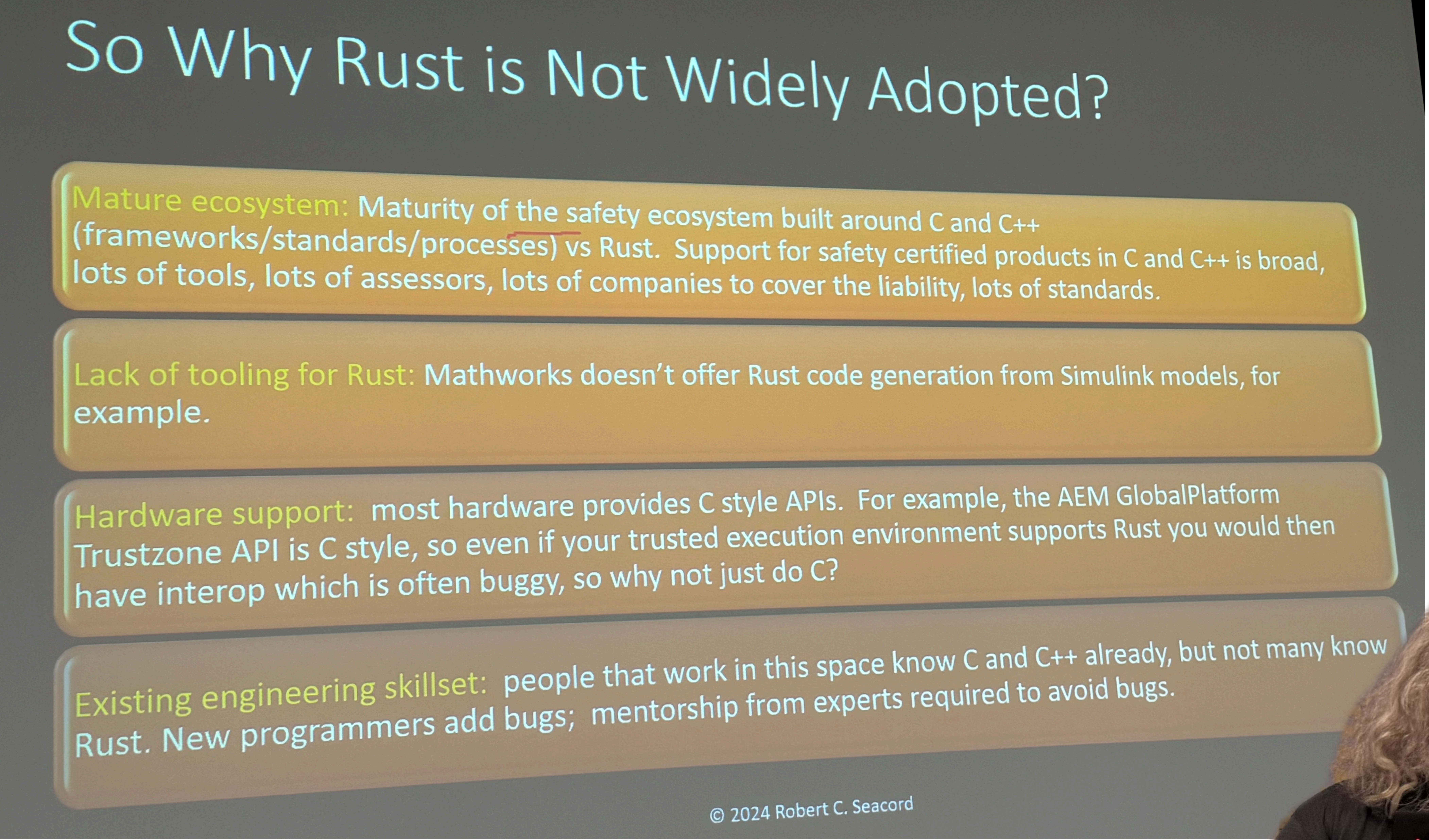
Clang Power Tools

Oxidizer SDK

Visual C++

Rust Tooling

🐦 @ciura_victor
🐘 @ciura_victor@hachyderm.io
🦋 @ciuravictor.bsky.social



So Why Rust is Not Widely Adopted?

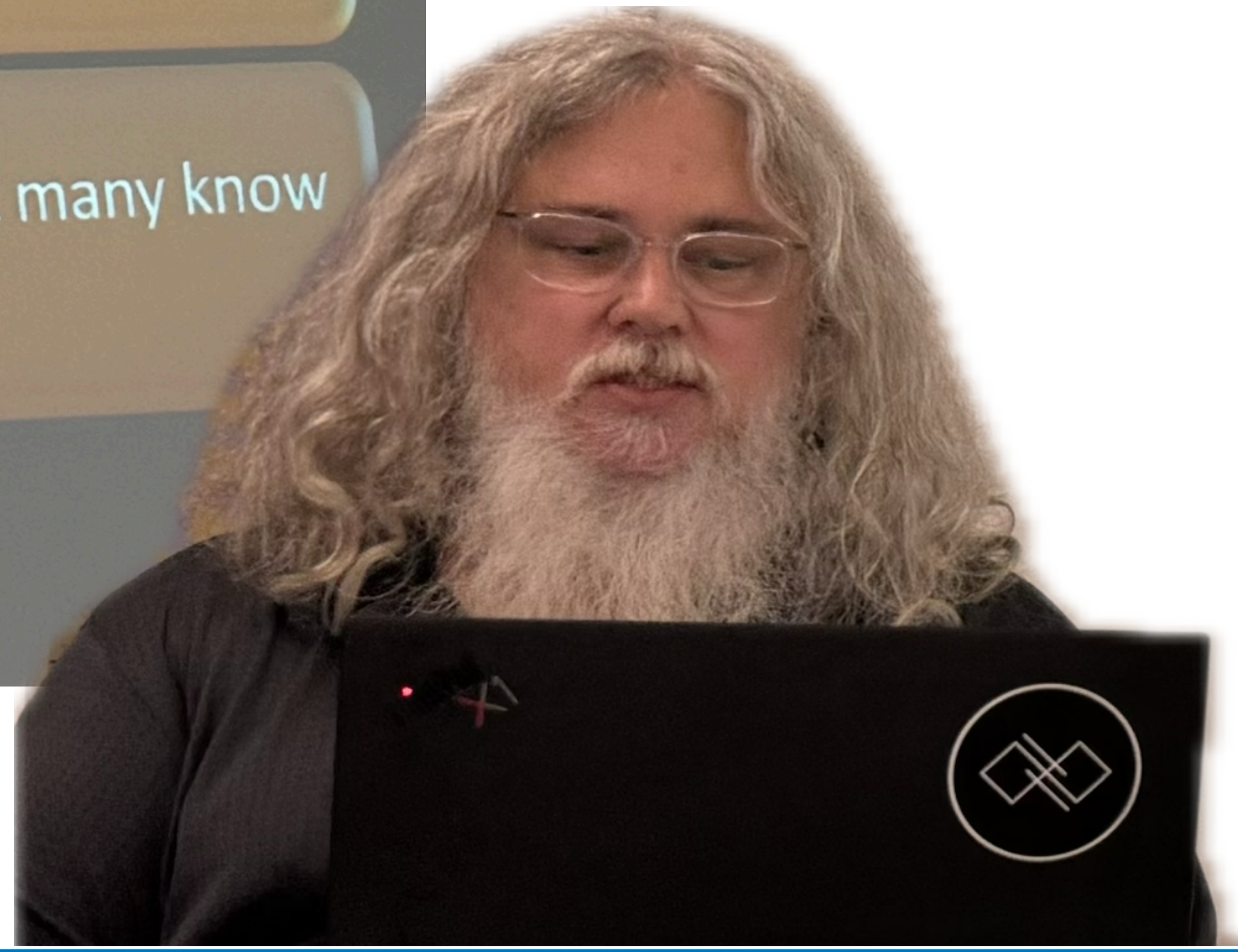
Mature ecosystem: Maturity of the safety ecosystem built around C and C++ (frameworks/standards/processes) vs Rust. Support for safety certified products in C and C++ is broad, lots of tools, lots of assessors, lots of companies to cover the liability, lots of standards.

Lack of tooling for Rust: Mathworks doesn't offer Rust code generation from Simulink models, for example.

Hardware support: most hardware provides C style APIs. For example, the AEM GlobalPlatform Trustzone API is C style, so even if your trusted execution environment supports Rust you would then have interop which is often buggy, so why not just do C?

