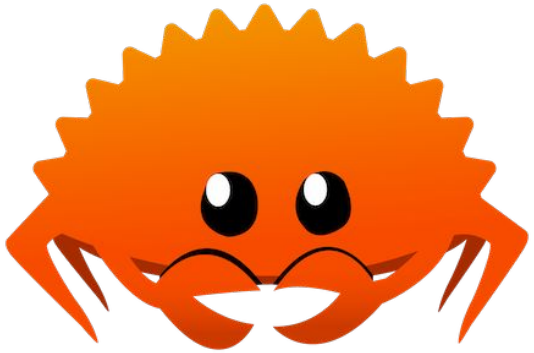
Mark Russinovich
@markrussinovich

Speaking of languages, it's time to halt starting any new projects in C/C++ and use Rust for those scenarios where a non-GC language is required. For the sake of security and reliability. the industry should declare those languages as deprecated.

10:50 PM · Sep 19, 2022



Varför Rust och hur?

En C++-veterans betraktelser

Vem är jag?

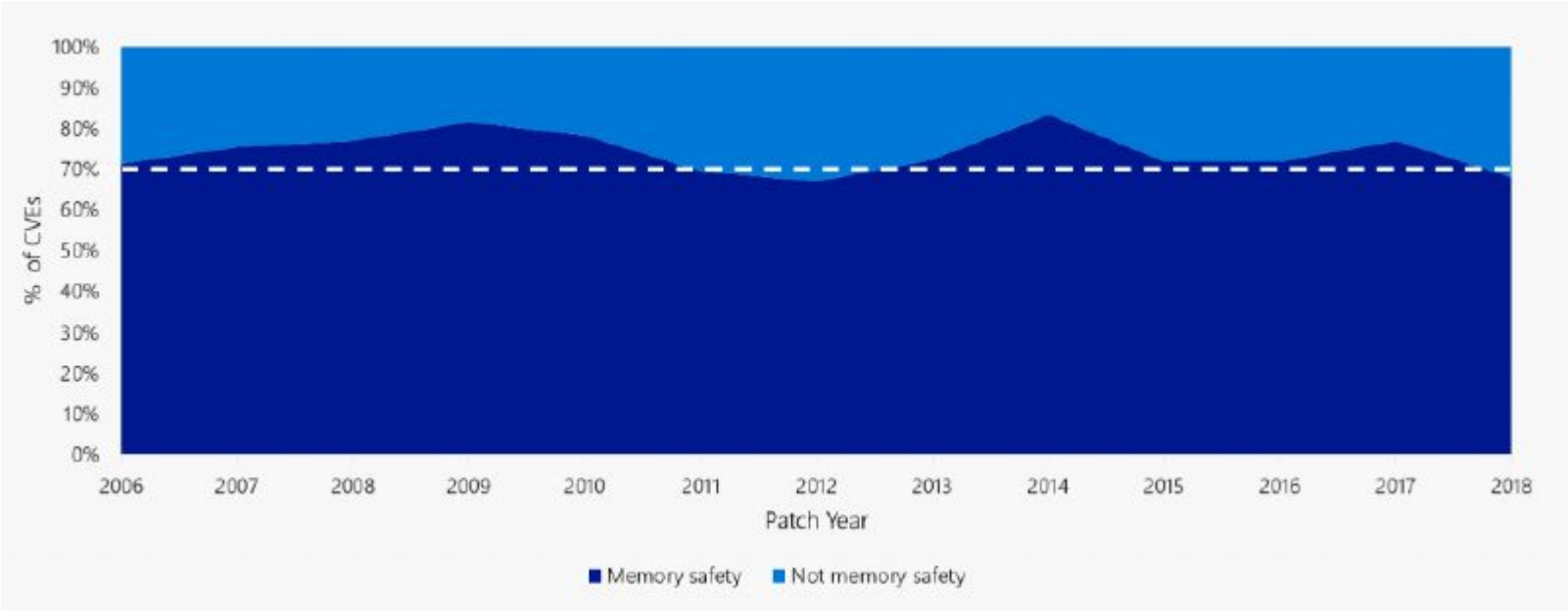
Thomas Johannesson

code42d.se

github.com/jordfras

Rusts USP

- Minnessäkerhet utan GC



Rusts USP

- Minnessäkerhet utan GC
- Kompilatorn garanterar att du gör rätt
  - Borrow checker

