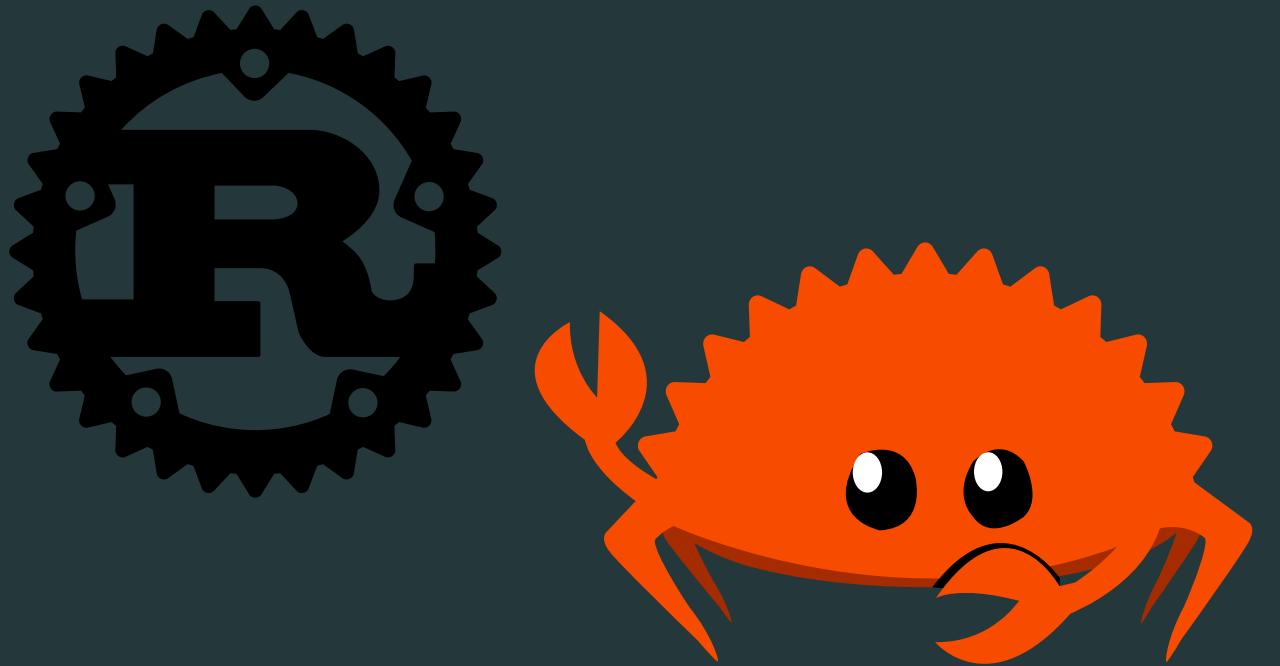


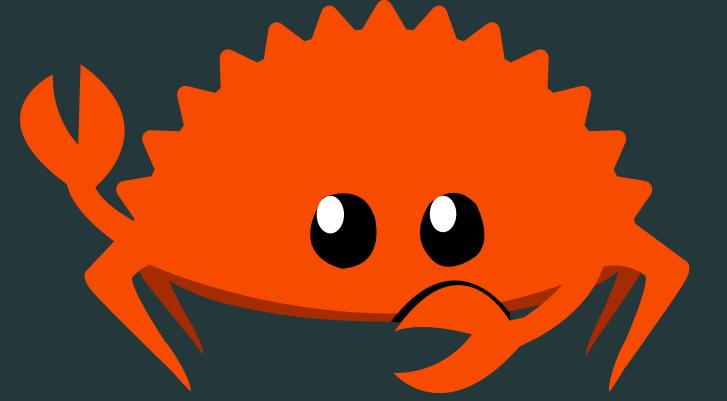
招待講演

Rust から広がる プログラム検証・テストの新展望

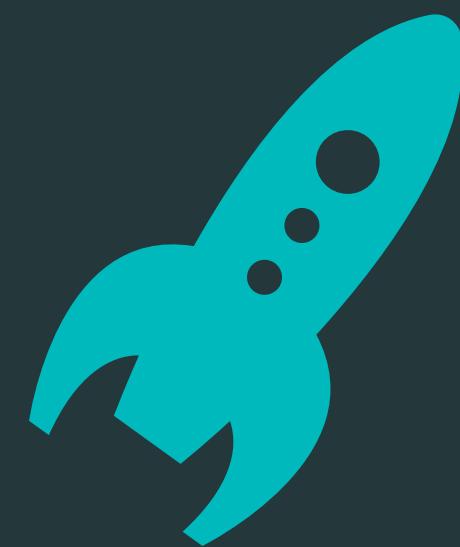
松下 祐介 京都大学 情報学研究科 五十嵐・末永研

2024.6.13 情報処理学会 プログラミング研究発表会





Rust は人類を
ソフトウェア開発の
新時代へと導く

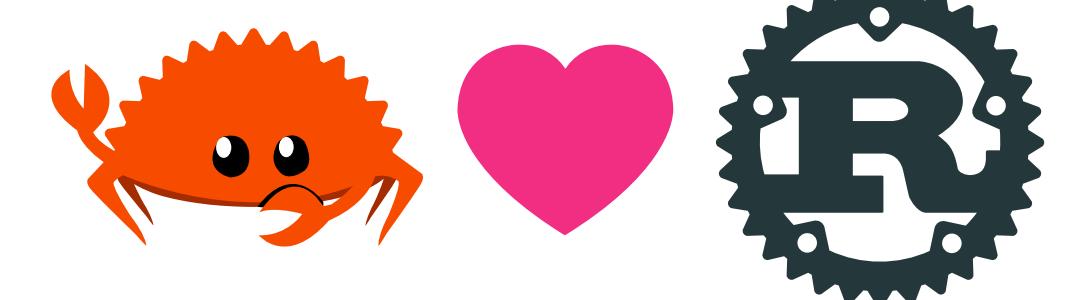


自己紹介



PLDI 2022 にて

松下 祐介 ソフトウェア科学者



現在 京都大学 情報学研究科
学振PD研究員 @五十嵐・末永研

2024 東大院 情報理工 CS専攻 博士号取得 @小林研

2019 東大 理学部情報科学科 卒業 @小林研
卒業研究 **RustHorn** 

現実のソフトウェア開発に役立つ科学的アイデア

この講演

Rust

預言

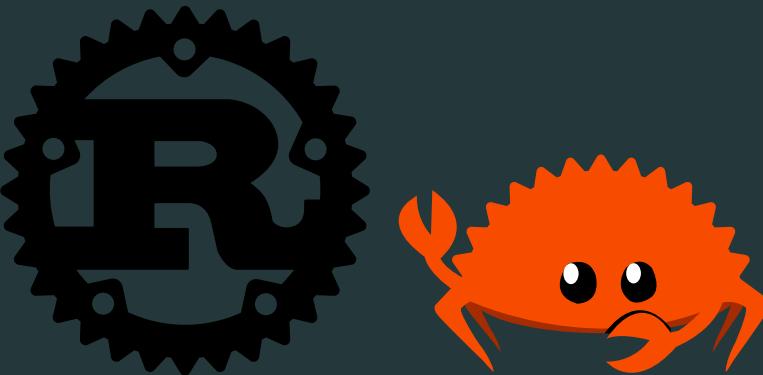
所有権

未来

白熱する

Rust プログラミング言語

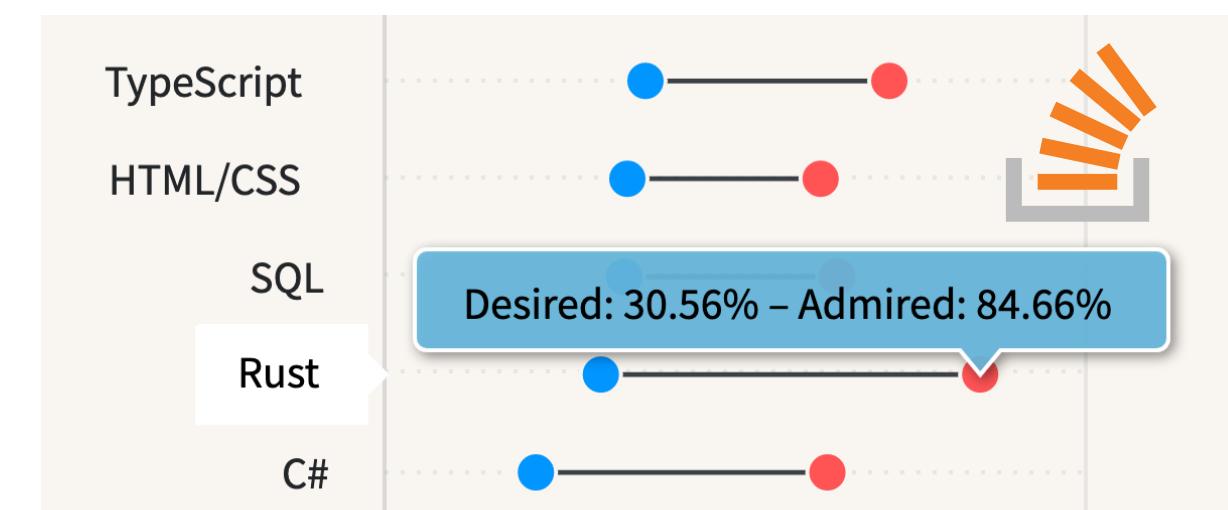
Rust プログラミング言語が今熱い



効率的で信頼できるソフトウェアを
誰もがつくれる言語

はじめる

[バージョン 1.78.0](#)