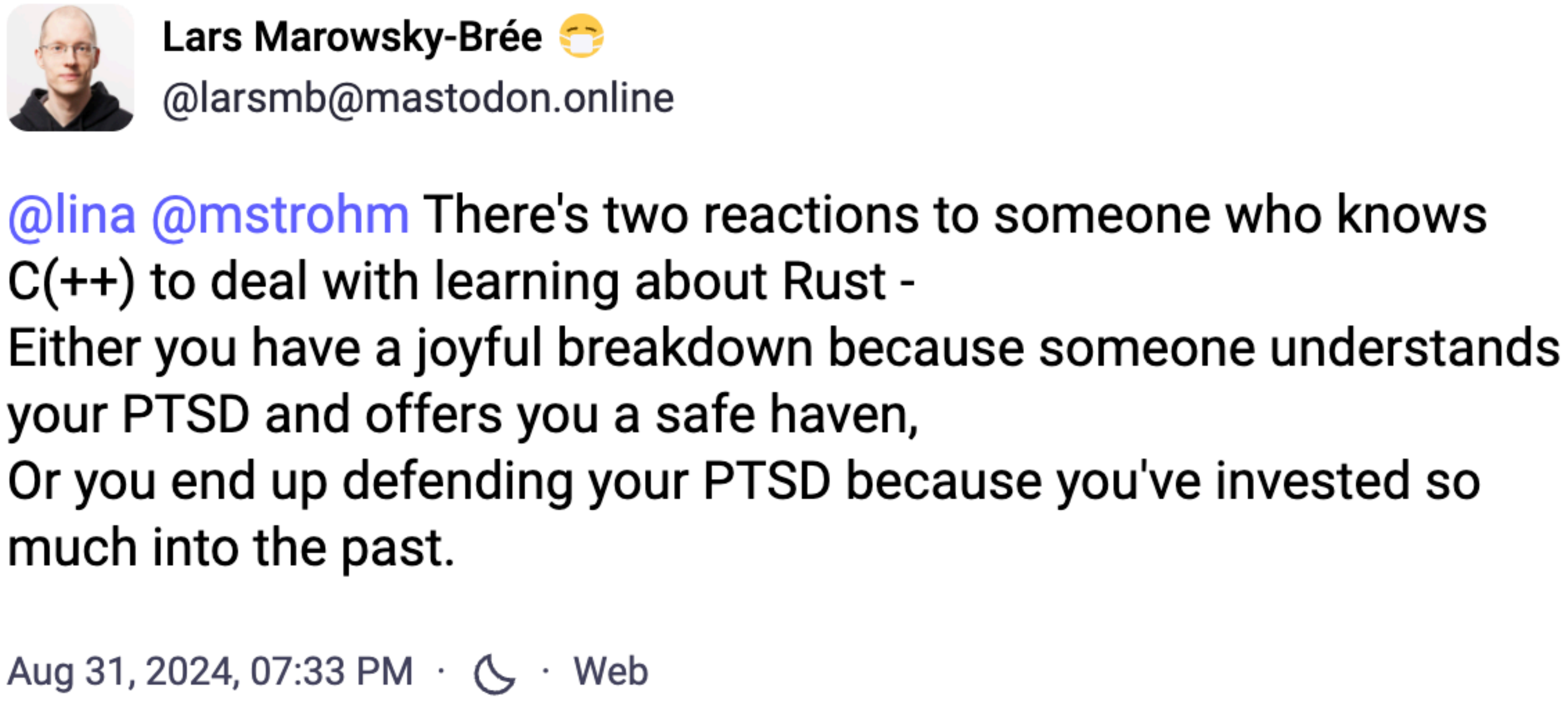
Existing engineering skillset: people that work in this space know C and C++ already, but not many know Rust. New programmers add bugs; mentorship from experts required to avoid bugs.

© 2024 Robert C. Seacord



ndctechtown.com/agenda/memory-safety-rust-vs-c

First Encounter



Lars Marowsky-Brée 😷
@larsmb@mastodon.online

@lina @mstrohm There's two reactions to someone who knows C(++) to deal with learning about Rust -
Either you have a joyful breakdown because someone understands your PTSD and offers you a safe haven,
Or you end up defending your PTSD because you've invested so much into the past.

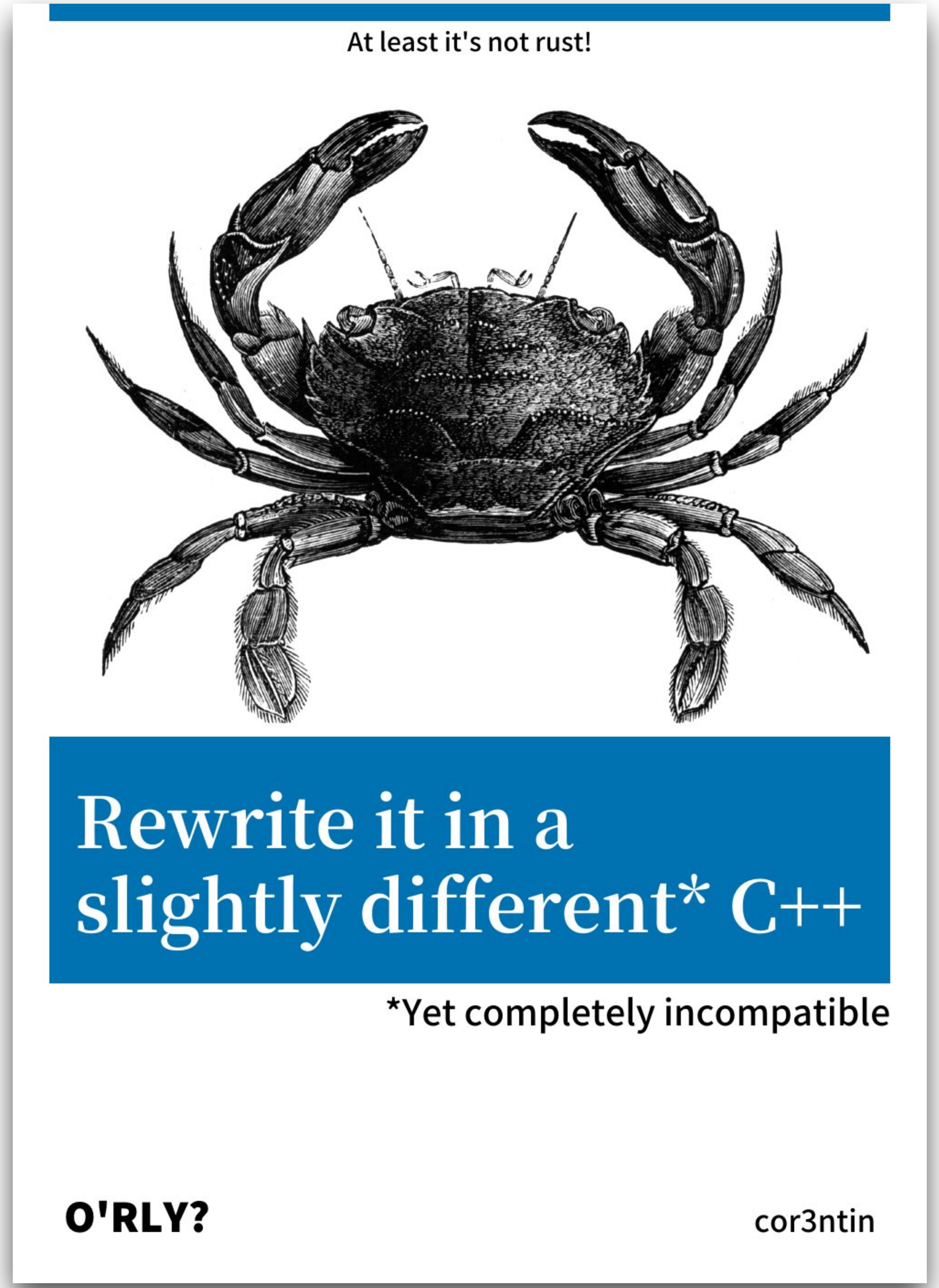
Aug 31, 2024, 07:33 PM · · Web

mastodon.online/@larsmb/113057830402545219

The C++ Playbook to Safety

Rewrite it in... ~~Rust~~ SafeC++? 🤦‍♂️

- A cakewalk and eating it too
- But we care about safety, right?
- Dogma... galore
- Down with Safety!
- To UB or not to UB? 💀
- Correct by confusion
- ++(C++) successor languages
- Borrowing the borrow checker


