

HOW TO RUST WHEN STANDARDS ARE DEFINED IN C



EDDY PETRISOR

/ 'ɛdi pɛtrɪʃɔr /

Embedded SW Engineer / Automotive Industry

@ eddy.petrisor+rust@gmail.com

 github.com/eddyp

 <http://ramblingfoo.blogspot.com/>

AUTOMOTIVE & STANDARDS

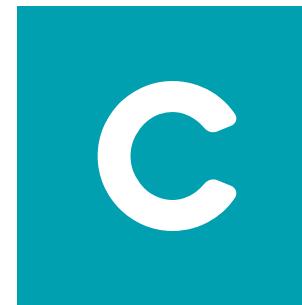


I CAN DO ANYTHING IN C!

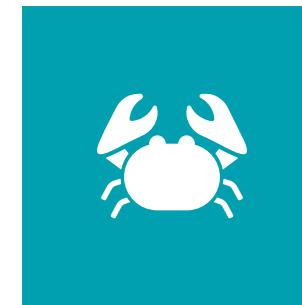




VALUES MATTER



SIMPLE



MEMORY SAFE

See video: [Scale By The Bay 2018: Bryan Cantrill, Rust and Other Interesting Things](#)

CAN WE RUST?

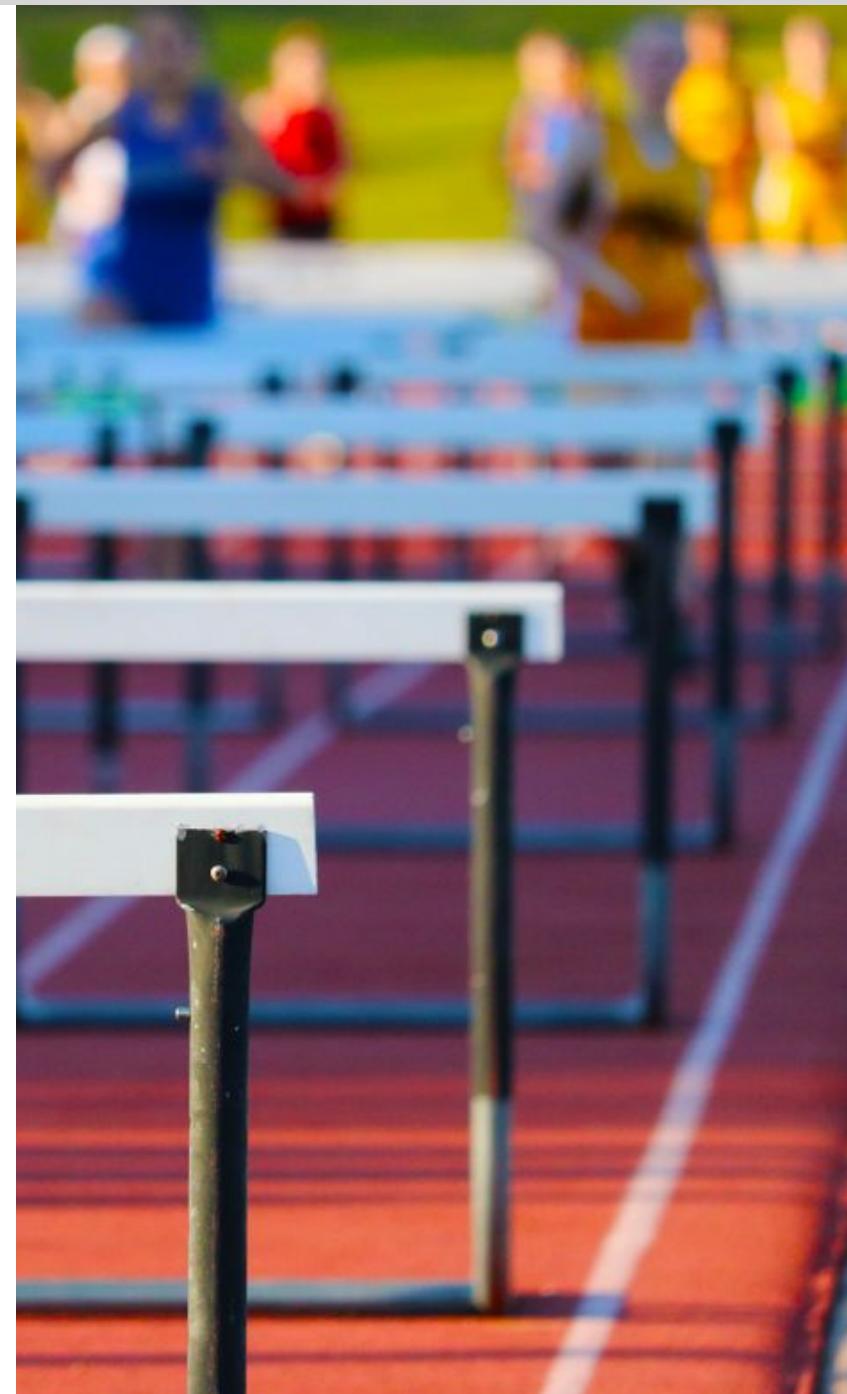
1 AUTOSAR STANDARD

2 C, ASM & RUST INTEGRATION

3 KNOWN (RUST) PROBLEMS

4 ENGINEERS & MANAGERS

5 LEGACY SOFTWARE



CAN WE RUST?

