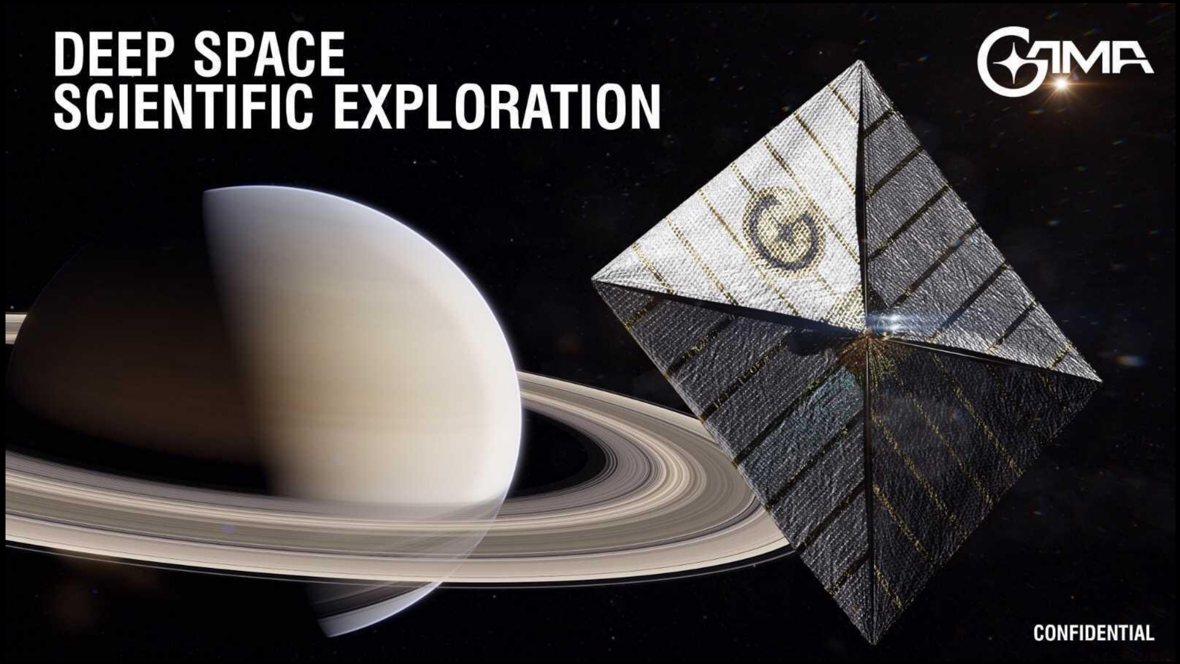
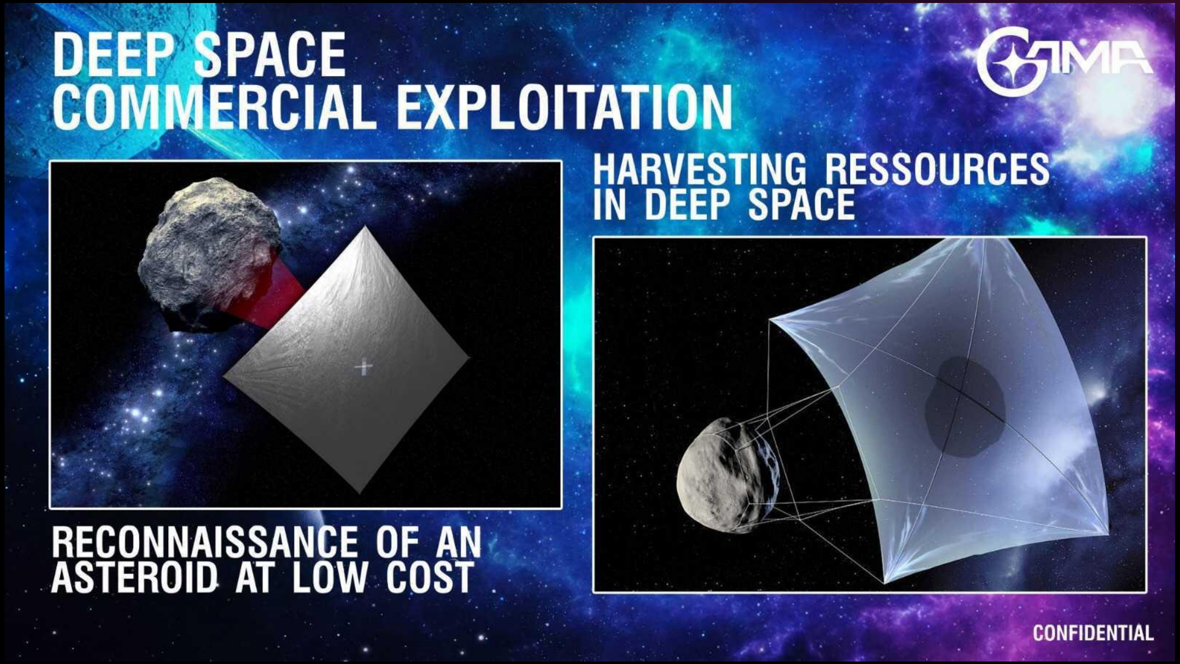
Rust for satellite flight software

Enabling a LEO solar sailing demonstrator at GAMA

Introduction

What is GAMA ?





