Rust
Wozu eine neue Programmiersprache?

Andreas Wallner

2024-11-04

german brand award '24 corporate brand of the year

Ich?

Seit > 10 Jahren bei Infineon Technologies

Principal Engineer Contactless Innovation / Software & Firmware

In “Contactless Innovation” Team als Design Lead
- Pre-Tapeout: System / Digital Design
- Post-Tapeout: Software

Unregelmäßig in Produktentwicklung

In anderen Teams als Feuerwehr 🚒

Verbesserung von Entwicklungsflows

Rust für open-source / privat / Infineon

Umfrage

Agenda

Rust

- Programmiersprache
- nativ / compiliert
- statisch, stark, linear, inferiert typisiert

- 2008 – Hobbyprojekt von Graydon Hoare
- 2010 – Mozilla
- 2015 – Erste Alpha Release
- 2020 – Rust Foundation

Warum Rust?

70%

https://www.chromium.org/Home/chromium-security/memory-safety/

https://msrc.microsoft.com/blog/2019/07/a-proactive-approach-to-more-secure-code/

How Cyber Thieves Use Your Smart Fridge As Door to Your Data

Published Jun 23, 2021 at 4:00 PM EDT | Updated Jun 23, 2021 at 4:04 PM EDT

By Alex J. Rouhandeh
Special Correspondent
FOLLOW

by Michael Kan
U.S. Correspondent

Smart teddy bears involved in a contentious data breach

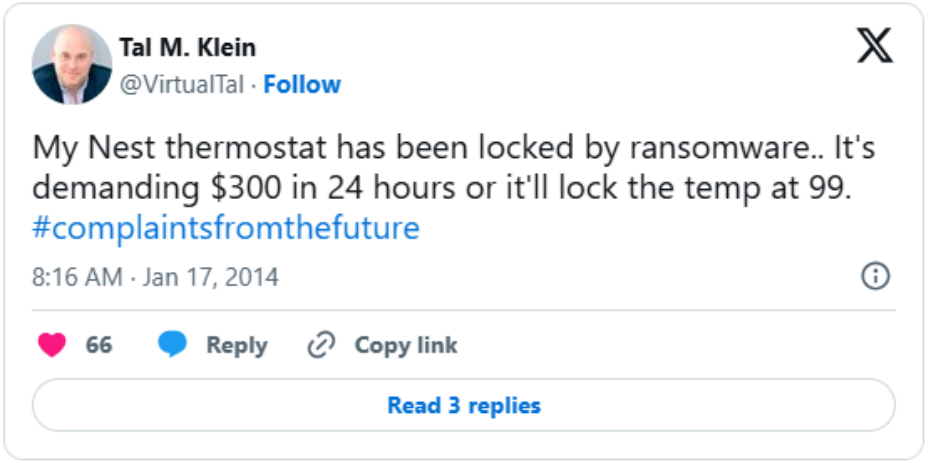
News
28 Feb 2017 • 4 mins

arXiv:2306.09017 (cs)

[Submitted on 15 Jun 2023 (v1), last revised 26 Jul 2023 (this version, v3)]

Who Let the Smart Toaster Hack the House? An Investigation into the Security Vulnerabilities of Consumer IoT Devices