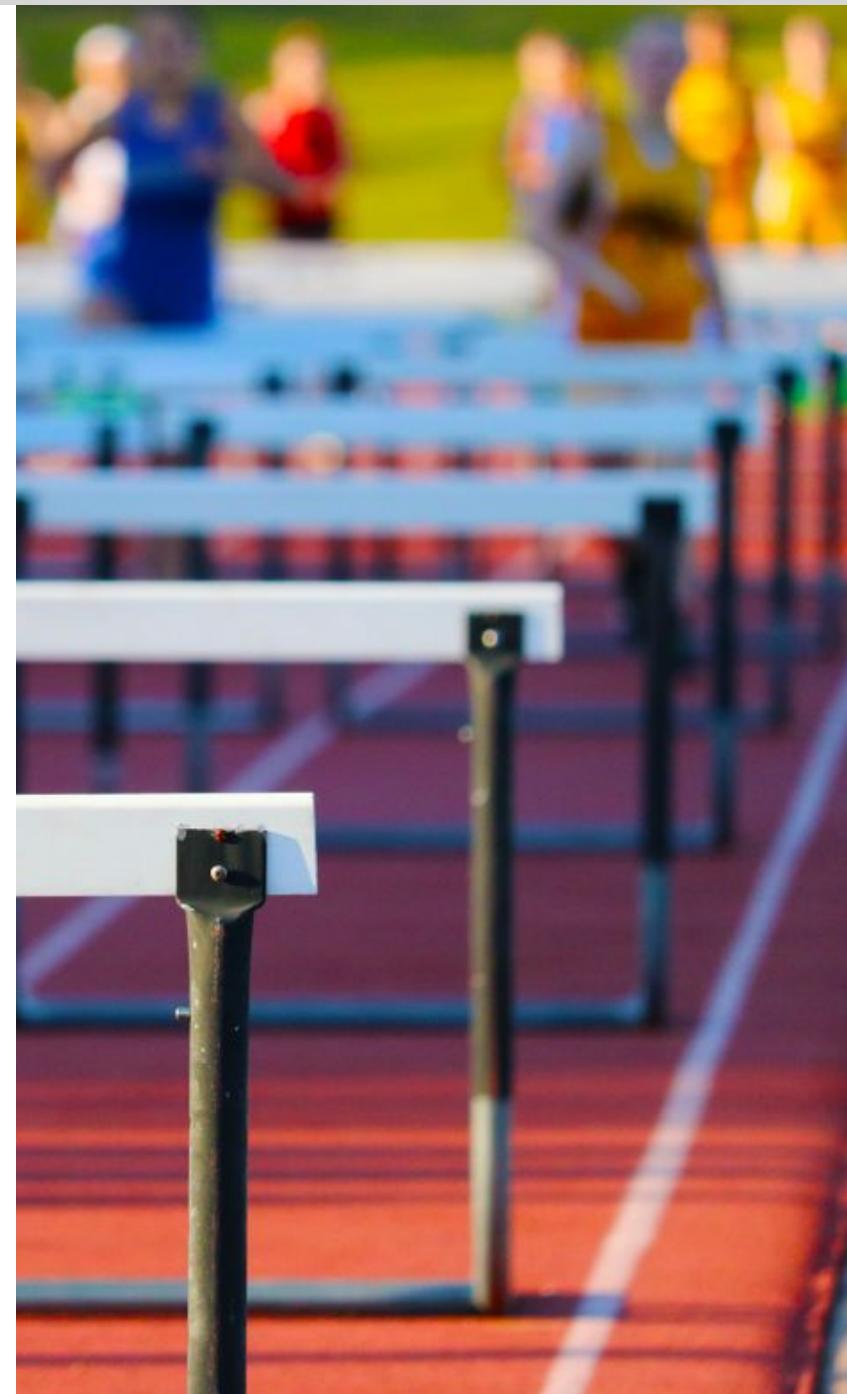
1 AUTOSAR STANDARD

2 C, ASM & RUST INTEGRATION

3 KNOWN (RUST) PROBLEMS

4 ENGINEERS & MANAGERS

5 LEGACY SOFTWARE



QUOTES FROM THE AUTOSAR STANDARD



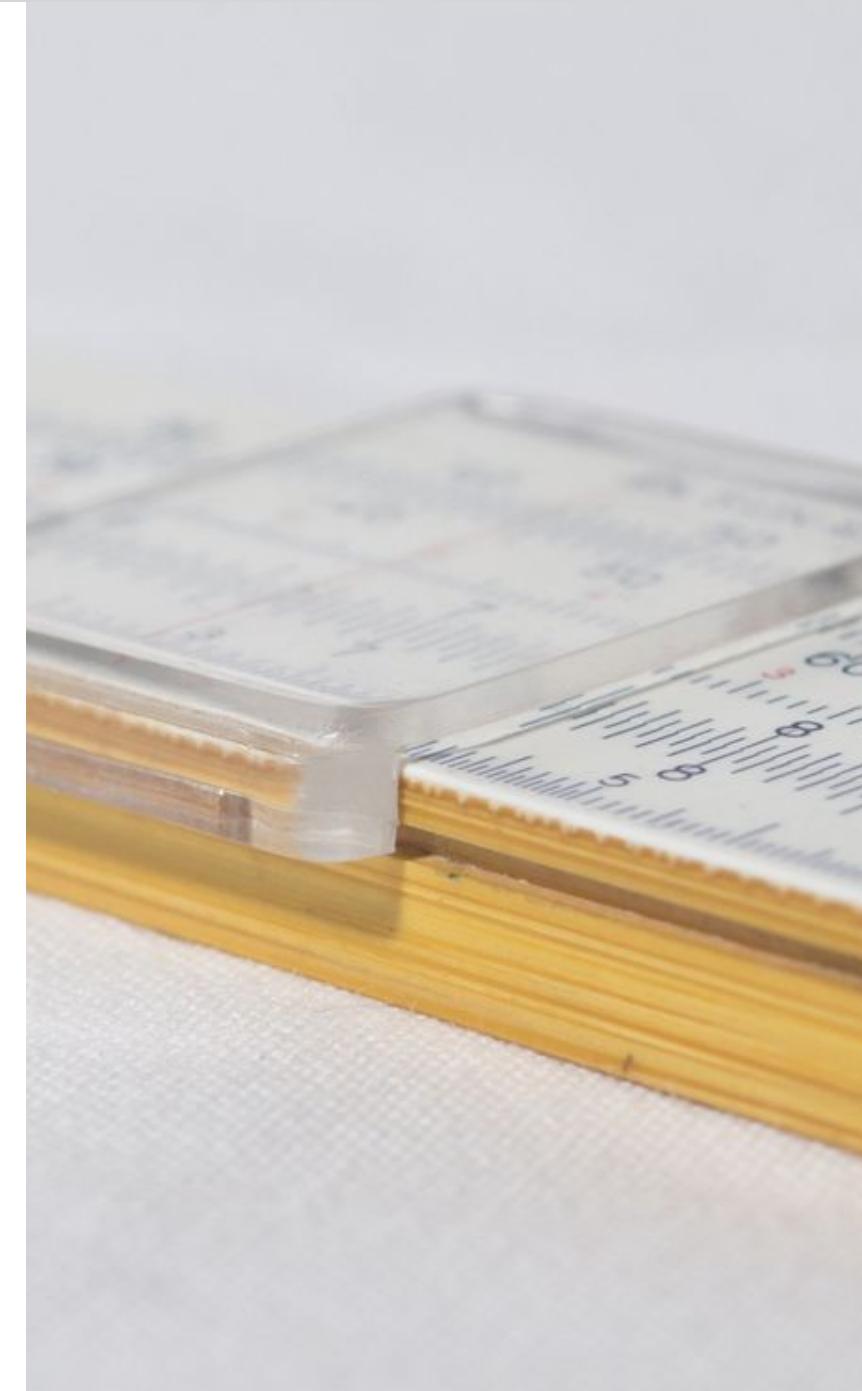
DISCLAIMER

This specification may be utilized or reproduced without any modification, in any form or by any means, for informational purposes only

SPECIFICATION OF OPERATING SYSTEM

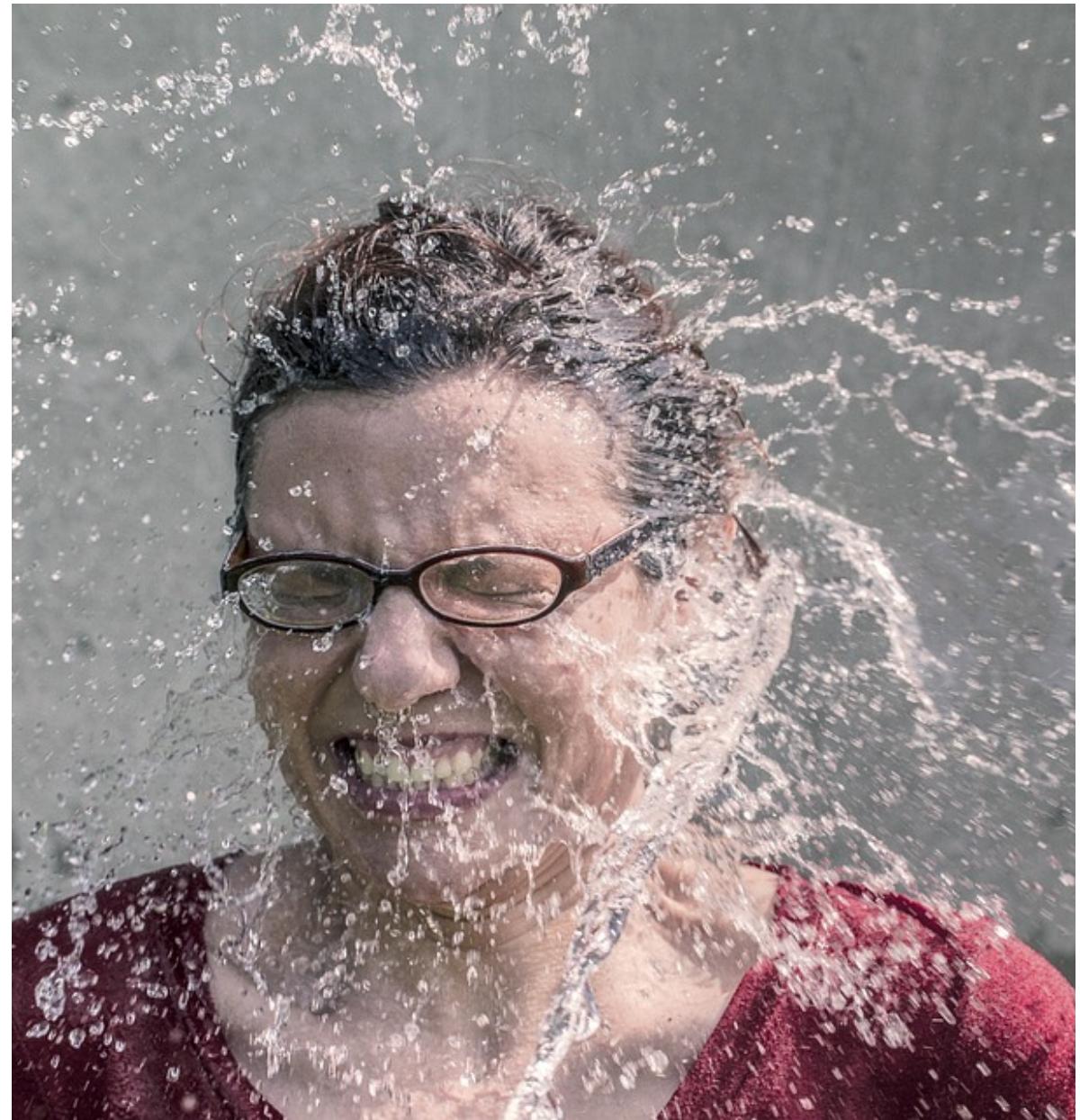
V5.0.0 R4.0 Rev 3 - 4.6.2 Programming Language

The API of the operating system is defined as **C89** [23] function calls or macros. **If other languages are used they must adapt to the C interface.** This is because C99[24] allows for internal dynamic memory allocation during subroutine calls. Most automotive applications are **static (non-heap based)**.



REALITY CHECK!

BSW171: [...] functionalities [...] which are disabled by static configuration shall not consume resources (RAM, ROM, runtime).



OPTIMIZING RAM USAGE IN C (PREPROCESSOR)

```
#if (OS_NO_OF_RESOURCES > 0U)

    #if (OS_NO_OF_RESOURCES <= 8U)
        typedef uint8_t res_t;

    #elif OS_NO_OF_RESOURCES <= 16U
        typedef uint16_t res_t;

    #elif OS_NO_OF_RESOURCES <= 32U
        typedef uint32_t res_t;

    #else
        /* typically somewhere else, maybe computed by a cfg generator */
        #define OS_RES_ARRAY_SIZE \
            ((OS_NO_OF_RESOURCES / 8) + (!!(OS_NO_OF_RESOURCES % 8)))
        typedef uint8_t res_t[OS_RES_ARRAY_SIZE];

#endif
```

OPTIMIZING RAM USAGE IN C - USING THE PREPROCESSOR

```
struct tcb_t {  
    ... TaskIdType id; /* also priority */  
    ... struct ctx_t ctx;  
  
    /* if no memory protection, stack is common */  
    #if defined(USE_PREEMPTION) || defined(USE_MEMORY_PROTECTION)  
    ... stack_info *stk; /* ISRs of the same prio share the stack */  
    #endif  
  
    #if (OS_NO_OF_RESOURCES > 0U)  
    ... res_t res_mask;  
    #endif  
  
    #if (OS_NO_OF_ALARMS > 0U)  
    ... alarm_t alarms;  
    #endif  
};
```

(AB)USING THE PREPROCESSOR / A POWERFUL TOOL

https://github.com/evidence/erika3/blob/GH63/pkg/kernel/oo/ee_ar_sched_table.c#L52

```
52 #include "ee_internal.h"
53
54 FUNC(StatusType, OS_CODE)
55 . osEE_st_start_rel
56 (
57 . P2VAR(OsEE_CounterDB, AUTOMATIC, OS_APPL_DATA) . p_counter_db,
58 . P2VAR(OsEE_SchedTabDB, AUTOMATIC, OS_APPL_DATA) . p_st_db,
59 . VAR(TickType, ...AUTOMATIC) . . . . . offset
60 )
61 {
62 . VAR(StatusType, AUTOMATIC) . ev;
63 . CONSTP2VAR(OsEE_SchedTabCB, AUTOMATIC, OS_APPL_DATA)
64 . . p_st_cb = osEE_st_get_cb(p_st_db);
65 . . CONSTP2VAR(OsEE_TriggerDB, AUTOMATIC, OS_APPL_DATA)
66 . . . p_trigger_db = osEE_st_get_trigger_db(p_st_db);
67 . . CONSTP2VAR(OsEE_TriggerCB, AUTOMATIC, OS_APPL_DATA)
68 . . . p_trigger_cb = p_trigger_db->p_trigger_cb;
69 #if (!defined(OSEE_SINGLECORE))
70 . CONST(CoreIdType, AUTOMATIC)
71 . . counter_core_id = p_counter_db->core_id;
72 . . /* Lock the Core Lock witch the counter is tied */
73 . . osEE_lock_core_id(counter_core_id);
74 #endif /* OSEE_SINGLECORE */
75 |
76 . if (p_trigger_cb->status > OSEE_TRIGGER_CANCELED) {
```

OPTIMIZING RAM USAGE IN RUST



EMULATE C PREPROCESSOR?