At least it's not rust!
Rewrite it in a slightly different* C++
*Yet completely incompatible
O'RLY?
cor3ntin

cor3ntin.github.io/posts/safety/

Safety

Rewrite it in... ~~Rust~~ SafeC++? 🤦

The Register®

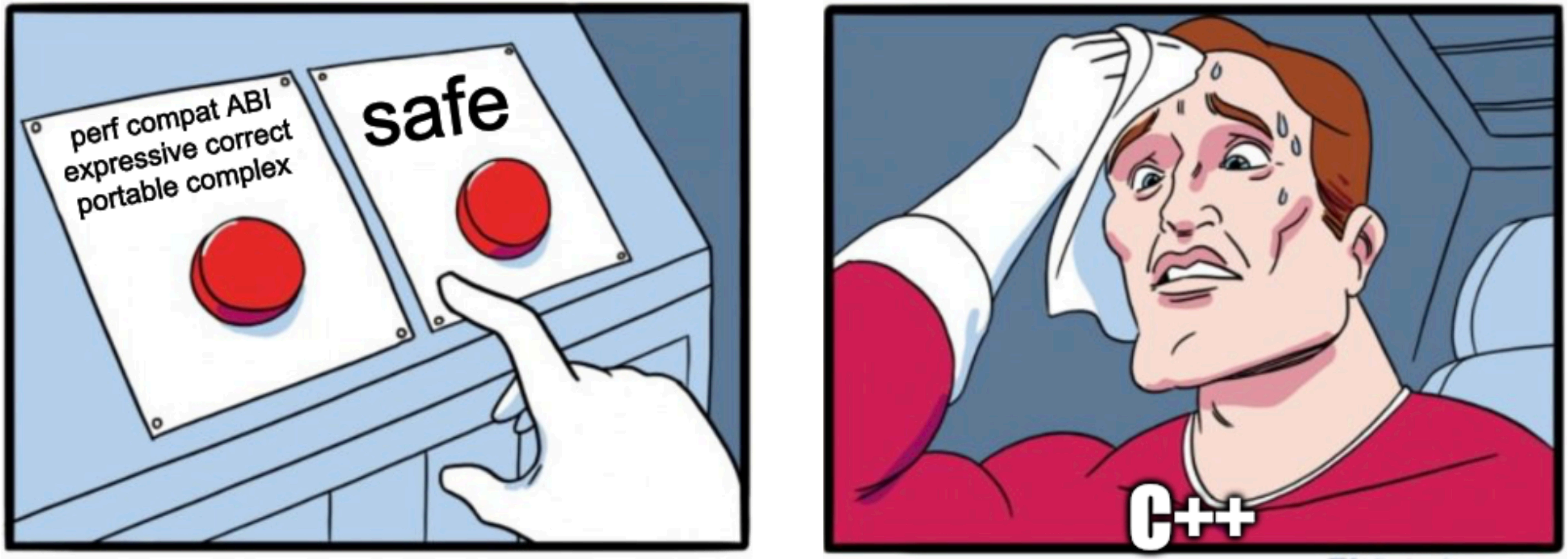
The empire of C++ strikes back with Safe C++

theregister.com/2024/09/16/safe_c_plusplus

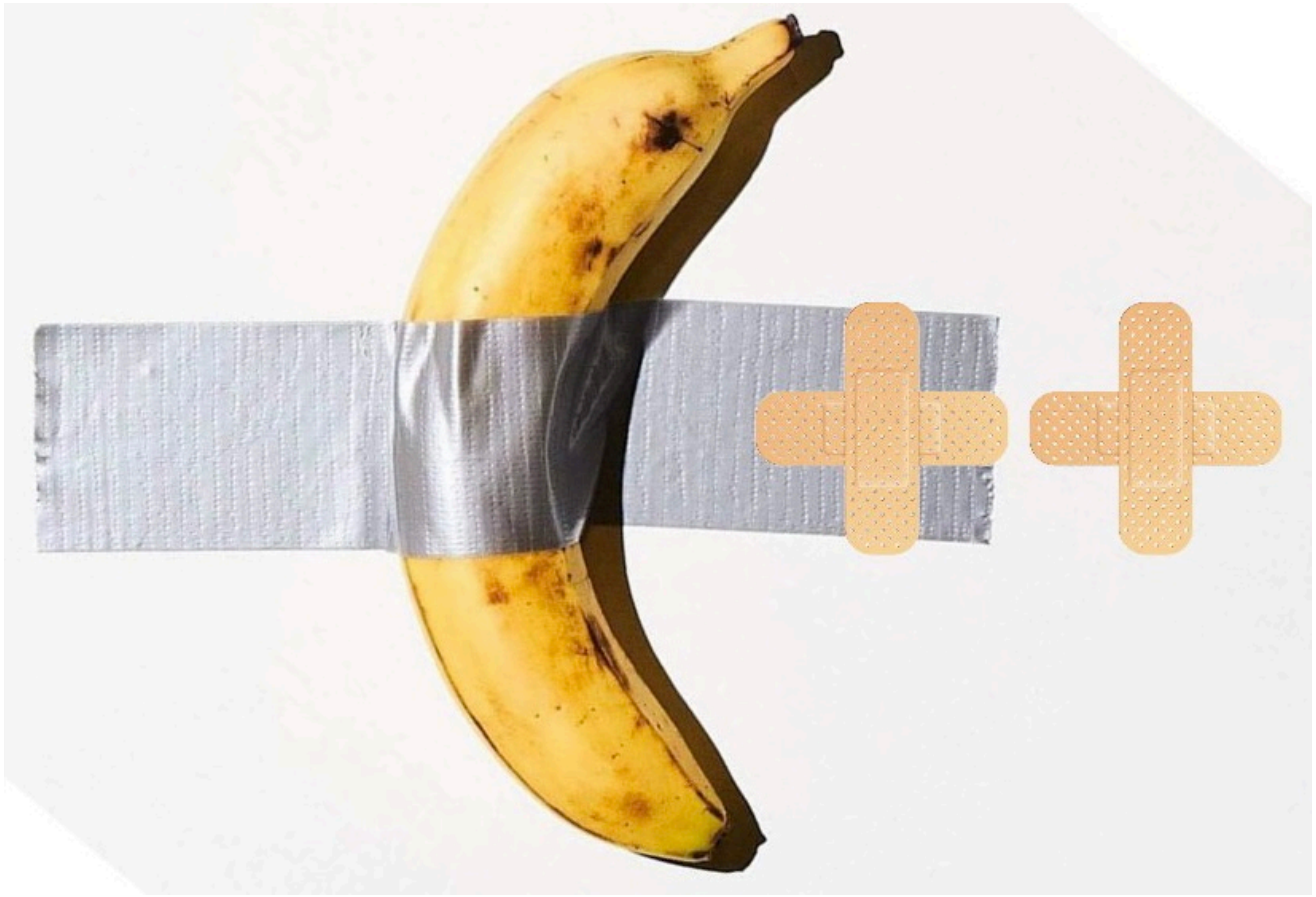
~~C++~~ Circle Compiler

safecpp.org/P3390

Choices


perf compat ABI
expressive correct
portable complex
safe
C++

New (C++) is the enemy of the old



twitter.com/tvaneerd/status/1387

Rust ❤️ C++

Not a zero-sum game.

We need to learn to play nice together... for a looong time!

Engineering

Engineering, not programming

"Software engineering is programming integrated over time."

abseil.io/swe-book/ch01

Path... 🦀

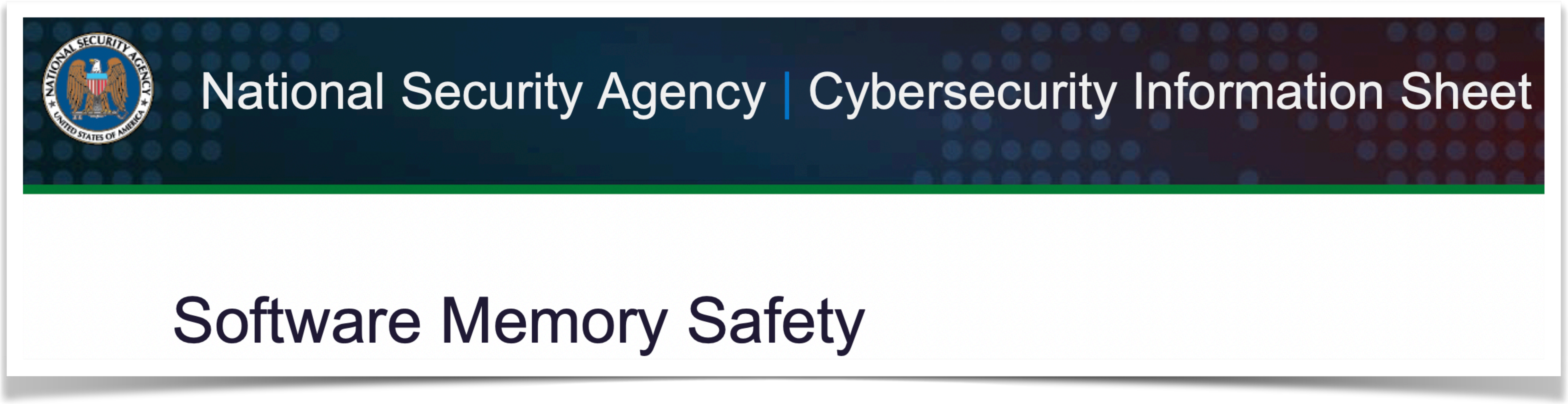
- Why teams want Rust
- Path to Rust
  - Learning
  - Bootstrapping
  - Engineering Systems
  - RiiR
- Interop aka. "not living in a bubble"
- Problems along the way, risk evaluation
- Early wins
- What's next?