Yang Li, Anna Maria Mandalari, Isabel Straw



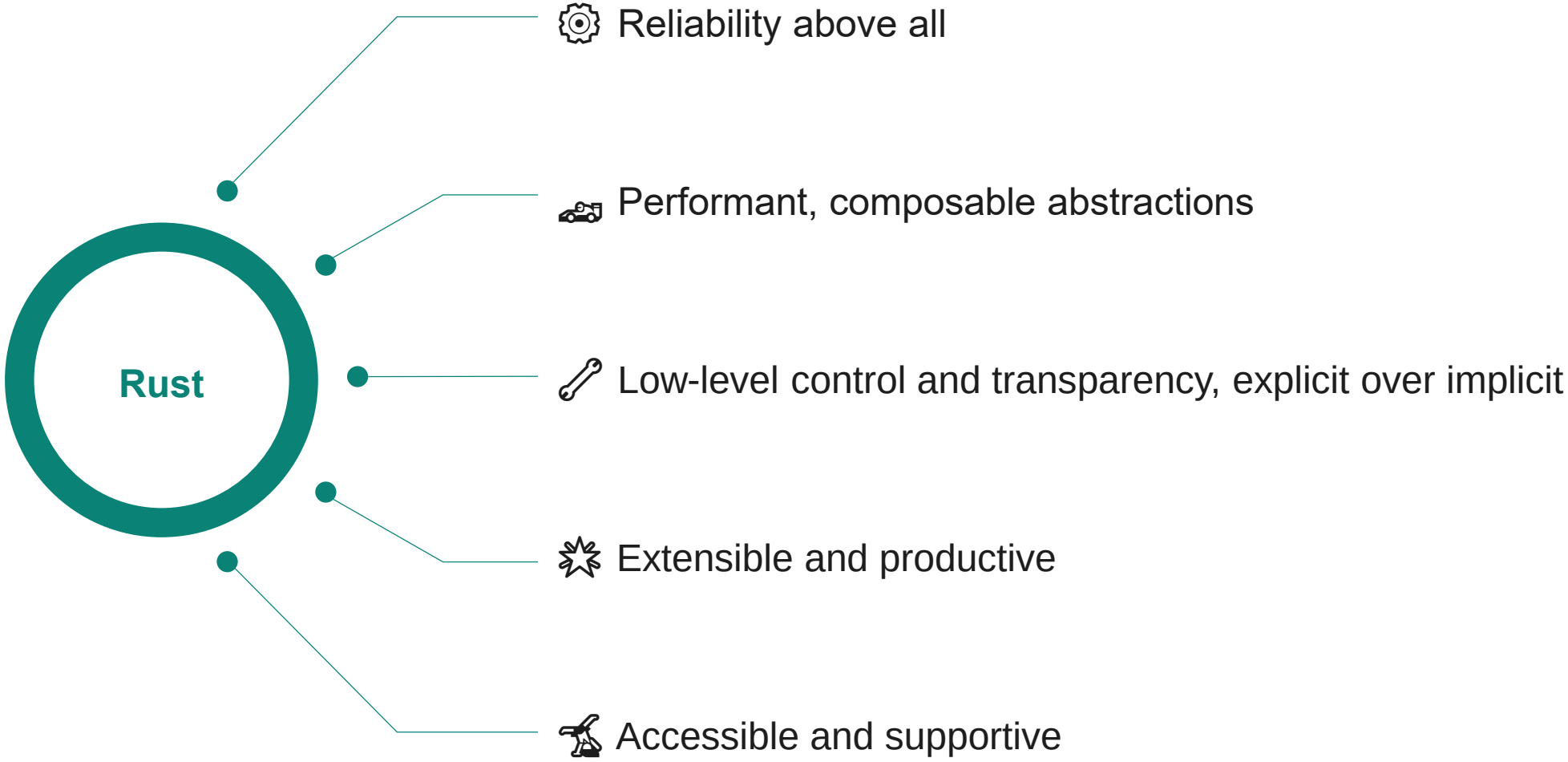
Safe Rust

kein “Undefined Behavior”

lesen/schreiben außerhalb Allokationen
use-after-free
data races
Uninitialisierte Variablen verwenden
Null-Pointer dereferenzieren
Zugriff auf nicht aktive Union-Member
(Ganzzahl Overflow)
...
nicht: Programmabbruch

https://doc.rust-lang.org/stable/reference/behavior-considered-undefined.html

Leitsätze



Rust anhand Beispielen

Borrowchecker - Entweder / Oder

Ein Besitzer

„Owner“

Viele konstante Referenzen	Eine veränderbare Referenz
„borrow“	„borrow“
„immutable reference“	„exclusive reference“
	„mutable reference“

Verletzung -> Compilierfehler

Speicherfreigabe: Verlassen des Scopes

Borrowchecker - Entweder / Oder

```rust
fn main() {
    let mut s = String::from("Hello");

    // Borrow `s` immutably
    let r1 = &s;
    let r2 = &s;

    // FEHLER: solange r1 & r2 existieren, kann keine mut referenz erstellt werden
    let r3 = &mut s;

    println!("r1: {}, r2: {}", r1, r2);

    // r1 & r2 sind nicht mehr in scope -> mut ref möglich
    let r3 = &mut s;
    r3.push_str(", world!");

    println!("r3: {}", r3);
}
```

Try it online

Borrowchecker - Entweder / Oder

```rust
struct Encoder<'a> {
    pub slice: &'a mut [u32],
}

fn main() {
    let mut v = vec![0; 255];
    let encoder = Encoder {
        slice: &mut v[20..25],
    };

    println!("{:?}", v);

    encoder.slice[0] = 0;
}
```

```
error[E0502]: cannot borrow `v` as immutable because it is
also borrowed as mutable
  --> code.tio:11:22
   |
8  |         slice: &mut v[20..25],
   |                      - mutable borrow occurs here
...
11 |     println!("{:?}", v);
   |                      ^ immutable borrow occurs here
12 |
13 |     encoder.slice[0] = 0;
   |     ---------------- mutable borrow later used here
```