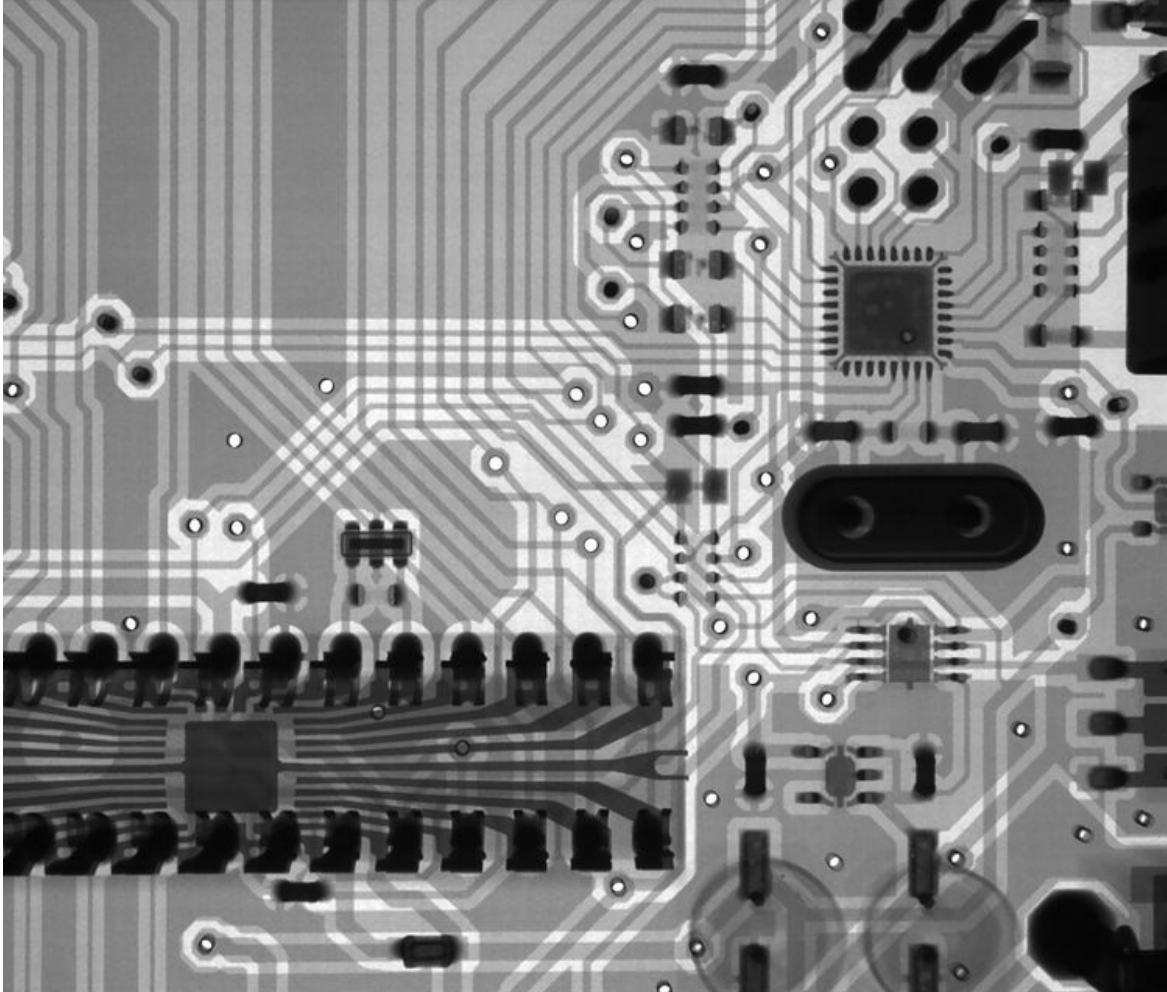
OPTIMIZING RAM USAGE IN RUST



**IS THIS BEHIND CLOSED
DOORS?**

It matters only if used in the interface

OPTIMIZING RAM USAGE IN RUST

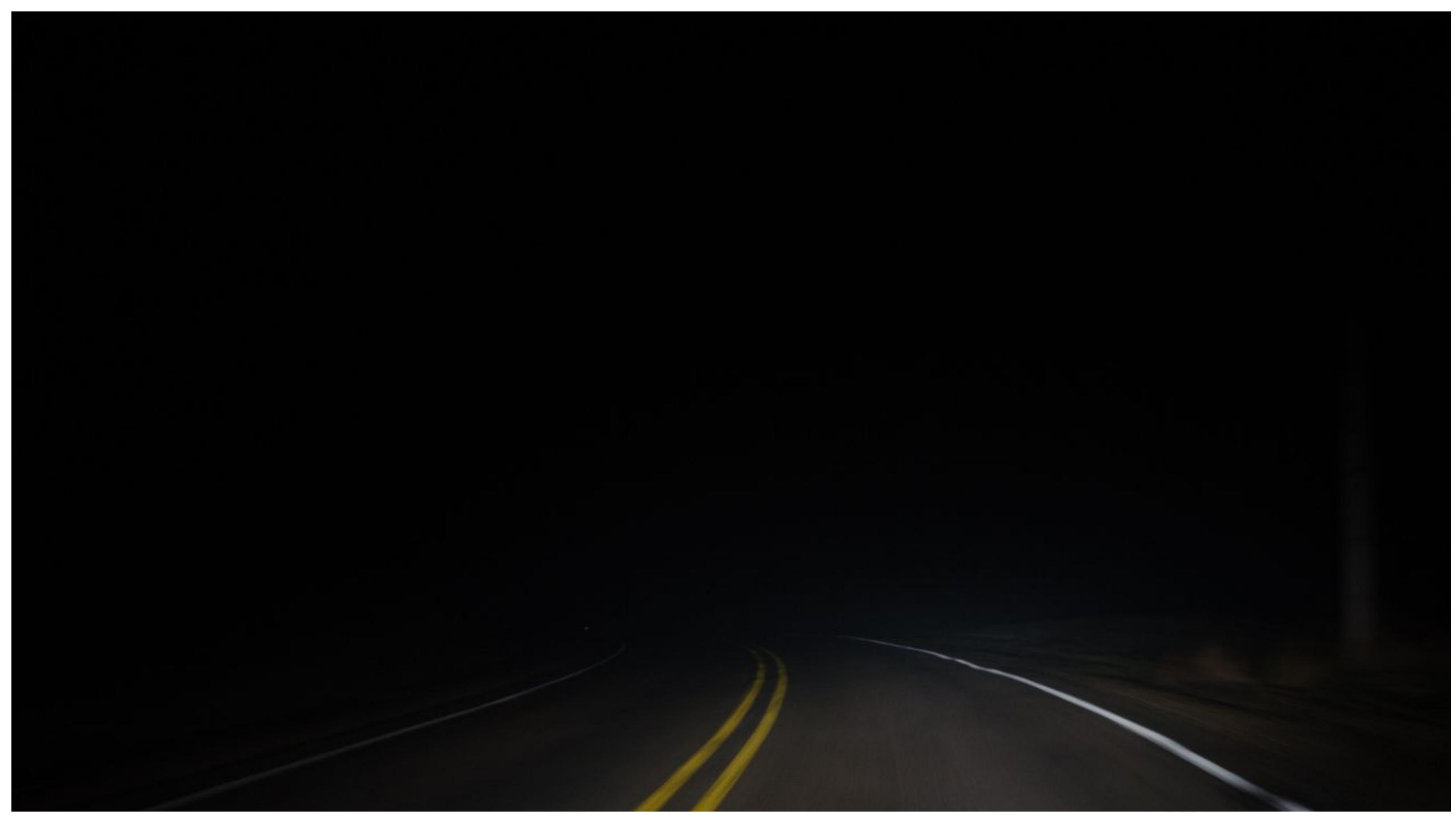


SMART COMPILER?

RUST: OPTIONAL STRUCT FIELDS

```
#[repr(C)]
pub struct TaskControlBlock {
    id: SafeTaskIdType,
    ctx: TaskContext,
    #[cfg(any(mem_protection, preemption))]
    stk: StackInfo,
    #[cfg(resources)]
    res_mask: ResourceType,
    #[cfg(alarms)]
    alarms: Alarms,
}
```





CAN WE RUST?

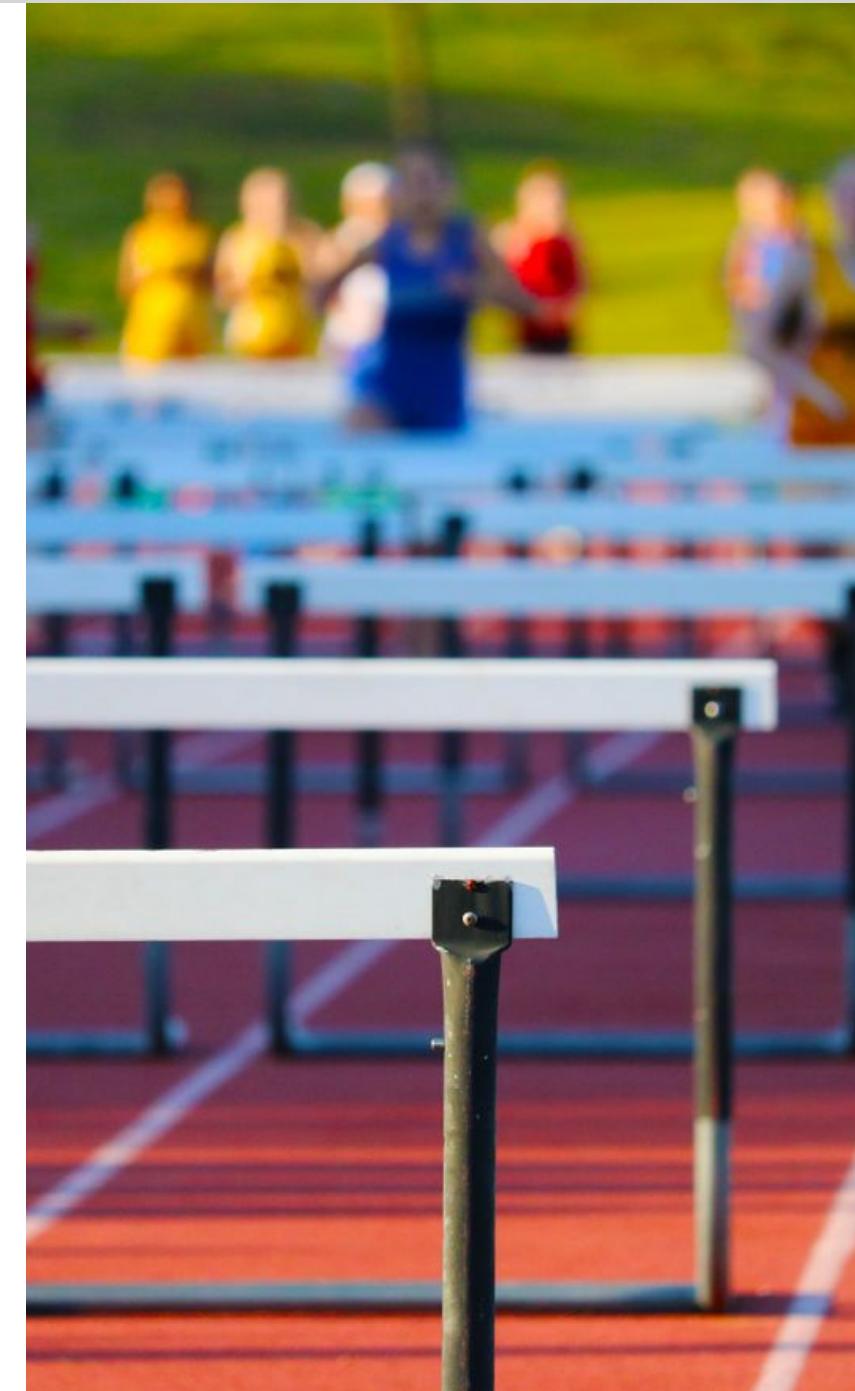
1 AUTOSAR STANDARD

2 C, ASM & RUST INTEGRATION

3 KNOWN (RUST) PROBLEMS

4 ENGINEERS & MANAGERS

5 LEGACY SOFTWARE



WHERE IS THE RUST CODE?



RUST
COMPONENT
SOURCES

RUST
COMPONENT -
BINARIES &
GENERATOR

CONFIGURATION

LINKING INTO
INTEGRATION
APP

THE BRIDGE BETWEEN C & RUST

Config files:
Os_cfg.h, Os_cfg.c



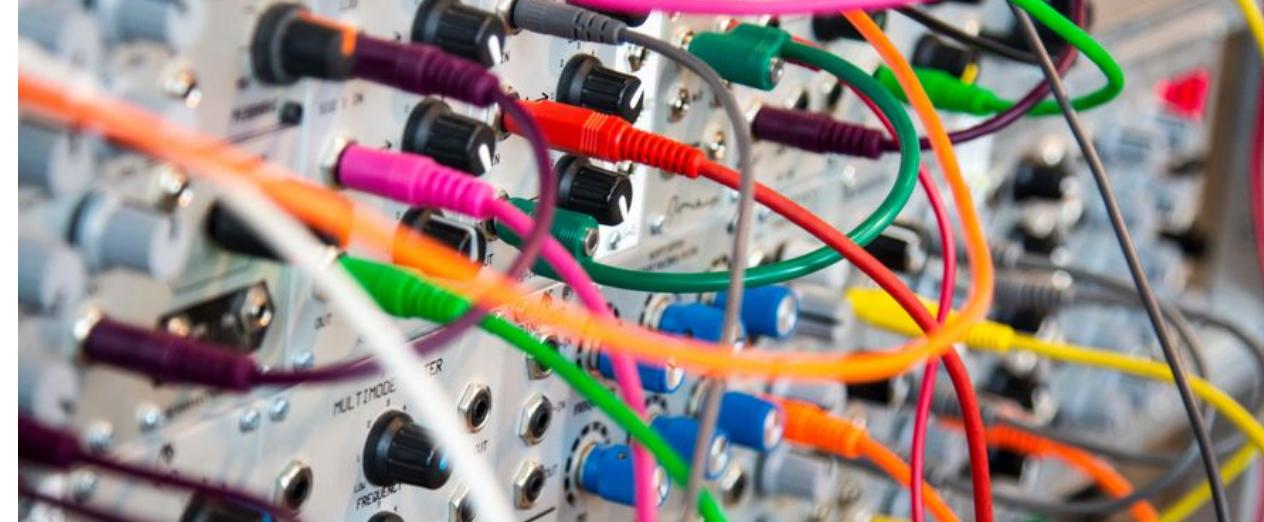


LET'S TAKE IT EASY ...