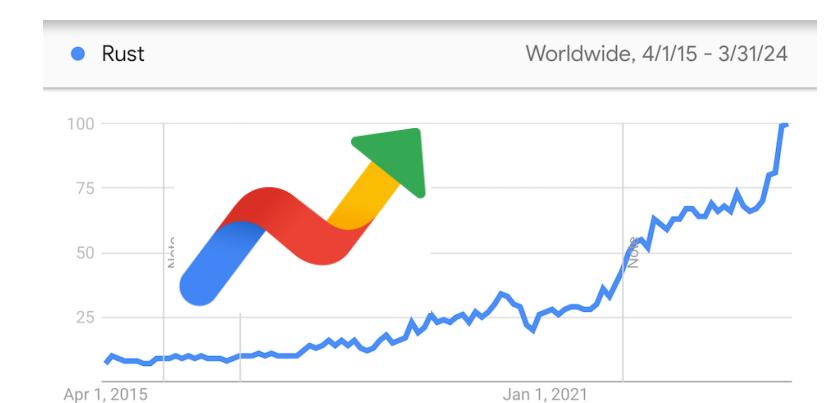


<https://survey.stackoverflow.co/2023/>

8年連続 最も“愛される”言語

<https://www.rust-lang.org/ja/>

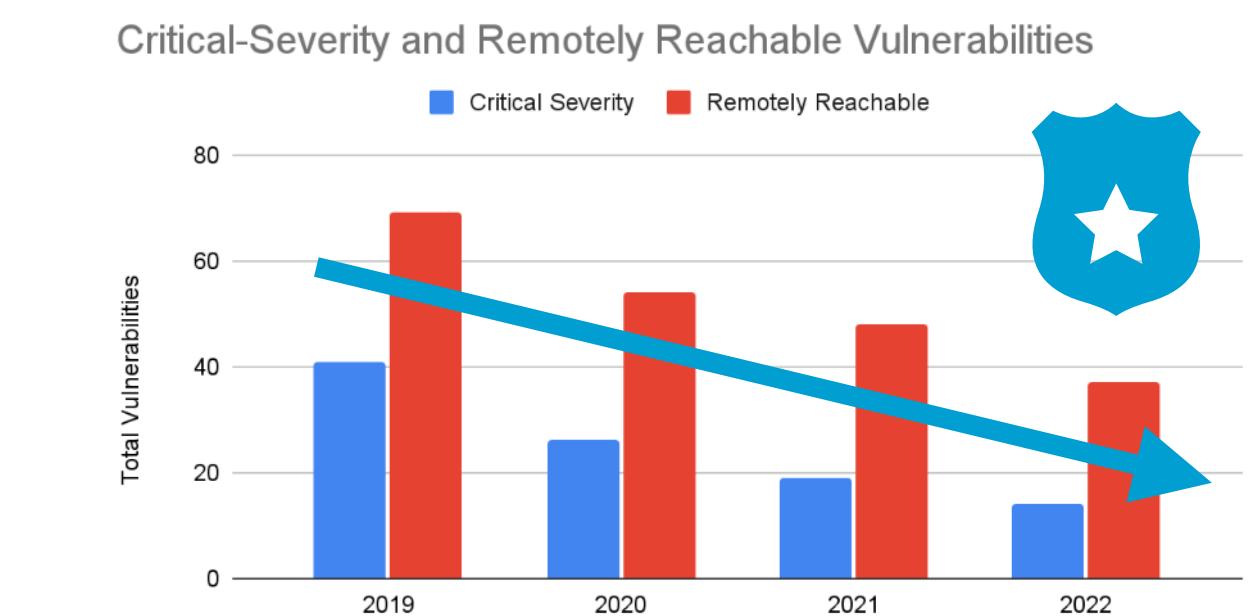
高性能 × 安全性 × 開発効率



<https://trends.google.com/trends/explore?date=2015-04-01%202024-03-31&q=%2Fm%2F0dsbp6>

年々高まる Rustへの関心

Rust がもたらす安全性



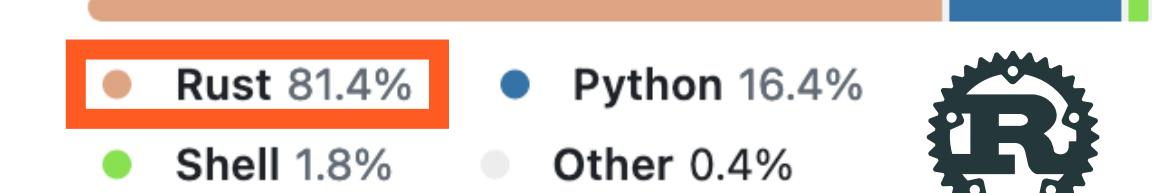
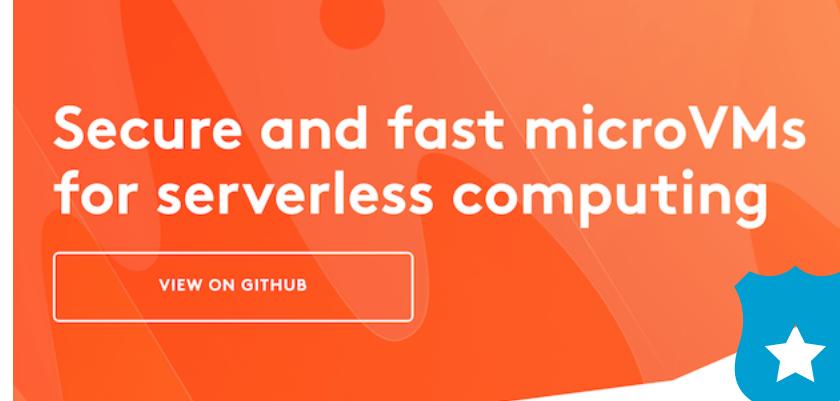
<https://security.googleblog.com/2022/12/memory-safe-languages-in-android-13.html>

Android は Rust を採用 → 脆弱性減

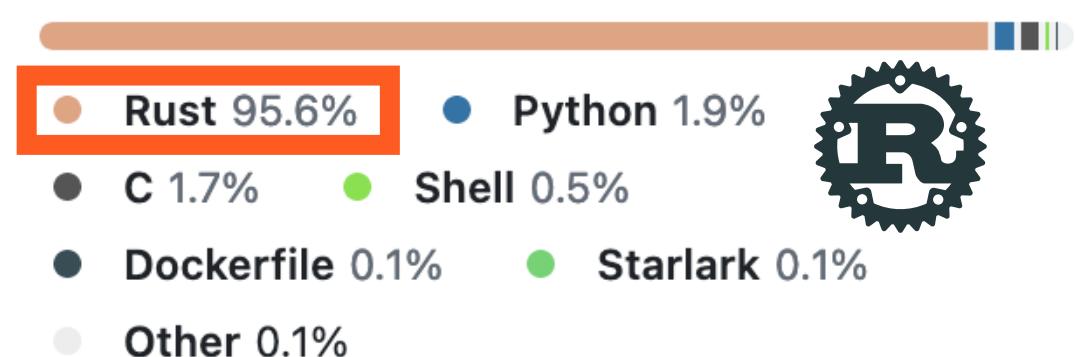
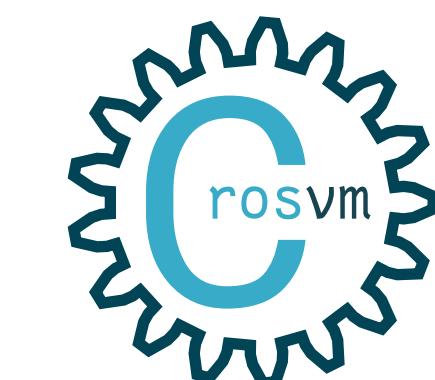


aws

Firecracker



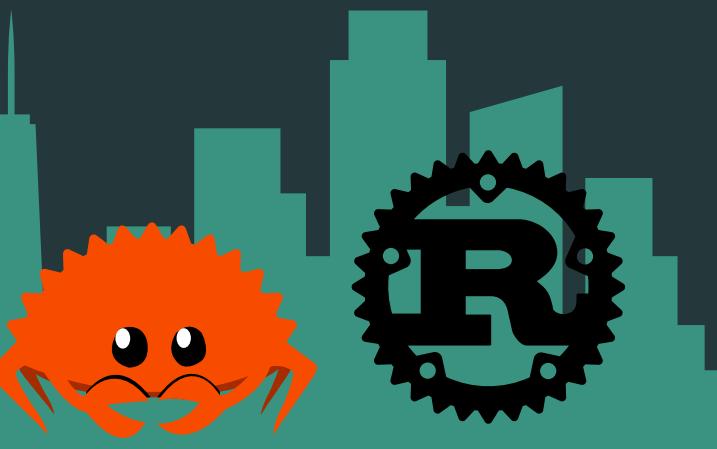
<https://firecracker-microvm.github.io/>