But Why?

But Why?

But Why?

Safe C++ 😳


National Security Agency | Cybersecurity Information Sheet

Software Memory Safety

media.defense.gov/2022/Nov/10/2003112742/-1/-1/0/CSI_SOFTWARE_MEMORY_SAFETY.PDF

Microsoft CVEs

Is CVE a Memory Safety Issue (RCE, EOP, Info Disclosure)?
100%
90%
80%
70%
60%
50%
40%
30%
20%
10%
0%
150
161
228
241
433
346
136
318
479
409
481
649
379
220
2016
2017
2018
2019
2020
2021
2022
🟦 Memory safety issue
🟧 Non-memory safety issue

Root cause of memory safety CVEs

100%
90%
80%
70%
60%
50%
40%
30%
20%
10%
0%
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
Heap Corruption
Heap OOB
Stack Corruption
Type Confusion
Uninitialized Use
Use After Free
Other

Year	Heap Corruption	Heap OOB	Stack Corruption	Type Confusion	Uninitialized Use	Use After Free	Other
2006	36	1	32	4	6	12	44
2007	35	1	24	4	4	16	30
2008	43	2	21	13	8	18	44
2009	45	4	22	30	8	22	41
2010	64	8	26	21	11	44	59
2011	30	5	13	14	6	57	103
2012	36	7	4	7	5	39	61
2013	35	13	11	15	6	113	120
2014	31	21	4	24	9	186	19
2015	60	51	3	26	37	188	68
2016	78	83	6	36	20	97	78
2017	110	100	10	77	94	86	78
2018	81	64	9	81	64	103	73
2019	133	109	14	81	42	98	105
2020	150	95	17	41	37	134	333
2021	175	68	7	16	22	65	135
2022	157	87	15	15	19	80	210
2023	175	123	13	12	20	103	135
2024	217	63	17	11	11	78	94

Embracing an Adversarial Mindset for C++ Security - Amanda Rousseau youtube.com/watch?v=glkMbNLogZE

Systems Language Overview

	Rust	C++	C
Object Lifetime	Statically Enforced	Not Enforced, unclear path forward.	No hope
Type Safety	Statically Enforced	Not enforced, unclear path forward.	No hope
Bounds Safety	Enforced at runtime when needed	Could be enforced for STL containers.	No hope
Uninitialized Safety	Statically Enforced	Not enforced, could be enforced w/ breaking change.	Stack could be enforced w/ breaking change.

Bug Classes vs. Mitigations

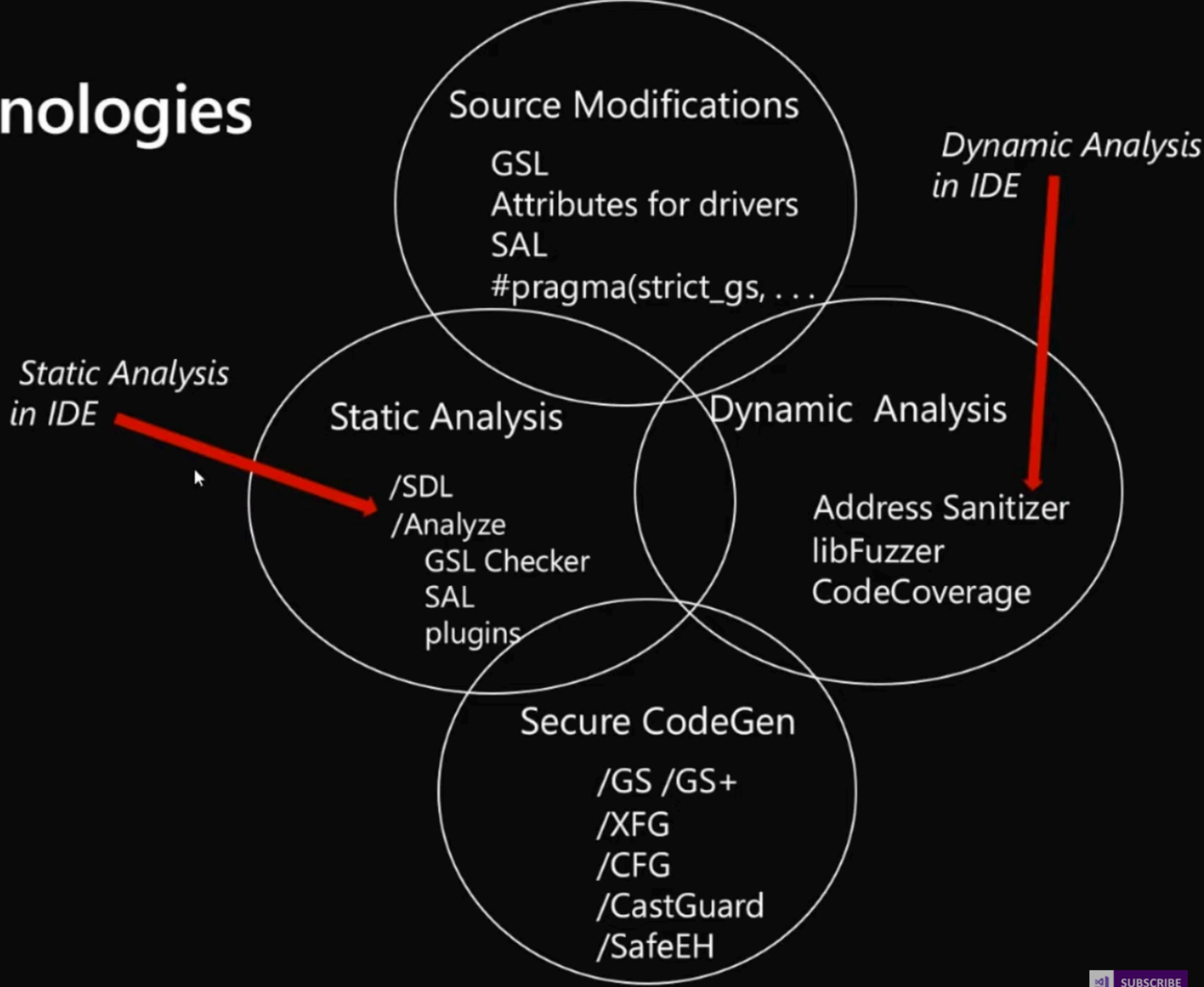
Vulnerability Class	Deterministic or Probabilistic Mitigation	% of Memory Safety Issue CVE's
Heap Linear Overflow	Deterministic	9.5%
Heap Non-Linear Overflow	Probabilistic	12%
Use-After-Free	Probabilistic	26%
Heap Linear Overread	Deterministic	5.3%*
Heap Non-Linear Overread	Probabilistic	14%*
Uninitialized Memory	Deterministic	12%
Type Confusion	Not Mitigated	14%

We can mostly solve these. Uninitialized doesn't require memory tagging.

We can make progress on these (both detection and exploitability).

CastGuard solves some type confusion, others have no apparent solution.

C++ Security Technologies



youtube.com/watch?v=i8_RfDAEjMs

Exploit Mitigation Timeline

SAFESEH

ASLR

Code Integrity (CI)

HEASLR

High Entropy Address Space Layout Randomization

Jit Hardening

CFG

Control Flow Guard

Sandbox

2003 2004 2006 2008 2012 2023 2015

GS Cookie

Return Flow Guard (RFG)