IN

THE EASY “MUSTS”



THE EASY “MUSTS” - LINKER FILES

```
linker.ld      x

1
2 SECTIONS
3 {
4     . = 0x10000;
5     .ostext : { librust.os.a(.text) }
6     .text : { *(.text) }
7     . = 0x80000000;
8     .data : { *(.data) }
9     .bss : { *(.bss) }
10 }
11
```

THE EASY “MUSTS” - NO MANGLING

```
lib.rs      x

75
76 // fn get_safe_task_id(unsafe_task_id: TaskIdType) -> Result<SafeTaskIdType, StatusType>
77
78 // C declaration: StatusType ActivateTask (TaskType TaskID);
79 #[allow(non_snake_case)]
80 #[no_mangle]
81 pub fn ActivateTask(TaskID: TaskIdType) -> StatusType {
82     match get_safe_task_id(TaskID) {
83         Ok(_safe_task_id) => {
84             // really activate task
85         }
86         Err(status) => return status,
87     }
88
89     StatusType::E_OK
90 }
```

THE EASY “MUSTS” - C REPRESENTATION

The Rustonomicon / Alternative representations: <https://doc.rust-lang.org/nomicon/other-reprs.html>

Cargo.toml | lib.rs | x

```
10
11 #[repr(C)]
12 pub struct TaskIdType
13 {
14     id: libc::uint32_t,
15 }
16
17 #[allow(non_camel_case_types)]
18 #[repr(C)]
19 pub enum StatusType {
20     E_OK = 0,
21     E_OS_ACCESS = 1,
22     E_OS_CALLEVEL = 2,
23     E_OS_ID = 3,
```

THE EASY “MUSTS” - FFI: WHAT TO AVOID

<https://doc.rust-lang.org/nomicon/other-reprs.html#reprc>

```
1  repr(C) will be nonsensical or problematic:  
2  
3  * ZSTs are still zero-sized even though  
4    * is not standard behavior in C  
5    * in C++ they should consume a byte  
6  * Enums with fields  
7    * aren't a concept in C or C++  
8    * a valid bridging is defined - unimplemented!  
9  * Tuple struct: like struct but fields aren't named  
10 [..]
```


ENUM: SHORT/SMALL/BEST & SIGNED/UNSIGNED

ABI compatibility matters

gcc USING the GNU Compiler X +

https://gcc.gnu.org/onlinedocs/gcc-8.2.0/gcc/Structures-unions-enu ... 🇮🇪 ⌂ ⚡ ⌄ ⌁ 1 ⌂ ⌄

- *The integer type compatible with each enumerated type (C90 6.5.2.2, C99 and C11 6.7.2.2).*

Normally, the type is `unsigned int` if there are no negative values in the enumeration, otherwise `int`. If `-fshort-enums` is specified, then if there are negative values it is the first of `signed char`, `short` and `int` that can represent all the values, otherwise it is the first of `unsigned char`, `unsigned short` and `unsigned int` that can represent all the values.

On some targets, `-fshort-enums` is the default; this is determined by the ABI.

Next: [Qualifiers implementation](#). Previous: [Hints implementation](#). Up: [C Implementation](#)



MORE UNDEFINED BEHAVIOR, NOW 100% ALIGNED

Technically correct, but still undefined behavior

```
3 ►
4 struct aXc {
5     uint32_t a;
6     uint16_t X;
7     uint32_t c;
8 };
9
10 //      0   1   2   3
11 // +-----+
12 // 0 | a | a | a | a |
13 // +-----+
14 // 4 | X | X |   |   |
15 // +-----+
16 // 8 | c | c | c | c |
17 // +-----+
18 ►
19
```

THE CURIOUS CASE OF DIAB STRUCT PACK - (2, 8, *)

```
#pragma pack(max_member_align, min_struct_align, byte-swap)
```

```
#pragma pack(2, 8[, byte-swap])
struct aXc_pack_maxmember2_minstruct8 {
    uint32_t a;
    uint16_t X;
    uint32_t c;
} vec_2_8[2];
```

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f	
//	+	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
//		0		1		2		3		4		5		6		7	
//	+	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
//	0x10		a		a		a		a		X		X		c		c
//	0x20		a		a		a		a		X		X		c		c
//	+	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	



THE CURIOUS CASE OF DIAB STRUCT PACK - (4, 8, *)

```
#pragma pack(max_member_align, min_struct_align, byte-swap)
```

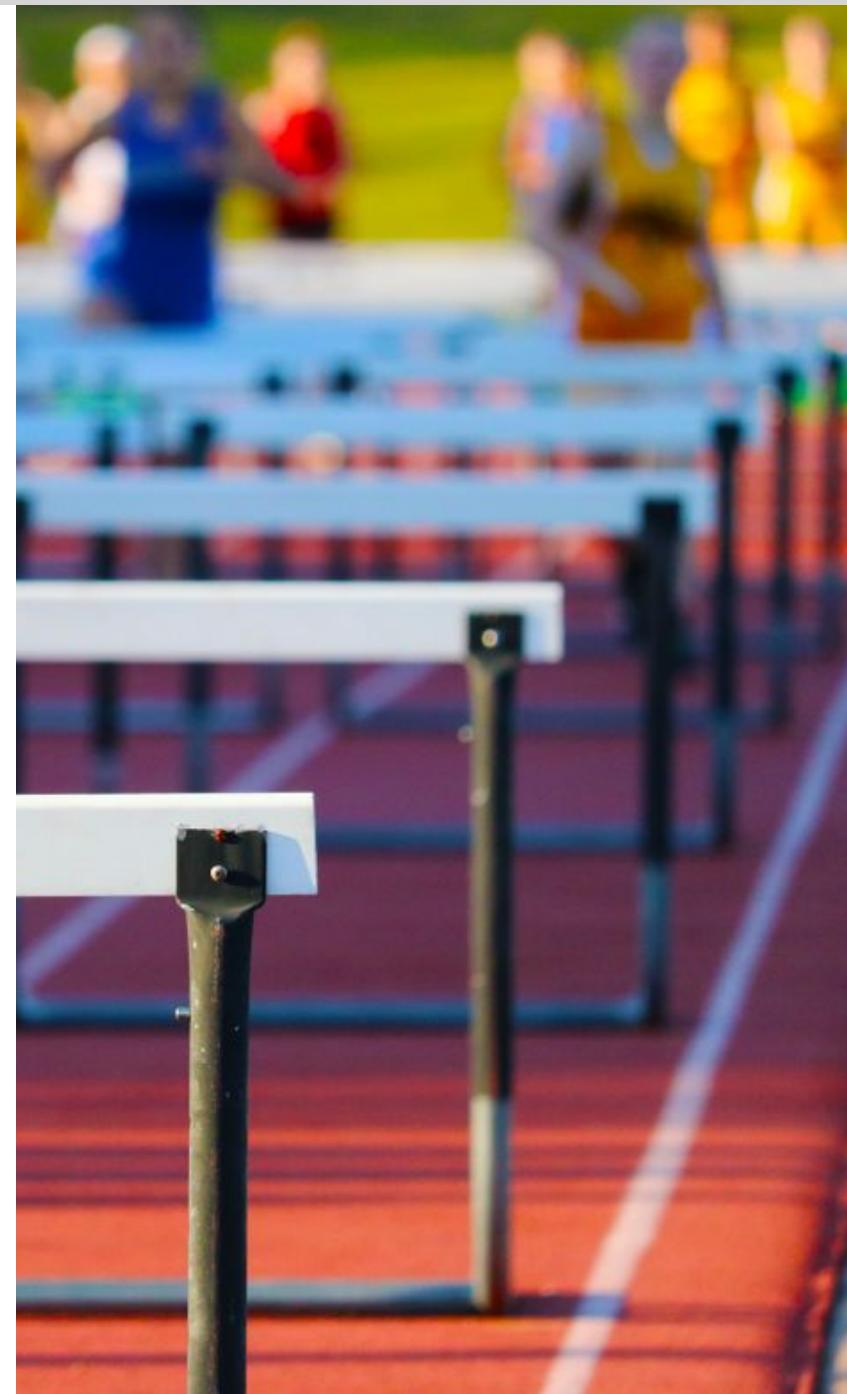
```
#pragma pack(4, 8[, byte-swap])
struct aXc_pack_maxmember4_minstruct8 {
    uint32_t a;
    uint16_t X;
    uint32_t c;
} vec_4_8[2];
```

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
//	0x10	a	a	a	a	X	X	.	.	c	c	c	c	.	.	.
//	0x20	a	a	a	a	X	X	.	.	c	c	c	c	.	.	.



CAN WE RUST?

