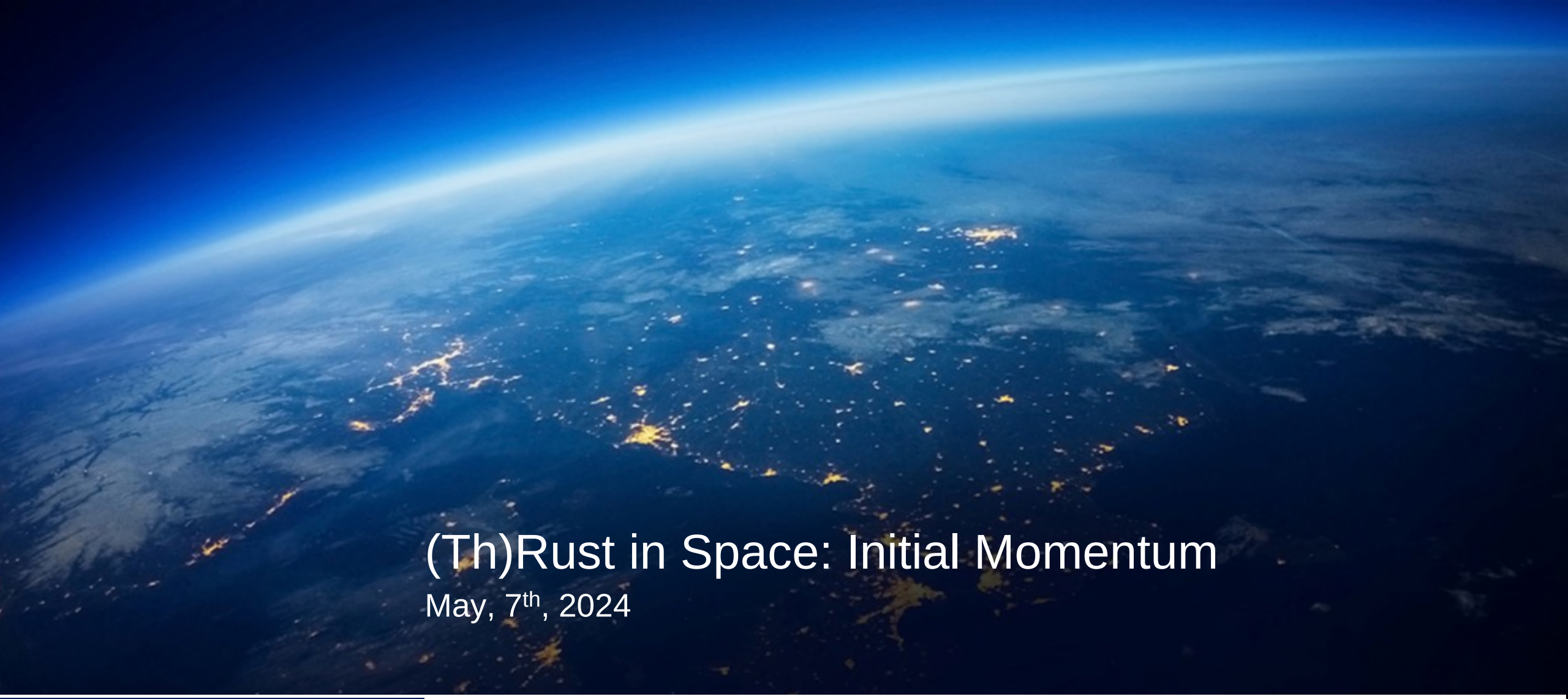


(Th)Rust in Space: Initial Momentum

May, 7th, 2024

DEFENCE AND SPACE

See Page 2 for Export Control Information concerning this document's content

AIRBUS

DEFENCE AND SPACE

Export Control Information

Section 1 **(not applicable in France, please go to section 3)**

This document contains Technical Information :

Yes ☐ **No** ☐

If No to section1: please complete Section 2

If Yes to section1: please complete Section 3 as applicable

Section 2 **(not applicable in France, please go to section 3)**

I confirm the document does not contain Technical Information and is « **Not-Technical** »
Name:
Date:

Section 3

3a. National and EU regulations Export Control Assessment

This document has been assessed against applicable export control regulations in

☑ France ☐ Germany ☐ Spain ☐ UK ☐ Other: [Specify the country]

☑ and does not contains Controlled Technology[1] and is therefore **« Not Listed / Not Controlled »**

☐ and contains Controlled Technology with export control classification [Insert classification number, e.g ML22x, xExxx, AMAx]

Note: Any transfer of this document in part or in whole must be made in accordance with the appropriate export control regulations. Prior to any transfer outside of the responsible legal entity, confirmation of an applicable export licence or authorisation must be obtained from the local Export Control Officer (ECO).

3b. US (ITAR/EAR) Export Control Assessment

☑ This document does not contains US origin Technical Data (Technology)

☐ This document contains « Technology » which is controlled by the U.S government under [USML category number / ECCN] and which has been received by [Legal entity] under the authority of [Licence number / ITAR exemption / EAR licence exception / NLR]

☐ This document contains technology which is designated as EAR99 (subject to EAR and not listed on the USML/CCL.)

Note: Any re-export or re-transfer of this document in part or in whole must be made in accordance with the appropriate regulation (ITAR or EAR) and applicable authorization. If in any doubt please contact your local ECO.