Building solar sails for cheap deep-space transportation

Introduction

GAMA missions

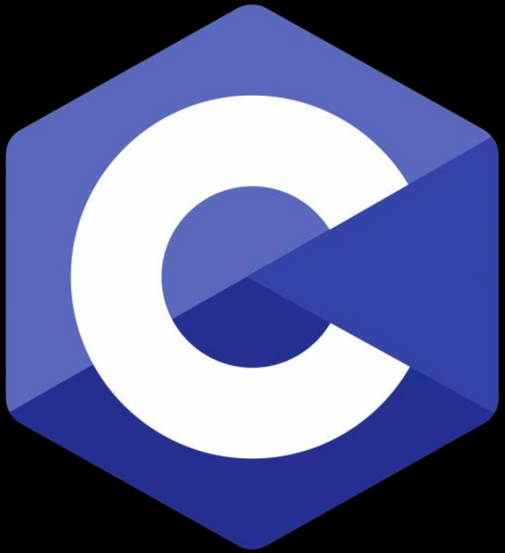
1. Demonstrating solar sail deployment

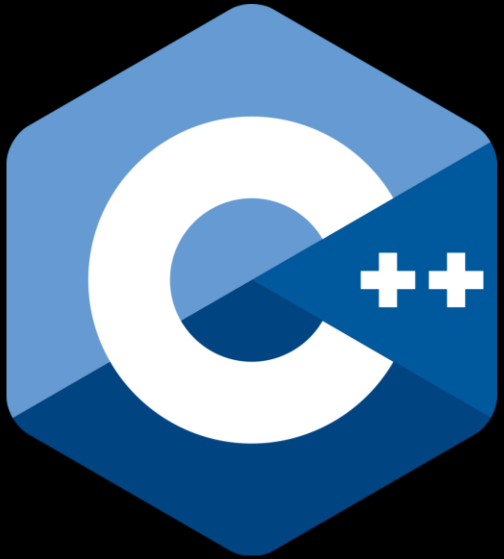
2. Demonstrating solar sail performance

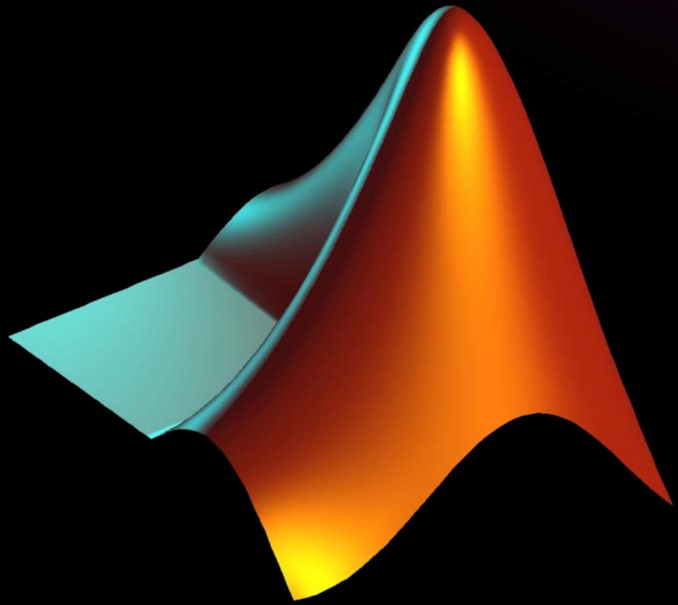
Introduction

Problematics

So what’s wrong with the usual approaches ?







Introduction

Rust programming language

Born at Mozilla in 2006 by Graydon Hoare

First stable release (v1.0) in 2015

Objective : Reduce memory management bugs

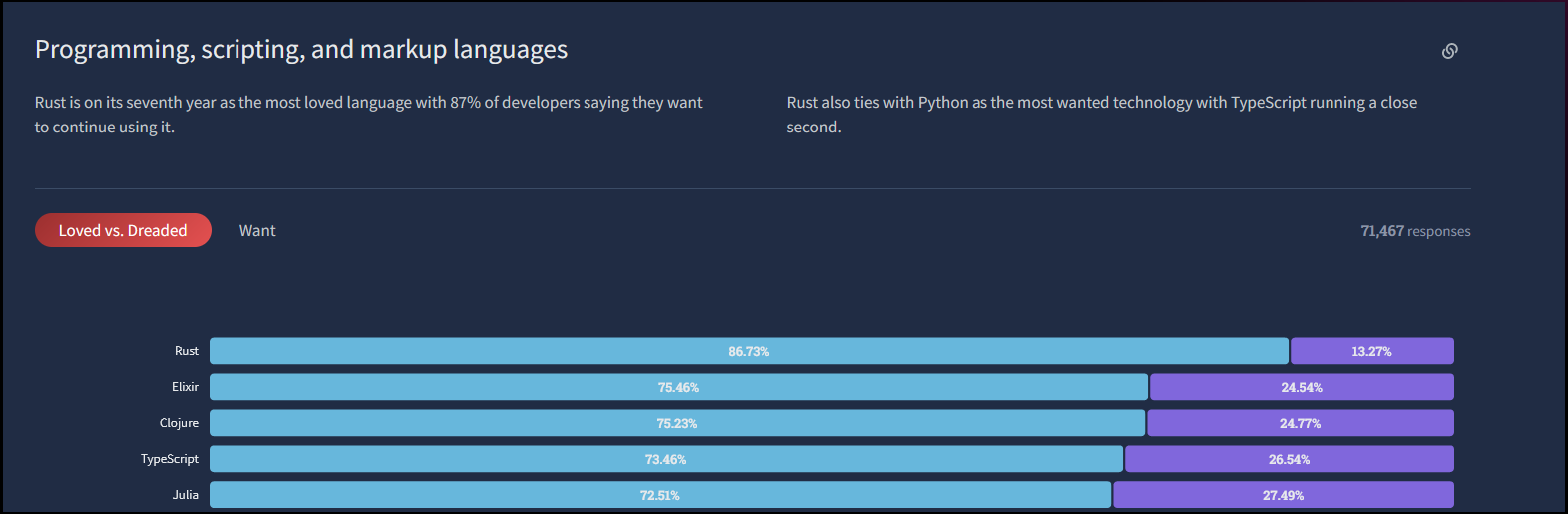
Known to be very simple and elegant while integrating modern features

https://survey.stackoverflow.co/2022/#technology-most-loved-dreaded-and-wanted

https://www.rust-lang.org/governance

Introduction

Rust programming language



Programming, scripting, and markup languages

Rust is on its seventh year as the most loved language with 87% of developers saying they want to continue using it.

Rust also ties with Python as the most wanted technology with TypeScript running a close second.

Loved vs. Dreaded | Want

71,467 responses

1st most loved language, 1st most wanted, 14th most popular

Introduction

Rust programming language

Ownership & lifetimes

Safe vs Unsafe Rust

Introduction

Rust in space ... ?