- 1 AUTOSAR STANDARD**
- 2 C, ASM & RUST INTEGRATION**
- 3 KNOWN (RUST) PROBLEMS**
- 4 ENGINEERS & MANAGERS**
- 5 LEGACY SOFTWARE**



RUST IS PERFECT!!!11



RUST GUARANTEES DATA-RACE FREE CODE & THREAD SAFE/MULTI-CORE SCALABILITY¹



EVEN FOR μ C^{2 3}

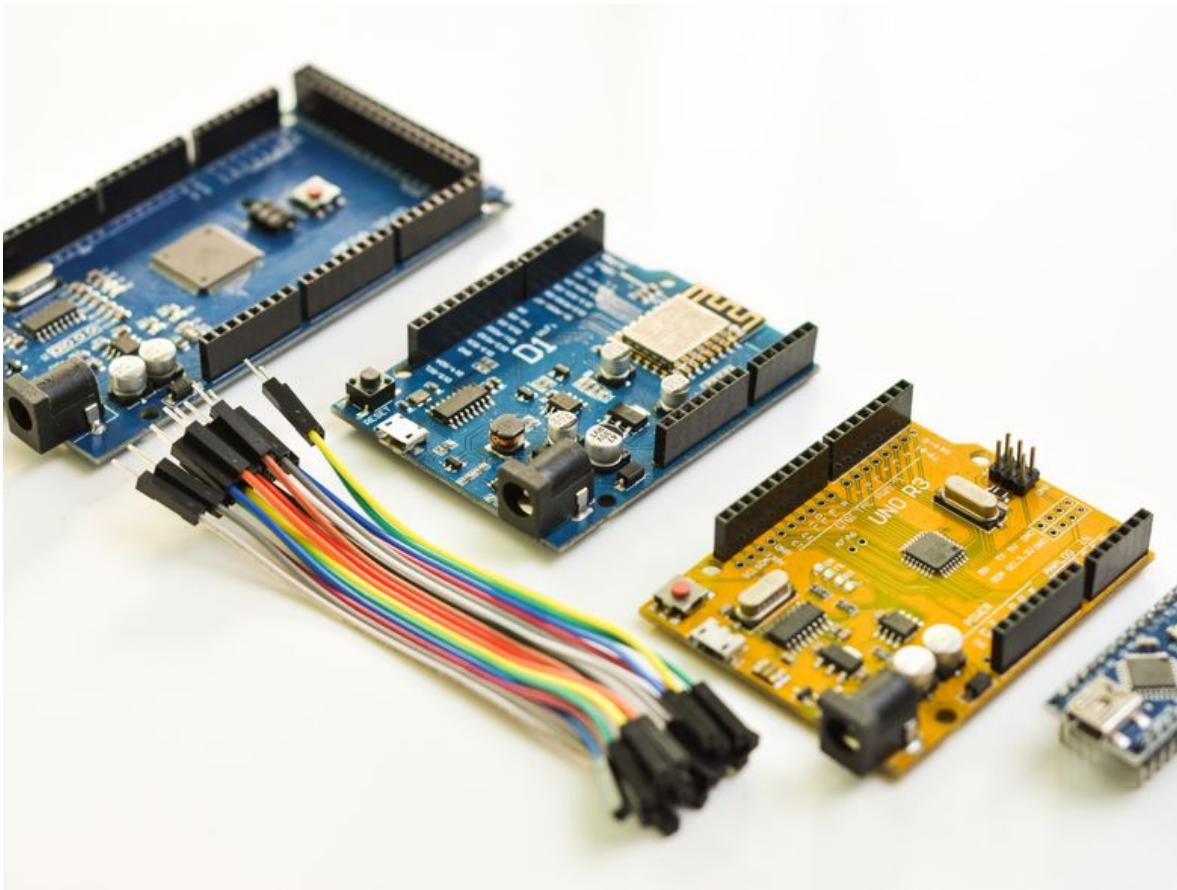
RUST GUARANTEES DATA-RACE FREE CODE & THREAD SAFE/MULTI-CORE SCALABILITY¹



EVEN FOR μC ^{2 3}

¹ with an OS/std, since that provides `Arc<>`, `Mutex<>` etc.

RUST GUARANTEES DATA-RACE FREE CODE & THREAD SAFE/MULTI-CORE SCALABILITY¹

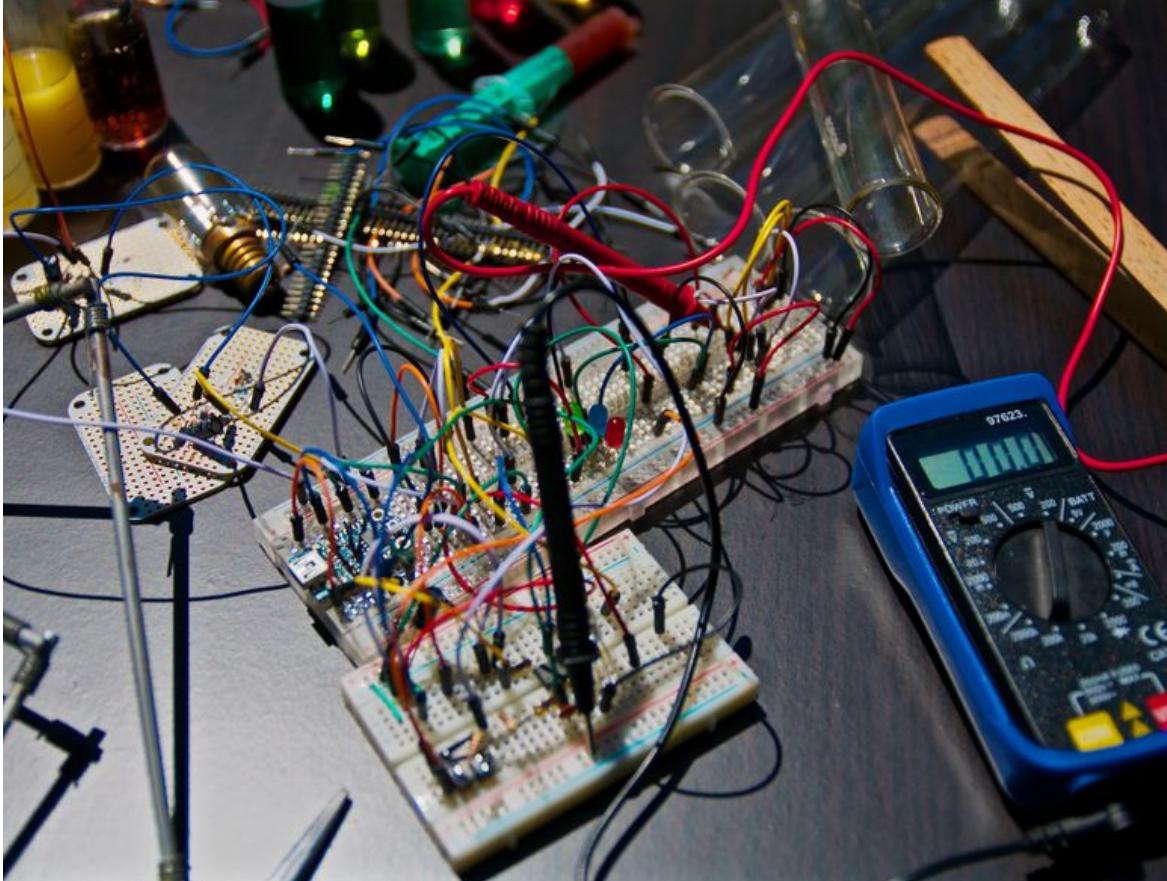


EVEN FOR µC^{2 3}

¹ with an OS/std, since that provides `Arc<>`, `Mutex<>` etc.

² if the µC is single core

RUST GUARANTEES DATA-RACE FREE CODE & THREAD SAFE/MULTI-CORE SCALABILITY¹



EVEN FOR μ C^{2 3}

¹ with an OS/std, since those provide `Arc<>`, `Mutex<>` etc.

² if the μ C is single core

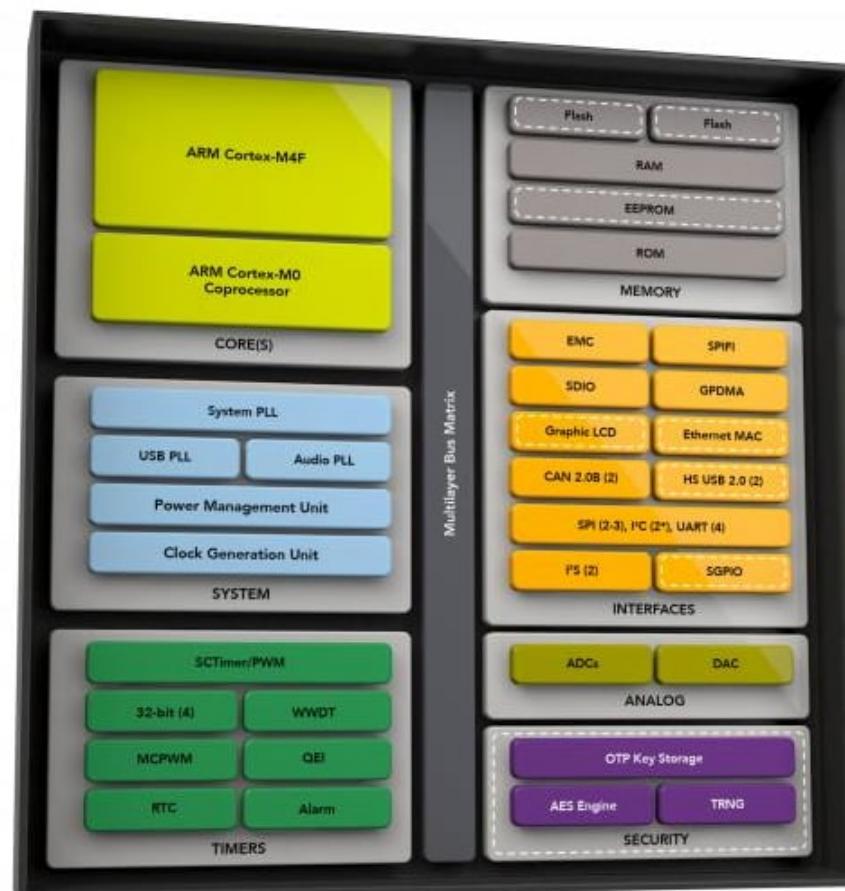
³ if you use a DSL (cortex-m-rt macros)