3c. Technical Rater Information

This document has been assessed by the following Technical Rater :
Assessed and classified by:
Date classification completed:

[1] "Controlled Technology" is defined as any Information necessary for the design, development, production, use, operation, maintenance or repair of export controlled goods. Examples of such Information are blueprints, plans, diagrams, models, engineering designs, manuals, requirements specifications and instructions etc. If in any doubt please contact your local Export Control Officer (ECO)

AIRBUS

About Me

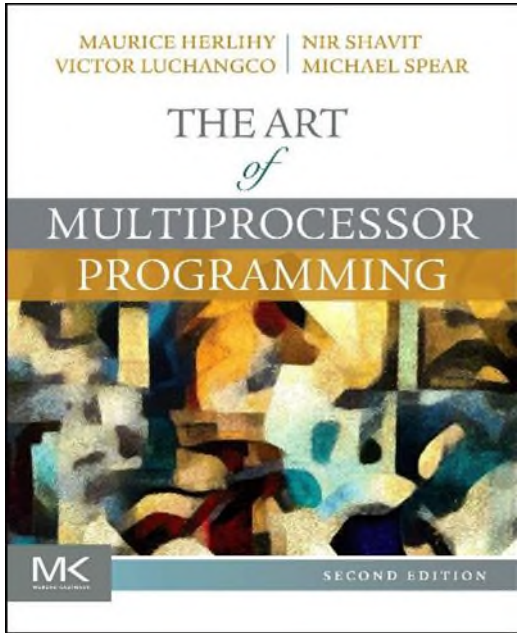
Michaël Melchiore
Digital Payload On-Board SW Specialist
Airbus Defense & Space @Toulouse, France
michael.melchiore@airbus.com

Data-intensive real-time SW for more than 15 years

- Focus on lower layers (middleware, framework)
- Worked in various industries (Railway, Radar and Space)
- Your team (reluctant) “Multicore *Artist*”

Discovered Rust in 2014, tried to bring it to work ever since…

- Succeeded in 2022 (!)



AIRBUS

Satellite Design Constraints

Space is an **hostile** operating environment

- Radiations, cosmic particles
- Electromagnetic fields
- Temperature gradients

Space is **expensive** to reach

- Size, Weight and Power (SWaP)
- Small, light and energy efficient

Specialized technologies are required, which started very small

- Small memories (a few KB)
- Very slow processors (20x slower)
- Slow communication links (a few Kb/s)

AIRBUS

Role of the On-Board SW

Control, regulate, monitor

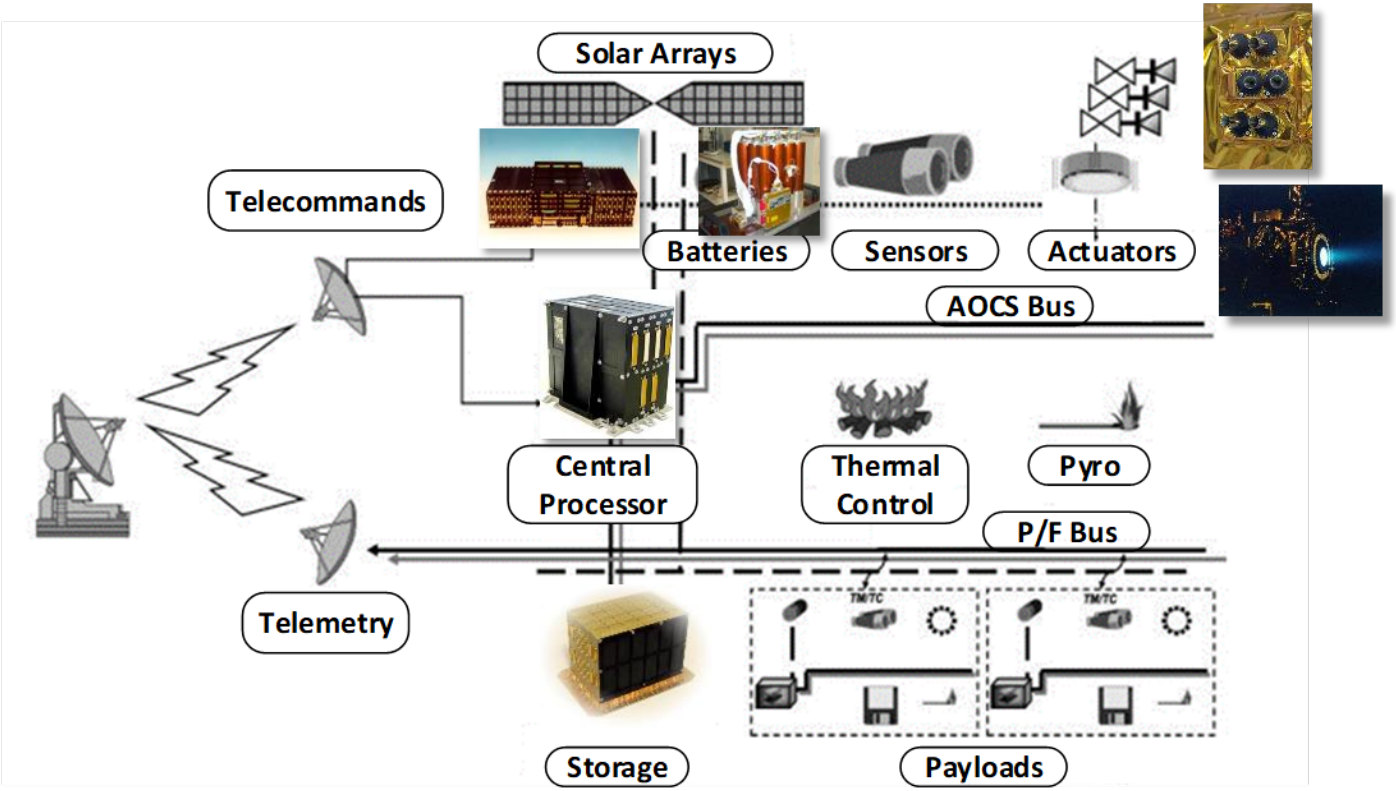
- Power/Temperature
- Orientation/Orbit

Manage the mission

- Manage equipment
- Take pictures (for example) & store data
- Downlink data to ground stations