Never been done before.

ferrous systems

ferrocene

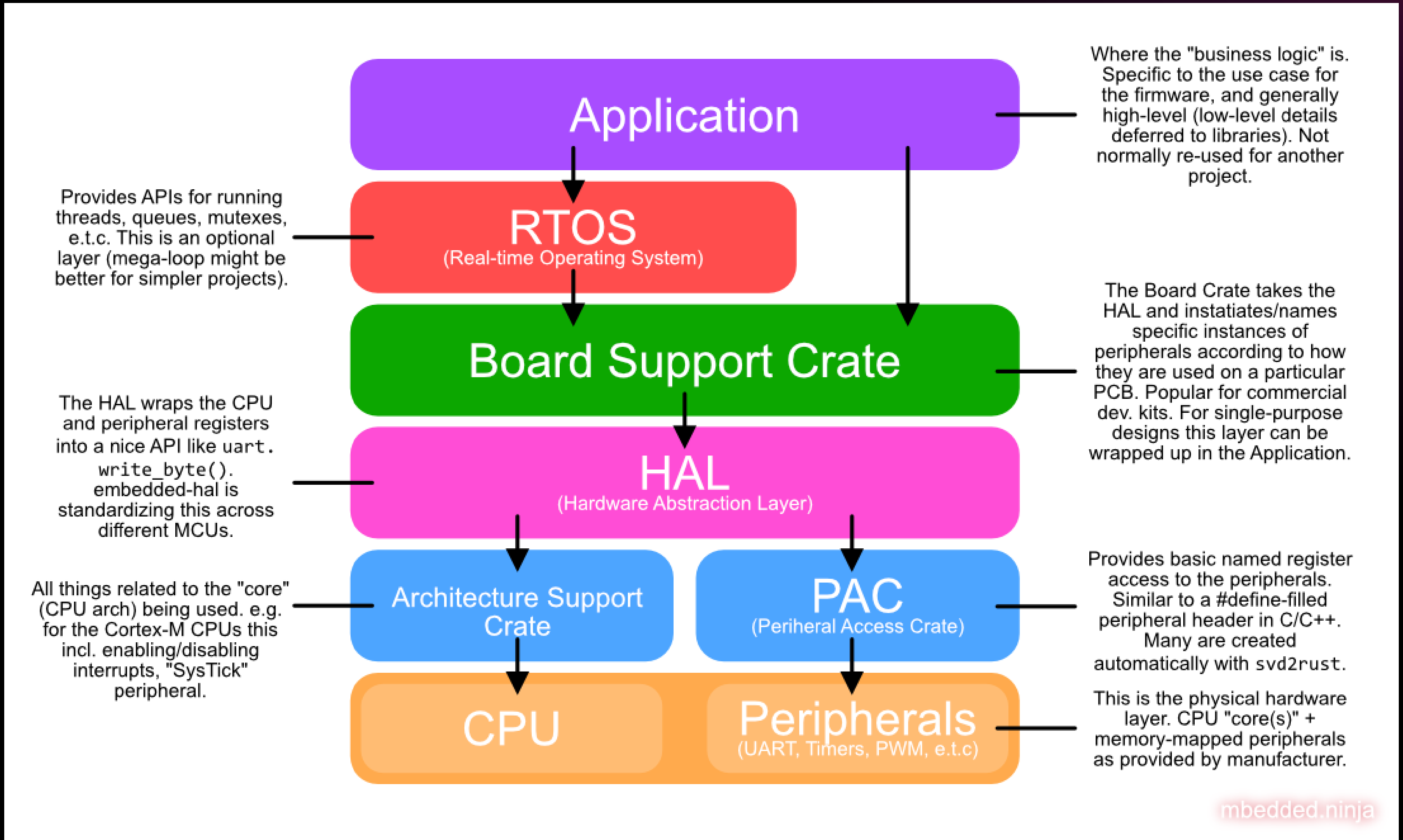
Few initiatives at CNES, ESA, NASA, SpaceX

Introduction

Embedded Rust stack

Copied from:

https://blog.mbedded.ninja/programming/languages/rust/running-rust-on-microcontrollers/#the-disadvantages-of-using-rust



Application

Where the "business logic" is. Specific to the use case for the firmware, and generally high-level (low-level details deferred to libraries). Not normally re-used for another project.

RTOS
(Real-time Operating System)

Provides APIs for running threads, queues, mutexes, e.t.c. This is an optional layer (mega-loop might be better for simpler projects).

Board Support Crate

The Board Crate takes the HAL and instatiates/names specific instances of peripherals according to how they are used on a particular PCB. Popular for commercial dev. kits. For single-purpose designs this layer can be wrapped up in the Application.

HAL
(Hardware Abstraction Layer)

The HAL wraps the CPU and peripheral registers into a nice API like `uart.write_byte()`. embedded-hal is standardizing this across different MCUs.

Architecture Support Crate

All things related to the "core" (CPU arch) being used. e.g. for the Cortex-M CPUs this incl. enabling/disabling interrupts, "SysTick" peripheral.

PAC
(Periheral Access Crate)

Provides basic named register access to the peripherals. Similar to a #define-filled peripheral header in C/C++. Many are created automatically with `svd2rust`.