DEP

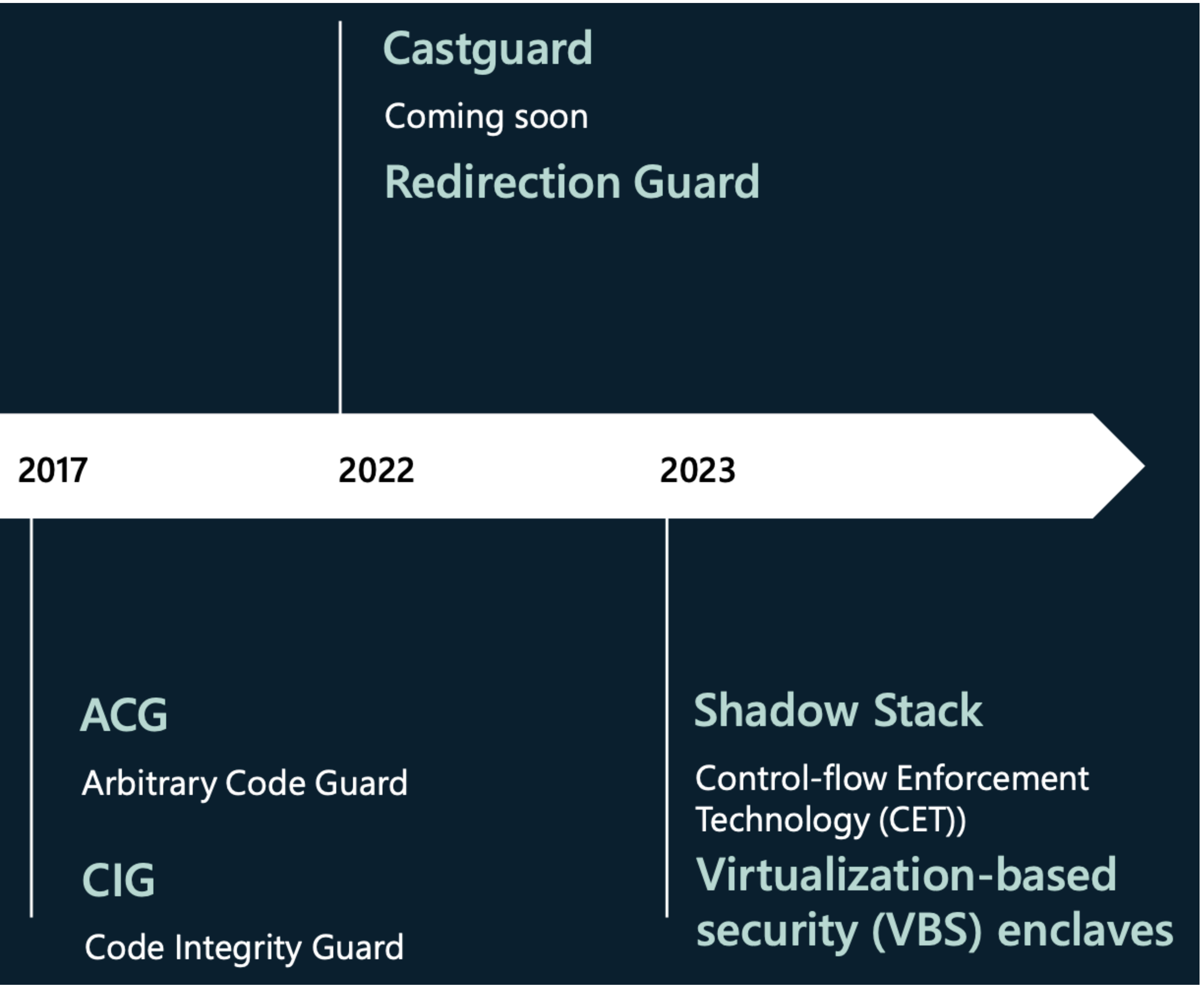
SEHOP

Structured Exception Handler Overwrite Protection

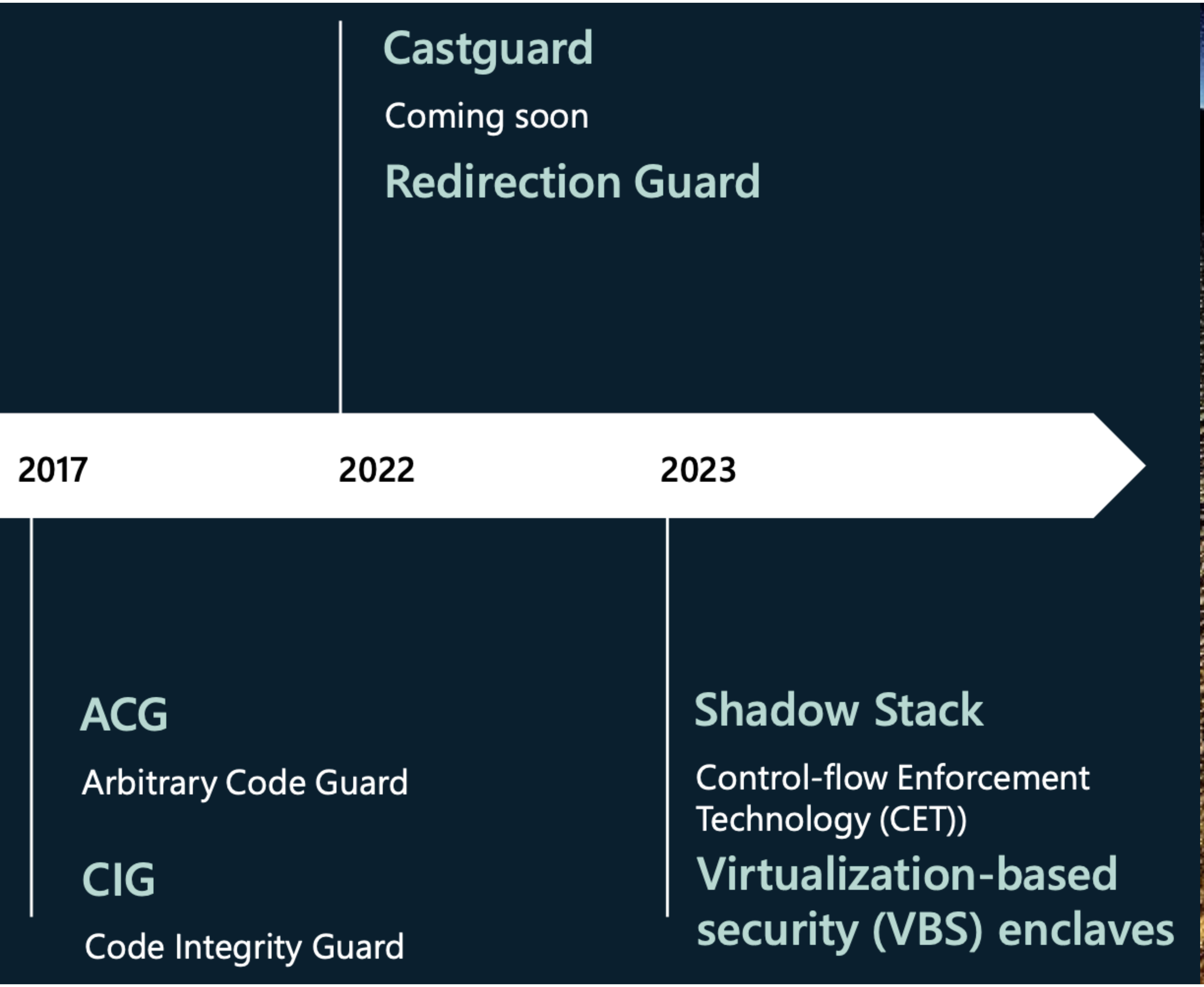
Isolated Heap

Delayed Free

Exploit Mitigation Timeline



Exploit Mitigation Timeline



2017 · 2022 · 2023

2017

ACG

Arbitrary Code Guard

CIG

Code Integrity Guard

2022

Castguard

Coming soon

Redirection Guard

2023

Shadow Stack

Control-flow Enforcement Technology (CET))

Virtualization-based security (VBS) enclaves

Microsoft

Ongoing efforts:

- Making step-changes in our **SDL** operations and making additional investments to meet the evolving needs of cloud and emerging technologies
- Completing our deployment of **CodeQL**, integrated with GitHub Copilot learnings
- Continue to invest in **hardening** C & C++ code
- Standardizing on **Rust** and other memory safe languages (MSLs)
- Contribute 💰 to support the work of the Rust Foundation
- Assist developers making the transition from C, C++, C# to Rust
  - we will continue to invest 💰 in Rust developer tooling

Hardening C & C++ code

Short term:

- tactical efforts to eliminate attack surface
- block exploit techniques
- statically analyze vulnerabilities
- dynamic analysis & fuzzing
- CLFS signing, heap mitigations, ASLR, CFG, UMFD

Hardening C & C++ code

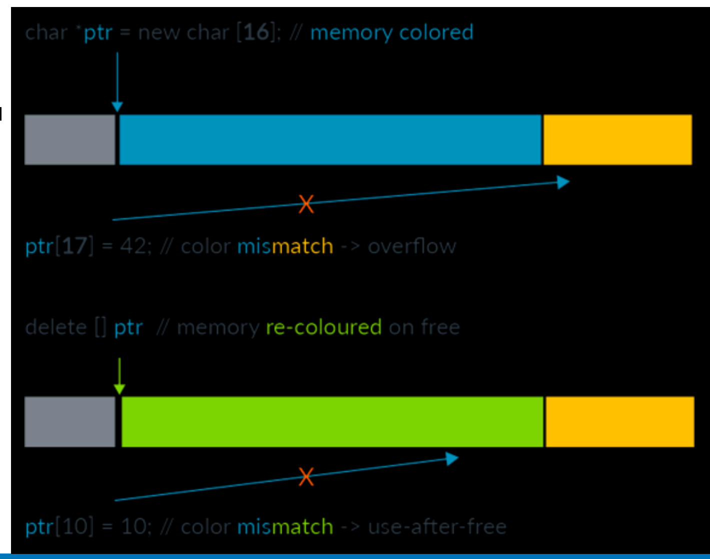
Long term:

- Combination of software and *hardware* mitigations to detect & eliminate the most common memory safety issues classes.
- InitAll / Pool Zeroing – Zero initialize stack variables and kernel pool allocations.
- CastGuard – Prevent illegal stack downcasts (type confusion).
- Memory Tagging – Broad impact to a variety of bug classes, hardware feature.