<https://github.com/google/crosvm>

現代の VM は Rust で開発 → 安全

Rust 製の色々なソフトウェア



<https://www.mozilla.org/en-US/firefox/new/>



<https://servo.org/>

Web ブラウザ描画エンジン



<https://github.com/sharkdp/bat>

コマンドラインツール



Deno

<https://deno.com/>

JavaScript 処理系



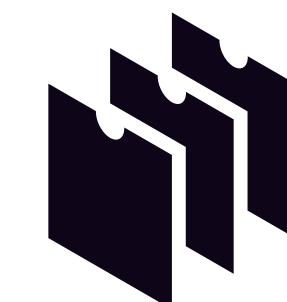
<https://datafusion.apache.org/>

SQL 処理系



<https://sui.io/>

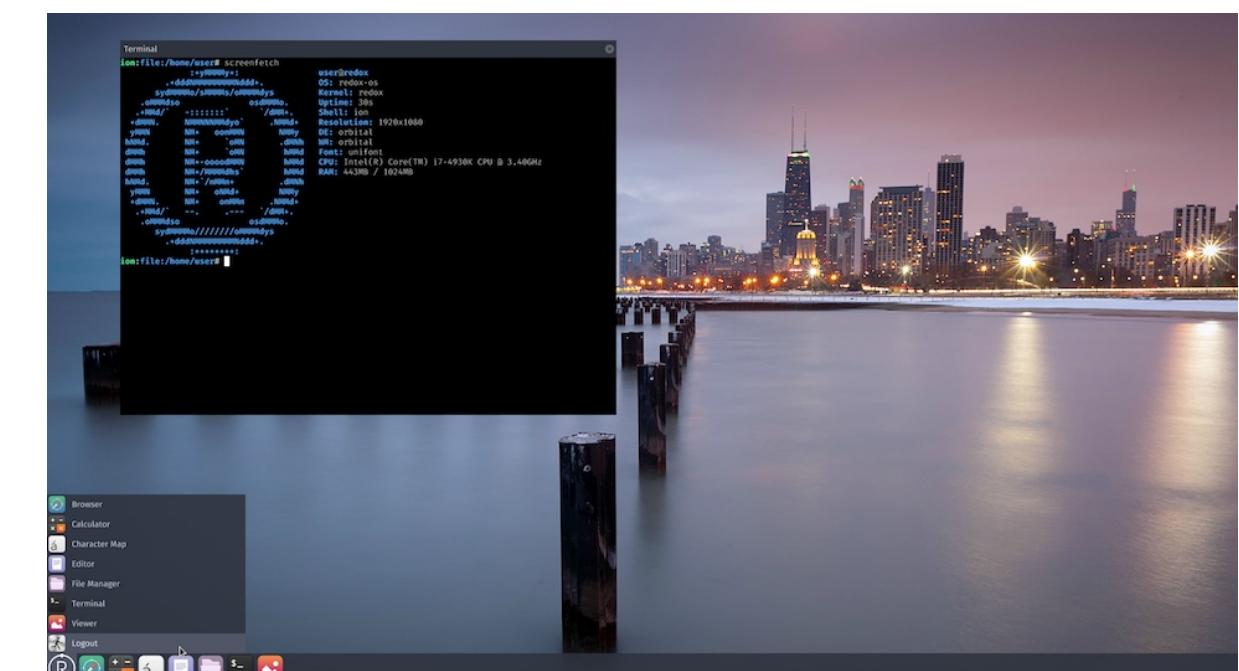
スマートコントラクト



Wasmer

<https://wasmer.io/>

Wasm 処理系



<https://www.redox-os.org/>

オペレーティングシステム

産業の Rust への期待



<https://foundation.rust-lang.org/>

Rust 財団 2021年誕生



世界のビッグテックが出資

Google Contributes \$1M to Rust Foundation to Support C++/Rust “Interop Initiative”

Google's generous contribution of \$1M USD will help the Rust Foundation make meaningful contributions to improve the state of interoperability between the Rust and C++ programming languages.

Google が C++/Rust 相互運用の改善に向けて100万ドルを寄付

2024.2

<https://foundation.rust-lang.org/news/google-contributes-1m-to-rust-foundation-to-support-c-rust-interop-initiative/>

\$1M Microsoft Donation to Fund Key Rust Foundation & Project Priorities

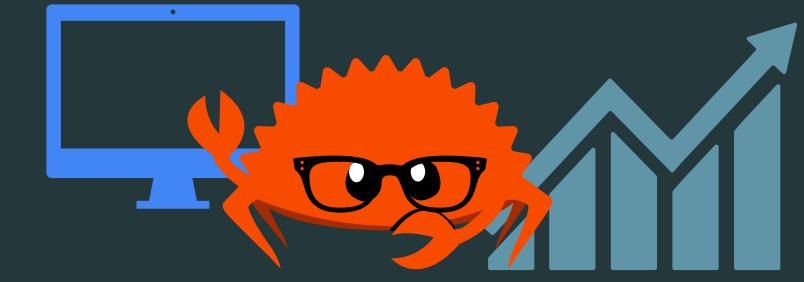
Announcing the allocation of a recent \$1M donation from Platinum Member Microsoft to support Rust Foundation and Rust Project priorities .

Microsoft が Rust の課題全般の解決に向けて100万ドルを寄付

2024.5

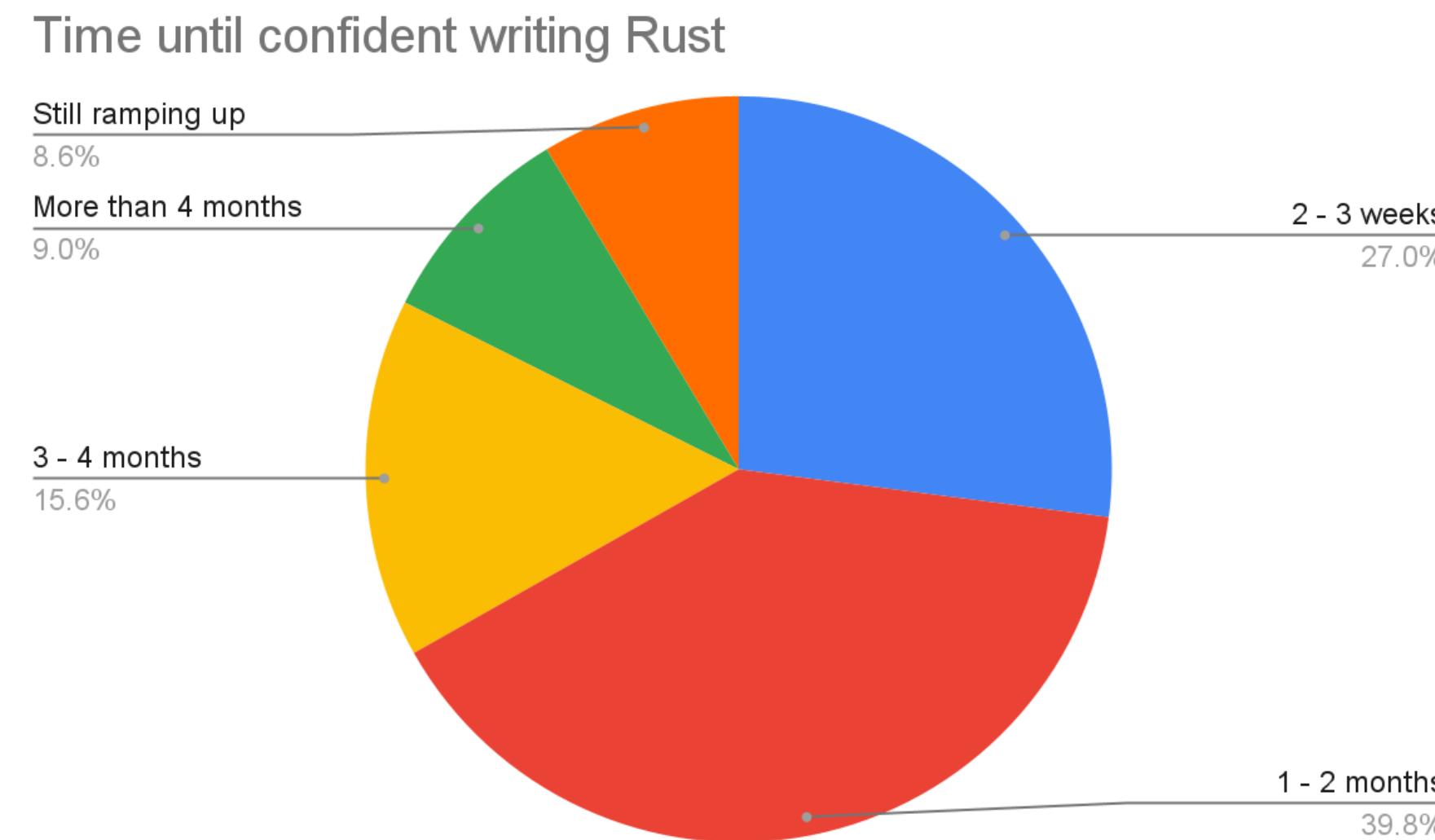
<https://foundation.rust-lang.org/news/1m-microsoft-donation-to-fund-key-rust-foundation-project-priorities/>

