

V O L V O

WHY AUTOMOTIVE SOFTWARE NEEDS MORE RUST

LTH | 3 October 2023

Julius Gustavsson, Volvo Cars | julius.gustavsson@volvocars.com

V O L V O

In a world where vehicles are
defined by software, the old tools
just won't cut it.

V O L V O

whoami

Julius Gustavsson

System Architect and Team Lead @ Volvo Cars - CSP

~ 20 Years of software development

- Worked in
 - Avionics
 - Telecom
 - Consumer Electronics
 - Automotive

Disclaimer:

These are mostly my personal thoughts and opinions and should not be seen as an official Volvo Cars position

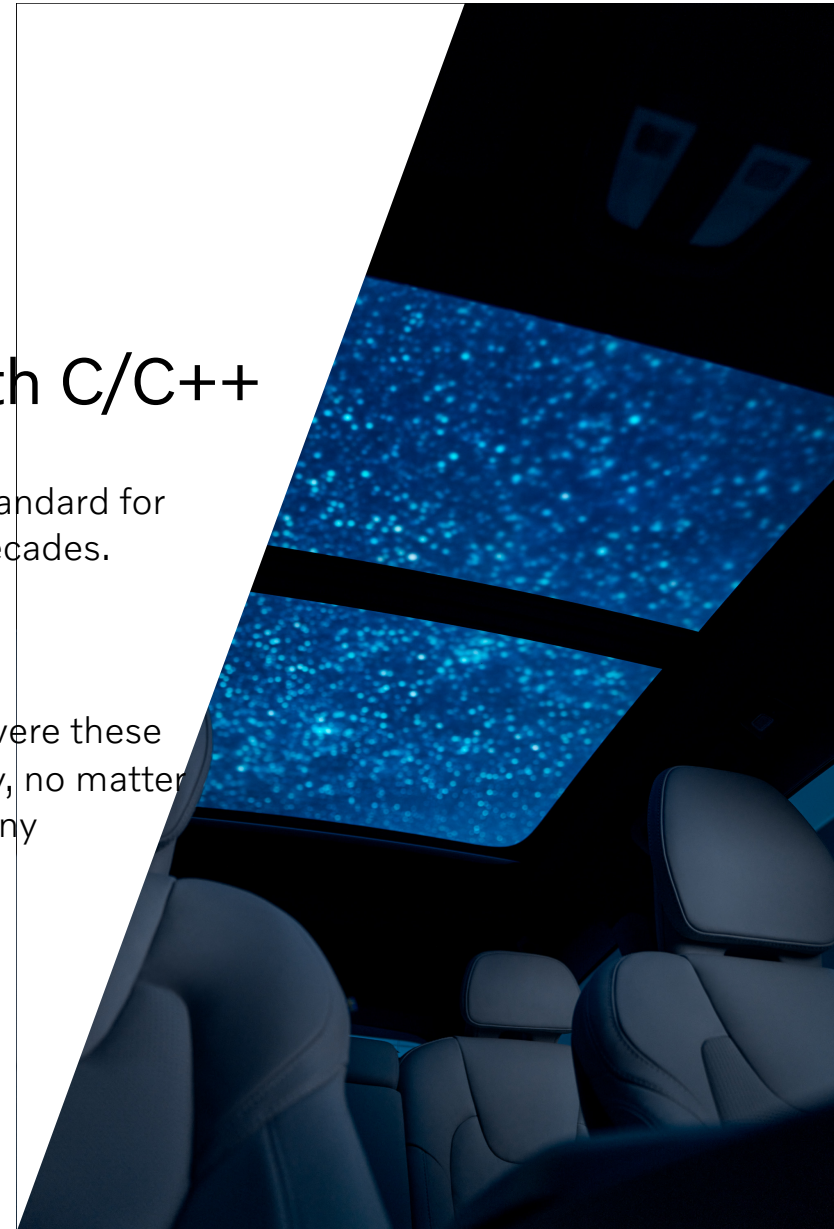
V O L V O

Systems Programming with C/C++

C/C++ is without a doubt the de facto standard for systems programming, going back decades.

But...

In every project I have worked on there were these obscure errors that never seem to go away, no matter the level of carefulness and scrutiny



”With undefined behavior, anything is possible!”

- Raph Lavien

V O L V O

Maybe you read it like this?



V O L V O

Or perhaps like this?



V O L V O

Spot the error

```
#include <stdio.h>

int main(int argc, char **argv) {
    (void)argc;
    (void)argv;
    unsigned long a[1];
    a[3] = 0x7ffff7a2e194UL;
    printf("%p\n", (void*)a);
    return 0;
}
```

```
$ gcc -g -Wall -Wpedantic -Wextra -o test_c
test.c
```

```
$ echo $?
```

```
0
```

```
$ ./test_c
0x7fff9cdc9990 test_c: iconv.c:91: iconv:
Assertion `!"Nothing like this should
happen"' failed. [1] 29794 abort (core
dumped) ./test_c
```


V O L V O

Spot the error (2)

```
#include <iostream>
#include <vector>
#include <string>

int main () {
    std::vector<std::string> v;
    v.push_back("Is");
    std::string &tmp = v[0];
    v.push_back("this");
    v.push_back("OK?");
    std::cout << tmp << '\n'; return 0;
}
```

```
$ g++ -g -Wall -Wpedantic -Wextra -o test_cpp
test.cpp
```

```
$ echo $?
0
```

```
./test_cpp [1] 17543 segmentation fault (core
dumped) ./test_cpp
```

Spot the error (3)

```
int process_heartbeat(SSL* ssl) {  
    short int len = 0;  
    char* inbuf = NULL;  
    char* outbuf = NULL;  
  
    len = SSL_read(ssl, &inbuf);  
    if (inbuf == NULL) {  
        return -1;  
    }  
  
    len = inbuf[0] << 8 | inbuf[1];  
    inbuf += 2;  
    outbuf = malloc(len + 2 + 1);  
  
    if (outbuf == NULL) {  
        return -1;  
    }  
  
    outbuf[0] = HEARTBEAT_RESP;  
    outbuf[1] = inbuf[0];  
    outbuf[2] = inbuf[1];  
  
    memcpy(outbuf + 3, inbuf + 2, len);  
    free(inbuf);  
    return SSL_write(ssl, &outbuf, len);  
}
```

This is a highly simplified version of the famous SSL “Heartbleed” vulnerability.

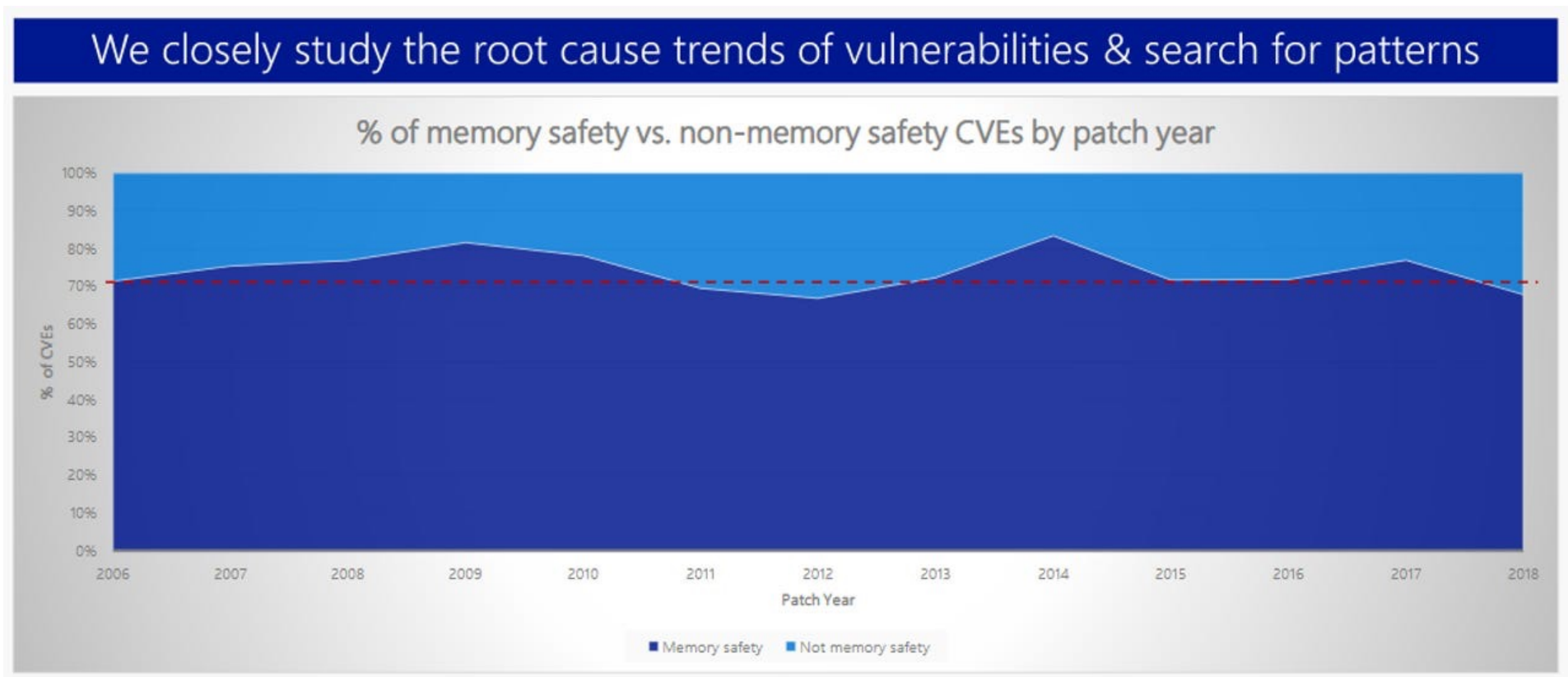
The “Heartbeat Request” message contains a two byte length field followed by that many bytes of payload.