Interact with operators

- Execute commands
- Provide reports



AIRBUS

On-Board SW Stakes

Real-time embedded SW developed in **C**

- **Critical** functions are executed cyclically (thermal@0.1Hz, attitude@10Hz)
- **Always** meet deadlines in every situation
- Fit into **limited execution resources** (memory, processing)

Control of **specialized/complex** HW

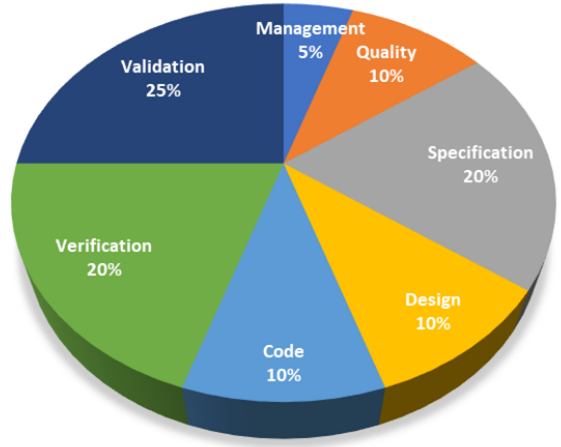
- Satisfy operating modes in every case, across all possible configurations
- Flexible configuration to adapt to failures during lifetime

Outstanding Non Func./Quality constraints to be met

- Strong reliance on **manual code reviews**
- **Recurring**, cost & time investment (3x feature impl. cost)

How to scale our workflow to tackle modern challenges?

Typical OBSW Cost Structure



ECSS	**E**uropean **C**ooperation for **S**pace **S**tandardization
MISRA	**M**otor **I**ndustry **S**oftware **R**eliability **A**ssociation

(Th)Rusting Forward

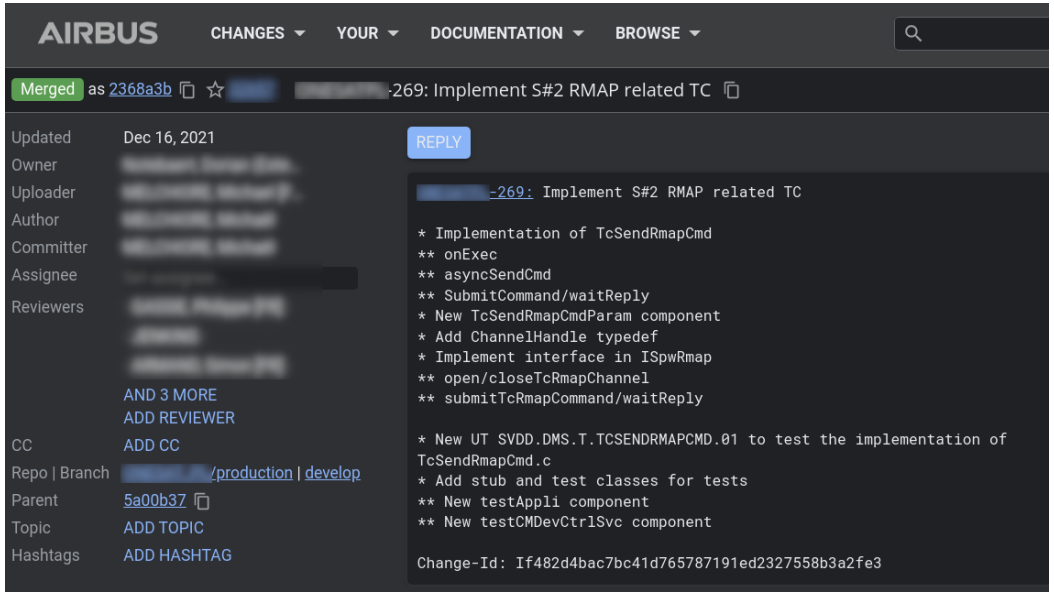
Study “Using game engine techniques and Rust to modernize On Board software” (OXYDE)

ESA Contract No.140066/22/NL/GLC/ov

Port telecom payload to Rust, using ECS design principle

Methodology

- Build factual evidence of potential migration trade-offs
- Leverage existing code review database
- Focus on business-specific functions, instead of rewriting everything from scratch, in Rust



SW Mitigation of HW Resets

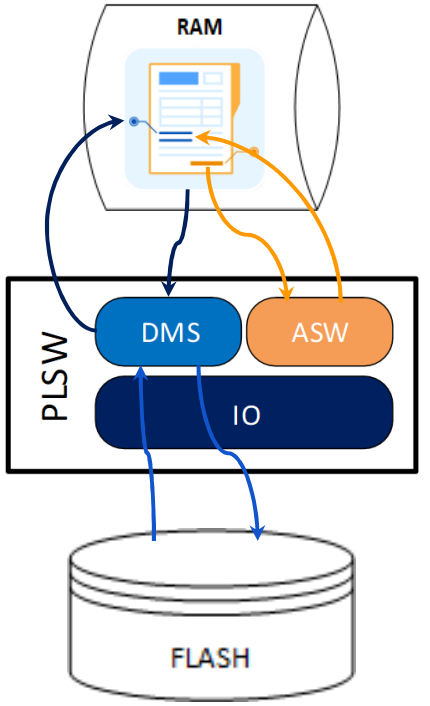
Minimize impact of SW/HW resets on mission