Rustに対する現場の声



1,000人以上の Google エンジニアに対するサーベイ

<https://opensource.googleblog.com/2023/06/rust-fact-vs-fiction-5-insights-from-googles-rust-journey-2022.html>



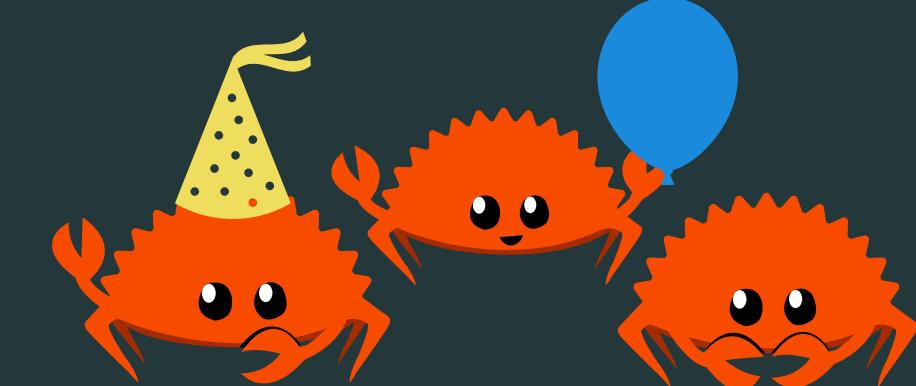
2ヶ月で 2/3以上が Rust 開発に自信

4ヶ月で 1/2以上が Rust で
他の言語より生産性を感じた

8割以上が Rust で
他の言語より正しさに自信

1/2以上が Rust は
レビューが非常に簡単と感じた

活発な Rust コミュニティ



Leadership council

Charged with the success of the Rust Project as whole,
consisting of representatives from top-level teams

Members

	Eric Holk GitHub: eholk Compiler team
	Jack Huey GitHub: jackh726 Language team
	Mara Bos GitHub: m-ou-se Library team
	Josh Gould GitHub: technetos Moderation team



<https://www.rust-lang.org/community>

熱心なコントリビュータ達

Language team

Designing and helping to implement new language features

Members

	Eric Huss GitHub: ehuss Dev tools team
	James Munns GitHub: jamesmunns Launching pad
	Mark Rousskov GitHub: Mark-Simulacrum Infrastructure team
	Niko Matsakis GitHub: nikomatsakis Team leader
	Josh Triplett GitHub: joshtipton
	Scott McMurray GitHub: scottmcm
	Tyler Mandry GitHub: tmandry Team leader
	Felix Klock GitHub: pnkfelix

Closed Resolve `await` syntax #57640
cramertj opened this issue on Jan 16, 2019 · 512 comments

mzji commented on Jan 18, 2019 · edited

Could postfix macro (i.e. `future.await!()`) still be an option? It's clear, concise, and unambiguous:

Future	Future of Result	Result of Future
<code>future.await!()</code>	<code>future.await!()</code> ?	<code>future?.await!()</code>

Also postfix macro requires less effort to be implemented, and is easy to understand and use.

36 23

chpio commented on Jan 18, 2019 · edited

Also postfix macro requires less effort to be implemented, and is easy to understand and use.

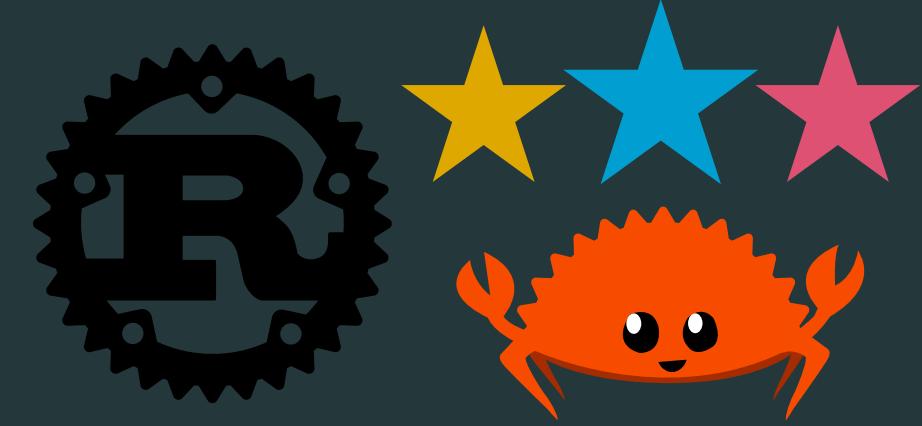
Also it's just using a [common lang feature](#) (or at least it would look like a normal postfix macro).

novacrazy commented on Jan 18, 2019 · edited

<https://github.com/rust-lang/rust/issues/57640>

言語設計についてオープンな議論

Rust はなぜ人気か



高性能 ⏳

直接のメモリ操作

GCがデフォルトでない

直接の並行制御

同期方法の指定

自由なメモリ順序

C/C++ と比べて

安全性 🛡️

実用的な所有権型

メモリ/スレッド安全性
の軽量な自動検証

借用・unsafe・
内部可変性で柔軟に

同等に高性能

圧倒的に安全

かなり快適

開発効率 📈

モダンな言語機能

パターンマッチ トレイト
衛生的マクロ クロージャ

便利な開発ツール

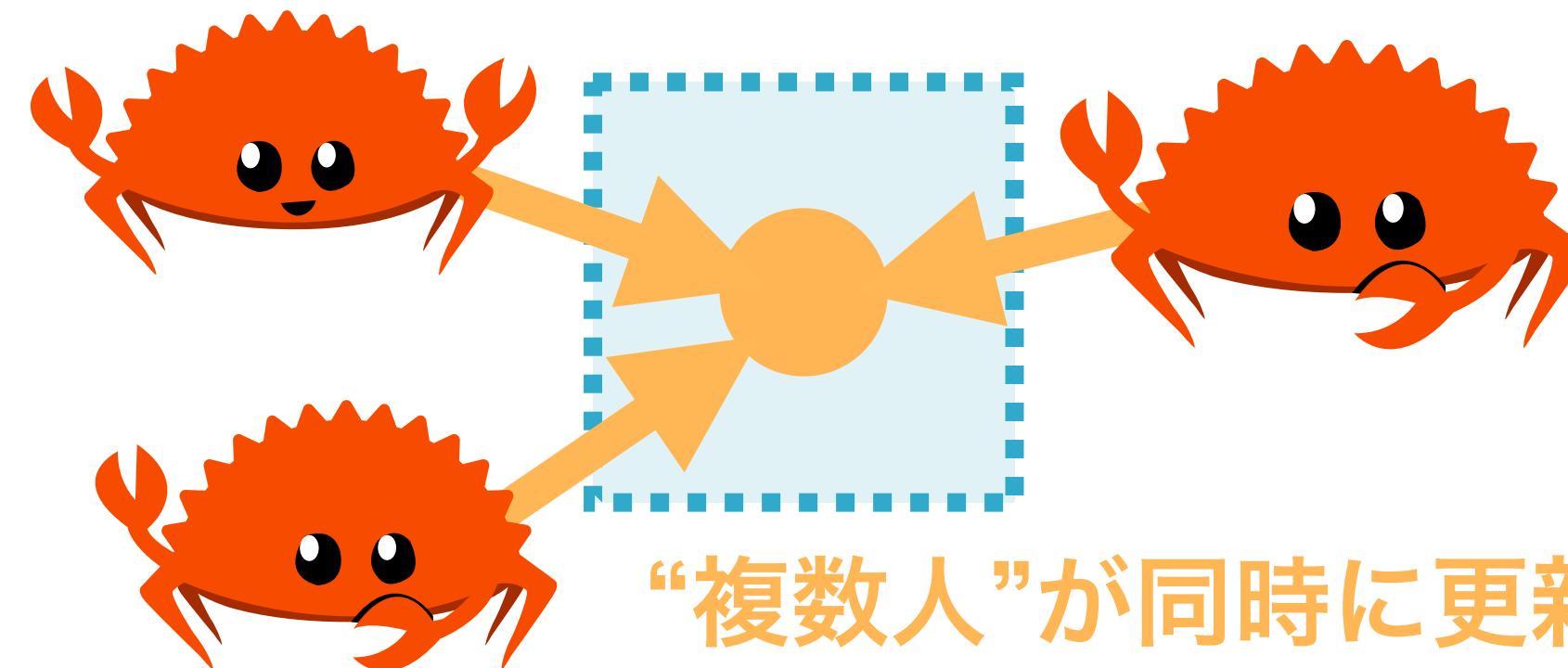
パッケージマネージャ
テスト自動化 IDE 連携

Rust は
所有権型を実用化 &
現実の開発を革新

Rust が広めた 所有権型の考え方

背景 メモリは難しい

記憶装置=メモリ



“複数人”が同時に更新

アクセス競合

ある箇所での更新が
予期せぬ影響をもたらす

メモリは深刻な**脆弱性**の原因

Use after free ダングリングポインタ
バッファオーバーラン/リード

CVE	Vendor	Product	Type
CVE-2019-7286	Apple	iOS	Memory Corruption
CVE-2019-7287	Apple	iOS	Memory Corruption
CVE-2019-0676	Microsoft	Internet Explorer	Information Leak
ZERO 2019-5786	Google	Chrome	Memory Corruption
CVE-2019-0808	Microsoft	Windows	Memory Corruption
CVE-2019-0797	Microsoft	Windows	Memory Corruption