Rusts USP

- Minnessäkerhet utan GC
- Kompilatorn garanterar att du gör rätt
  - Borrow checker
- Fearless concurrency

Rusts USP

- Minnessäkerhet utan GC
- Kompilatorn garanterar att du gör rätt
  - Borrow checker
- Fearless concurrency
- C++ med statisk kodanalys och runtime check på tester?

Utvecklarergonomi

- Modernt språk
  - Slutledning av typer (type inference)
  - Genomtänkta “defaults”
  - Enum är “summa-typ”
  - Mönstermatchning
  - Iteratorer
  - Kostnadsfria abstraktioner (zero-cost abstractions)
  - Felhantering med resultat-typ

Utvecklarergonomi

- Modernt språk
  - Slutledning av typer (type inference)
  - Genomtänkta “defaults”
  - Enum är “summa-typ”
  - Mönstermatchning
  - Iteratorer
  - Kostnadsfria abstraktioner (zero-cost abstractions)
  - Felhantering med resultat-typ

```
// Variable type is inferred from later usage
let mut things = Vec::new();
things.push("thing".to_string());
```

Utvecklarergonomi

- Modernt språk
  - Slutledning av typer (type inference)
  - Genomtänkta “defaults”
  - Enum är “summa-typ”
  - Mönstermatchning
  - Iteratorer
  - Kostnadsfria abstraktioner (zero-cost abstractions)
  - Felhantering med resultat-typ

```
let name: String = String::from("Rust");
// Cannot do this without declaring name `mut`
//name = String::from("Ferris");

// Need to explicitly clone to make a copy
let _other_name = name.clone();

// Default behavior is move
let _moved_name = name;
```

Utvecklarergonomi

- Modernt språk
  - Slutledning av typer (type inference)
  - Genomtänkta “defaults”
  - Enum är “summa-typ”
  - Mönstermatchning
  - Iteratorer
  - Kostnadsfria abstraktioner (zero-cost abstractions)
  - Felhantering med resultat-typ

```
enum Shape {
    Circle(f64),
    Rectangle { width: f64, height: f64 },
}

let shapes = vec![
     Shape::Circle(1.0),
     Shape::Rectangle { width: 2.0, height: 3.0 },
];
```

Utvecklarergonomi

- Modernt språk
  - Slutledning av typer (type inference)
  - Genomtänkta “defaults”
  - Enum är “summa-typ”
  - Mönstermatchning
  - Iteratorer
  - Kostnadsfria abstraktioner (zero-cost abstractions)
  - Felhantering med resultat-typ

```
for shape in shapes {
    match shape {
        Shape::Circle(radius) => println!("Circle with radius {}", radius),
        Shape::Rectangle { width: w, height: h, } => {
            println!("Rectangle with width {} and height {}", w, h)
        }
    }
}
```

Utvecklarergonomi

- Modernt språk
  - Slutledning av typer (type inference)
  - Genomtänkta “defaults”
  - Enum är “summa-typ”
  - Mönstermatchning
  - Iteratorer
  - Kostnadsfria abstraktioner (zero-cost abstractions)
  - Felhantering med resultat-typ

```
let numbers = vec![1, 2, 3, 4, 5];
let is_odd = |x: &i32| x % 2 != 0;
let numbers: Vec<_> = numbers
    .into_iter()
    .filter(is_odd)
    .map(|x| x * x)
    .collect();
println!("Odd squared numbers: {:?}", numbers);
```