- Identify vital execution context for critical business functions (≈500 MB)
- Periodically snapshot context into reliable storage
- Automatically restore latest archive at reboot

Key non-functional requirements

- Prevent data corruption due to concurrent context updates during backups
- Minimize Flash device power on cycles to maximize device lifetime

Verification requires continuous, extensive testing and peer reviews



ASW	**A**pplication **S**oft**W**are
DMS	**D**ata **M**anagement **S**ystem
PLSW	**P**ay**L**oad **S**oft**W**are

AIRBUS

Wrapping a SpaceWire C library

SpaceWire standards

- Spacecraft Communication Network (ECSS-E-ST-50-12C)
- Family of associated protocols (ECSS-E-ST-50-[51/52/53]C)
  - Non-trivial, dynamic packet structure

C library exposing a single, public opaque frame type

- Promote zero-copy design, ease error management
- Manage metadata associated with buffer content
  - Field offsets, derived attributes
- Expose getter/setters to read or write frame content

CCSDS	**C**onsultative **C**ommittee for **S**pace **D**ata **S**ystems
CPTP	**C**CSDS **P**acket **T**ransfer **P**rotocol
RMAP	**R**emote **M**emory **A**ccess **P**rotocol

CPTP

First byte transmitted

	Target SpW Address	....	Target SpW Address
Target Logical Address	Protocol Identifier	Reserved = 0x00	User Application
CCSDS Packet (First Byte)	CCSDS Packet	CCSDS Packet	CCSDS Packet
CCSDS Packet	...	...	CCSDS Packet
CCSDS Packet	CCSDS Packet (Last Byte)	EOP	

Last byte transmitted

RMAP Read Command

First byte transmitted

	Target SpW Address	....	Target SpW Address
Target Logical Address	Protocol Identifier	Instruction	Key
Reply Address	Reply Address	Reply Address	Reply Address
Reply Address	Reply Address	Reply Address	Reply Address
Reply Address	Reply Address	Reply Address	Reply Address
Initiator Logical Address	Transaction Identifier (MS)	Transaction Identifier (LS)	Extended Address
Address (MS)	Address	Address	Address (LS)
Data Length (MS)	Data Length	Data Length (LS)	Header CRC
EOP			

Last byte transmitted

Rust Binding V1

- Raw *_sys bindings (automatically) generated with bindgen
- Safe Rust binding to tackle code review feedbacks
  - (Truly) encapsulate C struct internals
  - Mandatory, ergonomic error checking
  - Enforced frame lifecycle with Builder pattern

spwapi.h

```c
// Provide storage for frame data
void frame_init(struct spw_frame * self,
                uint8_t * buffer,
                uint32_t buffer_size)

// Access internal buffer
uint8_t* frame_buffer(struct spw_frame const * self);

// Compute frame metadata
int frame_parse(struct spw_frame * frame, uint32_t size);

// Getters / setters
uint8_t * frame_cargo(struct spw_frame const * self,
                      uint32_t * cargo_length,
                      uint8_t ** crc);
```

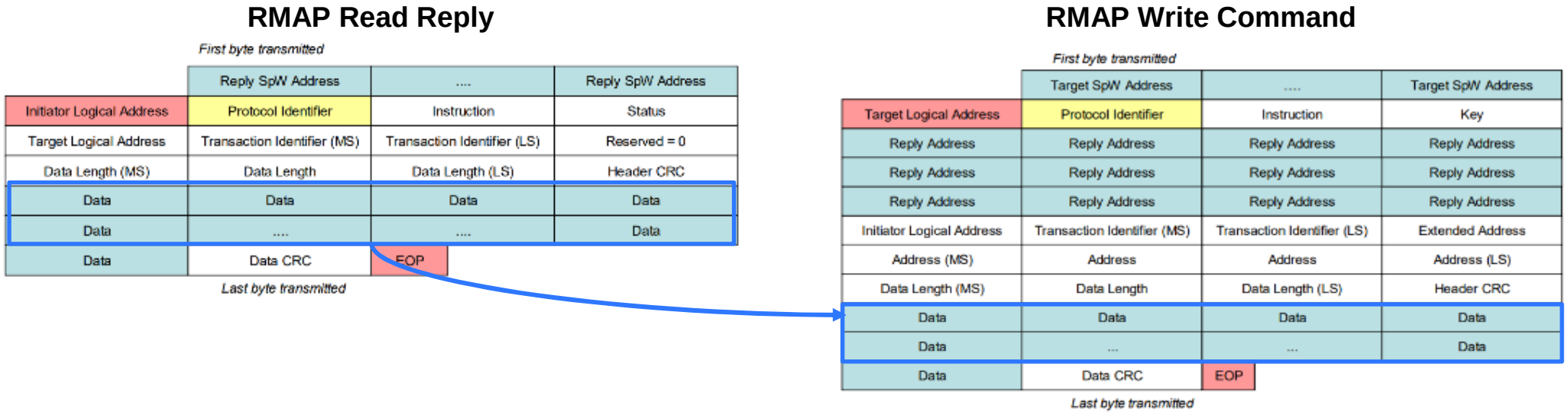
spwapi/src/lib.rs

```rust
use spwapi_sys::*;

pub struct FrameBuilder<'a> {
    pub(crate) _buffer: Buffer<'a>,
    pub(crate) frame: spw_frame,
}

impl<'a> FrameBuilder<'a> {
    pub fn with_buffer(mut buffer: Vec<u8>) -> Self {}
    pub fn with_slice(slice: &'a mut [u8]) -> Self {}

    // Internal buffer access
    pub fn buffer(&self) -> &[u8] {}
    pub fn buffer_mut(&mut self) -> &mut [u8] {}

    // Parse buffer content to compute metatadata
    pub fn parse(self, size: u32) -> Result<Frame, …> {}
}

pub struct Frame(pub(crate) FrameBuilder);

impl Frame {
    pub fn cargo(&self) -> Option<&[u8]>
    pub fn cargo_mut(&mut self) -> Option<&mut [u8]>
}
```

