Rust in regulated industries
The Renault/Ampere case.

MAY 2024

FRÉDÉRIC AMEYE
RENAULT GROUP - AMPERE SOFTWARE TECHNOLOGY

From Pwn2Own Vancouver 2022 - Team Synacktiv vs Tesla Model 3

Zero Day Initiative
9,71 k abonnés
S'abonner
82
Partager
Enregistrer

Rust could have prevented this.

Not joking. 70% of most severe security vulnerabilities could be prevented.

kudos to Tesla and Synacktiv, they do awesome engineering!

Many industries are more and more driven by software (or even relying their entire business on it).

Too bad, more software means more bugs and vulnerabilities.

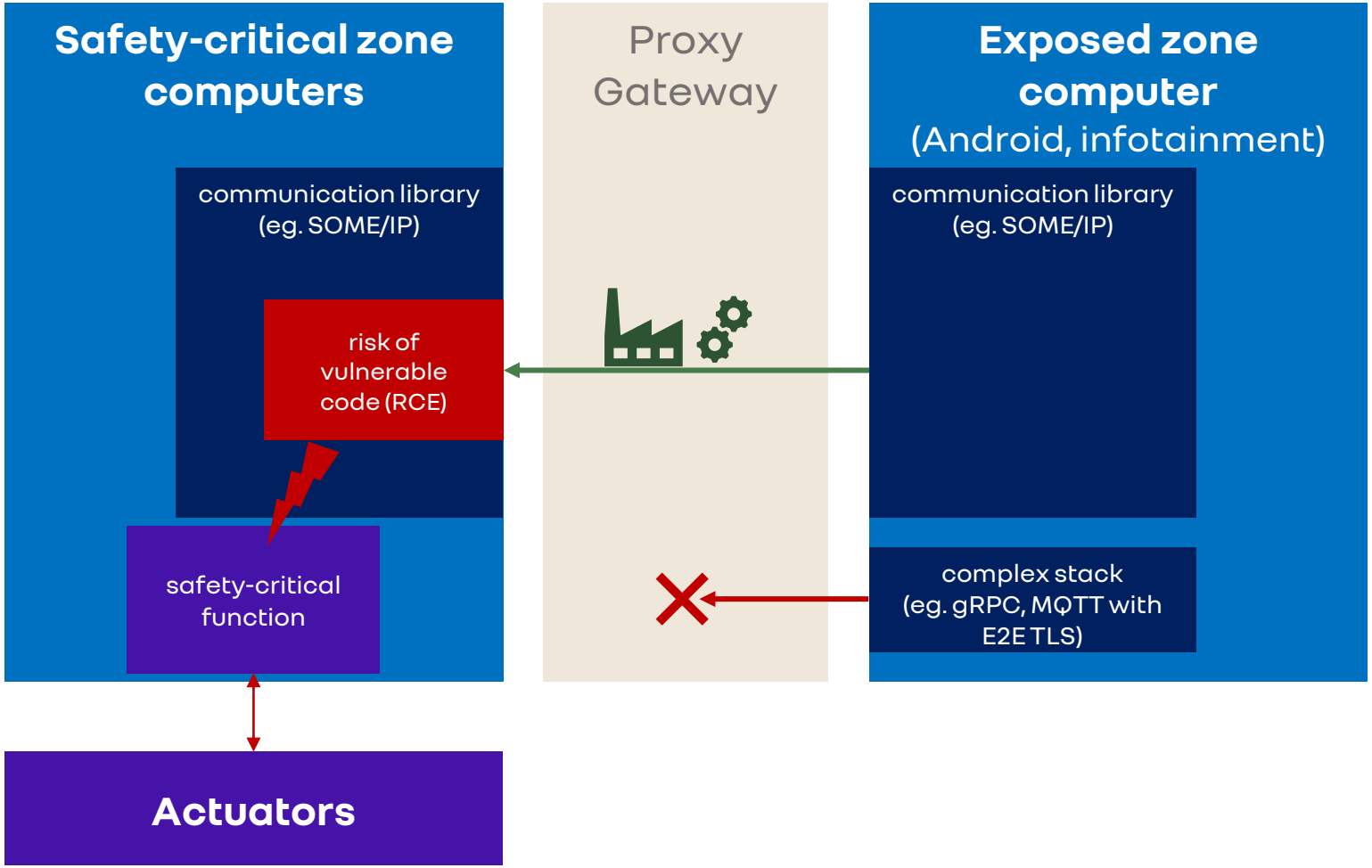
Rust quality & reliability properties seem like a great idea for carmakers.