Try it online

Borrowchecker - Entweder / Oder

```rust
fn main() {
    let s = String::from("hello");
    takes_ownership(s);
    println!("{s}");
}

fn takes_ownership(s: String) {
    todo!()
}
```

```
error[E0382]: borrow of moved value: `s`
 ⟶ src/main.rs:4:15
  |
2 |     let s = String::from("hello");
  |         - move occurs because `s` has type `String`, which does not implement the `Copy` trait
3 |     takes_ownership(s);
  |                     - value moved here
4 |     println!("{s}");
  |               ^^^ value borrowed here after move
 ...
```

Borrowchecker

+

Striktes Typsystem

+

Bounds checks

=

Safe Code

Variablen & Ausdrücke

```rust
fn main() {
    let int_val: i32 = 42;
    let float_val: f32 = 3.14;

    let mut f2 = 2.22;
    println!("f2: {}", f2);
    f2 = int_val as f64;
    println!("f2: {}", f2);

    let time = SystemTime::now().duration_since(SystemTime::UNIX_EPOCH).unwrap().as_secs();
    let is_odd = if time % 2 == 0 {
        println!("good number");
        false
    } else {
        println!("odd number");
        true
    };

    for i in 0..10 {
        println!("i: {}", i);
    }
}
```

Try it Online

Structs & Traits

```rust
pub trait GPIO {
    fn set_high(&mut self);
    fn set_low(&mut self);
    fn read(&self) -> bool;
}

pub struct ArduinoGPIO {
    pub pin: u8,
}

impl GPIO for ArduinoGPIO {
    fn set_high(&mut self) { todo!() }

    fn set_low(&mut self) { todo!() }

    fn read(&self) -> bool { todo!() }
}
```

- trait = Schnittstelle
- struct = Daten
- impl = Funktionalität

- Beliebig viele impls möglich
- externer code kann impls beinhalten

- Polymorphie auf Aufrufseite, nicht Definition

- embedded-rust crates

Structs & Traits

```rust
pub fn read_button(button: &ArduinoGPIO) -> bool {
    button.read()
}

pub fn blink_poly(led: &dyn GPIO) {
    led.set_high();
    led.set_low();
}

pub fn blink<T: GPIO>(led: &T) {
    led.set_high();
    led.set_low();
}
```

Funktionen können

- ein konkretes struct
- ein zur Laufzeit dynamisches struct
- ein zur Compile-Zeit dynamisches struct

übergeben bekommen

sum types (Enumerationen...)

```
enum Error {
    InvalidCommand,
    InvalidPin,
}

enum PinState {
    High = 1,
    Low = 0,
}

enum Command {
    Disconnect,
    SetPin(PinState),
    SetLeds(bool, bool, bool),
    SetPinAndLed { pin: PinState, led: bool },
}
```

Enumerationen sind nicht nur „Konstanten“

In C-wording sind enums eine Komination aus „enum“ & „union“

Eine Instanz

- kennt ihren aktuellen Zustand
- kann Daten beinhalten
- hat ihren eigenen Namespace

Pattern matching

```
fn process(&mut self, cmd: Command) {
    let leds_maybe_modified = match cmd {
        Command::Disconnect => exit(0),
        Command::SetPin(new) => {
            self.pin = new;
            false
        }
        Command::SetLeds(true, l2, l3) => {
            // spezialfall ?
            self.leds = [self.leds[0], !l2, !l3];
            true
        }
        Command::SetLeds(l1, l2, l3) => {
            let new = [l1, l2, l3];
            self.leds = new;
            true
        }
        Command::SmthgComplex { pin, led } => todo!()
    };
    /* ... */
}
```

- für Variablen / enum / struct / tuple / arrays
- „switch/case“ auf Steroid
- erlaubt das „Zerlegen“ von Daten
- Siehe Rust Book

Try it online

Pattern matching

- „if“ pattern matching für einzelne Fälle

```
fn example(&self, cmd: Command) {
    if let Command::SmthgComplex { led, ..} = cmd {
        println!("setting LED to {}", led);
    }
}
```