Preventing Frame Corruptions

A function can simultaneously handle hundreds of frames, up to 4 KB each

- Incrementally prepare a RMAP Write Command based on previous RMAP Read Reply cargo
- Share the buffer across related frames

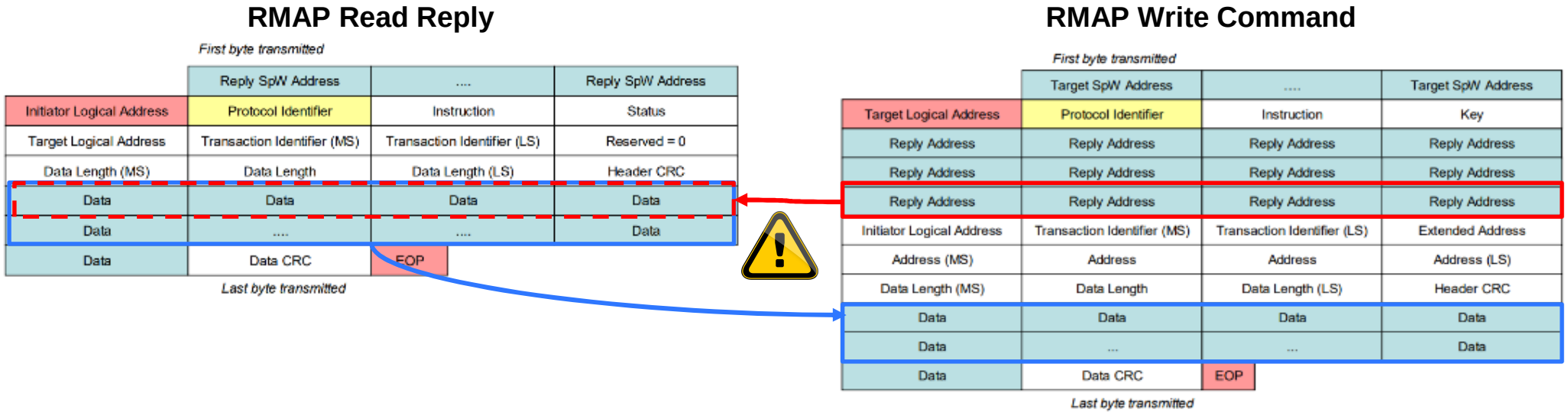


AIRBUS

Preventing Frame Corruptions

A function can simultaneously handle hundreds of frames, up to 4 KB each

- Incrementally prepare a RMAP Write Command based on previous RMAP Read Reply cargo
- Share the buffer across related frames