The server then creates a “Heartbeat Response” message by copying payload and sending it back

What happens when the length field lies?

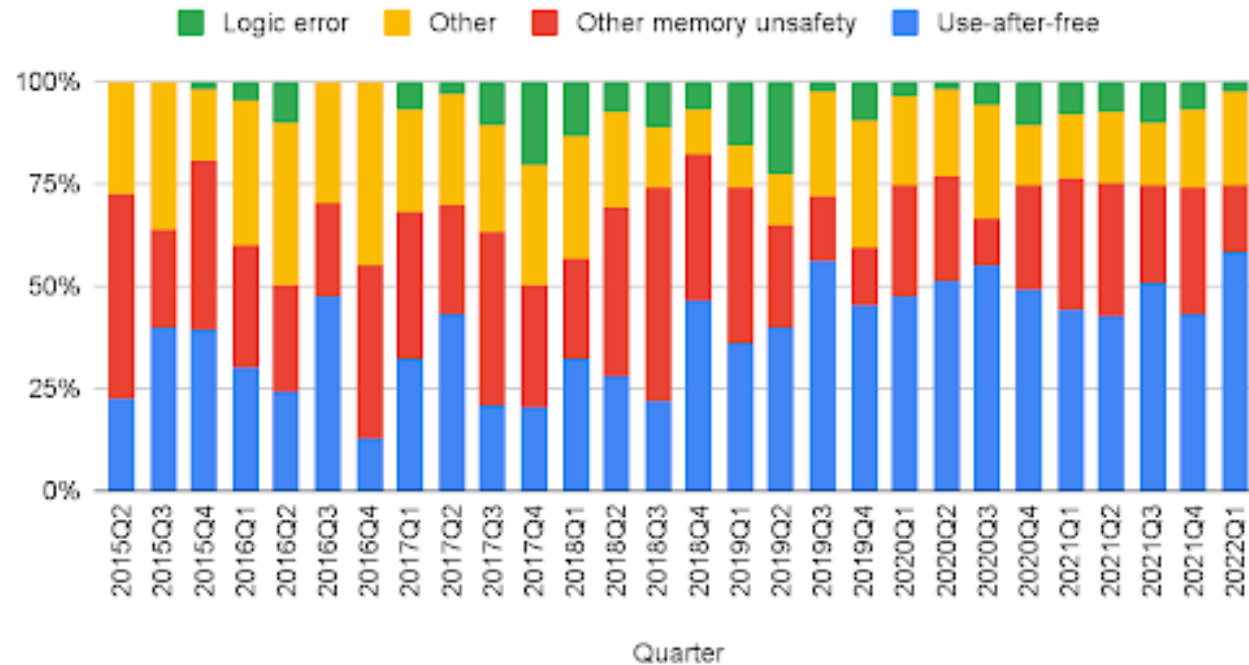
**BUT THIS IS NOT AN ISSUE IF YOU KNOW
WHAT YOU ARE DOING!**

Memory Safety Related CVEs in Microsoft Products



Memory Safety Related CVEs in Google Chrome

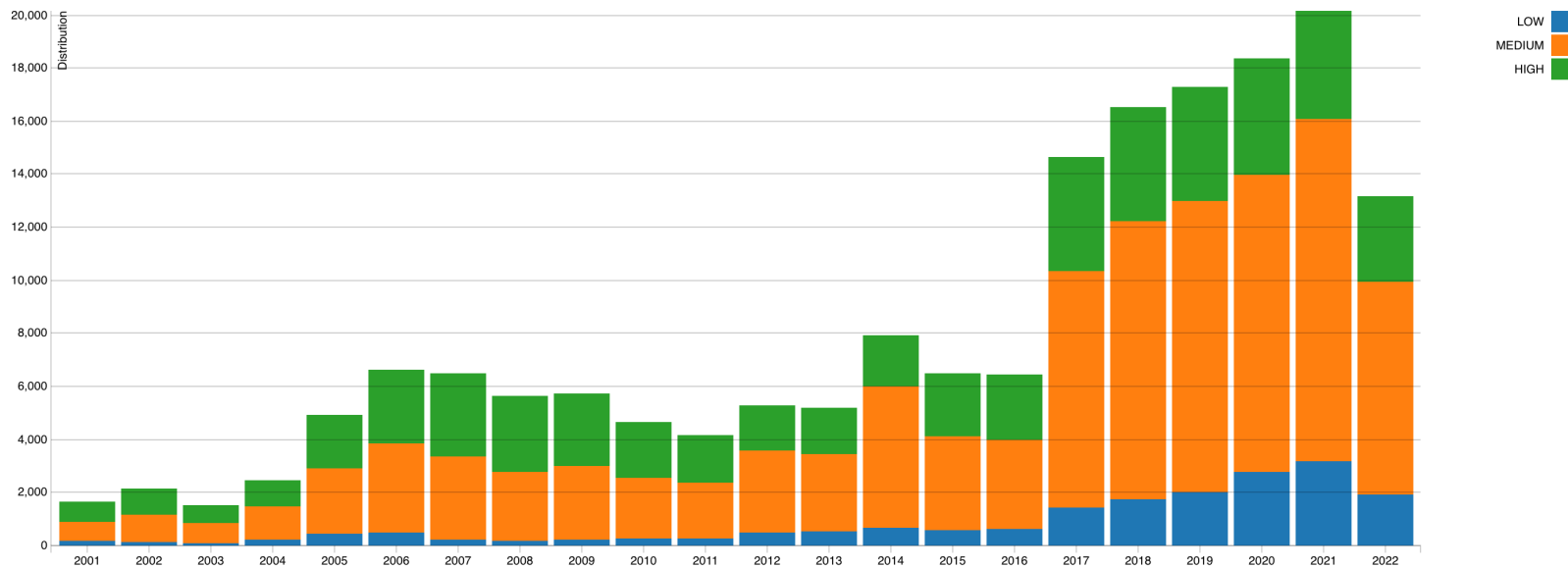
Bug types (high+ severity, impacting stable Chrome users)



And they are increasing over time

CVSS Severity Distribution Over Time

This visualization is a simple graph which shows the distribution of vulnerabilities by severity over time. The choice of LOW, MEDIUM and HIGH is based upon the CVSS V2 Base score. For more information on how this data was constructed please see the [NVD CVSS page](#).



”Memory issues in software comprise a large portion of the exploitable vulnerabilities in existence. NSA advises organizations to consider making a strategic shift from programming languages that provide little or no inherent memory protection, such as C/C++, to a memory safe language when possible.”

- NSA | Cybersecurity Information Sheet

”One key recommendation the US government is now advocating is that software developers ditch memory unsafe programming languages, such as C and C++, because they are the “leading cause of the world’s software vulnerabilities”.”

- The White House Cyber Security Chief | State of Open Source Conference

*”Roughly 60 to 70 percent of browser and kernel vulnerabilities—and security bugs found in C/C++ code bases—are due to memory unsafety, many of which can be solved by using memory-safe languages. While developers using memory-unsafe languages can attempt to avoid all the pitfalls of these languages, **this is a losing battle, as experience has shown that individual expertise is no match for a systemic problem.** Even when organizations put significant effort and resources into detecting, fixing, and mitigating this class of bugs, memory unsafety continues to represent the majority of high-severity security vulnerabilities and stability issues. It is important to work not only on improving detection of memory bugs but to ramp up efforts to prevent them in the first place.”*

- Consumer Reports / Future of Memory Safety

*ISO 26262 (the main automotive functional safety standard) emphasizes the use of **state of the art** with regards to technical solutions to achieve the objective of the standard.*

*ISO 26262 (the main automotive functional safety standard) emphasizes the use of **state of the art** with regards to technical solutions to achieve the objective of the standard.*

Can we still claim that C/C++ is state of the art?