<https://googleprojectzero.blogspot.com/p/0day.html>

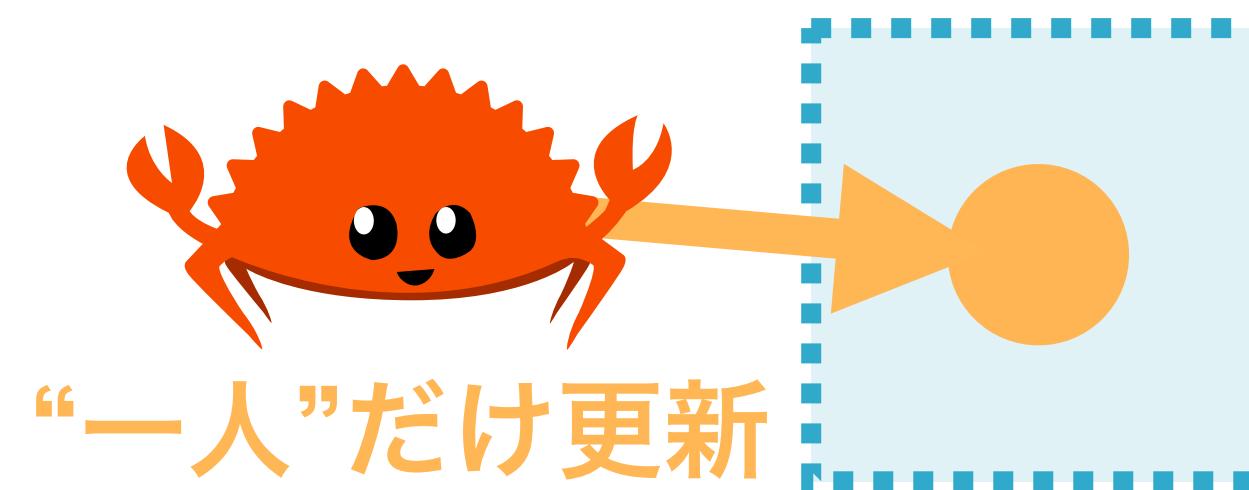
ゼロデイ攻撃の約7割がメモリ破壊

共有 xor 可変 – 所有権のもたらす保証

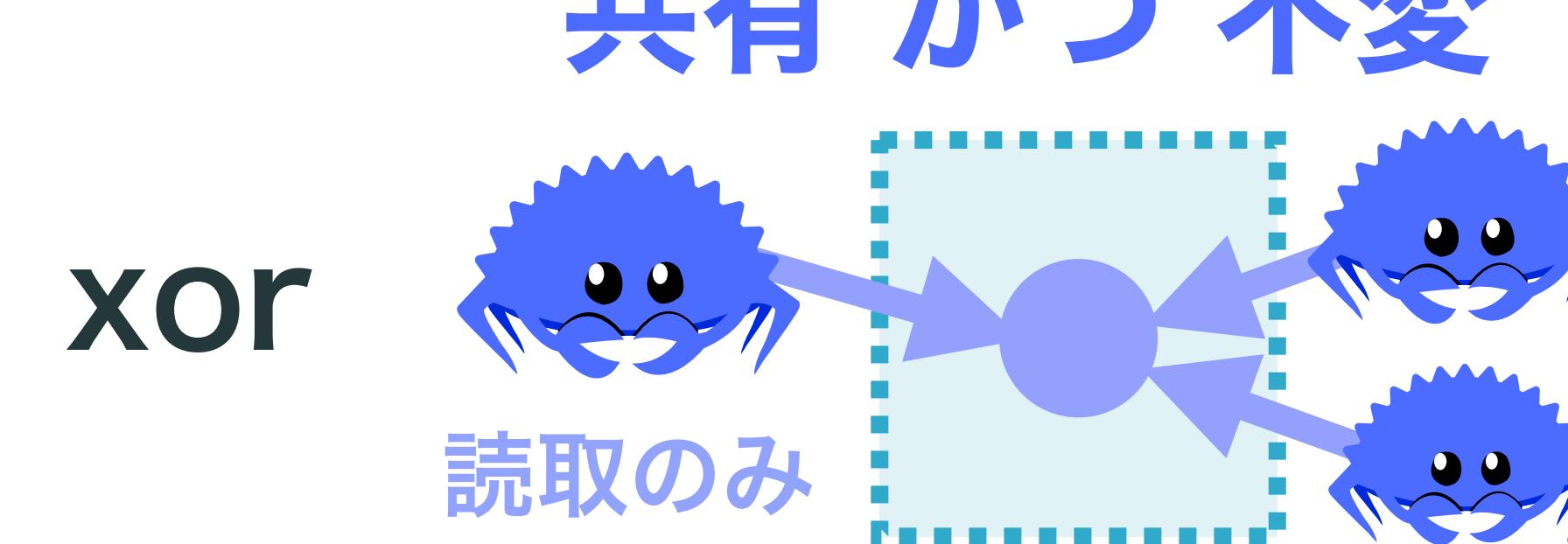
所有権 = メモリの特定領域へのアクセス権

各領域について常に

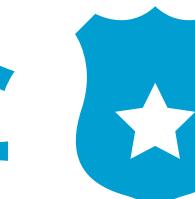
専有かつ可変



共有かつ不变



アクセス競合がない 安全



昔から所有権の動的検査はごく一般的

Mutex [Dijkstra '65] • Reader-Writer ロック [Courtois+ '72]

Rust の所有権型

型に所有権情報 共有 xor 可変 を静的に検証

アクセス競合がない メモリ/スレッド安全 

プログラマに優しい 自動 & 軽量 の静的検証

アカデミックな研究

リージョン型 [Tofte&Talpin '97]

所有権型 [Clarke+ '98]

Cyclone [Grossman+ PLDI'02]