But saying « it's better » is not enough to trigger changes in large corporations, isn't it?

Cybersecurity measures in cars strictly limits what we can do in the car architecture, and make them more expensive and complex.

Example on how Rust help us
save dozen millions€.

Fear of vulnerabilities make the cars more complex and expensive

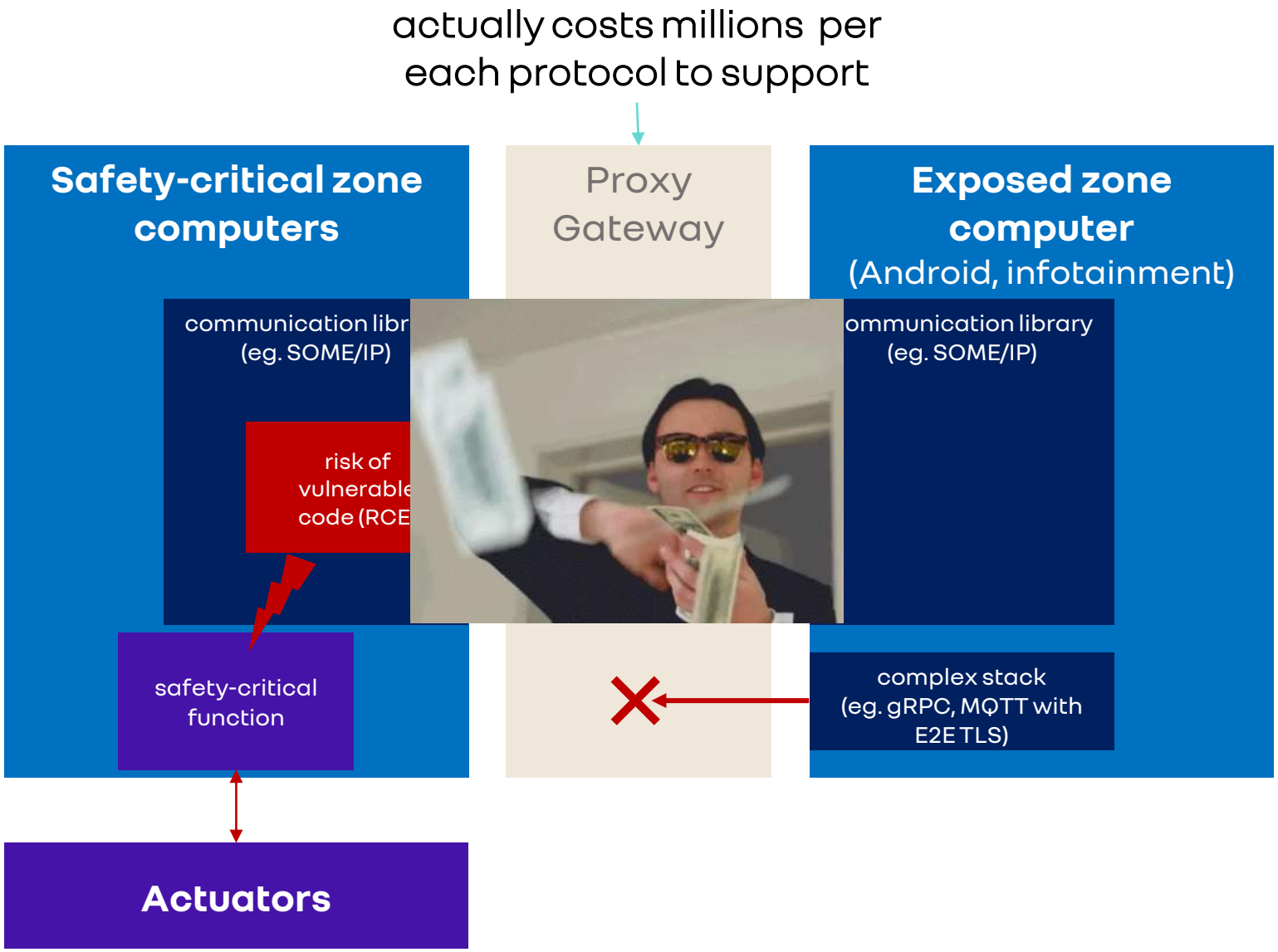


Rules before 2023:

simple protocols
→ allowed through full proxy
(packet inspection & reconstruction of all OSI stacks, payloads checking, Access Control Lists to OSI L7)

complex protocols
→ forbidden

Fear of vulnerabilities make the cars more complex and expensive



Rules before 2023:

simple protocols

→ allowed through full proxy

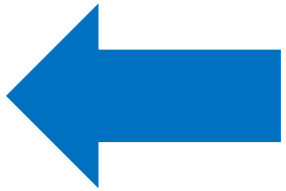
(packet inspection & reconstruction of all OSI stacks, payloads checking, Access Control Lists to OSI L7)

complex protocols

→ forbidden

One example of system impact:
fear of vulnerabilities make the cars more complex and expensive

- **low flexibility**
  (developer frustration)
- **cost & delay impact**
  of designing a proxy
- **performance impact**
  wasting MIPS and Watts
- sometimes **break E2E security properties** (proxy acts as SPOF/MiTM)
- vulnerabilities are still possible!

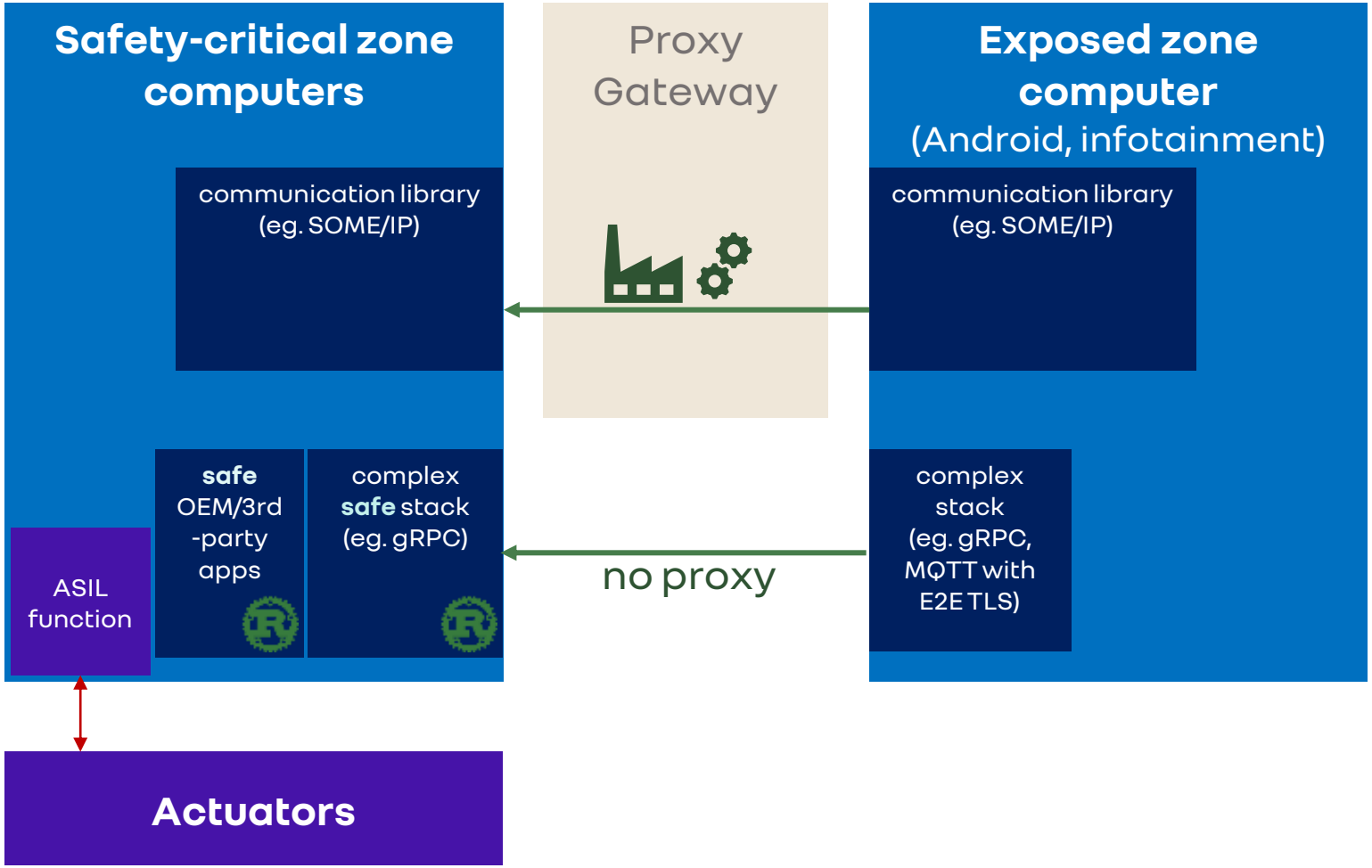


Before 2023:

simple protocols
→ allowed through full proxy
(packet inspection & reconstruction of all OSI stacks, payloads checking, Access Control Lists to OSI L7)

complex protocols
→ forbidden

Simplifying architectures through memory-safety guarantees



Renault SDV rules (MY2025):

unsafe protocols

→ allowed through full proxy

(packet inspection & reconstruction of all OSI stacks, payloads checking, Access Control Lists to OSI L7)

memory-safe protocols

→ allowed without proxy if:

- follows safe Rust coding rules
- protocol ensures E2E OSI L7 authentication & anti-replay
- apps using the protocol are also memory-safe

Our strategy:

If you use Rust,

you're allowed to
do more things than before,
at lower cost,
with more powerful tools,
and reduce technical debt.

We're not the only ones \o/

« [Google] Chrome was able to **move its QR code generator out of a sandbox** by adopting a **new memory-safe library** written in Rust [...] »

Consequence:

We made Rust mandatory for some SW.