V O L V O

Conclusion

If we want to overcome these issues we need to start considering other tools

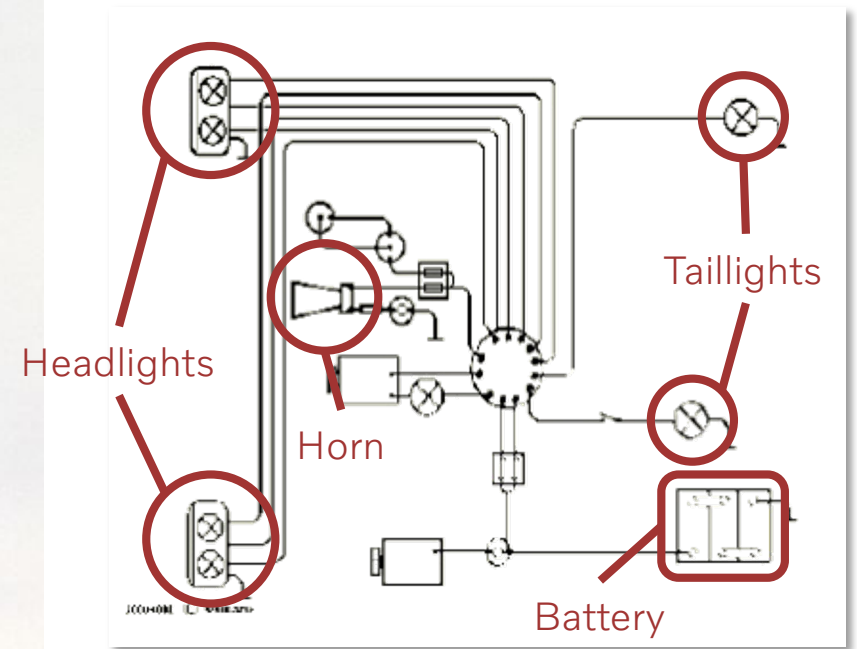
V O L V O

Automotive Software at Volvo Cars

Where are we now and how did we get here?

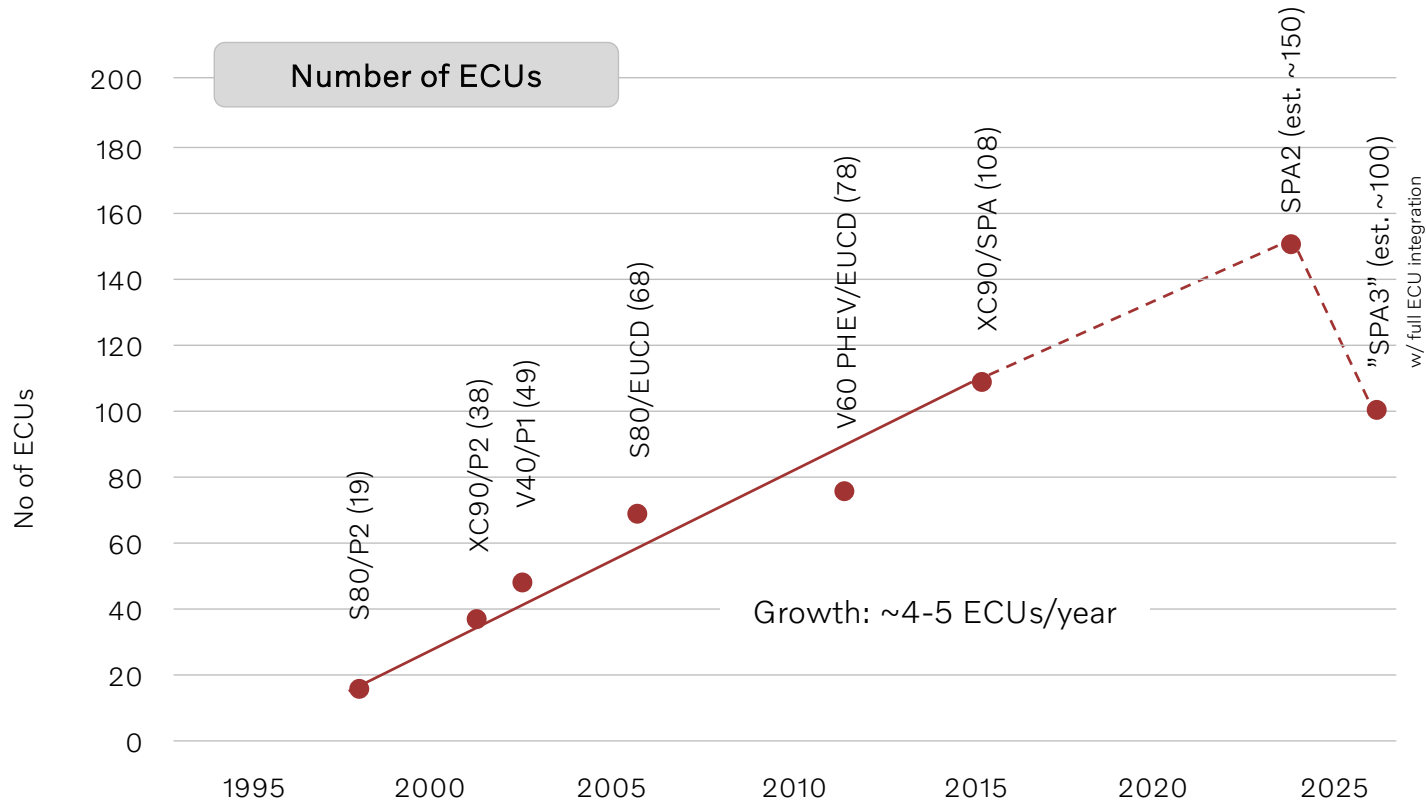
V O L V O

It all started in 1927 with Volvo ÖV4 (Jakob)



* Courtesy of Martin Hiller@ Volvo Cars

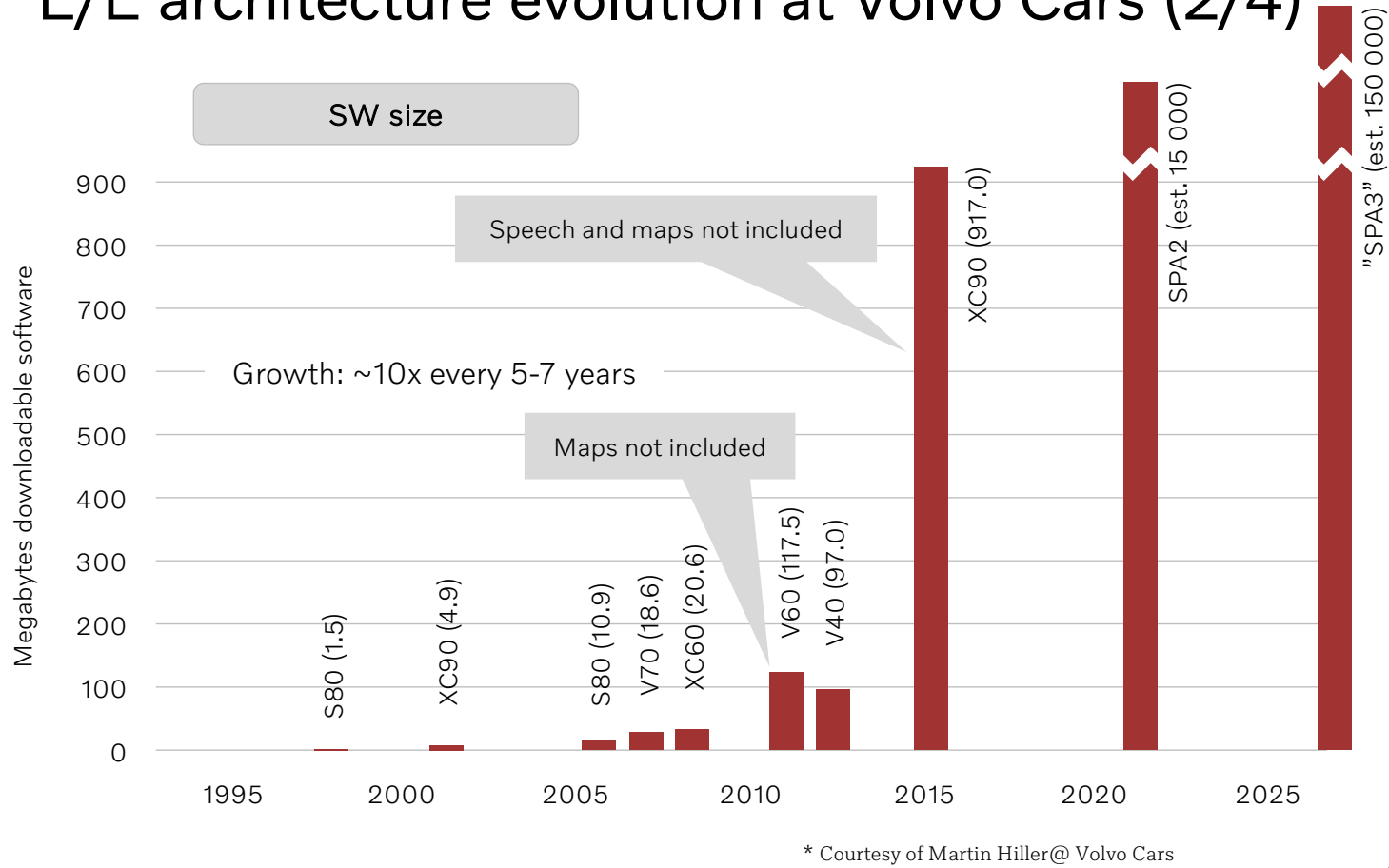
E/E architecture evolution at Volvo Cars (1/4)