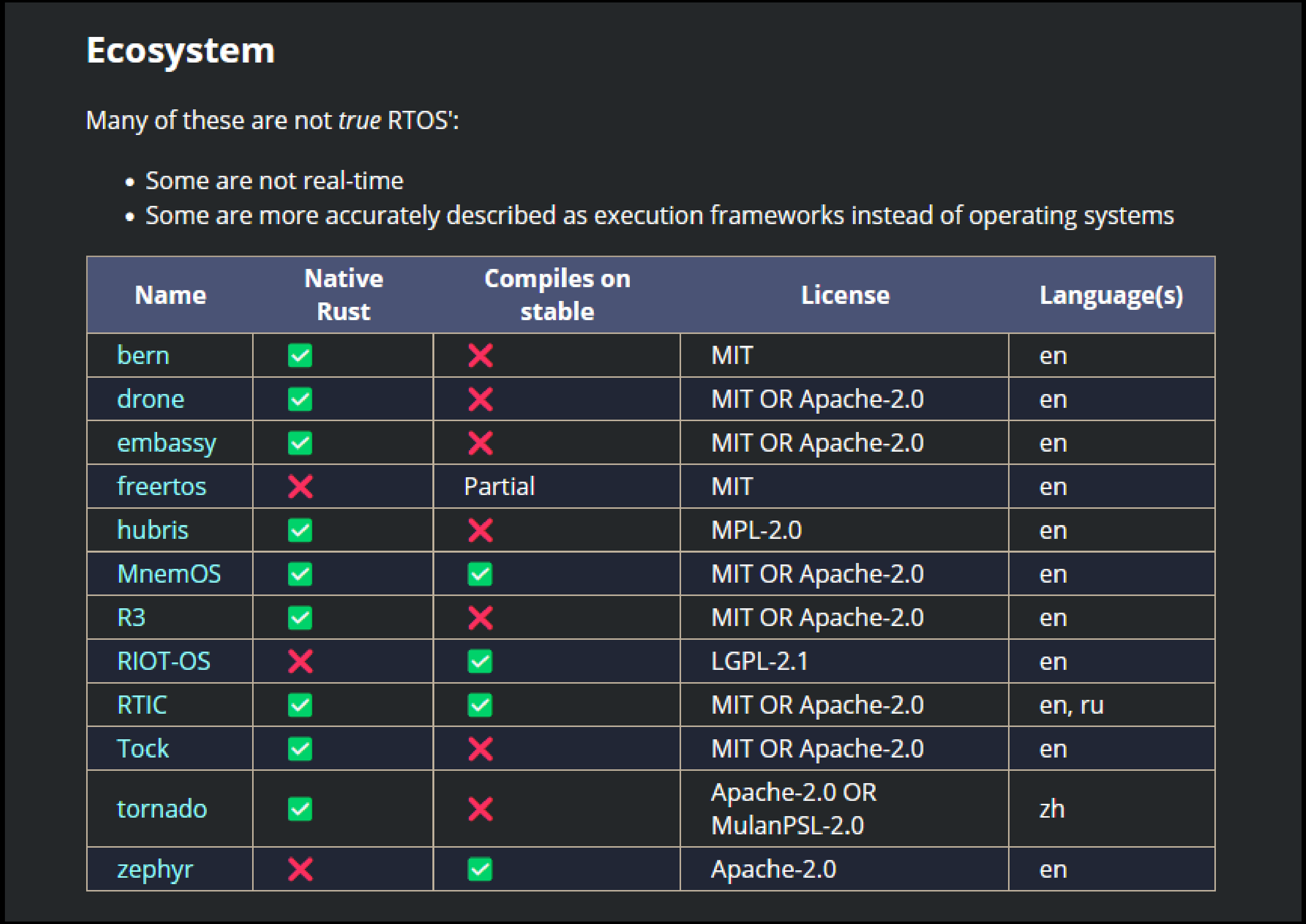
CPU

Peripherals
(UART, Timers, PWM, e.t.c)

This is the physical hardware layer. CPU "core(s)" + memory-mapped peripherals as provided by manufacturer.

Introduction

Rust for real-time systems



Ecosystem

Many of these are not *true* RTOS':

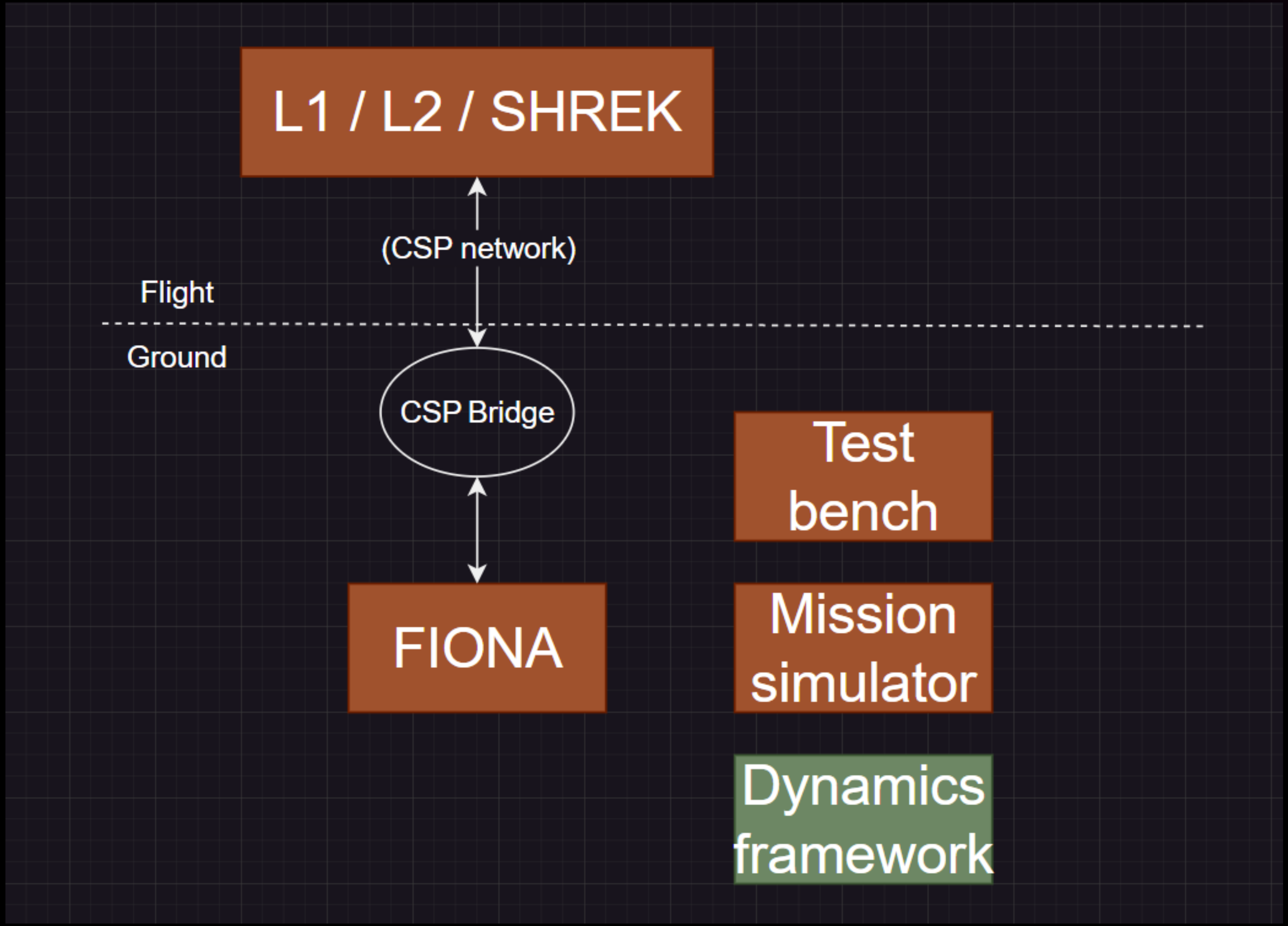
- Some are not real-time
- Some are more accurately described as execution frameworks instead of operating systems