Then, it naturally propagates at interfaces given its powerful capabilities & ecosystem.

Developers enjoy it,
time to market & quality improves,
control increases over the SW (sBOM)...

Better up-skilling?

A **good C/C++ developer** needs 10+ years of experience and will still make memory-management mistakes.

A **newbie Rust developer** will be catched by the compiler before making the same mistakes.
Training towards Rust is also working really great.

Do we use it?

- At Ampere, we think **distributed security**,
  by improving software development **guarantees**,
  is the way forward.

- We use Rust as part of our development processes, and **already use it in vehicles** that will be shipped 2026+.
  Proofs of Concepts ongoing for safety-critical Rust components.

- Rust is a **strong requirement in the AAOS-SDV OS** we're developing with Google (not only the infotainment part!).

- On-going strategy to have **Rust "by-default" for new developments**.

Rust is **not only about security**,
but also **improves quality** by significant factors,
in areas where **performance or reliability** are critical.

Go, Javascript, Kotlin, C++, asm, Java,.. still make sense for other areas!

And of course, Rust makes 100% sense when
dealing with untrusted data or exposed services.

Experiencing a switch towards Rust

AMPERE RG

- Finding a **« killer » area** where Rust could clearly be a **disruptive** change helps convincing large inertial entities. Saying « it's better » is not sufficient.
- Covering the « business security-critical » use-cases first, and closely controlling the boundaries towards non-Rust. **Rewrite everything is not the strategy!** We live with legacy.
- We observed **up to 20% faster time-to-market** in our internal developments (latency-critical networking code), and overall cost reduction throughout the dev cycle.
- Rust will **naturally propagate**, thanks to the areas where it's mandatory, and the increase in incentives ('Rust-by-default').