* Courtesy of Martin Hiller@ Volvo Cars



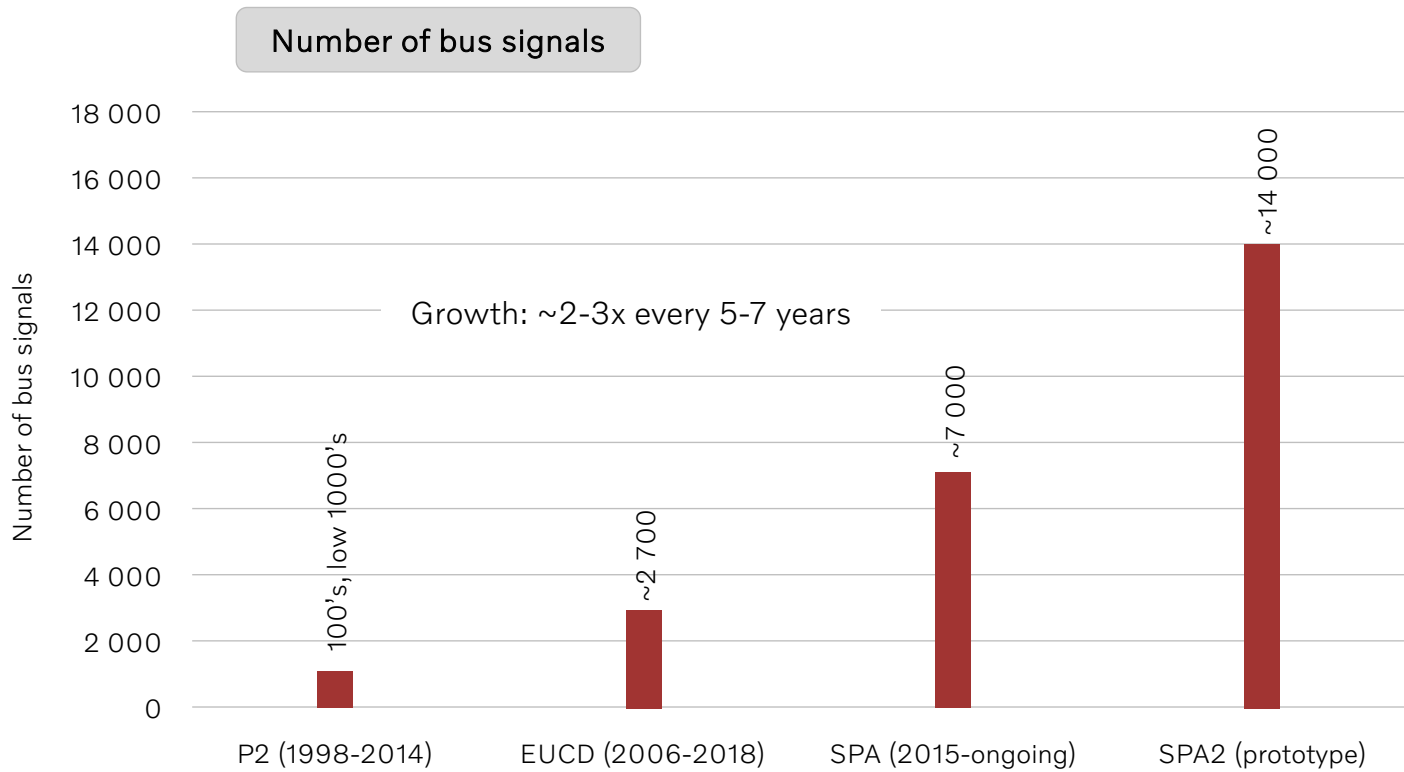
E/E architecture evolution at Volvo Cars (2/4)



Why Automotive Software Needs More Rust, Julius Gustavsson, Security Class: Public



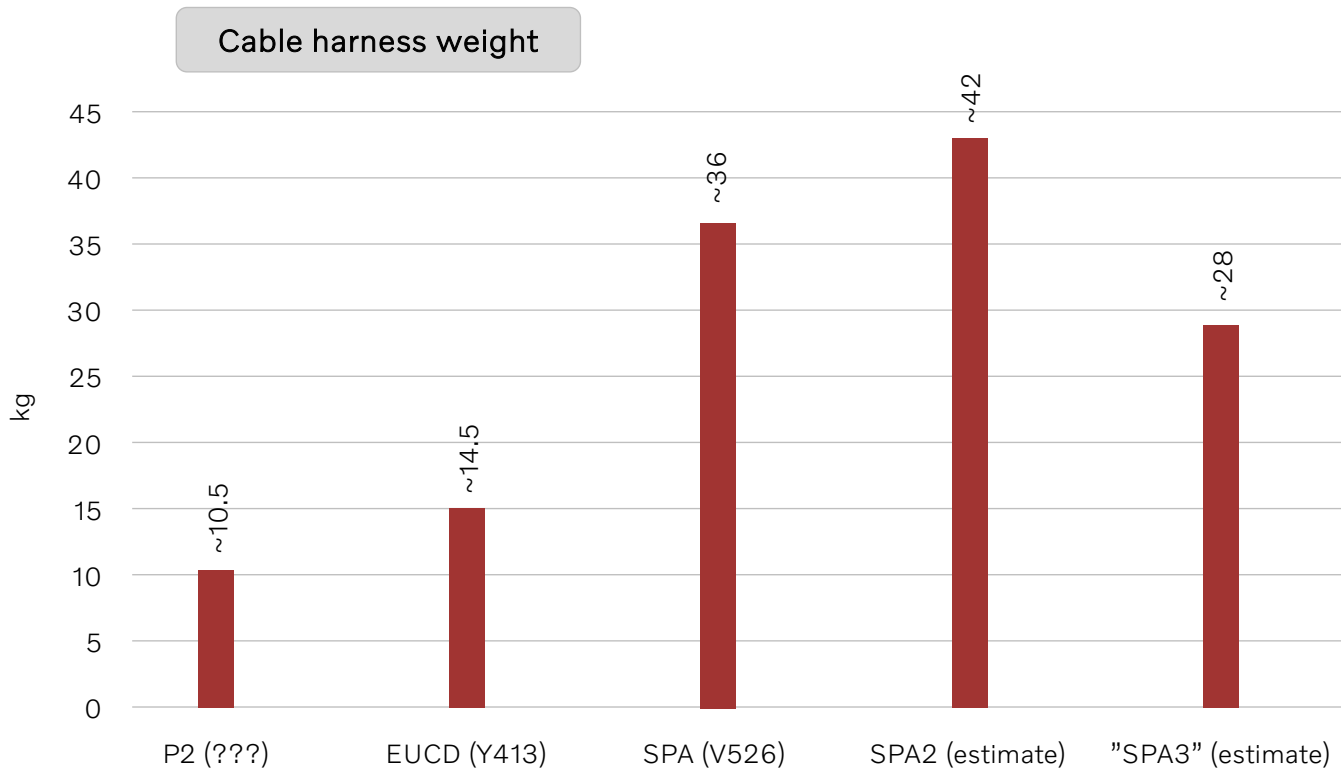
E/E architecture evolution at Volvo Cars (3/4)