Fehlerbehandlung

```rust
// normally, use std::result::Result
enum Result<T, E> {
    Ok(T),
    Err(E)
}

// and std::option::Option
enum Option<T> {
    Some(T),
    None
}

struct DivideByZero;
fn divide(x: u32, y: u32)
    -> Result<u32, DivideByZero>
{
    if y == 0 {
        return Result::Err(DivideByZero {});
    }

    Result::Ok((x as f32 / y as f32).round() as u32)
}
```

- „Result“ für mögliche Fehler
- „Option“ für mögliche Absenz
  - (array „.get()“)
  - auch für parameter
- Keine Spezialwerte für Fehler wie in C
- Trotzdem optimiert
- Statisch überprüft
- auch für Funktionen, die nichts zurückgeben
- Siehe Result, Option, und Rust Book

Fehlerbehandlung

```rust
fn main() -> Result<(), String> {
    match divide(10.0, 0.0) {
        Ok(result) => println!(„OK: {}", result),
        Err(e) => println!("Error: {}", e),
    }
    if divide(1.0, 0.0).is_err() {
        println!("that didn't go to plan...");
    }

    let _unsure = divide(10.0, random())?;

    let _with_fallback =
        divide(10.0, random()).unwrap_or(0.0);
    let _we_are_sure = divide(10.0, 1.0).unwrap();

    let my_option = divide(10.0, 0.0).ok();
    if let Some(result) = my_option {
        println!("Result: {}", result);
    }

    Ok(())
}
```

Try it online

Result / Option können benutzt werden via:

- Pattern matching
- Methoden zum Abfragen
- Weitergeben / Konvertieren des Fehlers
- Methoden zum Auspacken / fallback
- Konvertierung zwischen Result / Option

Newtypes

```rust
struct AdcCode12(i16);
impl AdcCode12 {
  pub fn new(value: i16)
      -> Result<Self, OutOfRangeError> {

    let max = 2i16.pow(12);
    let min = -(2i16.pow(12));
    if value >= min && value < max {
      return Ok(Self(value));
    }
    Err(OutOfRangeError {})
  }
  // ...
}
impl TryFrom<f32> for AdcCode12 {
  type Error = OutOfRangeError;
  fn try_from(value: f32) -> Result<Self, Self::Error> {
      todo!()
  }
}
```

- Nutzt striktes Typsystem
- Statt Variablennamen & Kommentar => Typ
- Kein Laufzeit-Overhead
- Marginal mehr Code bei Verwendung
- Siehe Rust by Example

Try it online

Macros

```
#[derive(Copy, Clone, Debug, PartialEq, Eq,
         PartialOrd, Ord)]
struct AdcCode12(i16);

fn func(code1: AdcCode12, code2: AdcCode12) {
    println!("code: {:?}", code1);
    println!("{}", code1 > code2);
}
```

Macros arbeiten auf Syntax-Baum Ebene

An vielen Stellen für die Code-Generierung verwendet

Manche als Marker (Copy, Sync, ...)

Erleichtern das Leben für

- Newtypes
- Debugging
- Konvertierungen
- ...

Siehe z.b. num_enum, bitflags, **serde**

Iterator

```rust
pub fn sum_squares_even(v: &[u32]) -> u32 {
    v.iter()
        .filter(|&vv| vv % 2 == 0)
        .map(|vv| vv * vv)
        .reduce(|r, vv| r + vv)
        .unwrap_or(0)
}

fn main() {
    let v = vec![4, 8, 7, 10, 42, 77];
    println!("{}", sum_squares_even(&v));
}
```

Oft Iterator statt Schleife

- Keine array-grenzen checks (...)
- Lazy
- Besser lesbar (?)

Oft verwendet als Resultat

User kann entscheiden ob eine neue Liste etc. gebraucht wird

Try it online

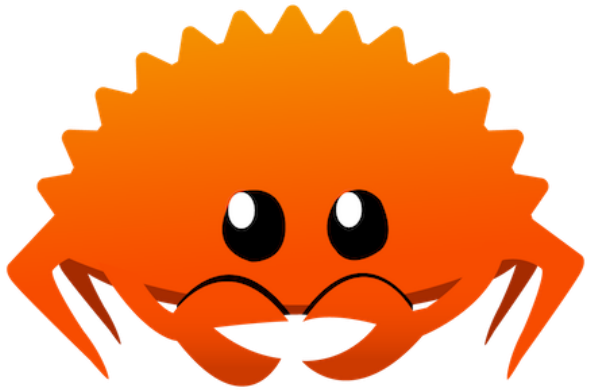
Anwendung & Ökosystem

Moderne Sprache

- Spagat zwischen low-level und high-level

- Namespaces, UTF-8 Strings, ...
- Große Standardbilbiothek
- Multithreaded / Asynchrone Programme

- Pakete / Module / Paketeverwaltung
  - expliziter Export / Sichtbarkeit
- Ökosystem
  - Cargo
  - crates.io / docs.rs / lib.rs
  - rustfmt
  - rustdoc
  - clippy
  - test
  - ...