Name	Native Rust	Compiles on stable	License	Language(s)
bern	✅	❌	MIT	en
drone	✅	❌	MIT OR Apache-2.0	en
embassy	✅	❌	MIT OR Apache-2.0	en
freertos	❌	Partial	MIT	en
hubris	✅	❌	MPL-2.0	en
MnemOS	✅	✅	MIT OR Apache-2.0	en
R3	✅	❌	MIT OR Apache-2.0	en
RIOT-OS	❌	✅	LGPL-2.1	en
RTIC	✅	✅	MIT OR Apache-2.0	en, ru
Tock	✅	❌	MIT OR Apache-2.0	en
tornado	✅	❌	Apache-2.0 OR MulanPSL-2.0	zh
zephyr	❌	✅	Apache-2.0	en

Source : arewertosyet.com

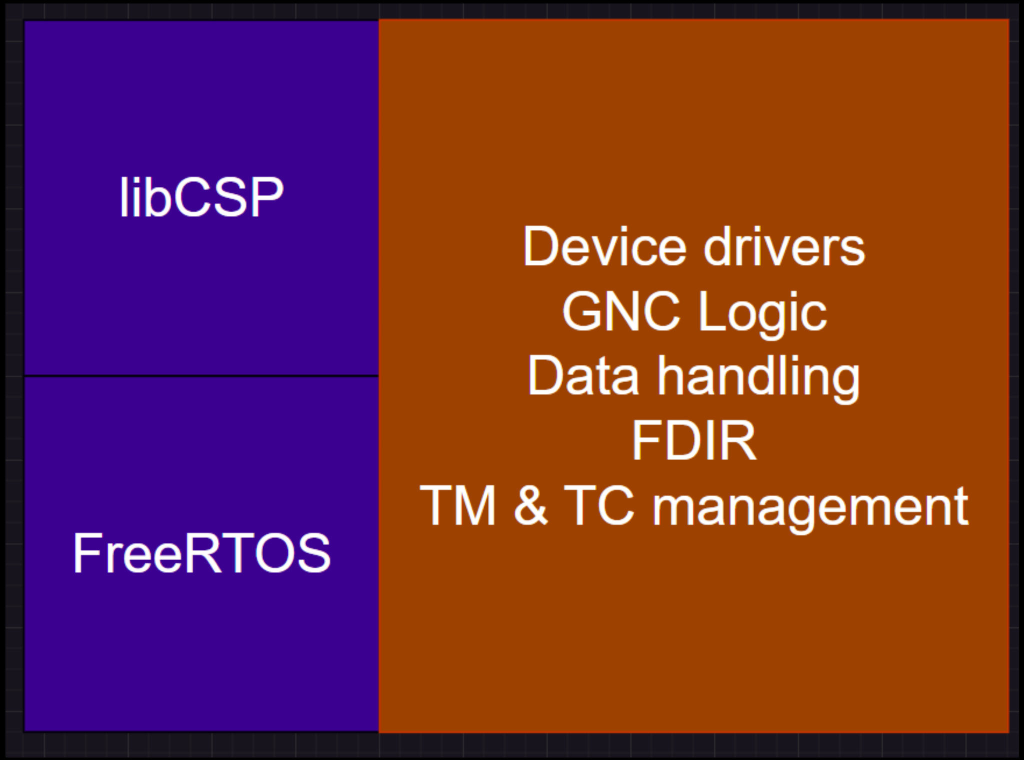
How we use Rust

Overview



How we use Rust

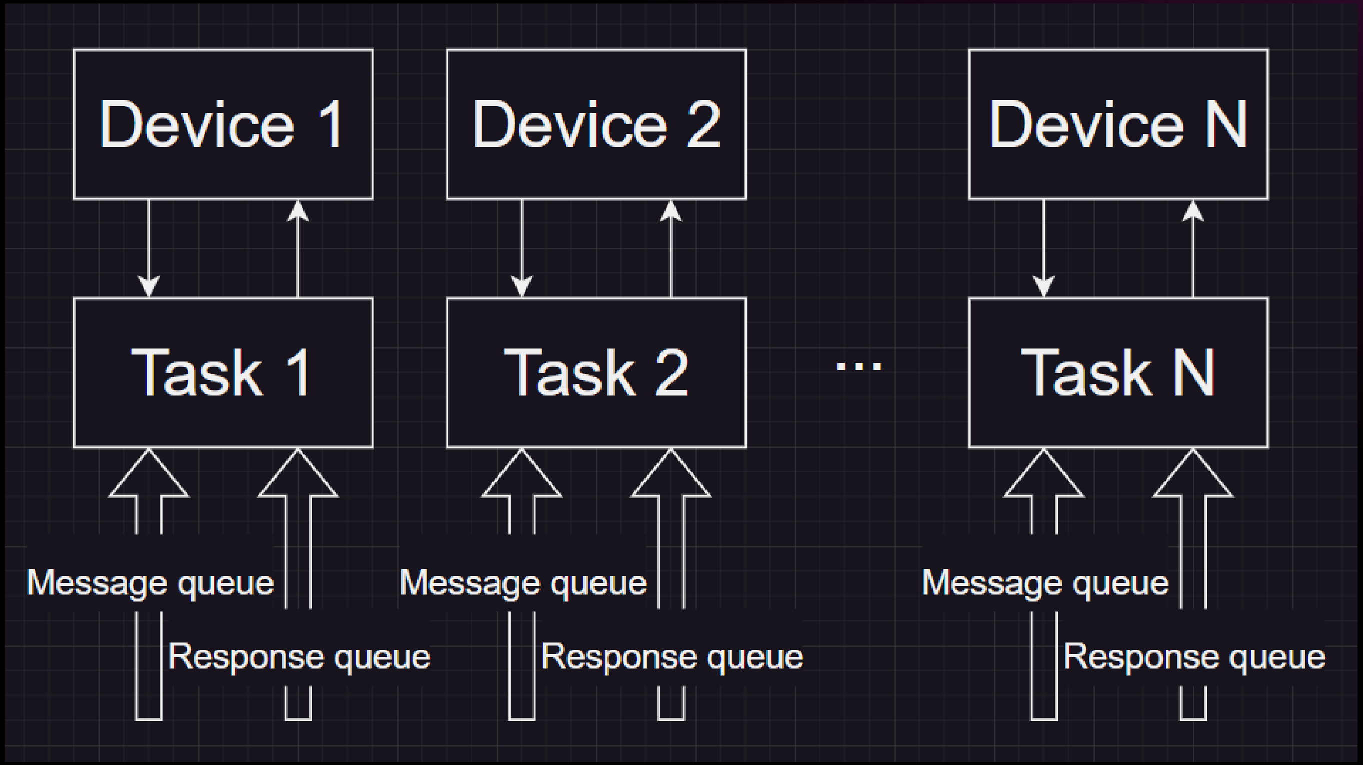
Flight Software



libCSP

FreeRTOS

Device drivers
GNC Logic