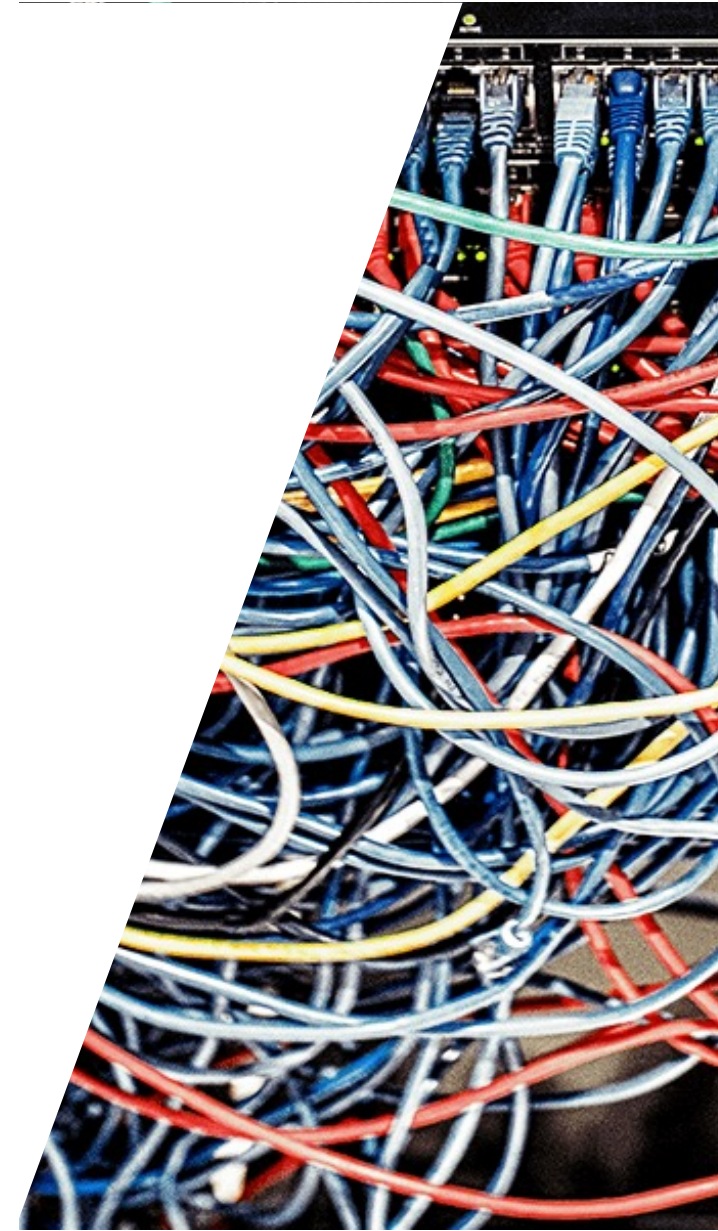
* Courtesy of Martin Hiller@ Volvo Cars



E/E architecture evolution at Volvo Cars (4/4)



* Courtesy of Martin Hiller@ Volvo Cars



V O L V O

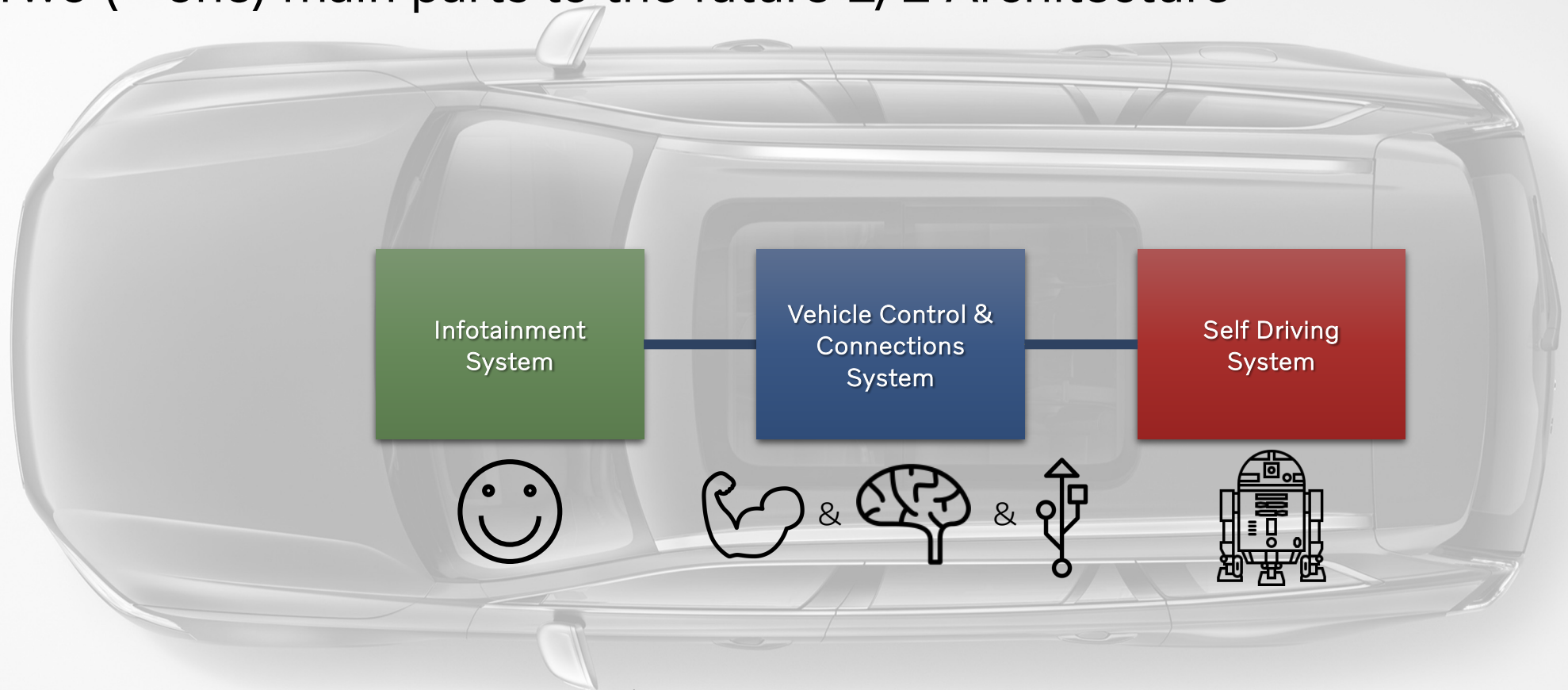
Where do we go from here?



* Courtesy of Martin Hiller @ Volvo Cars

V O L V O

Two (+ one) main parts to the future E/E Architecture



* Courtesy of Martin Hiller @ Volvo Cars

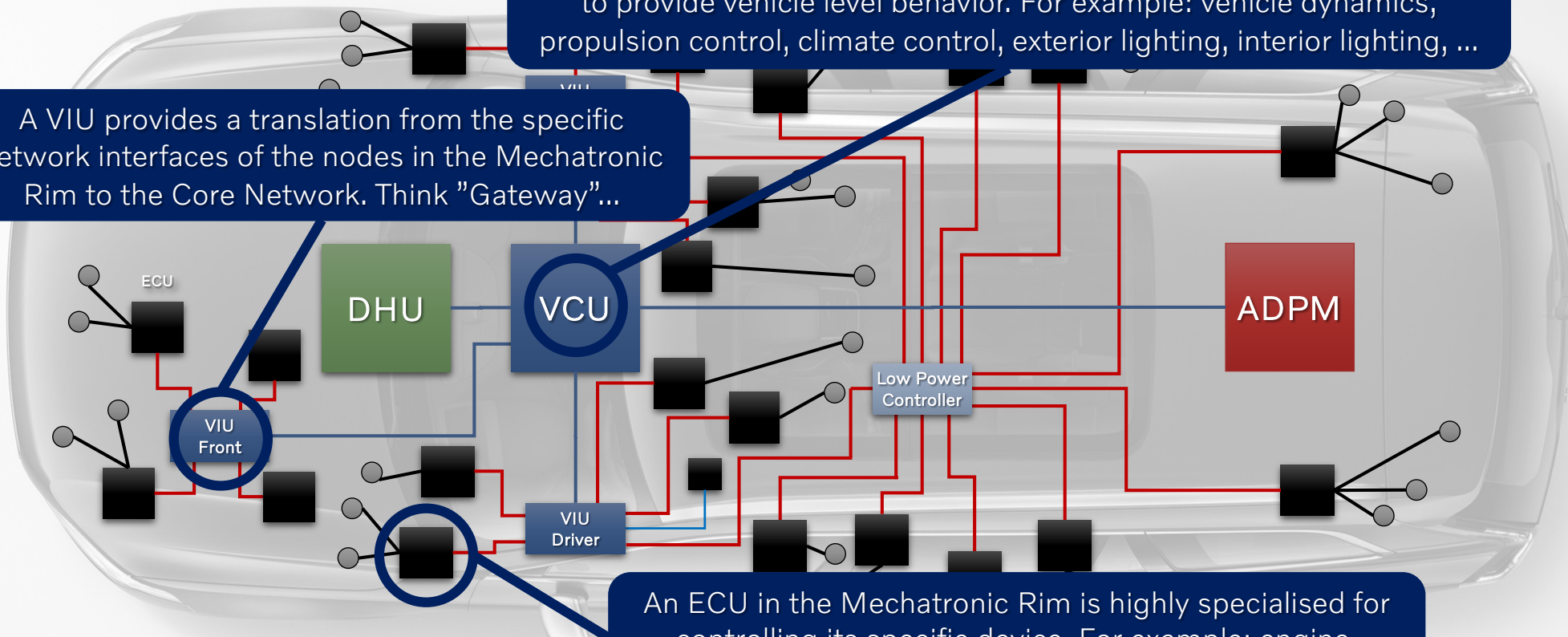
Step 1: Centralisation

Integrating domain masters and other compute h

Ethernet

The VCU coordinates fundamental capabilities in the Mechatronic Rim to provide vehicle level behavior. For example: vehicle dynamics, propulsion control, climate control, exterior lighting, interior lighting, ...

A VIU provides a translation from the specific network interfaces of the nodes in the Mechatronic Rim to the Core Network. Think "Gateway"...