独自進化



初めて本格的に実用化



Rust

2015~

ソフトウェア科学の画期

成功の鍵

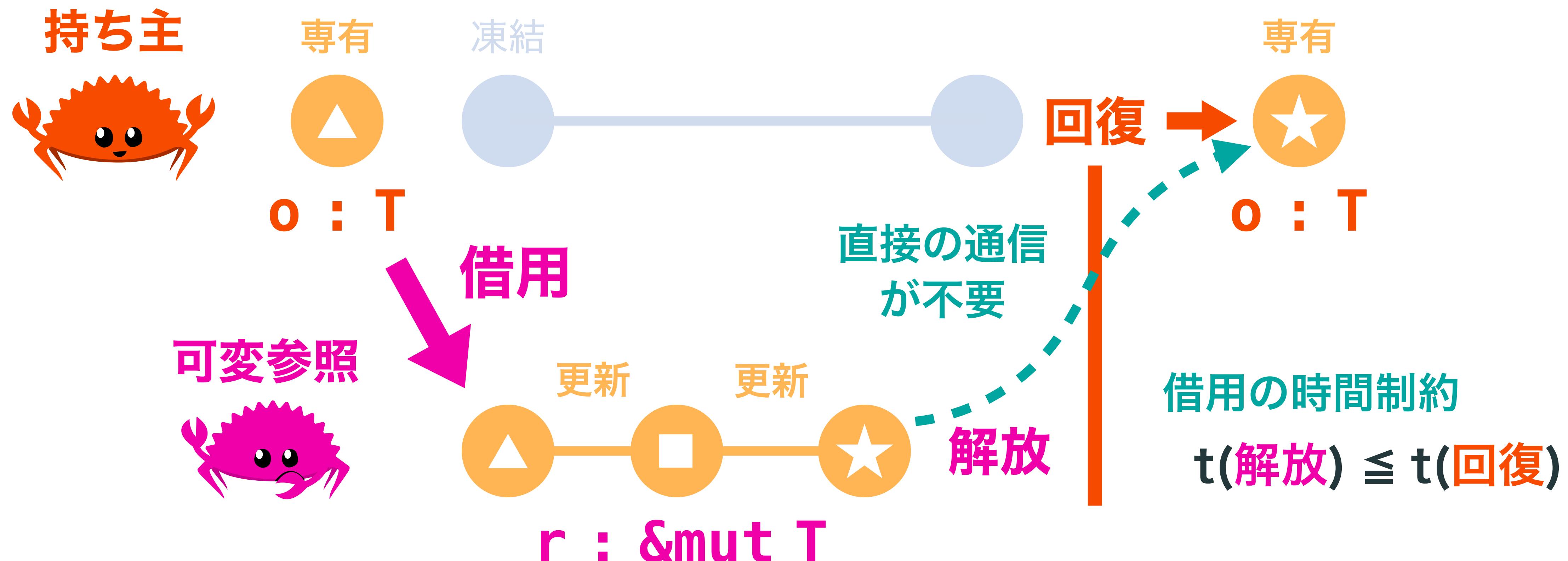
借用

unsafe

内部可変性

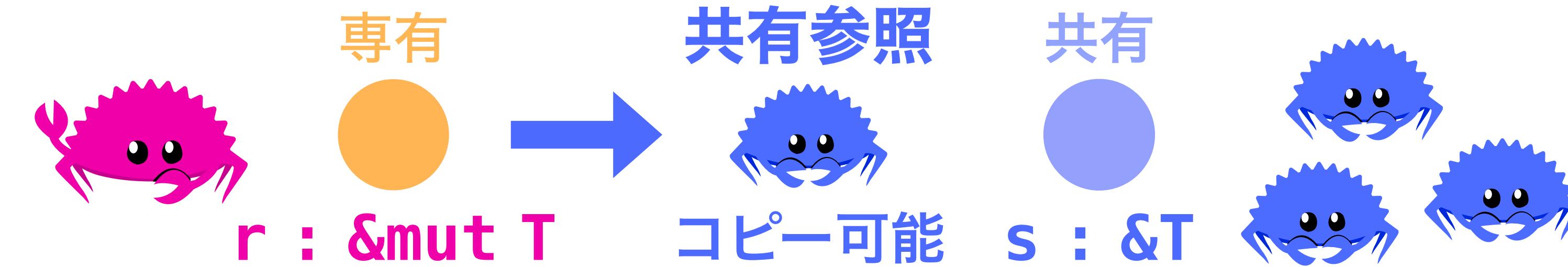
Rust の鍵#1 借用の基本アイデア

借用 = 所有権を一時的に借りること

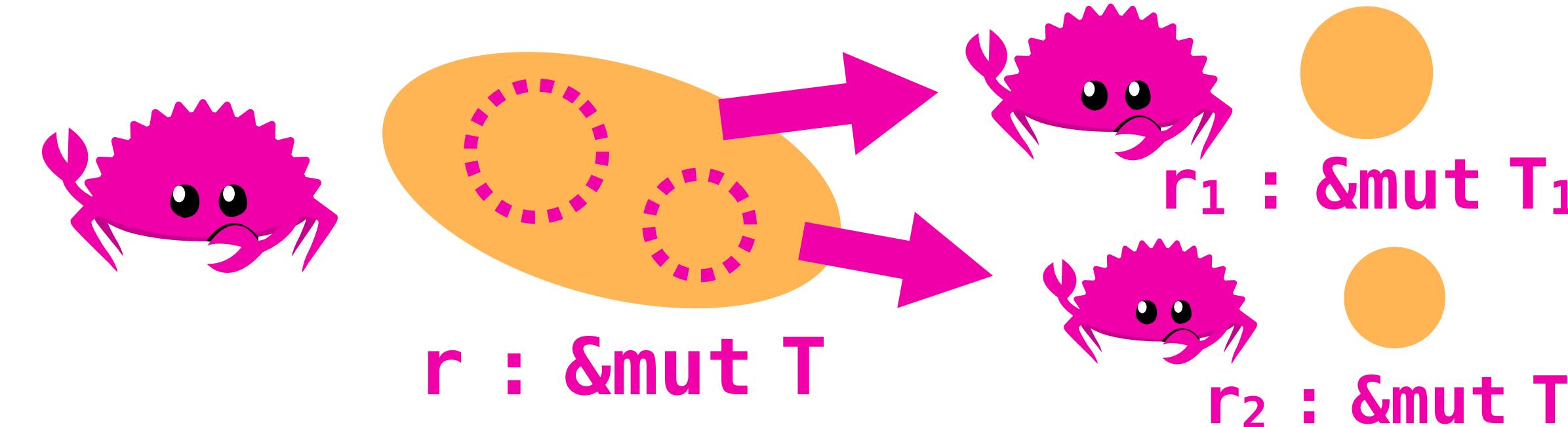


Rust の鍵#1 借用の色々なパターン

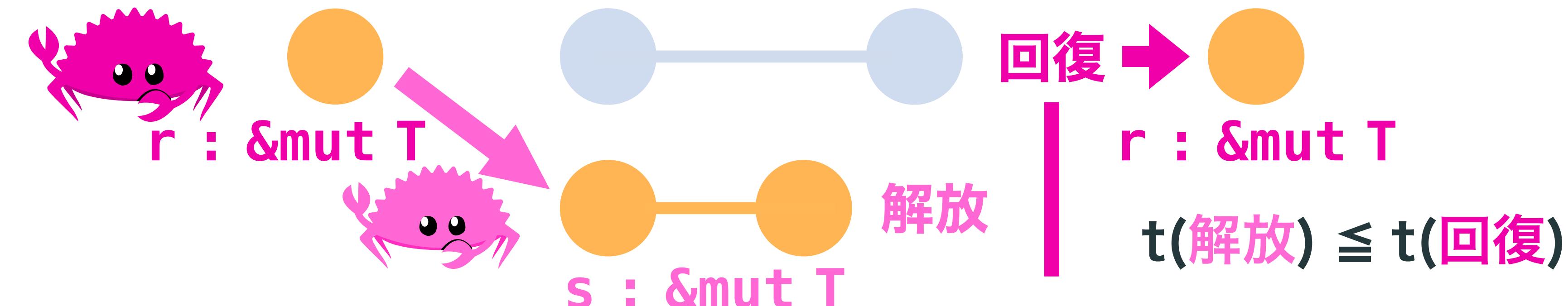
共有借用



借用の分割



再借用



Rust の鍵#1 借用検査

借用検査 = 借用の制約の自動静的検証

発展途上

特に $t(\text{解放}) \leq t(\text{回復})$

前世代

スコープ基準

$t(\text{解放}) = \text{スコープ終了}$

既存研究に直接由来

Cyclone [Grossman+ '02]

現世代

非字句的ライフタイム

$t(\text{解放})$ を生存解析で推論

ライフタイム = 連続する
プログラムポイントの区間



Niko Matsakis

次世代？

制御可能な借用検査

可変参照型で借用元を管理

自己借用への対応

```
fn new_widget(&self, name: String) -> Widget {  
    let name_suffix: &'name str = &name[3..];  
        // --- borrowed from "name"  
    let model_prefix: &'self.model str = &self.model[..2];  
        // ----- borrowed from "self.model"  
}
```

[https://smallcultfollowing.com/babysteps/blog/
2024/06/02/the-borrow-checker-within/](https://smallcultfollowing.com/babysteps/blog/2024/06/02/the-borrow-checker-within/)

Rust の鍵#2 unsafe による拡張

unsafe = 型検査で安全性を保証できない操作

特に 所有権検査を受けない 生ポインタ ***mut T** の操作

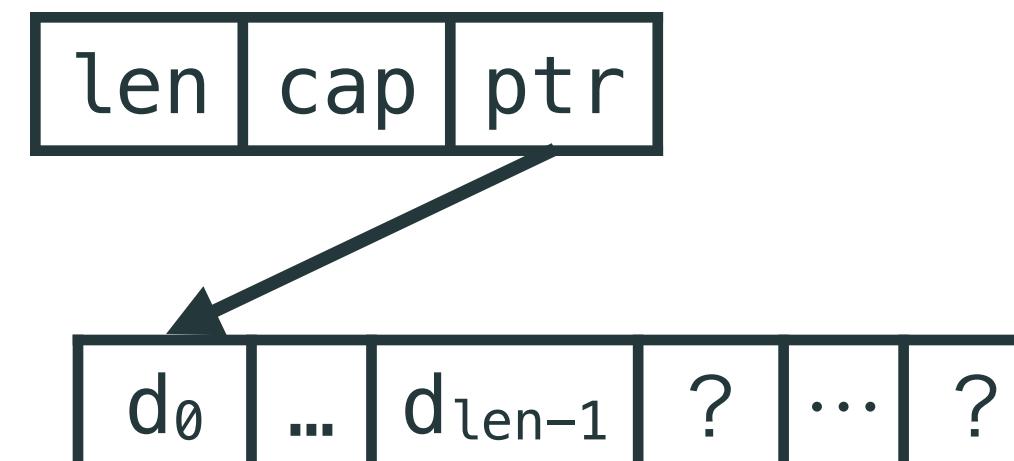
限界のある自動検査に対するバックドア

重要パターン **unsafe 実装を safe な型でカプセル化** 🔍

例

動的配列型

```
struct Vec<T> {  
    len : uint;  
    cap : uint;  
    ptr : *mut T;  
}
```



要素への可変参照を取得

```
fn index_mut(v : &mut Vec<T>,  
             i : uint) -> &mut T {  
    assert!(i < v.len); 動的な境界検査  
    unsafe { v.ptr.offset(i) } 🔍  
}
```