Data handling
FDIR
TM & TC management



Device 1

Device 2

Device N

Task 1

Task 2

…

Task N

Message queue

Response queue

Message queue

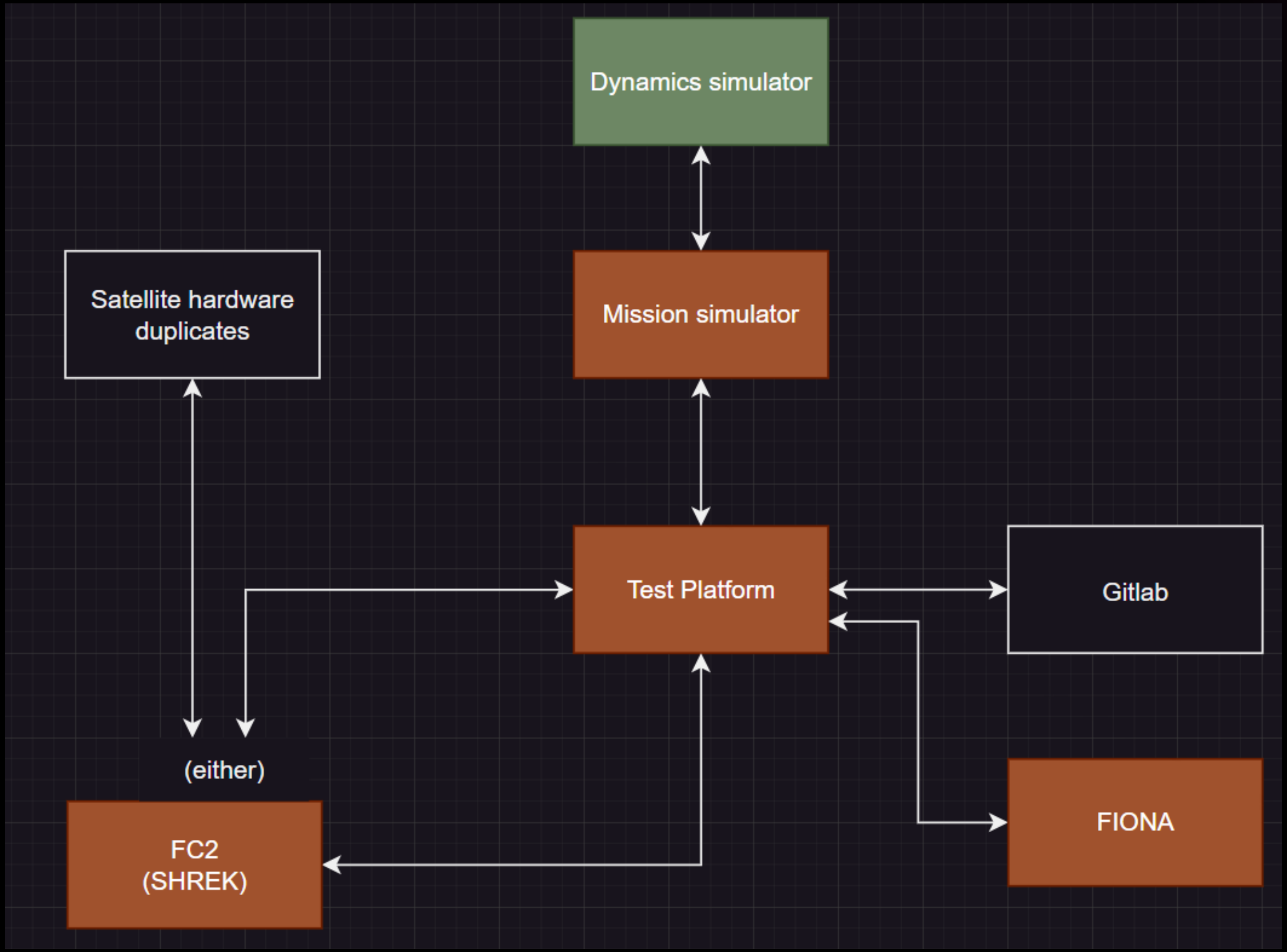
Response queue

Message queue

Response queue

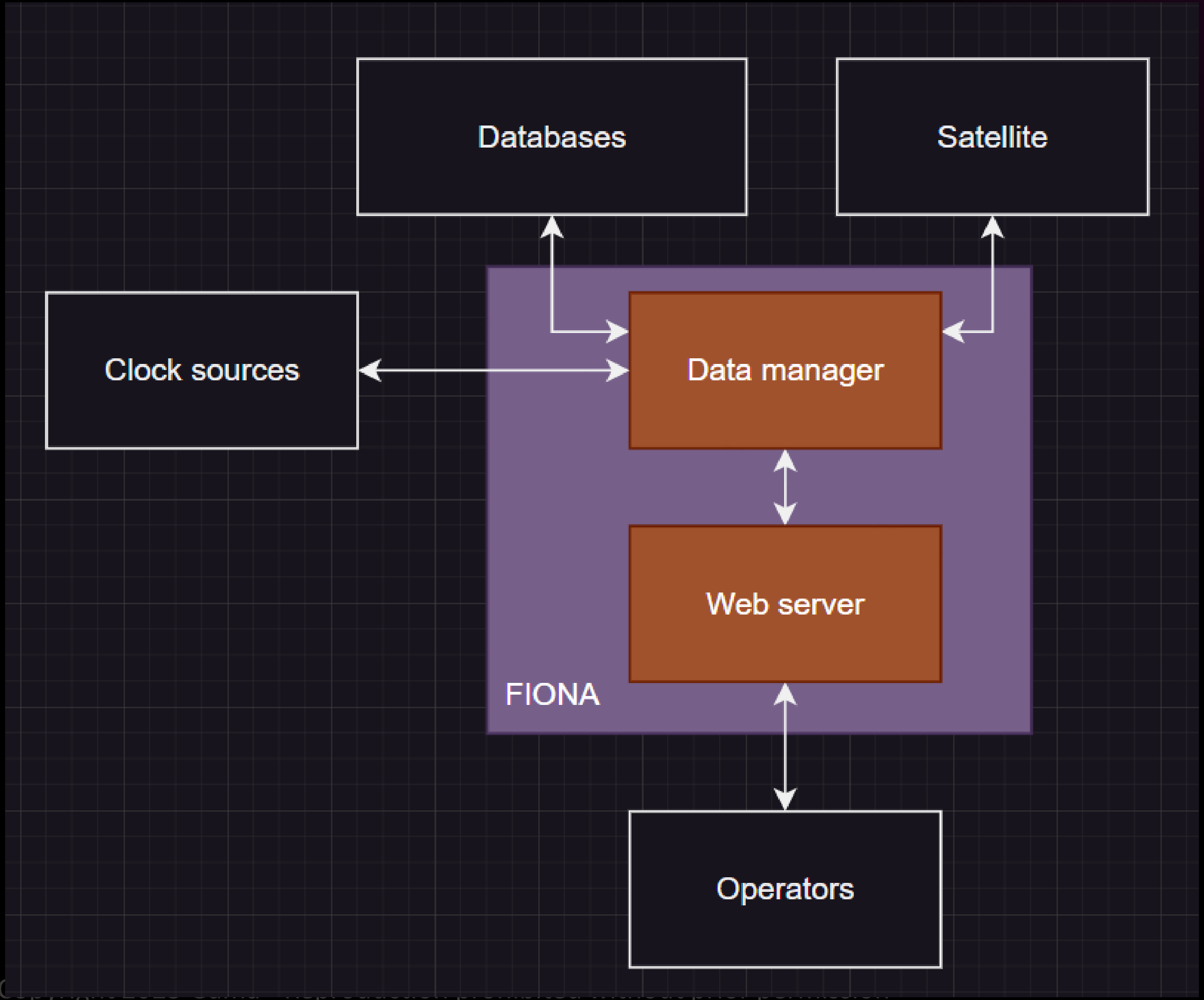
How we use Rust

Test bench



How we use Rust

Mission control software



How we use Rust

Product assurance

Property-based testing

Unit testing

Model checking

In-flight testing

HIL simulations

Fuzzing

Integration testing

Conclusion

Are we happy?

Mostly, yes

But… ?

Conclusion

Thank you for your time

chris@gamaspace.com / @chris_pwnorbitals:matrix.org