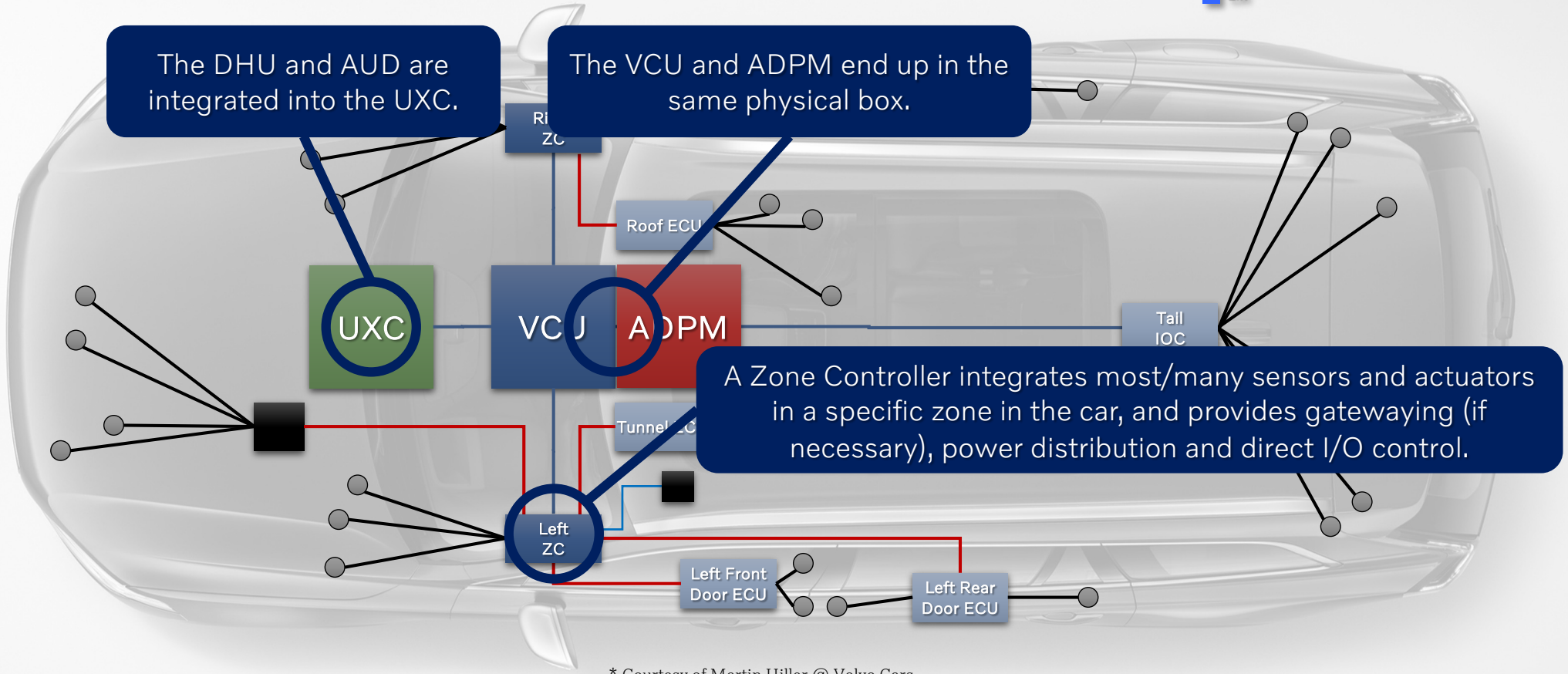
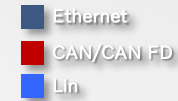


An ECU in the Mechatronic Rim is highly specialised for controlling its specific device. For example: engine, transmission, brakes, steering, doors, windows, seats, ...

* Courtesy of Martin Hiller @ Volvo Cars

Step 2: Integration

Integrating VIUs, power distribution and mechatronic ECUs into zone controllers

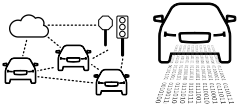

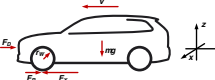









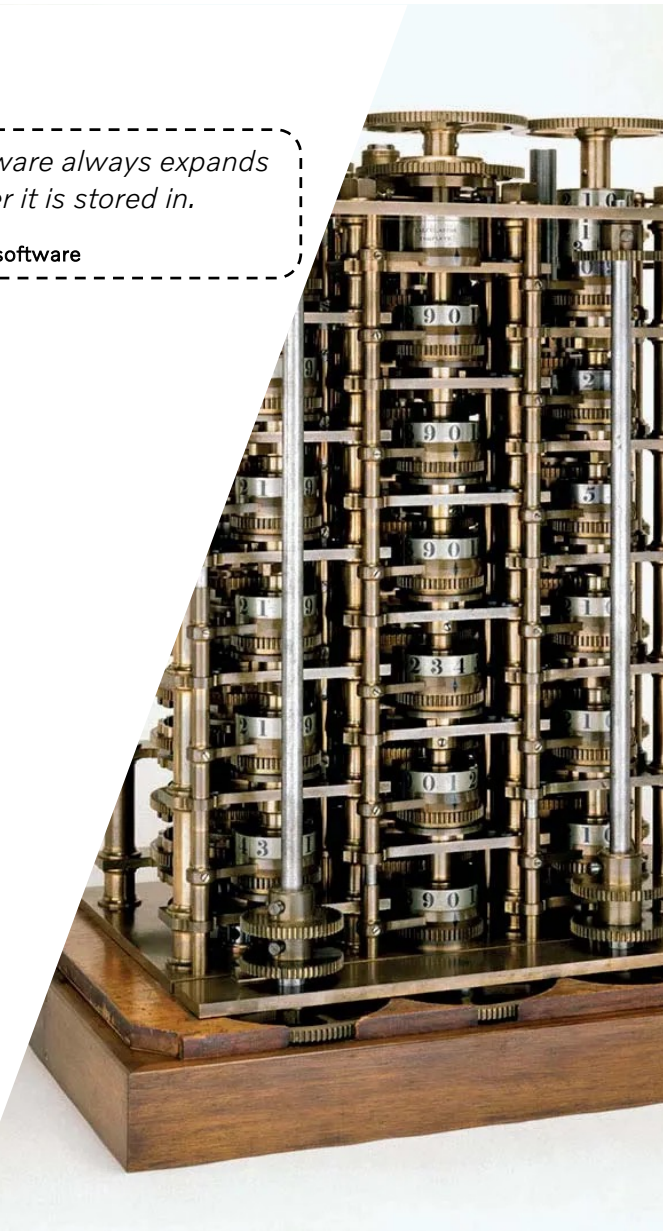
* Courtesy of Martin Hiller @ Volvo Cars

Computational needs – overview

Software is a gas. Software always expands to fit whatever container it is stored in.

- Nathan Myhrvold's 1st law of software

Functional area	Expected growth	CPU	GPU	NPU	ISP etc.	Other/ comments
Communication (internal & external) 		+				Network acc? Up to 50k packets/s coming into a core computer not too unrealistic
Traditional functions 		+	?		?	Assisted by vision? For example, viewing the road surface...
Infotainment & HMI 		++	++		++	AI/ML will surely come in here as well...
ADAS & Autonomy 		+++	+++	+++	+++	These guys are greedy... 500-1000 TOPS for "complete" stack
Other & unknown 		?	?	?	?	"Democratized" sensor data enables lots of cool new features...



Safety vs Security

Safety (as in personal safety)

- We need to ensure that our products are safe to use
- Hazard scenarios are identified and possible mitigations are put in place through various processes and methods
- C/C++ code is developed using a substantially limited subsets of the languages in an attempt only express well defined behavior
- The systems are extensively tested (using both formal and informal methods) to ensure that they behave as expected in all scenarios
- Once a system is safety certified, modifications are avoided as much as possible due to cost

Security (as in cyber security)

- We also need to ensure our products are safe from hacking
- Threat scenarios are identified and possible mitigation are put in place through various processes and methods
- C/C++ code is developed according to strict security guidelines in an attempt to minimize attack surface
- The systems are extensively tested (using both formal and informal methods) to ensure that they behave as expected in all scenarios
- A secure system is an update-to-date system where any identified vulnerabilities are patched.

V O L V O

Conclusion

The traditional tools and methods used by the automotive industry up until now are highly likely to be sub-optimal for tackling these new growth areas while maintaining or preferably increasing productivity.

Here we also should consider looking into new tool alternatives.