Rust の鍵#3 内部可変性で 共有 and 可変

内部可変性 = 共有中に制限つきで許される可変性

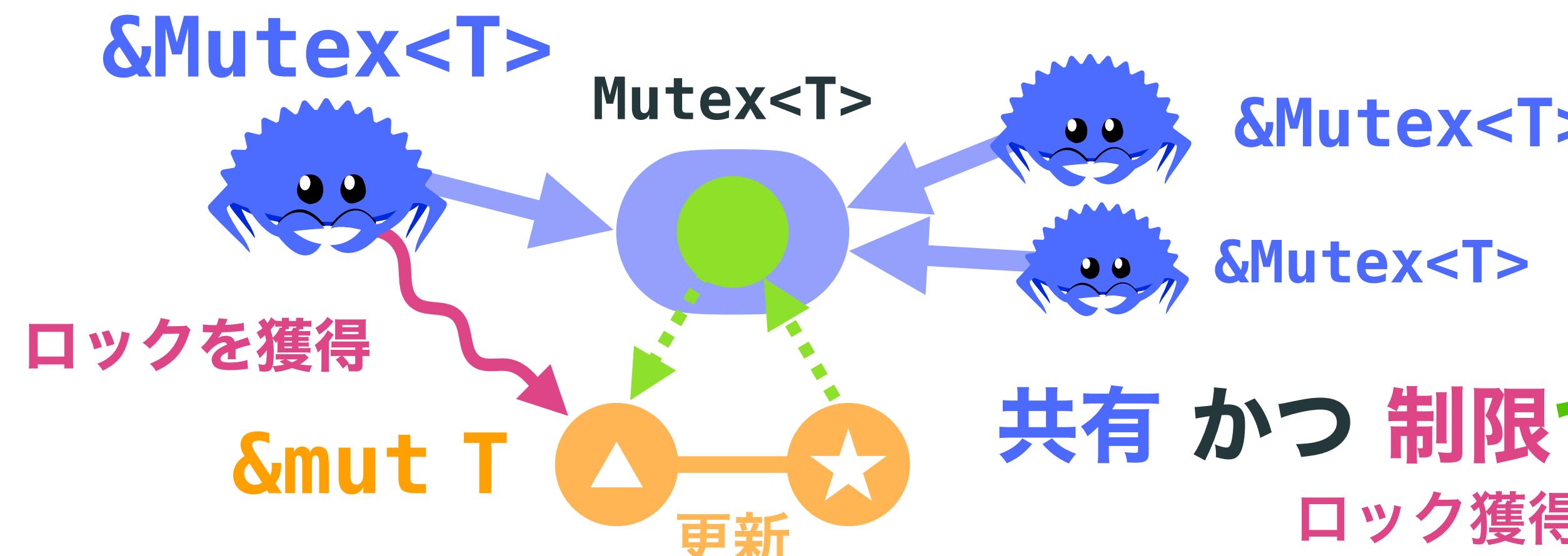
原則 共有中はデータが不变

`&int`
`&Vec<int>`

特定の型のみ内部可変性を持つ

`&Mutex<T>` `&RefCell<T>`
`&RwLock<T>` `&Cell<T>`

例



共有かつ制限つき可変
ロック獲得中 etc.

Rust 所有権型 まとめ

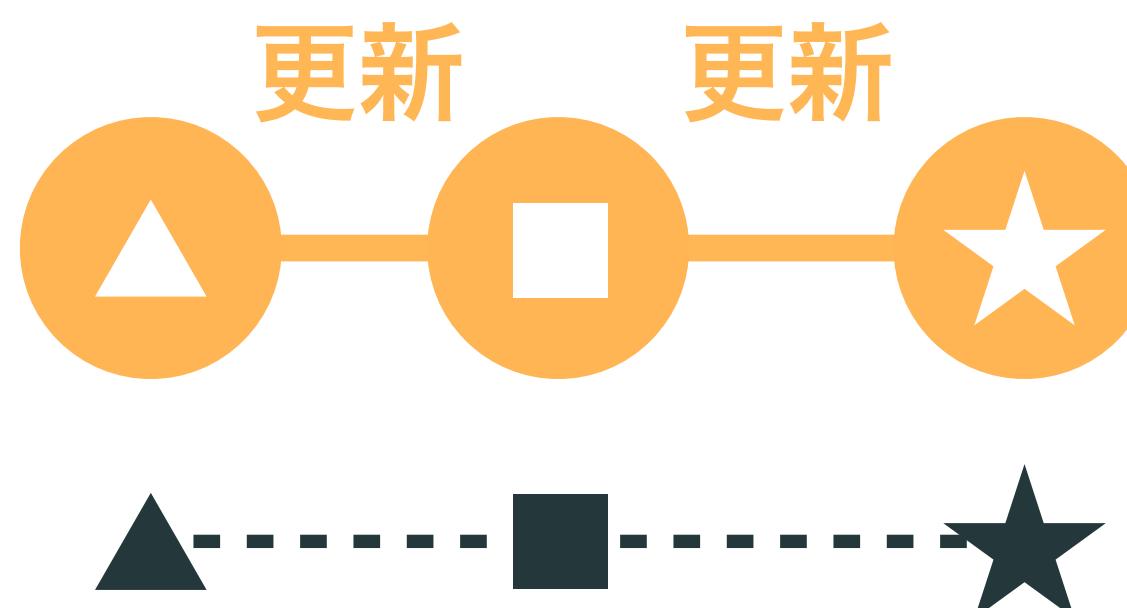
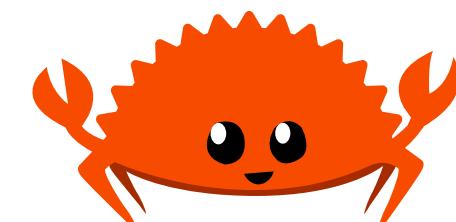
Rust の所有権型は
借用 & unsafe & 内部可変性
の力で柔軟な操作を実現

RustHorn が見つけた 預言のアイデア

所有権は世界を“関数型”にする

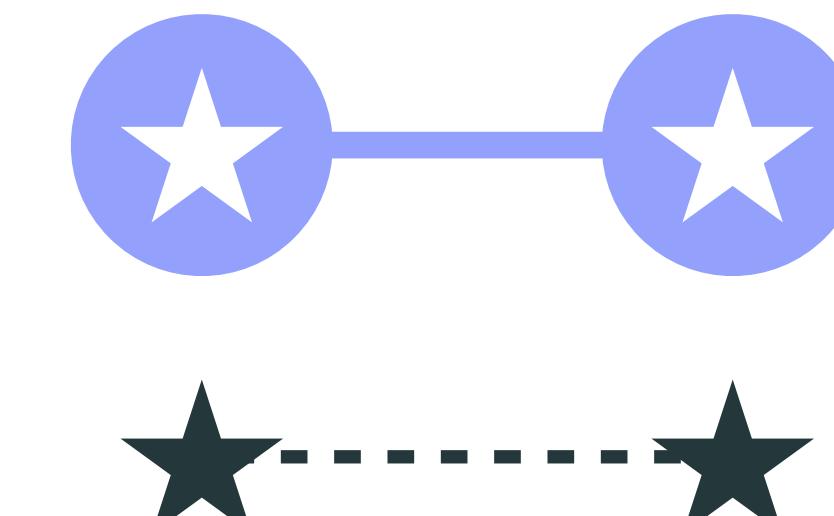
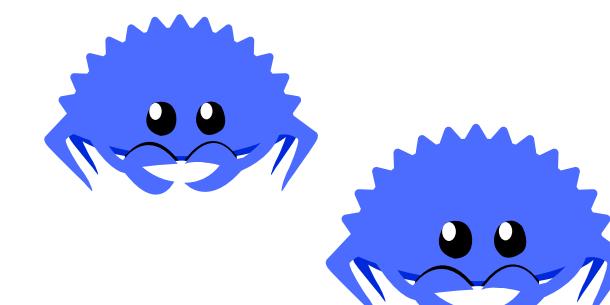
所有権
Rust プログラム → 関数型プログラム
様々なメモリ操作 メモリのない簡潔なモデル
効率的な検証