UNCOVERED USE CASES



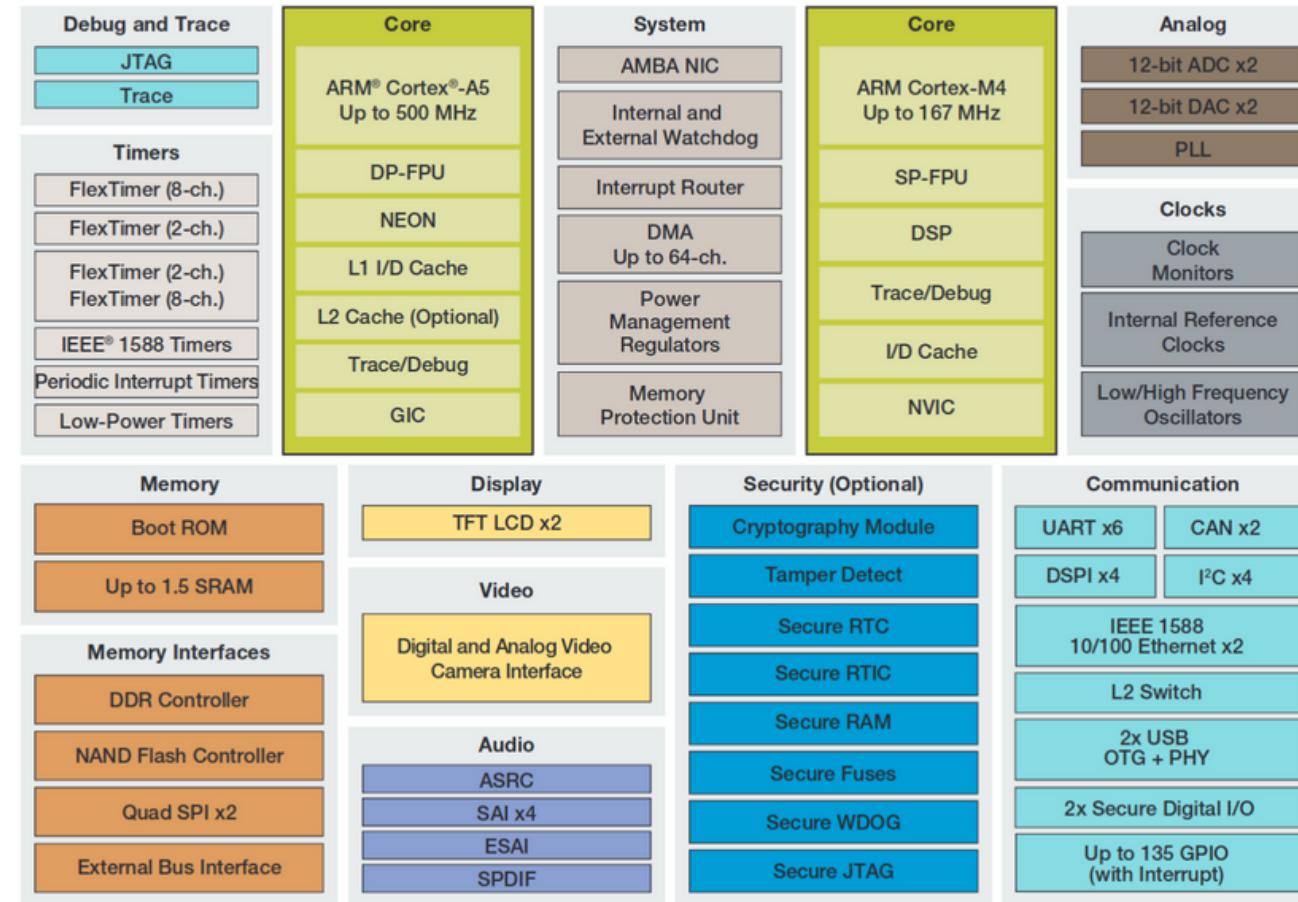
MULTICORE - LPC34XX: M4 + M0

Cortex-M4 based microcontrollers which include a Cortex-M0 coprocessor

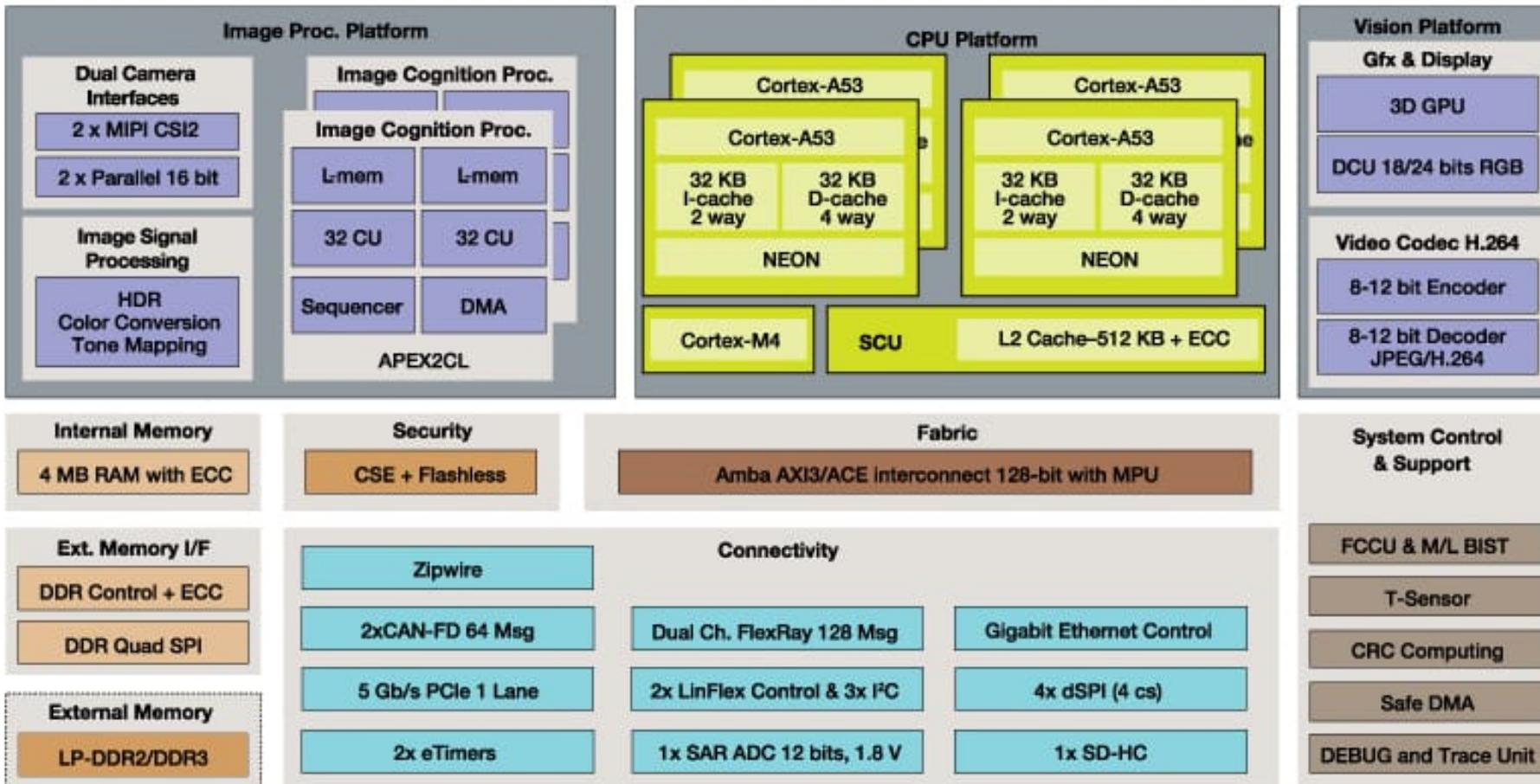


MULTICORE - VF6XX: A5 + M4

Vybrid VF6xx Block Diagram



MULTICORE - S32V234: 4xA53 + M4



**IT WILL WORK...
“ALL THE TIME,
80% OF THE TIME”**

`#[repr(packed), #[repr(align(N))]`



<https://doc.rust-lang.org/reference/type-layout.html>

#[REPRC(PACKED)] == #PRAGMA PACK(1)



#REPR[ALIGN(N)]



WHAT'S MISSING

between #[repr(packed)] and #[repr(align(N))]



SHORT ENUM

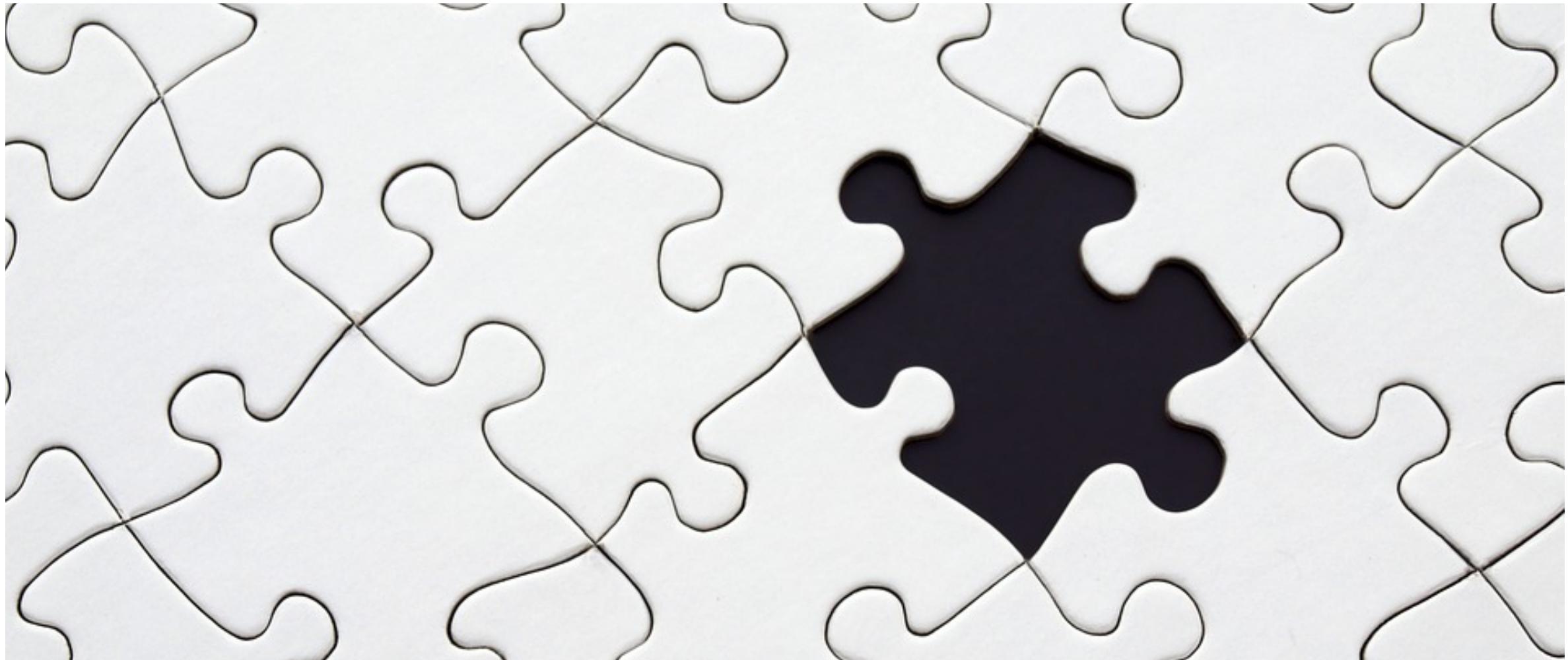


PACK != 1



**PACK &&
ALIGN**

EXTERN “C” STATIC(?) VARIABLE?



“#DEFINE” != “CONST”

```
#define MY_CONST 1234U
```

```
const MY_CONST:u32 = 1234u32;
```

MEMMAP.H MULTIPLE INCLUSION

https://www.autosar.org/fileadmin/user_upload/standards/classic/4-0/AUTOSAR_SWS_MemoryMapping.pdf



Specification of Memory Mapping
V1.4.0
R4.0 Rev 3

```
#define EEP_START_SEC_VAR_INIT_16
#include "MemMap.h"
static uint16 EepTimer = 100;
static uint16 EepRemainingBytes = 16;
#define EEP_STOP_SEC_VAR_INIT_16
#include "MemMap.h"
```

NIGHTLY - “DAMNIT EVERYTHING REQUIRES NIGHTLY D:”

#rust-embedded: jamesmunns: [...]some stuff [...] is preventing me from making the BBQueue::new() function const. I think it should be possible on nightly, but not possible on stable for now

durka42: [...] you can run `rustc --pretty=expanded` on nightly to see the code that is generated

21:18 SpaceManiac darkstalker: yes, with nightly feature 'specialization'

21:19 darkstalker damnit everything requires nightly D:

21:20 SpaceManiac eddyp: the other reason is that (IIRC) traits being known to have fields is a nightly feature

21:21 eddyp may I quote darkstalker?

21:26 darkstalker yes it's true. all the cool things require nightly

CAN WE RUST?