wouter@tweedegolf.com / @wassasin:matrix.org

(Support) Introduction

Who am I ?

GAMA

IPSA
ÉCOLE D'INGÉNIEURS
DE L'AIR, DE L'ESPACE ET DE LA MOBILITÉ DURABLE

IONSAT

O.N.E

https://declaverie.space/

Space / computing / engineering nerd

Currently head of GNC & Avionics at GAMA

(Support) Introduction

Flight software constraints

Reliability

Maintainability

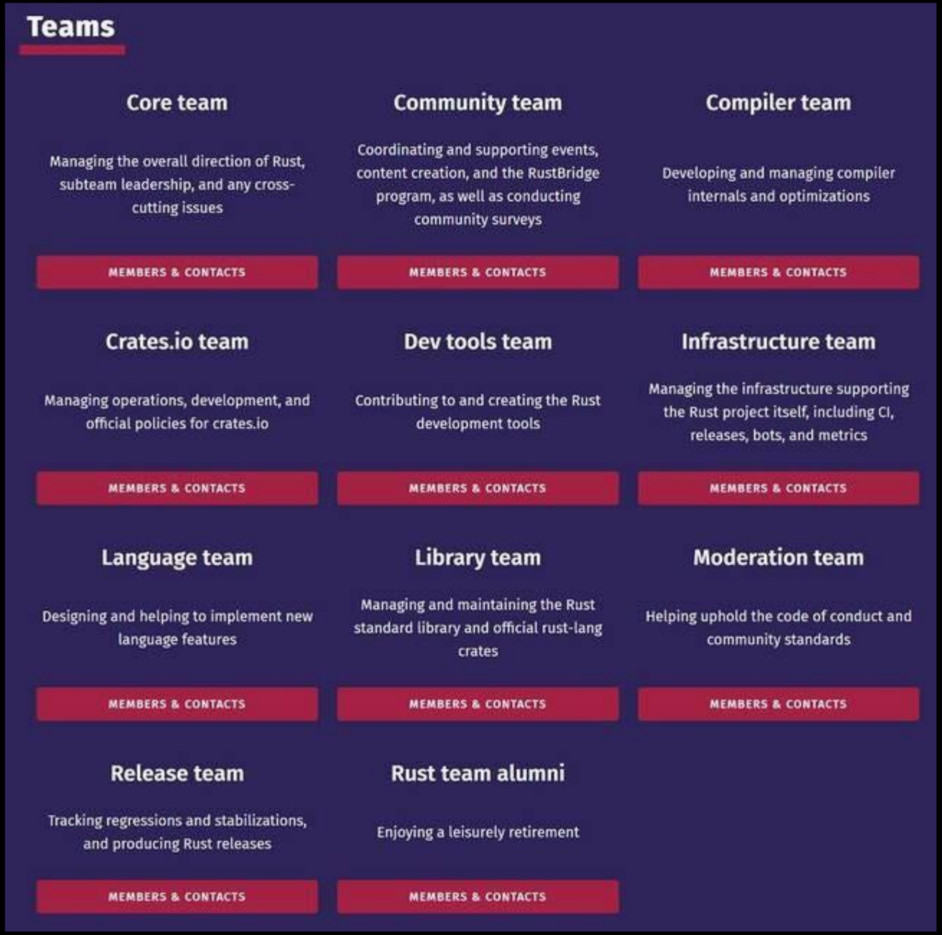
Testability

Hard real-time

Observability

Support

Rust governance



Teams

Core team
Managing the overall direction of Rust, subteam leadership, and any cross-cutting issues
MEMBERS & CONTACTS