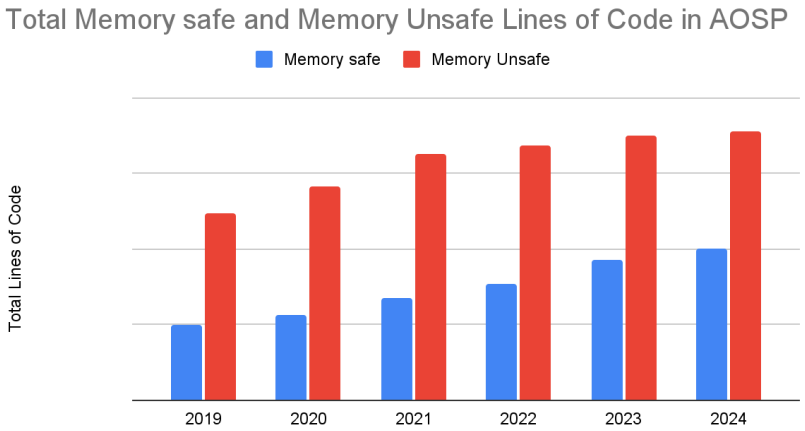


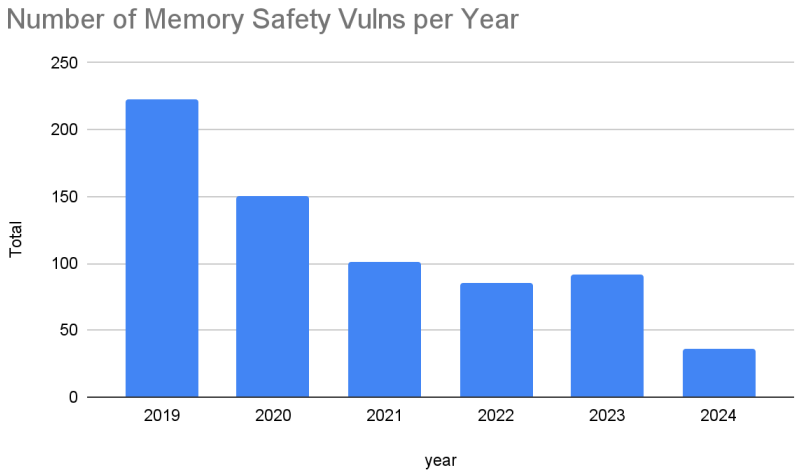
Akzeptanz in Industrie

- Verwendung stetig ansteigend (2023 ~1.45%, TIOBE Platz 13)
- Microsoft: ‘Best Chance’ at Safe Systems Programming (Video)
- Google: AOSP wechselt langsam auf Rust (2021, 2022, 2024)
- Amazon: Prime Video UI in Rust, Rust in EC2 & S3

- Volvo ECU für Elektroautos in Rust
- Infineon Automotive uCs mit Rust support (inklusive Aurix compiler)
- Elektrobit: AUTOSAR kompatible „basic software“