The Big Picture of where Rust helps

SW quality & performance	type model of Rust, compiler properties, as-per C++ performance	Better overall quality at same performance level	**better organization efficiency**
Time to Market / Efficiency	Tooling, ecosystem, less bugs at compile-time	Improved time-to-market	
Security vulnerabilities	Rust compiler properties, cargo tooling	50-70% less vulnerabilities worst ones nearly eliminated	**improved market penetration**
Supply chain security	Better control over supply chain (Cargo tools)	Better viewer over SBOM	**improved career opportunities**
Regulations Certification	Rust language properties	Reduces cost of Safety certifications / Security audits / formal proofing	
Engineers satisfaction	Modern tools on-par with the developer ecosystem	Better engineer satisfaction, less tooling friction, leading-edge technologies available	**decreased operating expenses**
Technical debt	Crates ecosystem, cargo tooling	Helps reduce technical debt	

- Rust is the **low-hanging fruit for cheaper and secure** automotive solutions, which improves time-to-market and overall satisfaction.
- **Ampere is demonstrating its effectiveness** and our roadmaps will spread it more and more ('Rust by default').
- The **regulated industries needs to embrace this opportunity**, and Ampere will continue to encourage its partners.
- Rust does not solve everything! Security certifications, sandboxing, defense-in-depth, great cryptography, logical bugs testing, security-by-design, KISS are complementary.