Hardening C & C++ code

Memory Tagging

- Helps developers catch bugs (eg. hardware ASAN), stops bugs from being exploitable if they ship to customers.
- Non-trivial CPU and memory overhead, but low enough to enable-by-default in production.
- Google will deploy to Android soon; Apple also expected to deploy.
- Microsoft is actively working w/ silicon partners on Memory Tagging designs that are scalable from small devices to large Azure servers.
- Goal: Enable by default for Windows to make a more reliable and secure OS.
- Goal: Support in Azure, for both Windows and Linux



```
char *ptr = new char [16]; // memory colored
ptr[17] = 42; // color mismatch -> overflow
delete [] ptr  // memory re-coloured on free
ptr[10] = 10; // color mismatch -> use-after-free
```

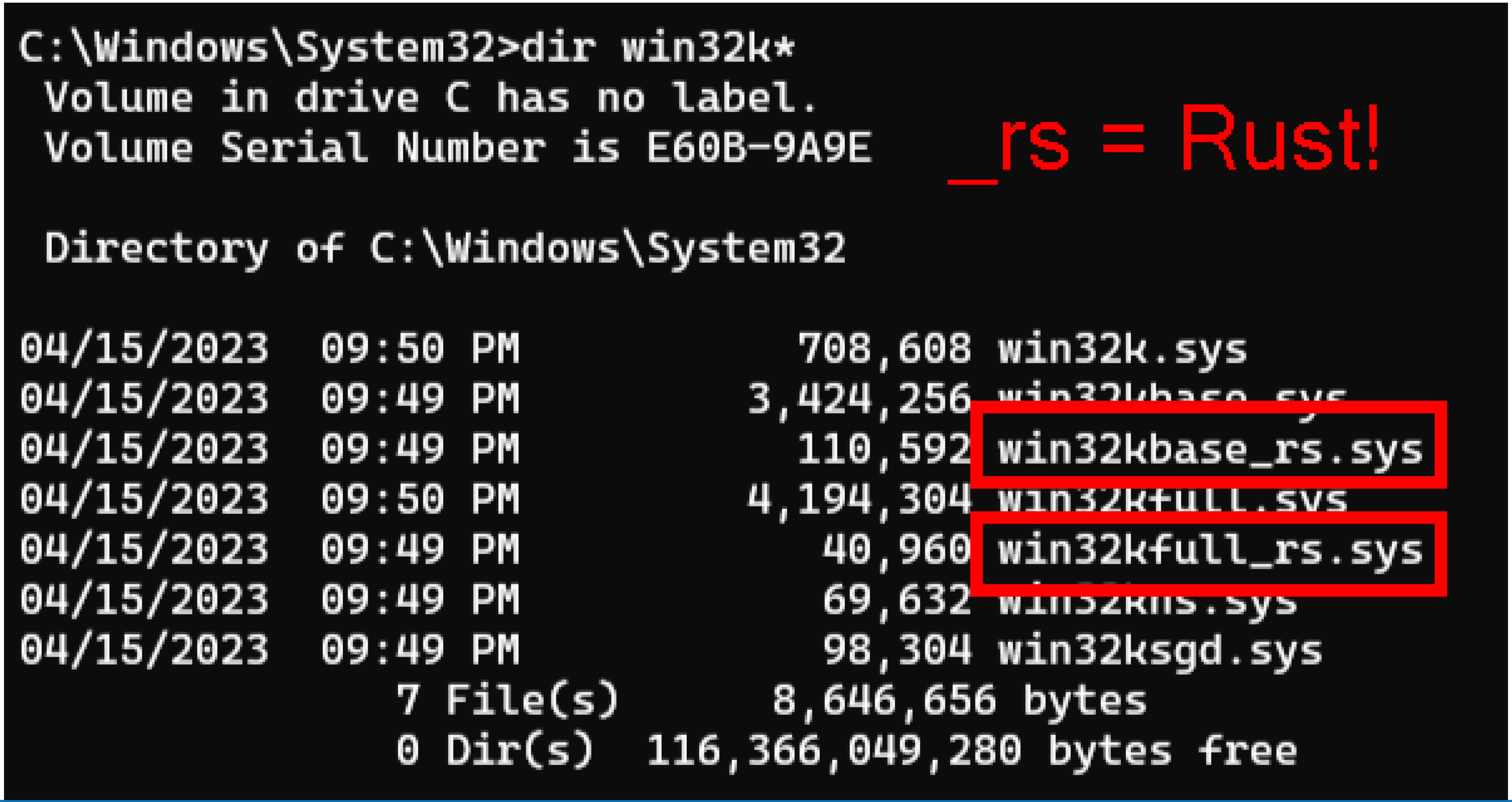
Rust... where?

Is this guy getting to the Rust part already?

😒

Oh, Really?

Rust already in the Windows 11 `kernel` (2023)



```
C:\Windows\System32>dir win32k*
 Volume in drive C has no label.
 Volume Serial Number is E60B-9A9E

 Directory of C:\Windows\System32

04/15/2023  09:50 PM           708,608 win32k.sys
04/15/2023  09:49 PM         3,424,256 win32kbase.sys
04/15/2023  09:49 PM           110,592 win32kbase_rs.sys
04/15/2023  09:50 PM         4,194,304 win32kfull.sys
04/15/2023  09:49 PM            40,960 win32kfull_rs.sys
04/15/2023  09:49 PM            69,632 win32kns.sys
04/15/2023  09:49 PM            98,304 win32ksgd.sys
               7 File(s)      8,646,656 bytes
               0 Dir(s)  116,366,049,280 bytes free
```

Rusty Windows

So this happened 👀 (public announcement - 2023)

Ported some **Windows 11** core components from C++ to Rust

- `DirectWrite`
- `GDI`
- `...` 🤫

youtube.com/watch?v=8T6ClX-y2AE&t=2703s

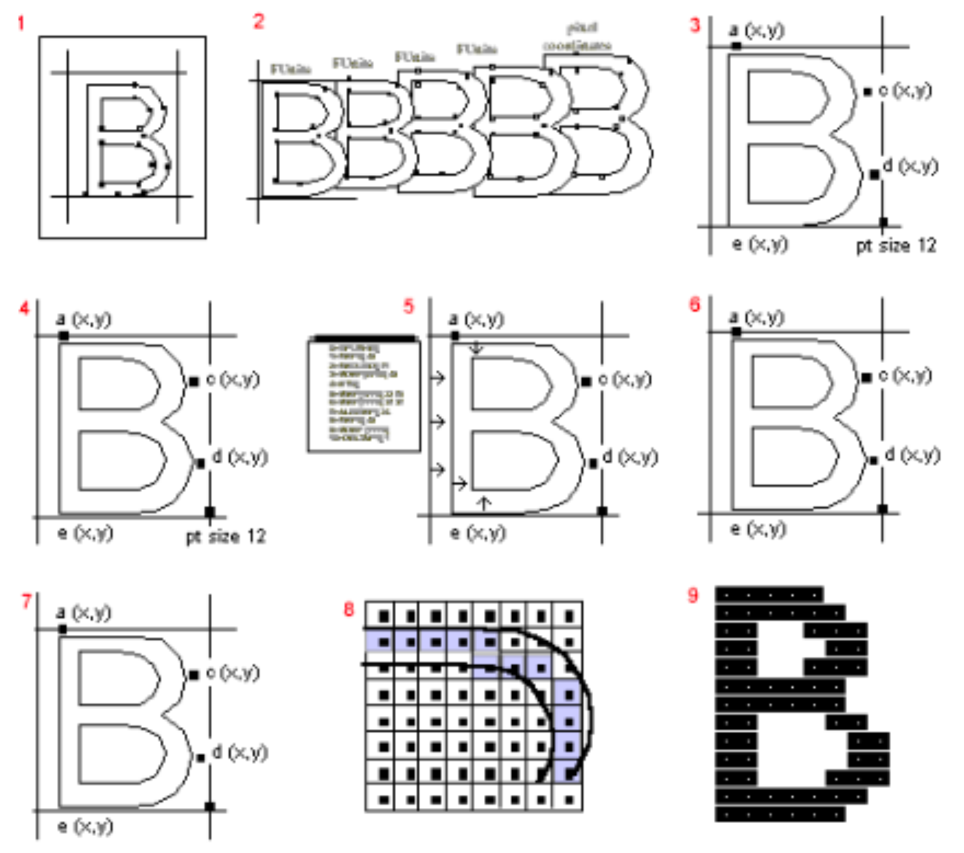
Rust in Windows

Learn by doing: **Exploration ➝ Flighting ➝ Production** (crawl ➝ walk ➝ run)

- Direct impact: Improve security
- Gain experience with transitioning to Rust in production
- Costs of learning Rust?
- Costs of porting to Rust?
- Costs of writing new Rust components?
- Is the full pipeline of Rust tooling ready?
- Dealing with debugging woes
- Performance targets, POGO, etc.
- Costs of maintaining a hybrid **C++/Rust** codebase?