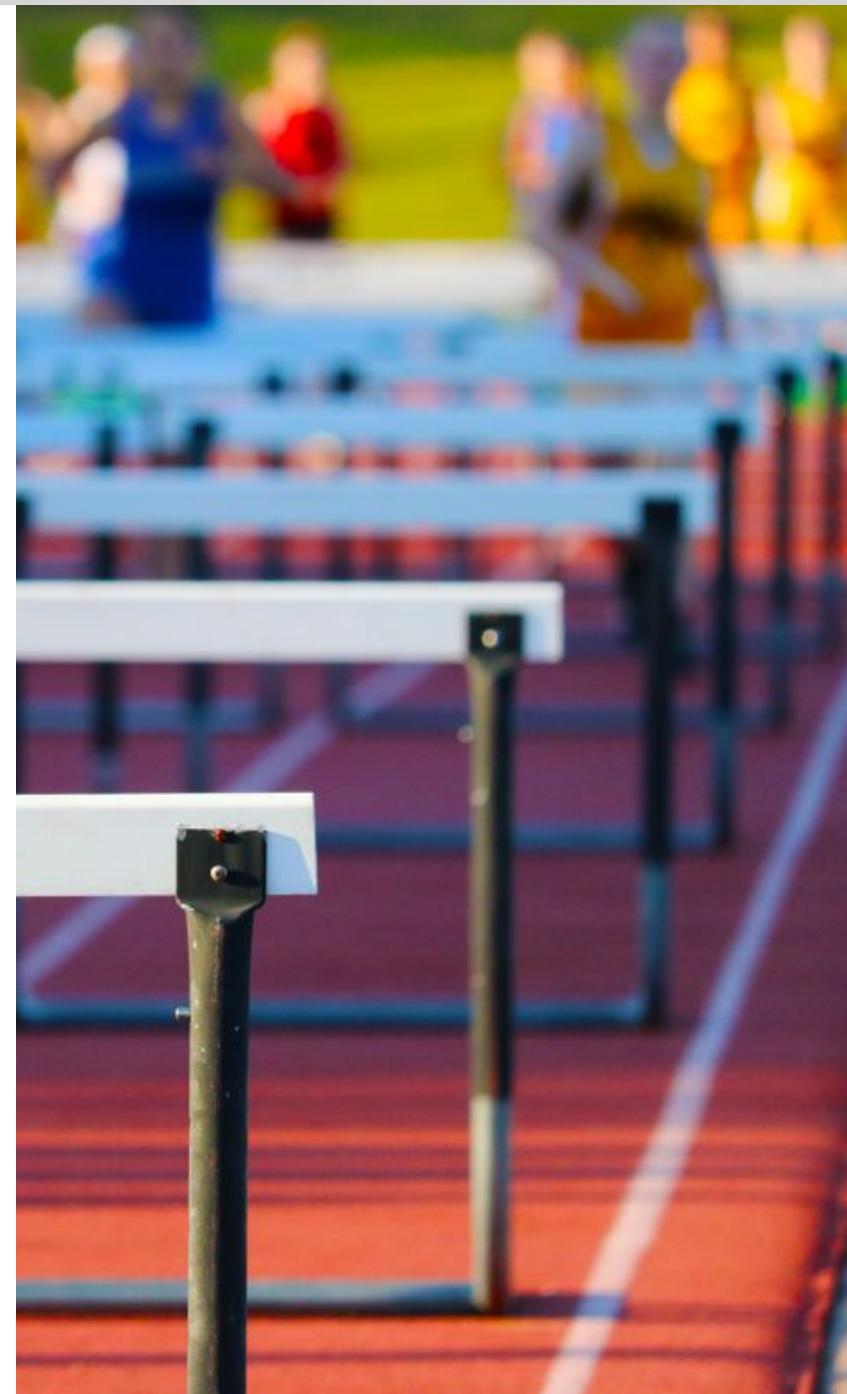
1 AUTOSAR STANDARD

2 C, ASM & RUST INTEGRATION

3 KNOWN (RUST) PROBLEMS

4 ENGINEERS & MANAGERS

5 LEGACY SOFTWARE



ENGINEERS, MUCH LIKE OLD DOGS



NEW TRICKS ARE HARD

HELLO, EMBEDDED RUST WORLD!

```
fn main() {
    println!("Hello, world!");
}
```

The screenshot shows a terminal window with four tabs, each displaying code or configuration:

- main.rs**: The source code for the main application.
- config**: The build configuration file.
- link.ld**: The linker script.
- Output**: The terminal output showing the build process.

main.rs content:

```
fn main() {
    println!("Hello, world!");
}
```

config content:

```
[target.thumbv7m-none-eabi]
rustflags = ["-C", "link-arg=-Tlink.ld"]

[build]
target = "thumbv7m-none-eabi"
```

link.ld content:

```
MEMORY
{
    FLASH : ORIGIN = 0x00000000, LENGTH = 256K
    RAM : ORIGIN = 0x20000000, LENGTH = 64K
}

/* The entry point is the reset handler */
ENTRY(Reset);

EXTERN(RESET_VECTOR);

SECTIONS
{
    .vector_table ORIGIN(FLASH) :
    {
        /* First entry: initial Stack Pointer value */
        LONG(ORIGIN(RAM) + LENGTH(RAM));
        /* Second entry: reset vector */
        KEEP(*(.vector_table.reset_vector));
    } > FLASH

    .text :
    {
        *(.text .text.*);
    } > FLASH

    /DISCARD/ :
    {
        *(.ARM.exidx.*);
    }
}
```

Output (terminal log):

```
rustc --target thumbv7m-none-eabi -C link-arg=-Tlink.ld main.rs
[...]
```

HELLO, EMBEDDED RUST WORLD!

```
fn main() {
    println!("Hello, world!");
}
```

The screenshot shows a terminal window with four tabs: `main.rs`, `config`, `main.rs`, and `link.ld`. The first tab contains the simple `Hello, world!` program. The second tab contains the build configuration, specifying the target as `thumbv7m-none-eabi` and rustflags as `[-C, "link-arg=-Tlink.ld"]`. The third tab shows the generated assembly code for the main function, which includes a panic info block and an infinite loop. The fourth tab contains the linker script (`link.ld`) which defines memory regions for FLASH and RAM, sets the entry point to the reset handler, and specifies sections for the vector table and text.

```
main.rs
fn main() {
    println!("Hello, world!");
}

main.rs
#![no_main]
#![no_std]

use core::panic::PanicInfo;

// The reset handler
#[no_mangle]
pub unsafe extern "C" fn Reset() -> ! {
    let _x = 42;
    // can't return so we go into an infinite loop here
    loop {}
}

// The reset vector, a pointer into the reset handler
#[link_section = ".vector_table.reset_vector"]
#[no_mangle]
pub static RESET_VECTOR: unsafe extern "C" fn() -> ! = Reset;

#[panic_handler]
fn panic(_panic: &PanicInfo<'_>) -> ! {
    loop {}
}

config
[target.thumbv7m-none-eabi]
rustflags = ["-C", "link-arg=-Tlink.ld"]

[build]
target = "thumbv7m-none-eabi"

link.ld
MEMORY
{
    FLASH : ORIGIN = 0x00000000, LENGTH = 256K
    RAM : ORIGIN = 0x20000000, LENGTH = 64K
}

/* The entry point is the reset handler */
ENTRY(Reset);

EXTERN(RESET_VECTOR);

SECTIONS
{
    .vector_table ORIGIN(FLASH) :
    {
        /* First entry: initial Stack Pointer value */
        LONG(ORIGIN(RAM) + LENGTH(RAM));
        /* Second entry: reset vector */
        KEEP(*(.vector_table.reset_vector));
    } > FLASH

    .text :
    {
        *(.text .text.*);
    } > FLASH

    /DISCARD/ :
    {
        *(.ARM.exidx.*);
    }
}
```

MANAGERS



INDUSTRY/MANAGEMENT: ADVERSE TO MONOCULTURE (OR OSS)

(unless is theirs)



CAN WE RUST?