Community team
Coordinating and supporting events, content creation, and the RustBridge program, as well as conducting community surveys
MEMBERS & CONTACTS

Compiler team
Developing and managing compiler internals and optimizations
MEMBERS & CONTACTS

Crates.io team
Managing operations, development, and official policies for crates.io
MEMBERS & CONTACTS

Dev tools team
Contributing to and creating the Rust development tools
MEMBERS & CONTACTS

Infrastructure team
Managing the infrastructure supporting the Rust project itself, including CI, releases, bots, and metrics
MEMBERS & CONTACTS

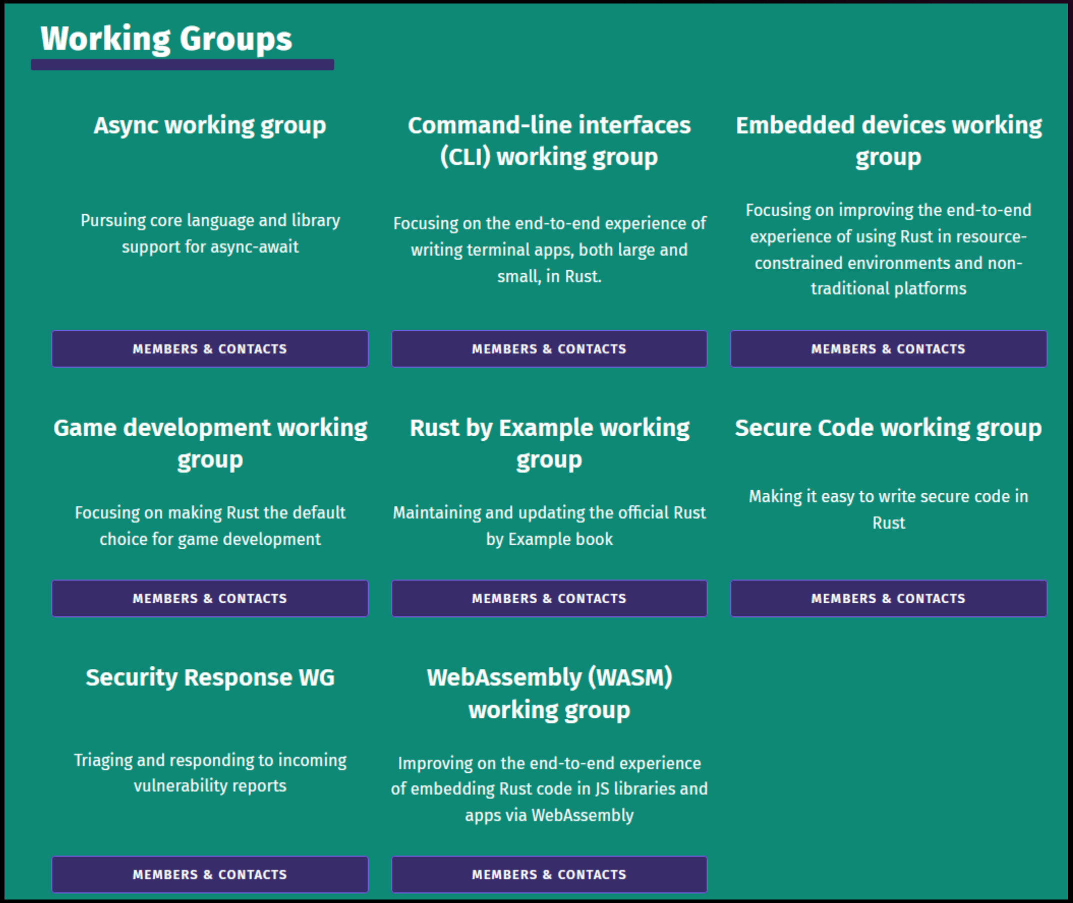
Language team
Designing and helping to implement new language features
MEMBERS & CONTACTS

Library team
Managing and maintaining the Rust standard library and official rust-lang crates
MEMBERS & CONTACTS

Moderation team
Helping uphold the code of conduct and community standards
MEMBERS & CONTACTS

Release team
Tracking regressions and stabilizations, and producing Rust releases
MEMBERS & CONTACTS

Rust team alumni
Enjoying a leisurely retirement
MEMBERS & CONTACTS



Working Groups

Async working group
Pursuing core language and library support for async-await
MEMBERS & CONTACTS

Command-line interfaces (CLI) working group
Focusing on the end-to-end experience of writing terminal apps, both large and small, in Rust.
MEMBERS & CONTACTS

Embedded devices working group
Focusing on improving the end-to-end experience of using Rust in resource-constrained environments and non-traditional platforms
MEMBERS & CONTACTS

Game development working group
Focusing on making Rust the default choice for game development
MEMBERS & CONTACTS

Rust by Example working group
Maintaining and updating the official Rust by Example book
MEMBERS & CONTACTS

Secure Code working group
Making it easy to write secure code in Rust
MEMBERS & CONTACTS

Security Response WG
Triaging and responding to incoming vulnerability reports
MEMBERS & CONTACTS

WebAssembly (WASM) working group
Improving on the end-to-end experience of embedding Rust code in JS libraries and apps via WebAssembly
MEMBERS & CONTACTS

Support

Rust future

- Flatten the learning curve
  - More expressive
  - Lift constraints
  - Improve features interplay (async, trait objects, …)

Support

References

- Official Rust book : https://doc.rust-lang.org/book/
- Rust embedded book https://docs.rust-embedded.org/book/start/registers.html
- Awesome-embedded-rust : https://github.com/rust-embedded/awesome-embedded-rust
- Rustonomicon for full reference : https://doc.rust-lang.org/nomicon/
- Unsafe code guidelines : https://rust-lang.github.io/unsafe-code-guidelines