DWrite - DWriteCore

DWrite is a full stack for text analysis, layout & rendering:



- DWrite ships in Windows (dwrite.dll)
- DWriteCore is cross-platform: Windows, Linux, Android, iOS, macOS
- Office depends on DWrite(Core)
- Rust port work began in 2020
- DWriteCore *internally* uses COM-like interfaces:
  - these were a good integration point for C++/Rust, and provided natural boundaries for incremental porting
- DWriteCore *public* APIs are all COM
  - Rust code is directly callable from app code, through COM interfaces

DWrite Internals

Total ported code ~= 152 KLOC (some modules not shown). (Precise counts are complicated, due to test code.)

All code is 100% safe code, except at C++ boundary

Not all parts of DWrite are shown; just those relevant to port

Layout (10 KLOC)

- Line layout, justification
- Text run management: bold, italics, font face, underline, etc.
- Font fallback: Most fonts don't contain all glyphs (e.g. emoji)

Shaping (36 KLOC) + OTLS (18 KLOC)

- Complex script-specific layout: Thai, Indic, Arabic, Hebrew, Hangul, etc.
- Mandatory for complex scripts
- Many are driven by hand-written FSMs
- Complex transformation rules stored in font files (OpenType)
- Transforms sequences of glyphs, e.g. ligatures, connected scripts

Unicode Analysis (6 KLOC)

- Very large property tables
- Defined by Unicode standard

Glyph Data + Glyph Rendering (24 KLOC)

- Computes vector curves, runs bytecode programs (!!) from font files to adjust them
- Rasterizes vector curves to bitmaps
- Provides metrics (advance width, x-height, side bearings)
- Scales bitmaps for high-density scripts (e.g. Chinese)

📆 Porting Time

- TrueType
  - ~ 2 months (1 dev, experienced in Rust) for the core functionality
  - ~ 2 months for exhaustive comparison testing and regression fixing
- Shaping + OTLS
  - ~ 2 months
  - ~ 1 month for comparison testing and regression fixing
  - ~ 2 weeks for performance improvements
- Layout
  - ~ 1.5 months
  - ~ 2 weeks for testing / regression fixing
- Unicode analysis
  - ~ 2 weeks
  - Low rate of regressions; very data-oriented

Win32k GDI - Rust

Ported the GDI REGION components:

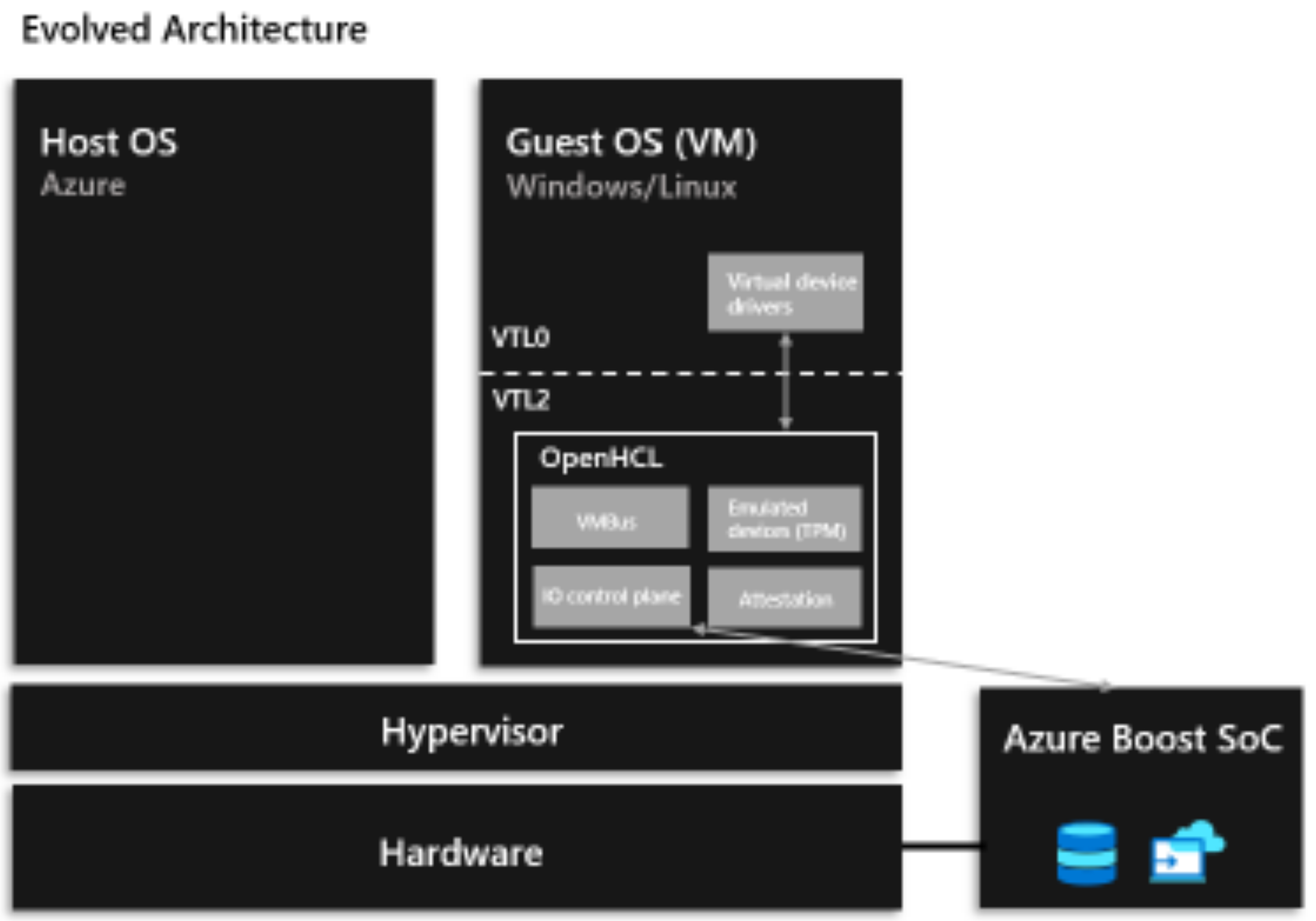
- Models overlapping controls (e.g., windows) in GDI
- "Leaf node" data type: few dependencies, many dependents
- Old (late 80s, early 90) and perf critical (designed for i286/i386)
- Maintenance nightmare: open-coded vector resizing and ref-counting
- Windows boots with the Rust version, and all GDI tests pass
- In flight testing, to prove viability

Win32k GDI - Rust

- Performance of ported code is excellent
  - Office tests, micro-benchmarks
- This work has driven some contributions to upstream Rust project
- Lots of calls to extern C/C++ functions => still a lot of unsafe code
- Unsafe area is reducing as we port more and more code to Rust

OpenHCL - OpenVMM

- OpenHCL is a para-virtualization layer built from the ground-up in Rust
- Azure Boost is an accelerator system that offloads server virtualization processes traditionally performed by the hypervisor and host OS onto purpose-built software and hardware.
- OpenHCL
  - a privileged guest compatibility layer
  - in Azure Boost
  - in Trusted Launch VMs
  - in Azure confidential VMs



Evolving Azure's virtualization model:
techcommunity.microsoft.com/t5/windows-os-platform-blog/openhcl-evolving-azure-s-virtualization-model

Oxidation

More oxidation 🦀 efforts in progress...

C/C++ ➡️ Rust ⬅️ C#

🤐

C++ Interop

“Based on historical vulnerability density statistics, Rust has proactively prevented hundreds of vulnerabilities from impacting the Android ecosystem. This investment aims to expand the adoption of Rust across various components of the platform.”
– Dave Kleidermacher, Google Vice President of Engineering, Android Security & Privacy

“While Rust may not be suitable for all product applications, prioritizing seamless interoperability with C++ will accelerate wider community adoption, thereby aligning with the industry goals of improving memory safety.”
– Royal Hansen, Google Vice President of Safety & Security

foundation.rust-lang.org/news/google-contributes-1m-to-rust-foundation-to-support-c-rust-interop-initiative/

Rust and C++ interoperability

It’s important for Rust to be able to call C++ functions (and vice-versa) in a way that meets the following criteria:

Choose… some?