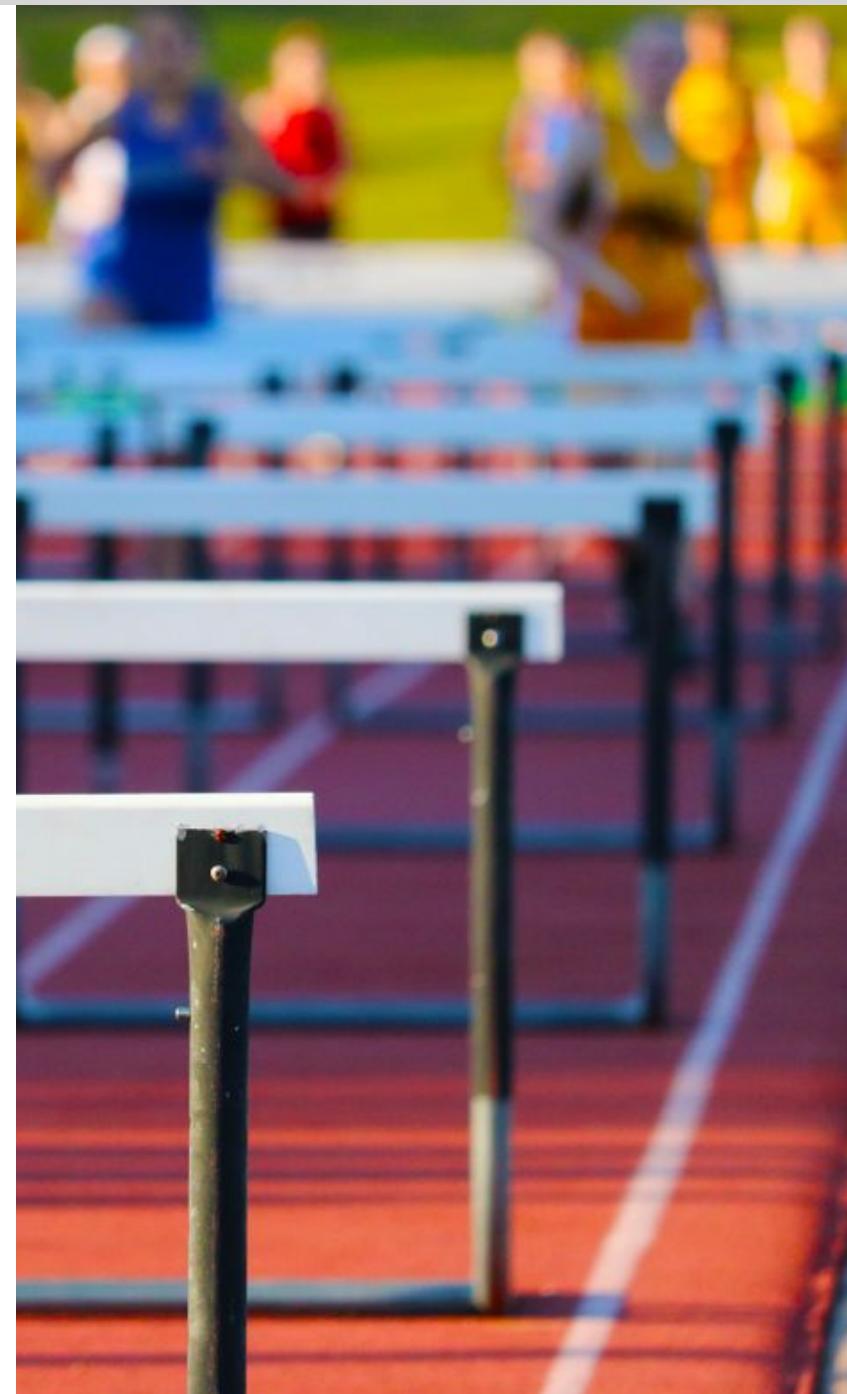
1 AUTOSAR STANDARD

2 C, ASM & RUST INTEGRATION

3 KNOWN (RUST) PROBLEMS

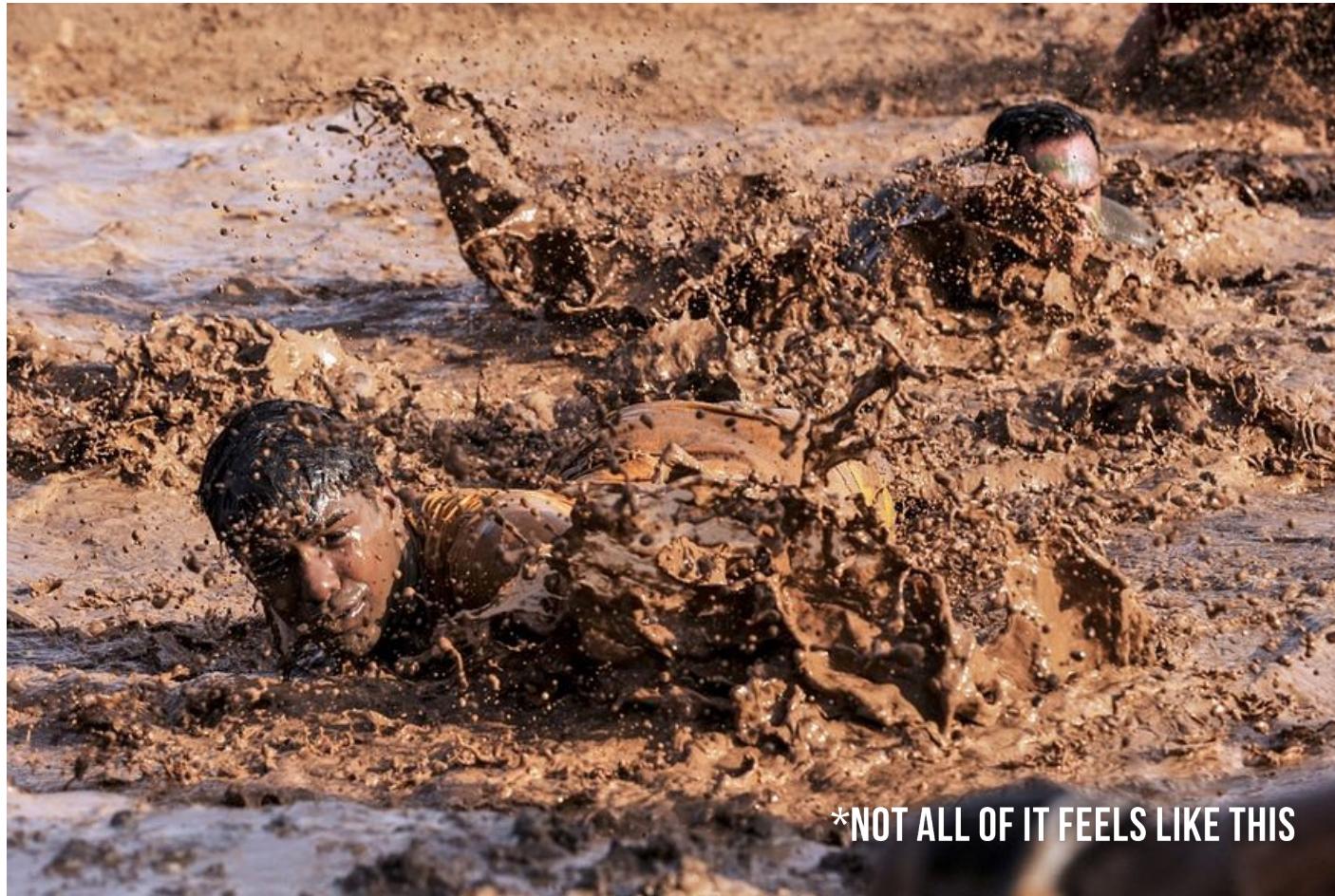
4 ENGINEERS & MANAGERS

5 LEGACY SOFTWARE



LEGACY SOFTWARE

Sometimes you feel like taking a shower after touching it...

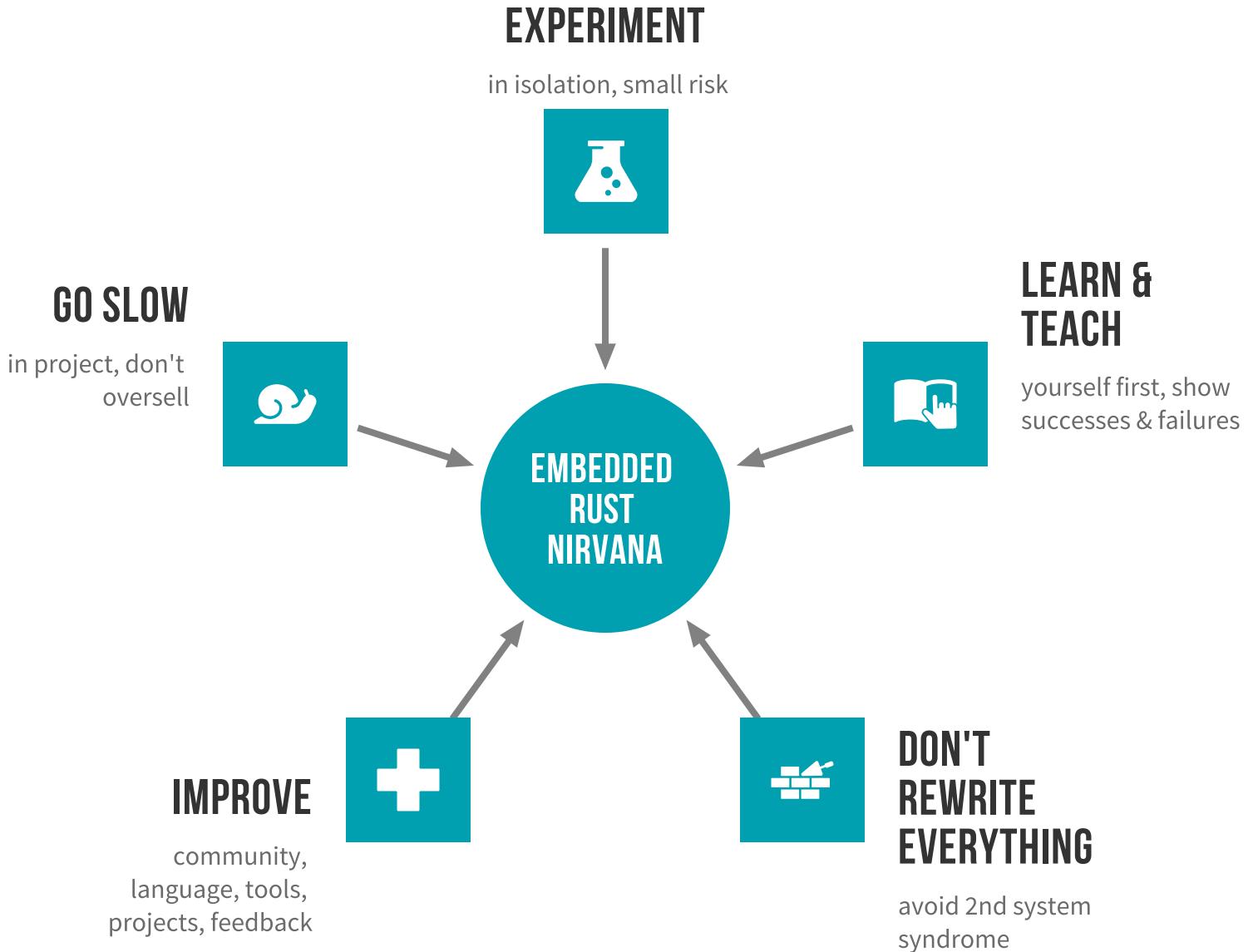


*NOT ALL OF IT FEELS LIKE THIS



CONCLUSIONS

WHAT CAN YOU DO?



WHAT CAN THE (EMBEDDED) RUST COMMUNITY DO?





STABILIZATION (BEYOND CORTEX M)

ASM INTERFACE - NOT JUST ARM

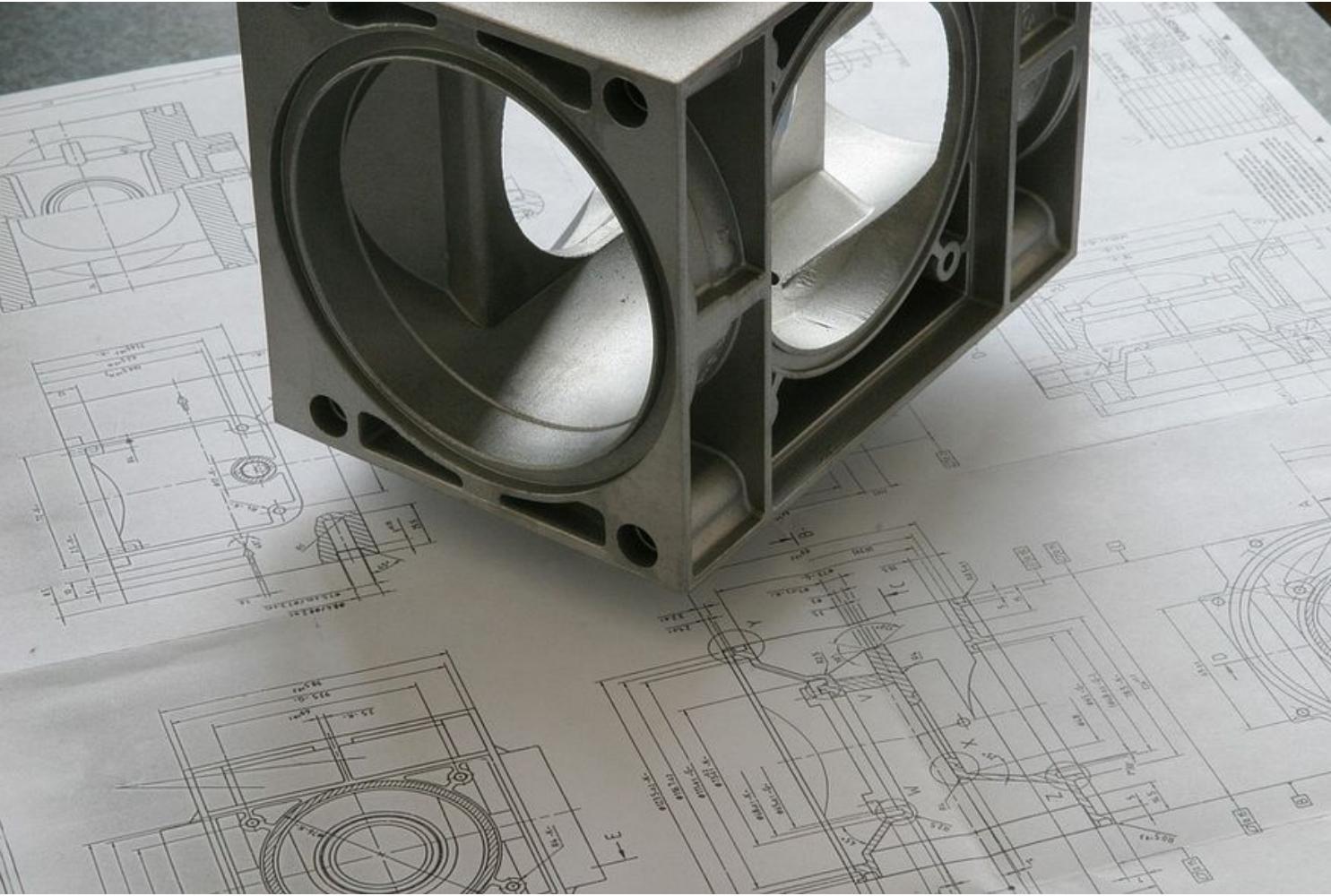
SAMPLES

HOPEFULLY, LESS CREEPY THAN THESE >>>



Provide more std-like examples: Sema4<> mutex?

NO FORMAL SPEC



WE NEED ALTERNATIVE RUSTC IMPLEMENTATIONS

Mutabah's Rust Compiler - <https://github.com/thepowersgang/mrustc> - might be an easier entry