AIRBUS

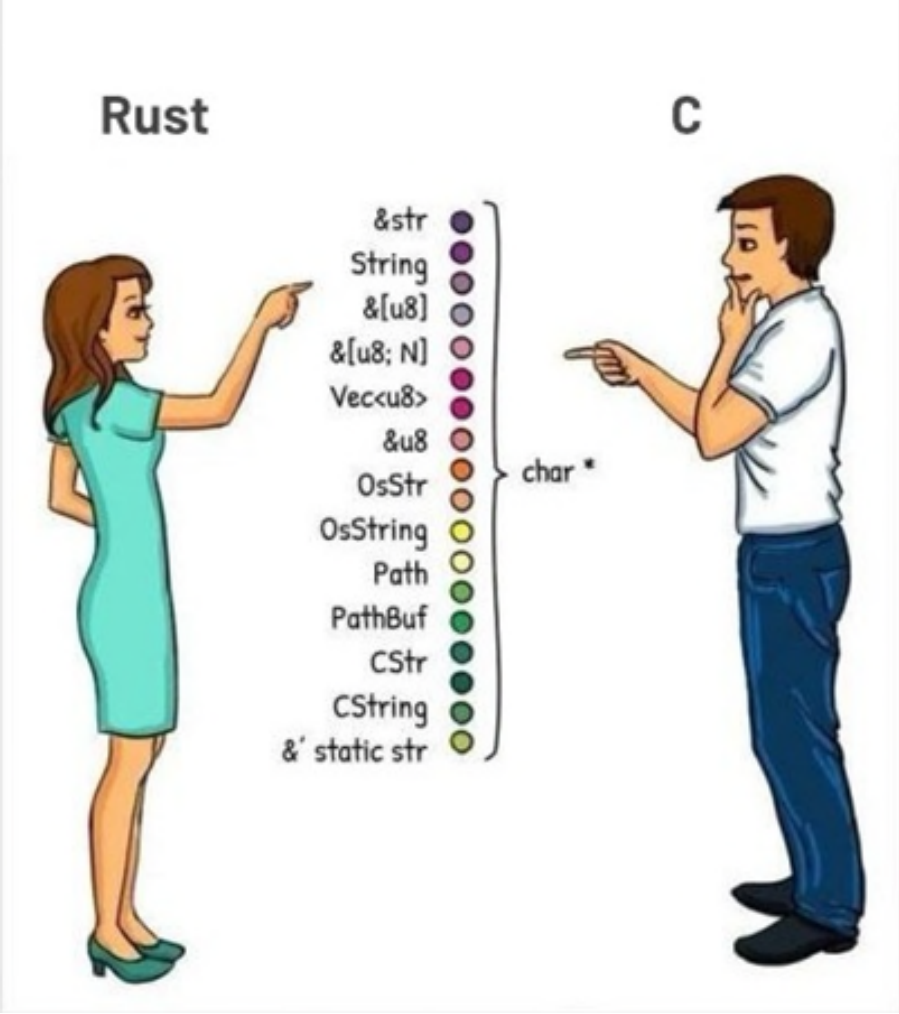
Fearless Optimization with Rust

In Rust, owned and shared buffers are different types…

… so we can build different API to handle the consequences

- Help users AND developers

```rust
pub(crate) enum Buffer<'a> {
    Slice(&'a mut [u8]),
    Vec(Vec<u8>),
}

pub struct FrameBuilder<'a> {
    pub(crate) _buffer: Buffer<'a>,
    /* … skipped … */
}

impl<'a> FrameBuilder<'a> {
    pub fn with_buffer(mut buffer: Vec<u8>) -> Self {}
    pub fn with_slice(slice: &'a mut [u8]) -> Self {}
}

pub struct Frame(pub(crate) FrameBuilder);

impl Frame {
    pub fn cargo(&self) -> Option<&[u8]> -> {}
    pub fn cargo_mut(&mut self) -> Option<&mut [u8]> -> {}

    // Get back the builder to process another frame
    pub fn reuse(mut self) -> FrameBuilder<'a> {}
}
```



AIRBUS

Smart Flash Power Management (1/5)

```
pub struct FlashSession;
```

Step by step guide of the design process

1. Define an active Flash session type

Smart Flash Power Management (2/5)

Step by step guide of the design process

1. Define an active Flash session type
2. Restrict Flash accesses to active Sessions only

```rust
pub struct FlashSession;

impl FlashSession {
    pub fn send(&mut self, …)-> Result<…> {}
    pub fn receive(&mut self, …) -> Result<…> {}
}
```

Smart Flash Power Management (3/5)

Step by step guide of the design process

1. Define an active Flash session type
2. Restrict Flash accesses to active Sessions only
3. Define a Flash endpoint type to manage device sessions

```rust
pub struct FlashSession;

impl FlashSession {
    pub fn send(&mut self, …)-> Result<…> {}
    pub fn receive(&mut self, …) -> Result<…> {}
}

pub struct FlashEndpoint;
```

```rust
impl FlashEndpoint {
    pub fn new_session(&mut self, …) -> Result<FlashSession, …> {}
}
```

Smart Flash Power Management (4/5)

Step by step guide of the design process

1. Define an active Flash session type
2. Restrict Flash accesses to active Sessions only
3. Define a Flash endpoint type to manage device sessions
4. Encode session lifecycle events in type systems

```rust
pub struct FlashSession(Sender<FlashSessionEvent>);

impl FlashSession {
    pub fn send(&mut self, …)-> Result<…> {}
    pub fn receive(&mut self, …) -> Result<…> {}
}

pub struct FlashEndpoint {
    sender: Sender<FlashSessionEvent>,
}


impl FlashEndpoint {
    pub fn new_session(&mut self, …) -> Result<FlashSession, …> {
        /* … skipped … */
        self.sender.send(FlashSessionEvent::Created);
        /* … skipped … */
    }
}

pub enum FlashSessionEvent { Created, Finished }

impl Drop for FlashSession {
    fn drop(&mut self) {
        self.0.send(FlashSessionEvent::Finished);
}
```

Smart Flash Power Management (5/5)

Step by step guide of the design process

1. Define an active Flash session type
2. Restrict Flash accesses to active Sessions only
3. Define a Flash endpoint type to manage device sessions
4. Encode session lifecycle events in type systems
5. Process lifecycle events in a cyclic function to control power

```rust
pub struct FlashSession(Sender<FlashSessionEvent>);

impl FlashSession {
    pub fn send(&mut self, …)-> Result<…> {}
    pub fn receive(&mut self, …) -> Result<…> {}
}

pub struct FlashEndpoint {
    sender: Sender<FlashSessionEvent>,
    receiver: Receiver<FlashSessionEvent>,
}

impl FlashEndpoint {
    pub fn new_session(&mut self, …) -> Result<FlashSession, …> {
        /* … skipped … */
        self.sender.send(FlashSessionEvent::Created);
        /* … skipped … */
    }
}

pub enum FlashSessionEvent { Created, Finished }

impl Drop for FlashSession {
    fn drop(&mut self) {
        self.0.send(FlashSessionEvent::Finished);
}

pub fn control_flash_power(endpoint: &mut FlashEndpoint) {}
```

Exclusive Execution Context Access

```rust
#[derive(Resource)]
pub struct WorkingArea(Option<Data>);

pub struct ArchiveSession {
    data: Option<Data>,
    /* ... skipped ... */
}

#[derive(Resource)]
pub struct ArchiveManager{
    /* ... skipped ... */
}

impl ArchiveManager {
    pub new_session(&mut self, area: &mut WorkingArea)
      -> Option<AreaSession> {

        let data = area.data.take()?;
        /* ... skipped ... */
    }
}
```

```rust
pub struct AreaAvailable(Data);

impl Drop for ArchiveSession {
    fn drop(&mut self) {
        let data = self
            .data
            .take()
            .expect("Active session data is always Some");
        self.sender.send(AreaAvailable(data));
    }
}
```

```rust
pub fn unlock_working_area(
    mng: Res<ArchiveManager>,
    mut area: ResMut<WorkingArea>) {

    if let Ok(evt) = mng.0.try_recv() {
        area.data = Some(evt.0);
    }
}
```

Key takeaways

- Simple concepts compounds to solve traditionally complex issues
- Requirement verification burden transferred to the compiler!