V O L V O

Now let's talk about Rust

The biggest improvement of the status quo during my career



V O L V O

How does Rust fare?

```
fn main() {  
    let mut a = [0u64; 1];  
    a[3] = 0x7ffff7a2e194u64;  
    println!("{:p}", &a);  
}
```

```
$ rustc -g -o test1 test1.rs
```

```
warning: this expression will panic at run-time
```

```
--> test1.rs:3:5 0|
```

```
3 | a[3] = 0x7ffff7a2e194u64;
```

```
0 |   ^^^^ index out of bounds: the len is 1  
    but the index is 3
```

```
$ ./test1
```

```
thread 'main' panicked at
```

```
'index out of bounds: the len is 1 but the  
index is 3', test1.rs:3:5
```

```
note: Run with `RUST_BACKTRACE=1` for a  
backtrace.
```

V O L V O

How does Rust fare?

```
fn main() {  
    let mut v = Vec::new();  
    v.push("Is");  
    let tmp = &v[0];  
  
    v.push("this");  
    v.push("Ok");  
    println!("{}", tmp);  
}
```

```
$ rustc -g -o test2 test2.rs
```

```
error[E0502]: cannot borrow `v` as mutable  
because it is also borrowed as immutable  
--> test2.rs:7:5  
0 |  
5 |     let tmp = &v[0];  
0 |         - immutable borrow occurs  
here  
  
6 |  
7 |     v.push("You");  
0 |     ^ mutable borrow occurs here  
  
...  
11| }  
   | - immutable borrow ends here
```

What does Rust bring to the table?

- Safe by default
 - Eliminates an entire class of bugs
- Automatic memory management without GC
 - Works just as well with or without an OS
- Can communicate with C/C++ with little or no overhead
 - No need to rewrite everything. Focus on the things that gives value
- Possibility to “bypass” some of the compiler checks by using the `unsafe` keyword
 - Unsafe code is clearly marked and easy to audit
- Combines the best concepts from both imperative and functional languages into a compelling package
 - Brings various interesting patterns into the systems domain
- A powerful static type system
 - Invariants and beviour can be enforced at compile time
 - Thread-safety is completely solved on type level!
- Tests are first class citizens and world class tooling at your fingertips
 - Lowers the barrier to modern DevSecOps practices
- Vibrant open-source community that produces some of the best-in-class third-party components out there
 - Composable high quality components that can significantly reduce development time

V O L V O

How does that apply to Safety & Security?

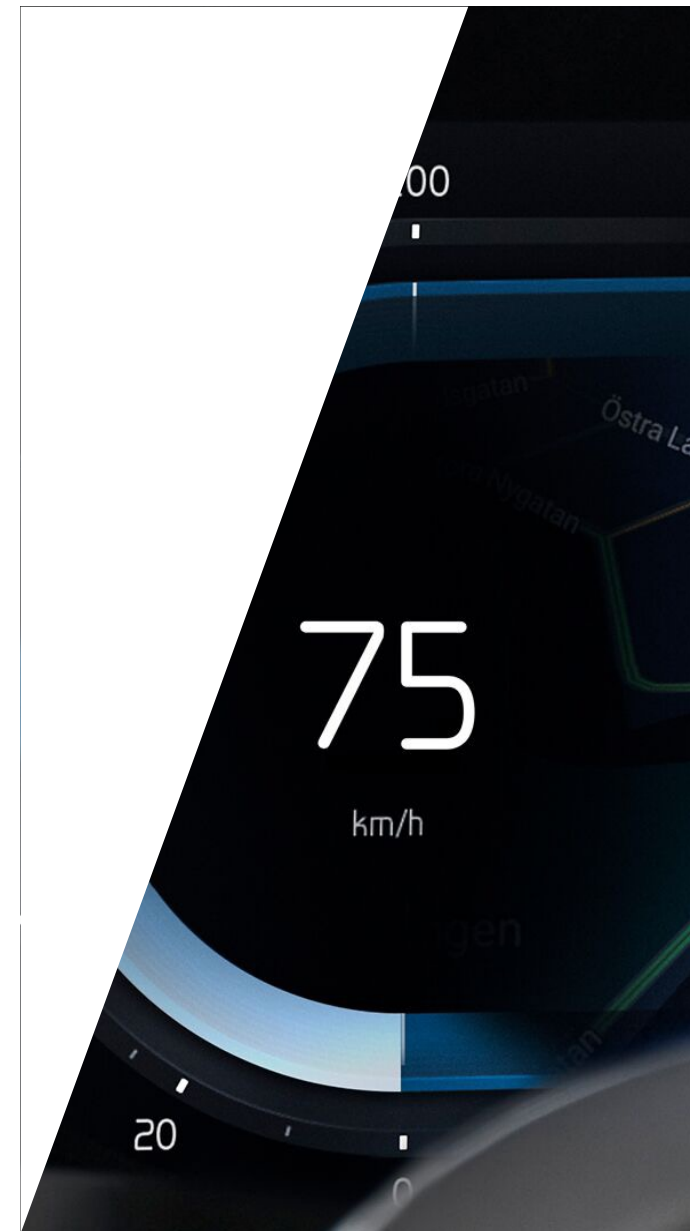
Rust has the potential to combine the two in a unique way:

The powerful type system can be used to encode various safety and security invariants that the compiler will enforce at compile time

- Changes and refactoring can be done with confidence because the compiler will have your back if you slip up
- Re-certification should become a simpler and cheaper process

Correctness up front at lower cost/effort

- Less need for hotfix updates
- When a hotfix is needed, it should be easier to ensure that all safety invariants are still upheld

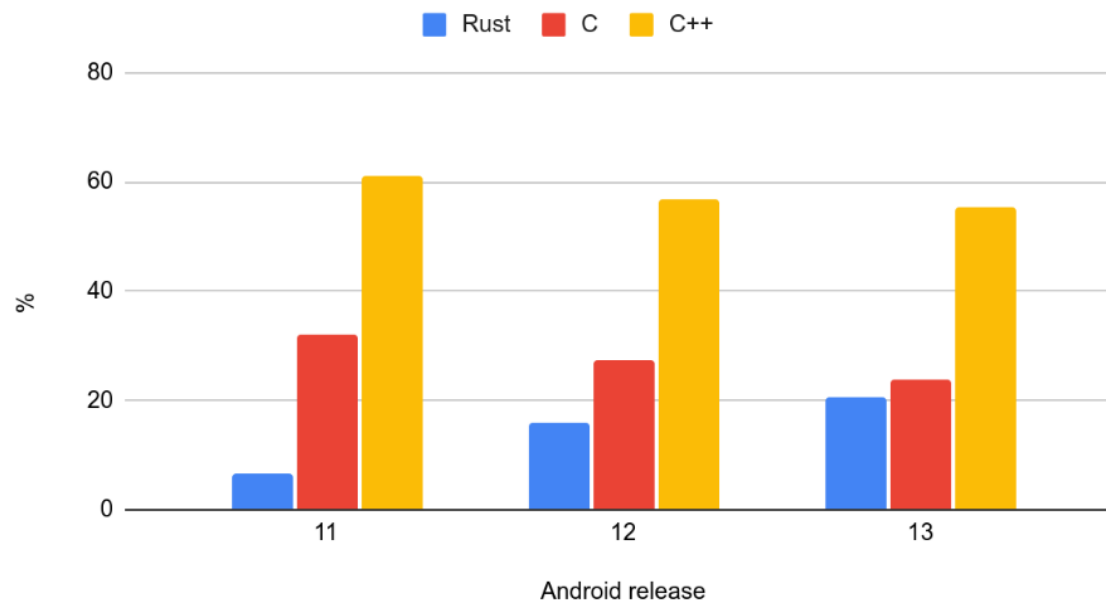


V O L V O

How should we adopt Rust in a large legacy codebase?

Focus on new code rather than rewrites

New Native Code



” In Android 13, about 21% of all new native code (C/C++/Rust) is in Rust. There are approximately 1.5 million total lines of Rust code in AOSP”

...

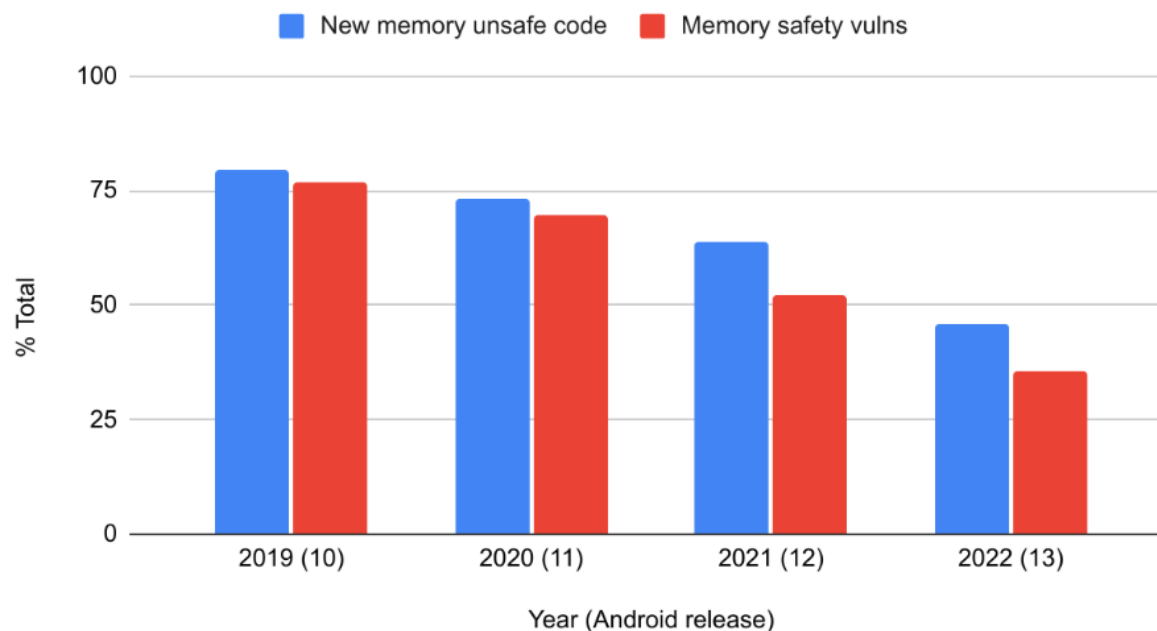
“To date, there have been zero memory safety vulnerabilities discovered in Android’s Rust code.”