Utvecklarergonomi

- Modernt språk
  - Slutledning av typer (type inference)
  - Genomtänkta “defaults”
  - Enum är “summa-typ”
  - Mönstermatchning
  - Iteratorer
  - Kostnadsfria abstraktioner (zero-cost abstractions)
  - Felhantering med resultat-typ

```
match "not_a_number".parse::<i32>() {
    Ok(result) => println!("Result: {}", result),
    Err(e) => println!("Error: {}", e),
}
```

Utvecklarergonomi

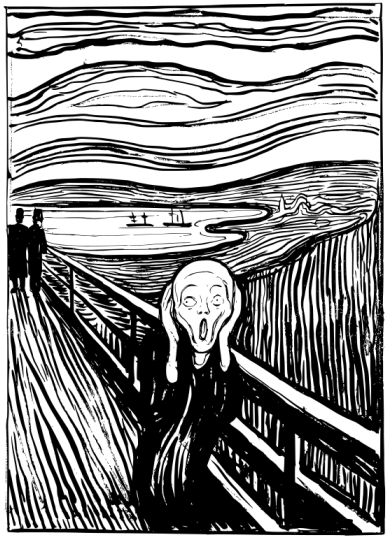
- Modernt språk
  - Slutledning av typer (type inference)
  - Genomtänkta “defaults”
  - Enum är “summa-typ”
  - Mönstermatchning
  - Iteratorer
  - Kostnadsfria abstraktioner (zero-cost abstractions)
  - Felhantering med resultat-typ
- “Bygger det så funkar det”

Utvecklarergonomi

- Modernt språk
  - Slutledning av typer (type inference)
  - Genomtänkta “defaults”
  - Enum är “summa-typ”
  - Mönstermatchning
  - Iteratorer
  - Kostnadsfria abstraktioner (zero-cost abstractions)
  - Felhantering med resultat-typ
- “Bygger det så funkar det”
- Hjälpsamma kompileringsfel

Utvecklarergonomi

- Modernt språk
  - Slutledning av typer (type inference)
  - Genomtänkta “defaults”
  - Enum är “summa-typ”
  - Mönstermatchning
  - Iteratorer
  - Kostnadsfria abstraktioner (zero-cost abstractions)
  - Felhantering med resultat-typ
- “Bygger det så funkar det”
- Hjälpsamma kompileringsfel
- Pakethantering



Utvecklarergonomi

- Modernt språk
  - Slutledning av typer (type inference)
  - Genomtänkta “defaults”
  - Enum är “summa-typ”
  - Mönstermatchning
  - Iteratorer
  - Kostnadsfria abstraktioner (zero-cost abstractions)
  - Felhantering med resultat-typ
- “Bygger det så funkar det”
- Hjälpsamma kompileringsfel
- Pakethantering

Utvecklarergonomi

- Modernt språk
  - Slutledning av typer (type inference)
  - Genomtänkta “defaults”
  - Enum är “summa-typ”
  - Mönstermatchning
  - Iteratorer
  - Kostnadsfria abstraktioner (zero-cost abstractions)
  - Felhantering med resultat-typ
- “Bygger det så funkar det”
- Hjälpsamma kompileringsfel
- Pakethantering
- Verktyg

Utvecklarergonomi

- Modernt språk
  - Slutledning av typer (type inference)
  - Genomtänkta “defaults”
  - Enum är “summa-typ”
  - Mönstermatchning
  - Iteratorer
  - Kostnadsfria abstraktioner (zero-cost abstractions)
  - Felhantering med resultat-typ
- “Bygger det så funkar det”
- Hjälpsamma kompileringsfel
- Pakethantering
- Verktyg
- Enkelt att korskompilera

Mindre bra

- Nytt, annorlunda, kräver studier

Mindre bra

- Nytt, annorlunda, kräver studier
- Gyttjebrottning med borrow checker

Mindre bra

- Nytt, annorlunda, kräver studier
- Gyttjebrottning med borrow checker
- Kompileringstider något sämre

Mindre bra

- Nytt, annorlunda, kräver studier
- Gyttjebrottning med borrow checker
- Kompileringstider något sämre
- Rör sig så snabbt att det kan vara svårt att hänga med

Mindre bra

- Nytt, annorlunda, kräver studier
- Gyttjebrottning med borrow checker
- Kompileringstider något sämre
- Rör sig så snabbt att det kan vara svårt att hänga med
- C++-interoperabilitet möjlig men långt från perfekt

Hur införa Rust?