Conclusions

Rust provides a promising framework for modern OBSW development

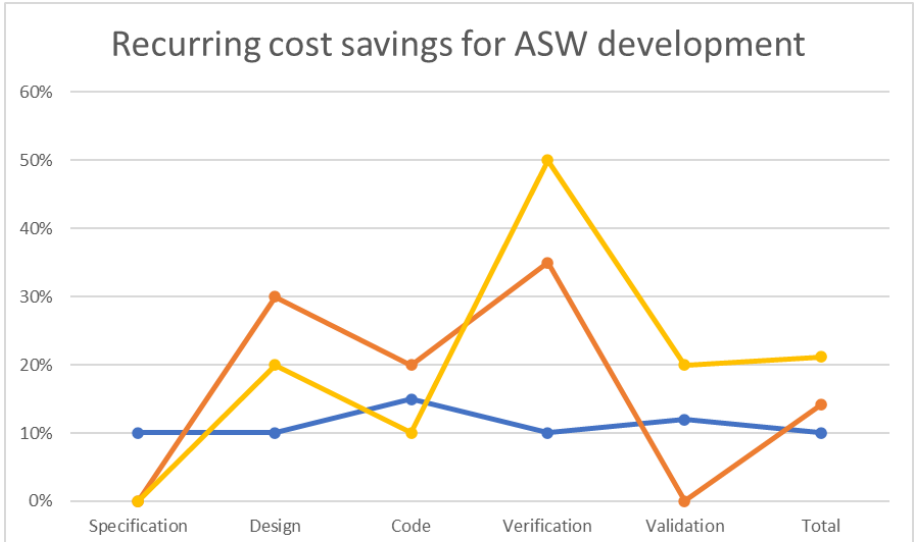
- Excellent tooled-up ecosystem focused on code quality and productivity
- Demonstrated ability to leverage existing C assets
- Expressive & flexible type system with unique semantics…
- …leveraged by everyone to build efficient & reliable SW

Upcoming activities

- Re-visit constellation customer feature requests
- Complete performance baseline