AMPERE

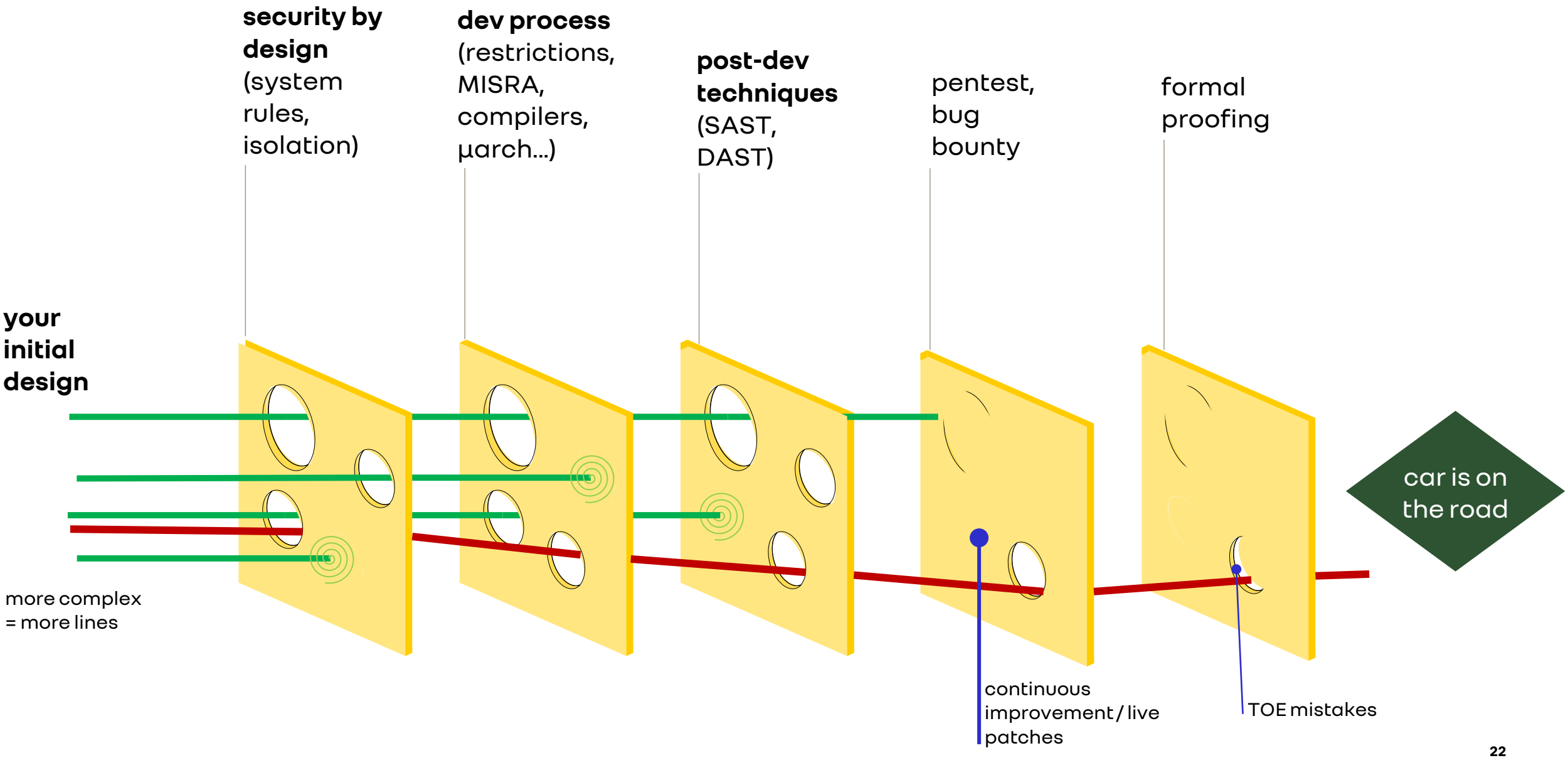
@fredericameye

Thank you

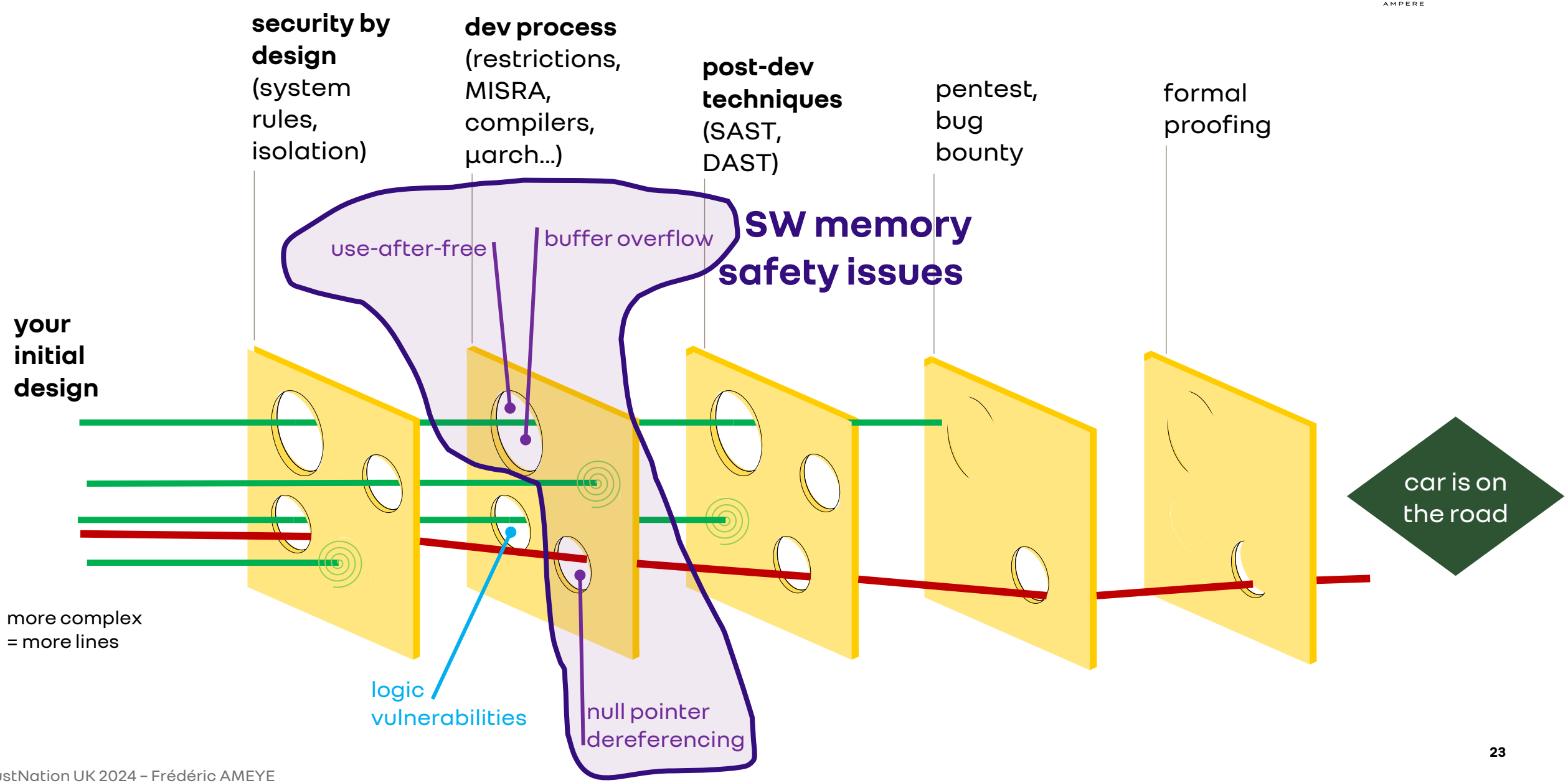
frederic.ameye@renault.com

AMPERE.CARS

Focus on security



Focus on security



Focus on security

security by design (system rules, isolation)

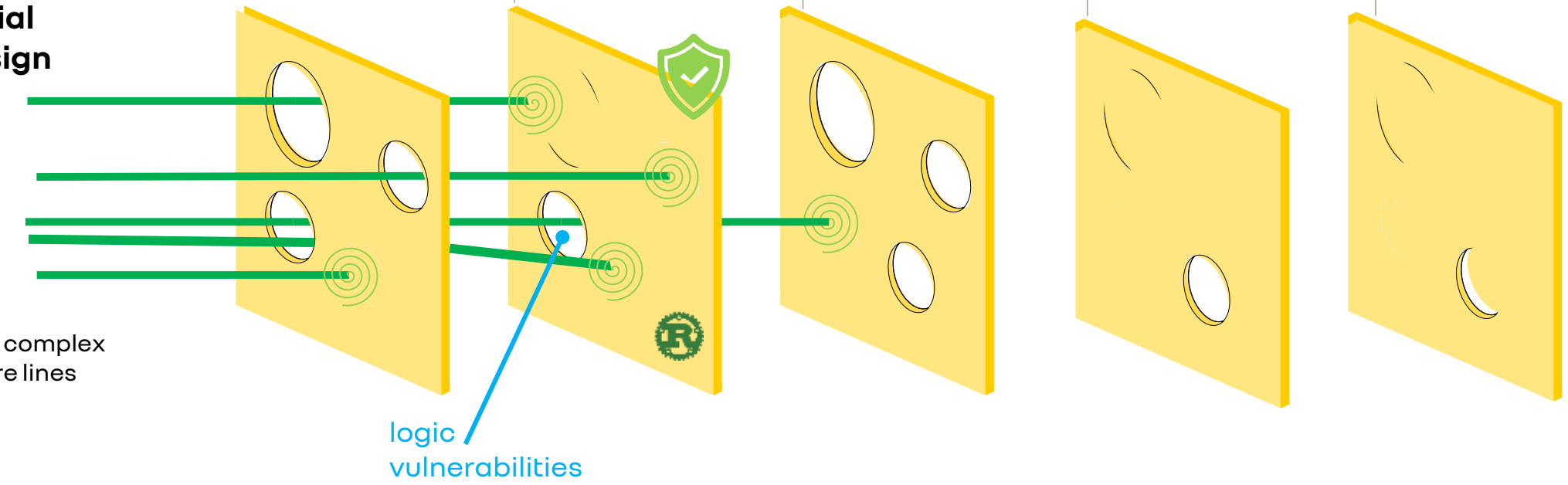
dev process **with Rust using safe guidelines**

post-dev techniques (SAST, DAST)

pentest, bug bounty

formal proofing

your initial design



more complex
= more lines

logic vulnerabilities

car is on the road

What's next?

- Defining Rust **supply chain attacks mitigations**
  - This is a trending topic at the Rust Foundation
  - Ampere is also defining a few rules

- Starting using **Rust for safety-critical components** (ADAS)
  - Less for cyber properties, but for better code behavior
  - Better defining formal guarantees on memory safety
  - How to link to 'what we always did'? Eg. MISRA rules...

- Pushing **memory-safety towards suppliers**:
  - hypervisors (& mostly their paravirtualized drivers)
  - low-level SoC firmware vendors (accelerators, BootROMs)
  - exposed components (eg. FOTA, crypto parsing, embedded HSMs, switches FW)