- [Google Security Blog](#)

V O L V O

The Android vulnerability trend is finally broken!

Memory unsafe code and Memory safety vulnerabilities



“While correlation doesn’t necessarily mean causation, it’s interesting to note that the percent of vulnerabilities caused by memory safety issues seems to correlate rather closely with the development language that’s used for new code.”

- Google Security Blog

”Rust has an ownership model that guarantees both memory safety and thread safety, at compile-time, without requiring a garbage collector. This allows users to write high-performance code while eliminating many bug classes. Though Rust does have an unsafe mode, its use is explicit, and only a narrow scope of actions is allowed. (14 Mar 2023)”

- NIST / Software Quality Group - Safer Languages

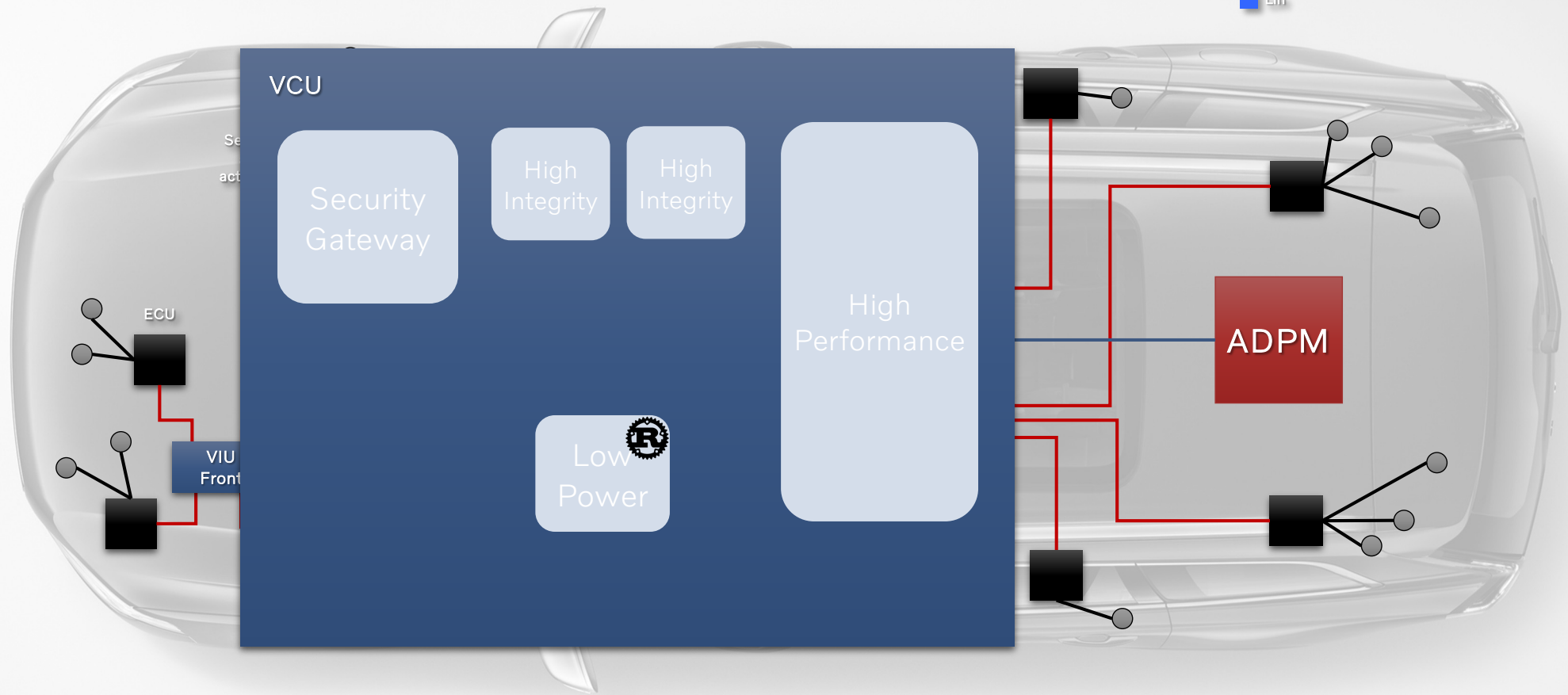
V O L V O

How are we currently using Rust at Volvo Cars?



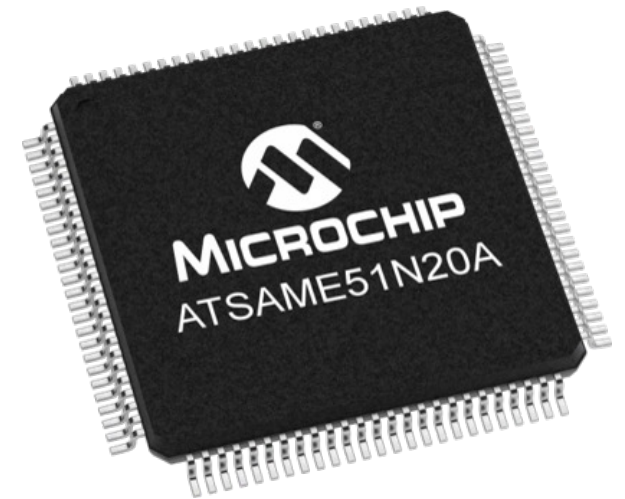
The VCU Low-Power Domain

- Ethernet
- CAN/CAN FD
- Lin



Functionality

- Always-on
 - Low power consumption is super important
- Powers on the VCU based on different criteria
 - Robustness is key! We want the car to always turn on when it should and never when it shouldn't
- Facilitates communication between different subsystems during sleep
 - Real-time characteristics
- Can only power the system on and not off
 - Not safety critical
- Limited functional scope
 - Suitable for a small team. Perfect as a test bed for Rust!



Implementation

- 100% Pure Rust Application based on [RTIC](#)
- HW drivers developed by Grepit AB in collaboration with the [atsamd](#) HAL project
- Custom development boards (also developed by Grepit AB) allows us to test our firmware to almost 100% before deploying it on the VCU
- A fully automated CI/CD pipeline based on [Zuul CI](#) where each code change is extensively tested on our custom hardware before being promoted to higher level testing on the VCU hardware



Reflections so far

- Enormous productivity boost!
 - The combination of the Rust language, Cargo build system/pkg manager, and vibrant third party eco system is really a “1 + 1 + 1 = 5” proposition
- The so called “steep learning curve” has not turned out to be a real issue
 - But make sure the team has access to Rust experts to avoid getting stuck
 - On the contrary, on-boarding new people on the team has been easier than before
 - The language and tools enables newcomers to “dive-into” new unfamiliar code-bases without fear
- The in-flow of error reports seems to be much lower
 - Still too early to tell, but the long tail of quality issues is still nowhere to be seen
- Memory footprint and performance is on par or better than C/C++
 - Binary size tends to be larger though
- The available tooling integrates seamlessly with cargo
 - Huge productivity boost!
 - But there are also gaps in the available tooling
- Instead of asking: “Should we use Rust for this?”
- Ask: “Is there a valid reason *not* to use Rust for this?”

V O L V O

But is that the whole story?



It's not a panacea

- It is a powerful but also rather complex language
 - A number of novel concepts you need to understand before you can become productive
- Let the compiler guide you, otherwise you're in for a treat!
 - You may have to un-learn various unsafe practices that you get away with in other languages
- Although the over all lead-time to a fully working product is shorter, initial progress can be slower
 - Before your program even compiles you are forced to think through the implications of your design choices
- Third-party eco system is still young and incomplete in many areas (particularly automotive)
 - Either an obstacle or an opportunity depending how you look at it
- Platform support is still early on many automotive platforms although rapidly improving (e.g. QNX and Infineon Aurix support added during 2023)
- Not yet qualified for safety critical applications according to automotive standards
 - Important work being done the [Ferrocene](#) project
 - Certification from TÜV expected within days!
 - [AUTOSAR](#) and [SAE](#) have both started Rust work groups
 - 2023 H2 will most likely be huge milestone

”Want to increase innovation? Lower the cost of failure!”

- Joichi Ito



Thank You!

Questions?