- Börja stegvis
  - Uppmuntra gräsrotsinitiativ
  - Kompetensutveckling
  - Porta perifert verktyg
  - Ta chansen i ny produkt om möjligt

Hur införa Rust?

- Börja stegvis
  - Uppmuntra gräsrotsinitiativ
  - Kompetensutveckling
  - Porta perifert verktyg
  - Ta chansen i ny produkt om möjligt
- Befintlig kodbas
  - Nya komponenter
  - Prestandakritiska moduler
  - Håll gränssnittet litet

Hur införa Rust?

- Börja stegvis
  - Uppmuntra gräsrotsinitiativ
  - Kompetensutveckling
  - Porta perifert verktyg
  - Ta chansen i ny produkt om möjligt
- Befintlig kodbas
  - Nya komponenter
  - Prestandakritiska moduler
  - Håll gränssnittet litet
- Använd AI-assistenter

Hur införa Rust?

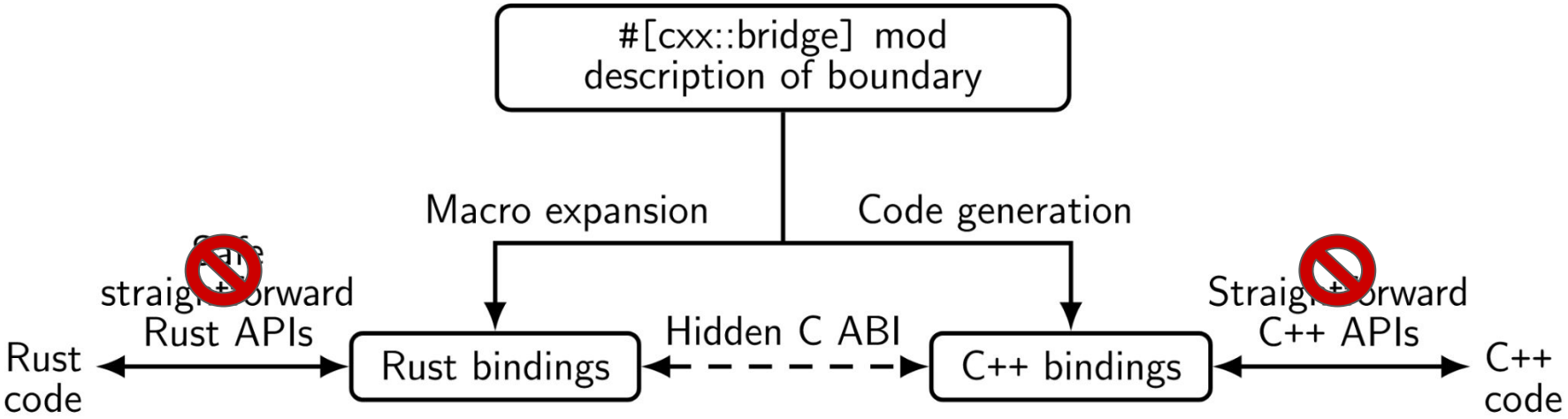
- Börja stegvis
  - Uppmuntra gräsrotsinitiativ
  - Kompetensutveckling
  - Porta perifert verktyg
  - Ta chansen i ny produkt om möjligt
- Befintlig kodbas
  - Nya komponenter
  - Prestandakritiska moduler
  - Håll gränssnittet litet
- Använd AI-assistenter
- Ta hjälp

Interoperabilitet med C++

- Bygga Rust som `static_lib`
- Flera verktyg, inget är perfekt
  - bindgen
  - cpp!
  - cxx
  - autocxx

Interoperabilitet med C++

- Bygga Rust som `static_lib`
- Flera verktyg, inget är perfekt
  - bindgen
  - cpp!
  - cxx
  - autocxx



Frågor