Currently identified enablers on which we want to collaborate

- Spaceworthy OS/HW support
- Ecosystem maturity growth



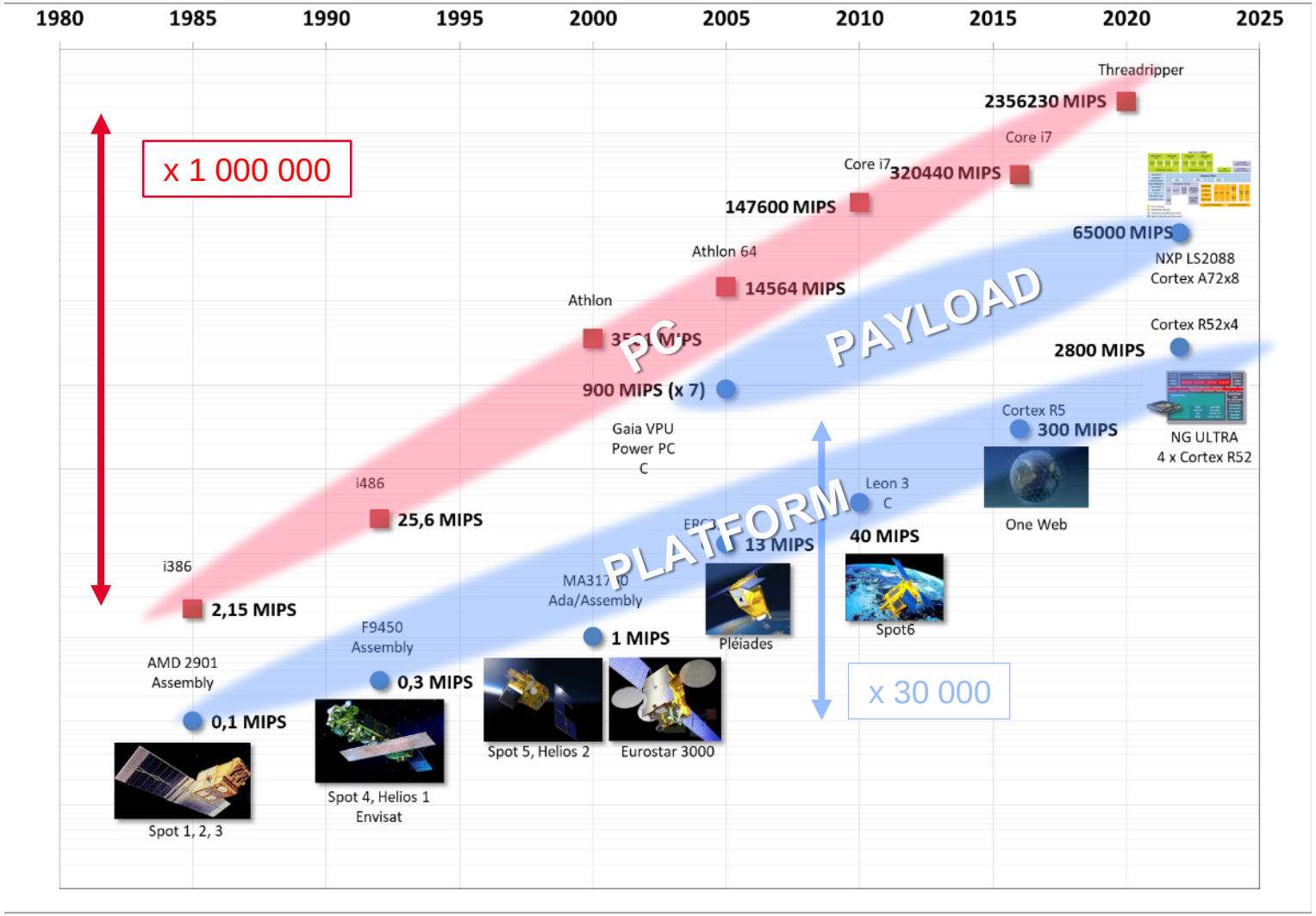
RTEMS™

ferrocene

AIRBUS

Thank you

A Brief Timelapse about Satellite Processors



AIRBUS

First (Baby) Steps

"Taking Rust for a Test Drive" by Ferrous Systems

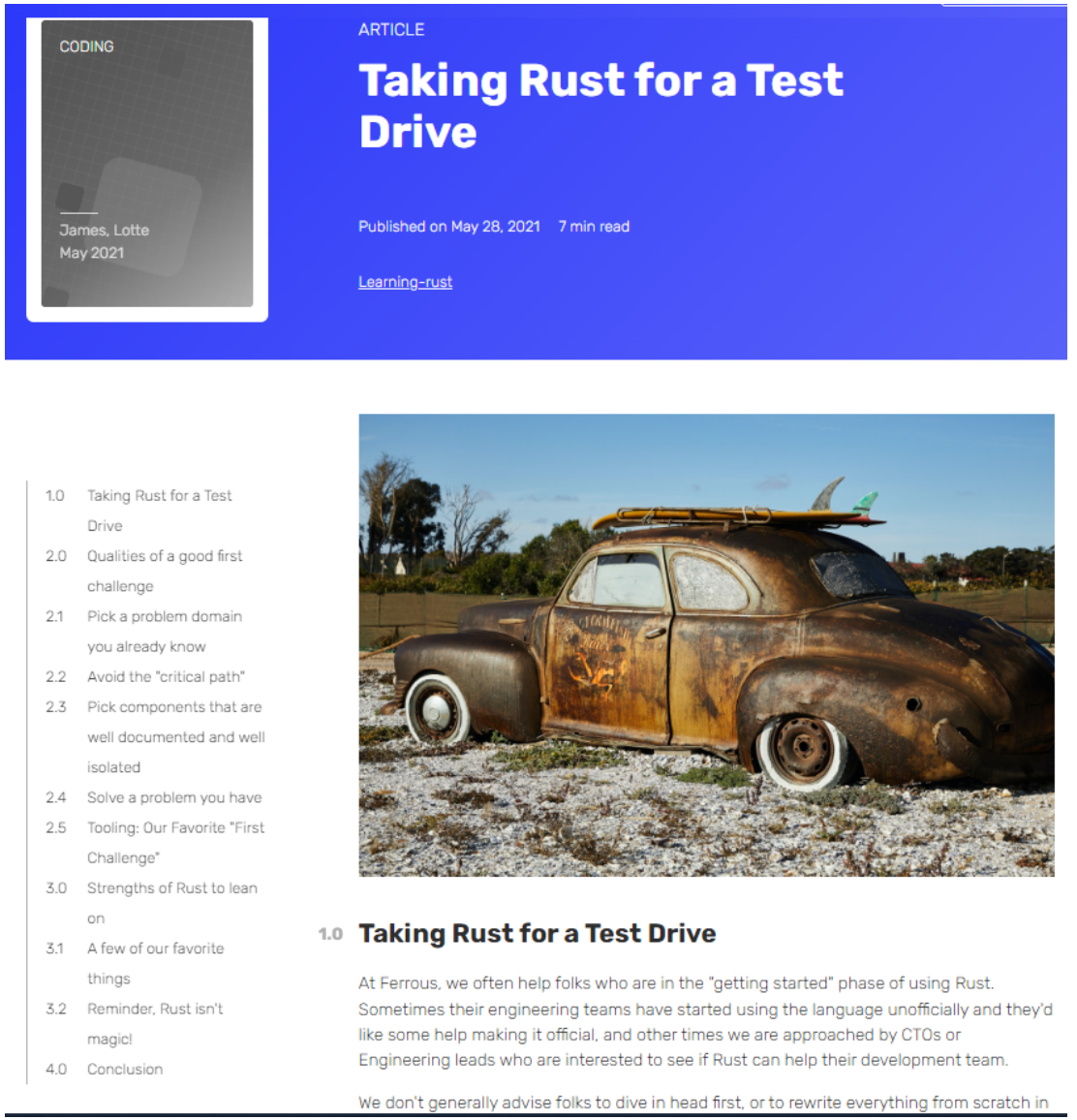
"Adopting Rust to Achieve Business Goals" by Tweede Golf

Key Insights

- Start small and early
- Focus on quick, concrete first results
- Consider your SW larger environment (tests, tools…)
- Seek professional training

Do not underestimate human factors!

- Introduce Rust into your company's landscape
- Build practical experience with the language/ecosystem
- Critical mass needed to start formal projects



AIRBUS