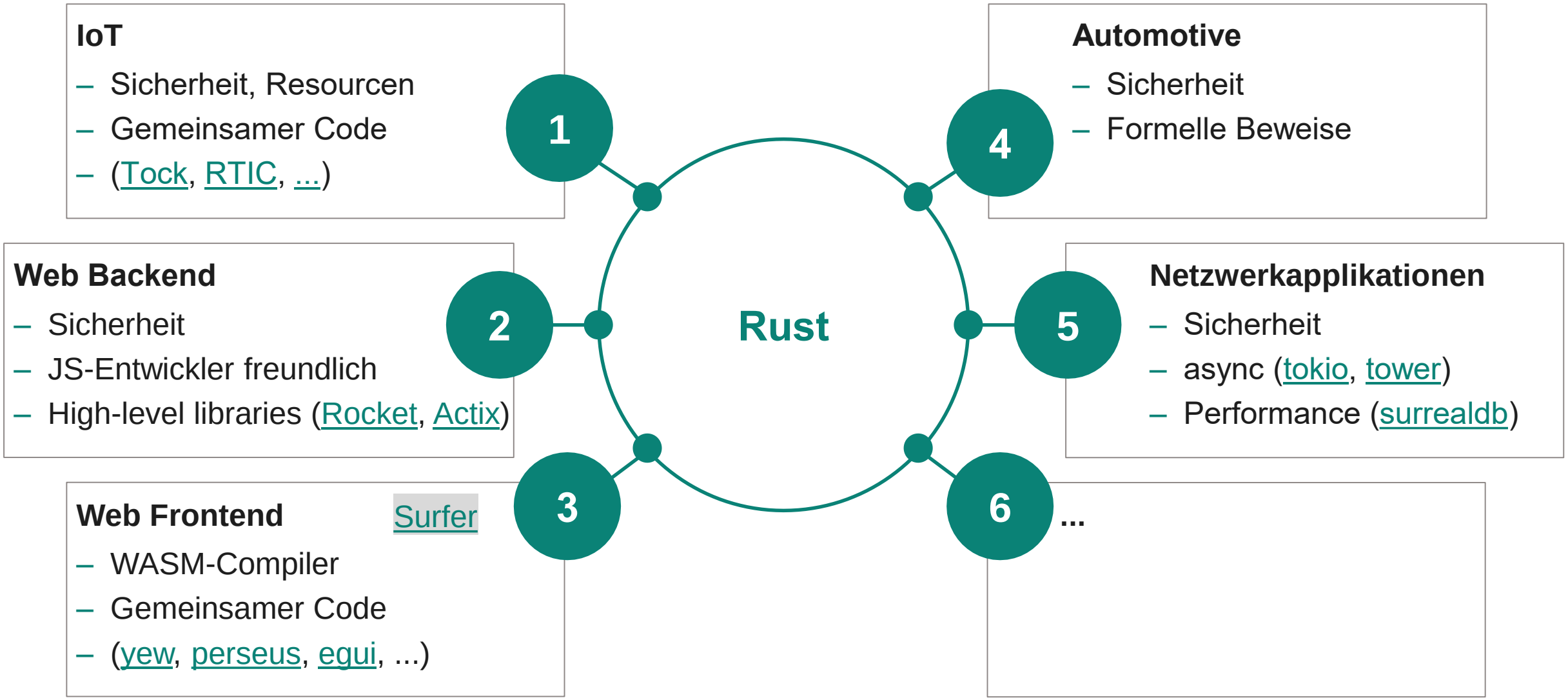
- EU IoT: Cybersecurity Guideline
  - memory safety noch nicht vorgeschrieben
- Memory-safe languages empfohlen/verlangt von
  US CISA, NSA, FBI, AUS CSC, CAN CCS, UK CSC, NZ CSC
- Code aktiv nach Rust portieren: DARPA TRAKTOR





Quelle: Google

Verwendungsgebiete



Rust im Unterricht

- Rust-Bedarf steigend, auch im Embedded Bereich
- Regulatorien werden dies weiter fördern

- Rust als erste Sprache: ... Nein?
- Steile Lernkurve?
  - Vieles ohne tiefes Wissen möglich
  - Verwenden von std braucht trotzdem gewisse Grundkonzepte
  - Compiliert erst wenn „fehlerfrei“ / „safe“
  - Sehr gute Fehlermeldungen
- Interessant: Pointer...

- Online resourcen: Compiler, Kurse, etc.

- https://rust-edu.org/
- https://www.youtube.com/watch?v=lQ36K1htRDY&list=PL0Ur-09iGhpwMbNiVTBeHmIjs0GuIXhNg&ab_channel=LukasKalbertodt

Rust @ Pädagigische Hochschule

- 631.0MK51 „Einführung in Rust – Moderne Programmierung für die nächste Generation“
- Mi 2025-04-09 @ Infineon Graz

- Anmeldung: 2024-12-18 – 2025-01-24
- ... PH Online

- Noch kein Link, aber
- https://www.phst.at/fortbildung-beratung/fortbildung/sekundarstufe-berufsbildung/
  => Stichwort „Technik“

Ressourcen

- https://www.rust-lang.org/learn
- https://doc.rust-lang.org/book/
- https://doc.rust-lang.org/rust-by-example/
- https://rust-lang-nursery.github.io/rust-cookbook/intro.html
- https://google.github.io/comprehensive-rust/
- https://tourofrust.com/
- https://fasterthanli.me/articles/a-half-hour-to-learn-rust

Questions & answers


Infineon