- No need for excessive unsafe keyword
- No perf overhead (avoid marshaling costs, eg. copying strings)
- No boilerplate or re-declarations / No C++ annotations
- Broad type support - with safety
- Ergonomics - with safety
- Works with dynamic libraries (including the weirdness* of Windows DLLs, CRT)
- Plays well with C++ ABI
- Easily automated
- Hybrid build systems (CMake, cargo, ...)

Rust and C++ interoperability

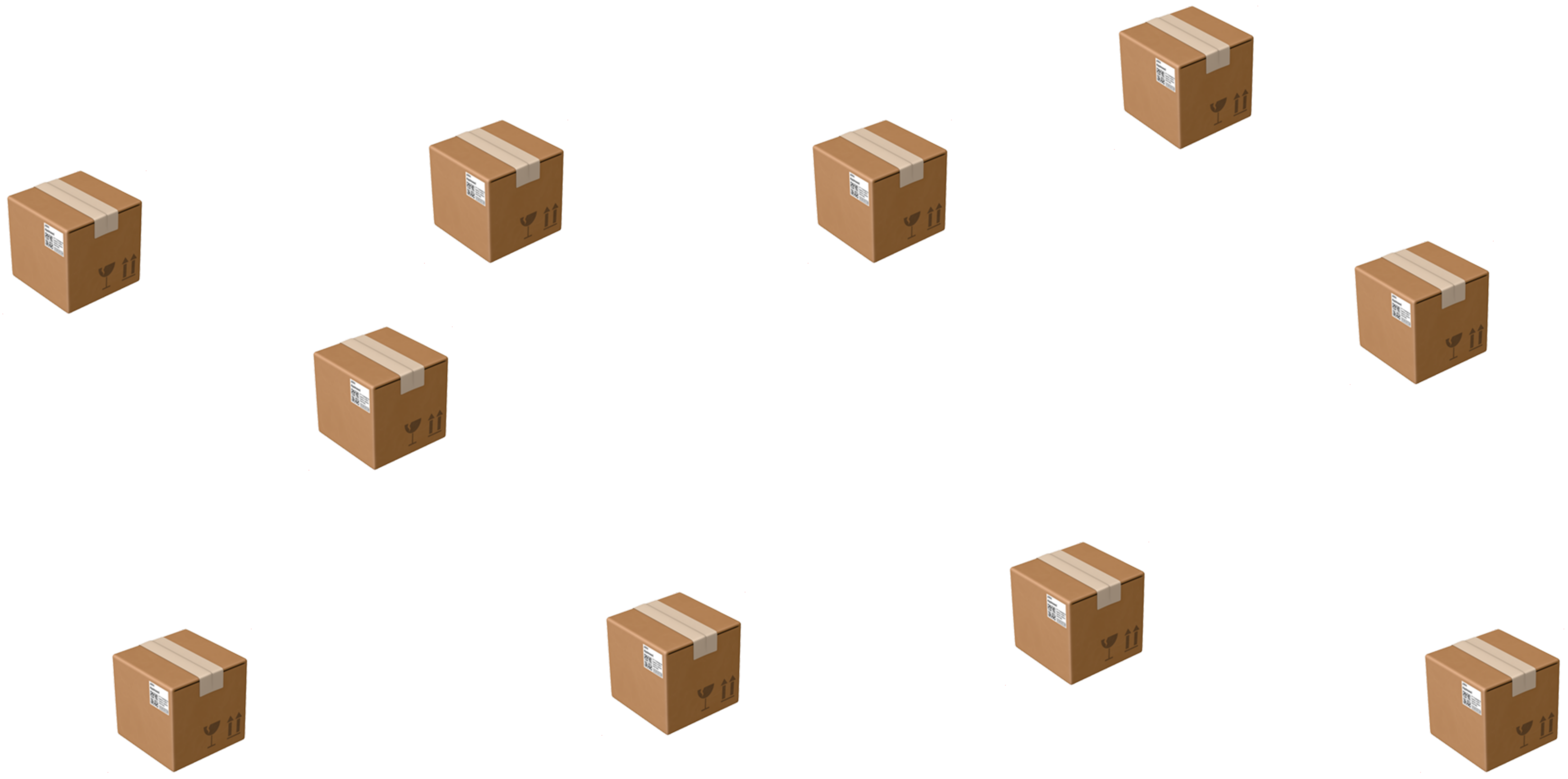
There's (some) progress in Rust community in solving some of these problems.

⚙️ cxx, autocxx, bindgen, cbindgen, diplomat, crubit, etc.

But way more work is needed here!

🚧 💰

Internal Crate Registry



Crates Audit

`cargo vet` ➡️ `audits.toml`

Tool to help projects ensure that third-party Rust dependencies have been audited by a trusted entity.

Downsides:

- review format is too sparse
- just marking a crate version as safe-to-run or safe-to-deploy
- add optional notes (but no more details)
- no ability to track internal crate usage, relative to reviews

mozilla.github.io/cargo-vet/

googleblog.com/2023/05/open-sourcing-our-rust-crate-audits

Rust Crate Review System

A system that records guidance from Microsoft developers on using Rust crates, both public and internal ones.

Questions this system helps answer:

- What Rust crates should my project use, or not use?
- How should I evaluate public Rust crates I'm considering using, and record the evaluation?
- What are the preferred crates for particular purposes?
- How to keep a rigorous SBOM posture for the project?

Rust Crate Review System

Rust Crate Review System

- External crate dependencies, come with the inherent risk, in the form of potential security issues, stability issues, and support and maintenance related issues.

Rust Crate Review System

- External crate dependencies, come with the inherent risk, in the form of potential security issues, stability issues, and support and maintenance related issues.

Rust Crate Review System

- External crate dependencies, come with the inherent risk, in the form of potential security issues, stability issues, and support and maintenance related issues.
- Proactively ensure that MSFT Rust ecosystem is built on the stable and thriving part of the OSS Rust ecosystem, lowering the risk of being affected eg. by a vulnerability reported on a hobby-project crate, with a single owner who is not maintaining it anymore.

Rust Crate Review System

- External crate dependencies, come with the inherent risk, in the form of potential security issues, stability issues, and support and maintenance related issues.

- Proactively ensure that MSFT Rust ecosystem is built on the stable and thriving part of the OSS Rust ecosystem, lowering the risk of being affected eg. by a vulnerability reported on a hobby-project crate, with a single owner who is not maintaining it anymore.

Rust Crate Review System

- External crate dependencies, come with the inherent risk, in the form of potential security issues, stability issues, and support and maintenance related issues.

- Proactively ensure that MSFT Rust ecosystem is built on the stable and thriving part of the OSS Rust ecosystem, lowering the risk of being affected eg. by a vulnerability reported on a hobby-project crate, with a single owner who is not maintaining it anymore.

- A set of unbiased Rust crate evaluation criteria, used for assessment of adoptability of third-party crates by any internal Rust project, lowering the company's vulnerability naturally introduced by depending on third-party OSS solutions.

Rust Crate Review System

- External crate dependencies, come with the inherent risk, in the form of potential security issues, stability issues, and support and maintenance related issues.
- Proactively ensure that MSFT Rust ecosystem is built on the stable and thriving part of the OSS Rust ecosystem, lowering the risk of being affected eg. by a vulnerability reported on a hobby-project crate, with a single owner who is not maintaining it anymore.
- A set of unbiased Rust crate evaluation criteria, used for assessment of adoptability of third-party crates by any internal Rust project, lowering the company's vulnerability naturally introduced by depending on third-party OSS solutions.

Rust Crate Review System

- External crate dependencies, come with the inherent risk, in the form of potential security issues, stability issues, and support and maintenance related issues.
- Proactively ensure that MSFT Rust ecosystem is built on the stable and thriving part of the OSS Rust ecosystem, lowering the risk of being affected eg. by a vulnerability reported on a hobby-project crate, with a single owner who is not maintaining it anymore.
- A set of unbiased Rust crate evaluation criteria, used for assessment of adoptability of third-party crates by any internal Rust project, lowering the company's vulnerability naturally introduced by depending on third-party OSS solutions.
- A unified, unbiased, highly automatable crate scoring system used throughout all orgs at Microsoft.

ONE DOES NOT SIMPLY
REWRITE IN RUST
makeameme.org

But we still do it!

Unleashing 🦀 The Ferris Within

EuroRust

October 2024

🐦 @ciura_victor
🐘 @ciura_victor@hachyderm.io
🦋 @ciuravictor.bsky.social

Victor Ciura
Principal Engineer
Rust Tooling @ Microsoft