可変 だが 完全所有



値の変化をトラック可能

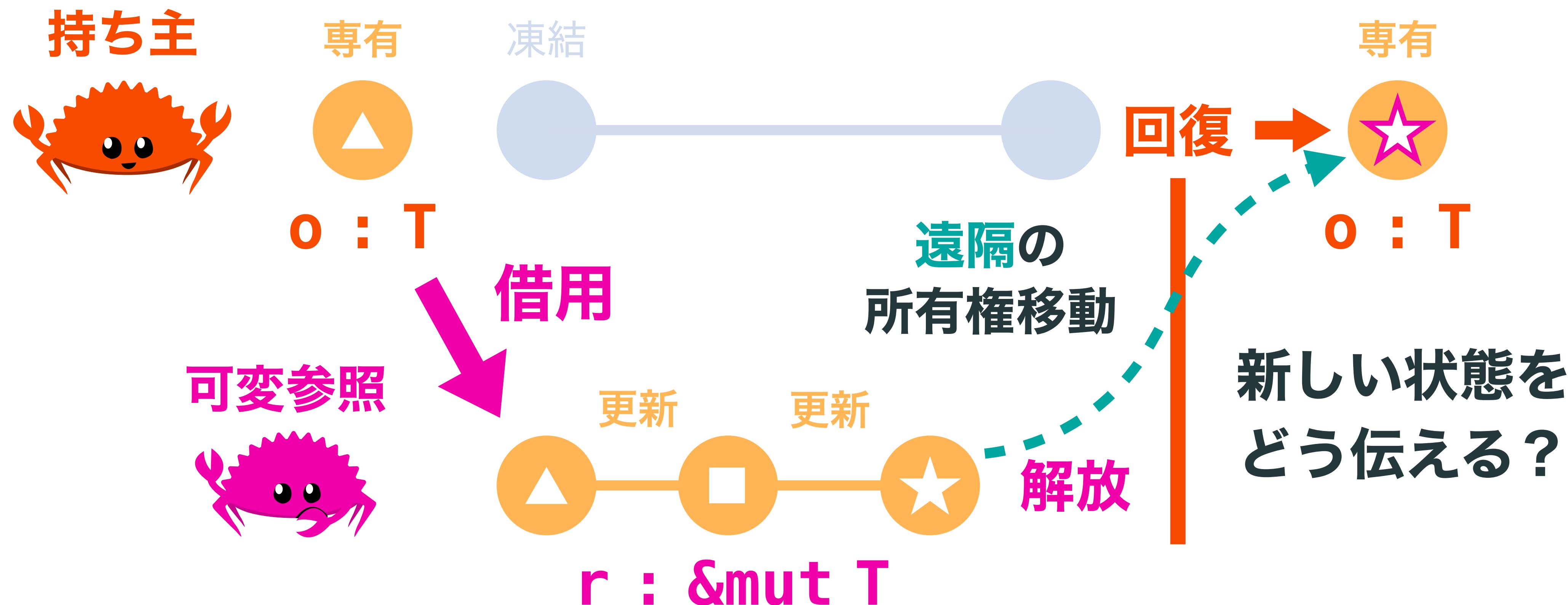
共有 だが 不変



単に不变な値として表現

借用の難しさ

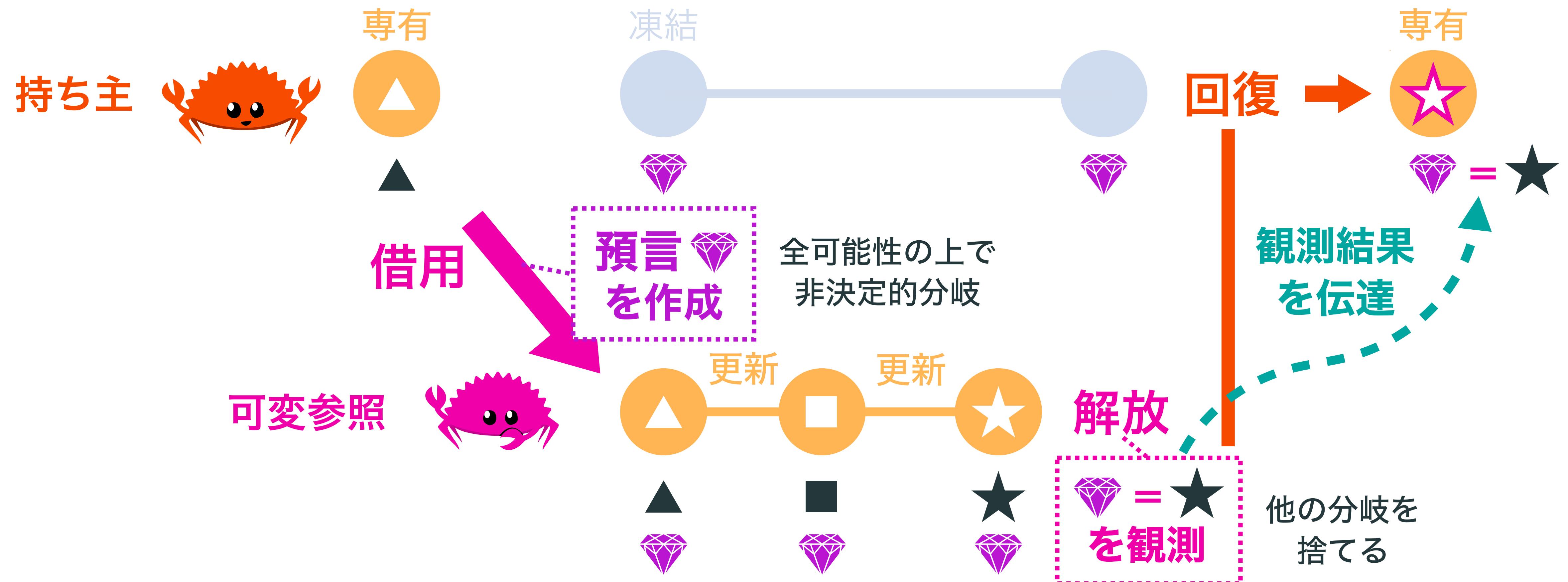
借用は遠隔の情報伝達を許すから難しい





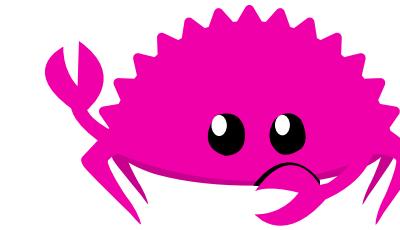
借用も一般に関数型で表現可能

最終的な値を“預言”



預言で与える借用操作の仕様

[True]

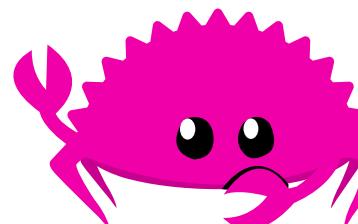


単純な更新

fn push(v : &mut Vec<T>, a : T)

[$\hat{v} = *v ++ [a]$]
預言

[$i < v.len$]



借用の分割

fn index_mut(v : &mut Vec<T>, i : uint) -> &mut T

[$*res = *v[i] \wedge \hat{v} = *v \{ i \leftarrow \hat{res} \}$]
大預言 小預言
get set



RustHorn と後継研究



全自动検証器 RustHorn

[松下+ ESOP'20/TOPLAS'21]

Rust → 制約付きホーン節
一階述語論理 + 最小不動点



半自動検証器 Creusot

<https://github.com/creusot-rs/creusot>

[Denis+ ICFEM'22]

注釈付き Rust → Why3



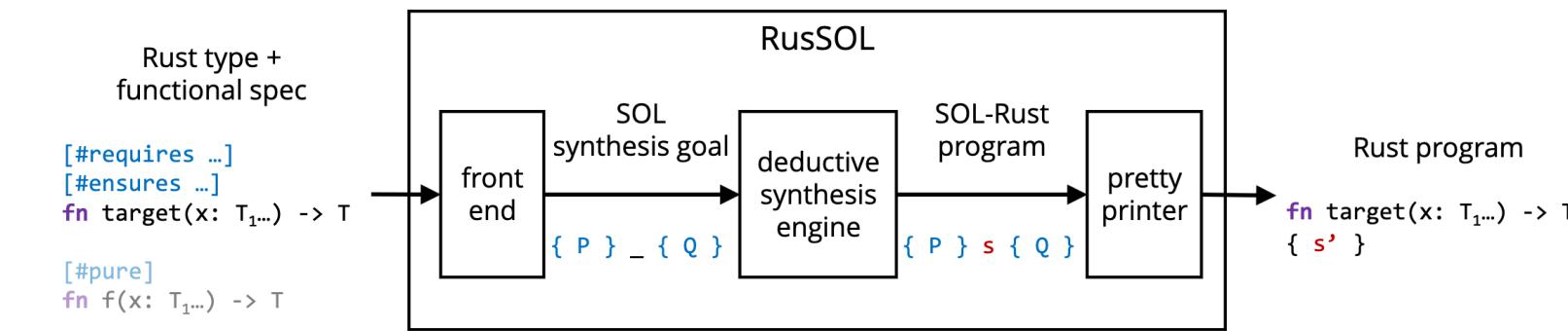
<https://gitlab.inria.fr/why3/why3>

応用例 Rust 製 SAT ソルバの検証

<https://github.com/sarsko/CreuSAT>

プログラム生成器 RusSOL

[Fiala+ PLDI'23]



関数的仕様 → Rust



生成



意味論的基礎 RustHornBelt

[松下+ PLDI'22 卓越論文賞]

預言による表現の正しさを
分離論理 Iris 上の意味論で広く証明

Coq で機械化



預言の力で借用は
綺麗な関数型の世界で
表現できる

Rust から広がる テスト・検証の現在と未来

安全性のためのテスト・検証

ソフトウェアの**安全性**は現実的な問題

バグの発見 — **厳密な証明** の両方が大事
テスト 検証

ファジング

有界モデル検査

実行時検証

演繹的検証

ストレステスト

コンコリックテスト

静的型検査

性質ベーステスト

記号実行

モデル検査

半自動検証

バグの発見 厳密な証明

Rust 向けのテストの現在

Rust インタプリタ Miri

<https://github.com/rust-lang/miri>

一種のサニタイザの機能

unsafe Rust の“深い”アクセス競合を検出

Stacked Borrows [Jung+ POPL'20] が理論的基礎

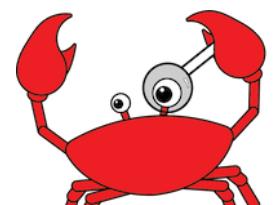
現実の多くの
バグを発見

- [Debug for vec_deque::Iter accessing uninitialized memory](#)
- [Vec::into_iter doing an unaligned ZST read](#)
- [From<&\[T\]> for Rc creating a not sufficiently aligned reference](#)
- [BTreeMap creating a shared reference pointing to a too small allocation](#)
- [Vec::append creating a dangling reference](#)
- [Futures turning a shared reference into a mutable one](#)
- [str turning a shared reference into a mutable one](#)
- [rand performing unaligned reads](#)
- [The Unix allocator calling posix_memalign in an invalid way](#)
- [getrandom calling the getrandom syscall in an invalid way](#)

課題 “深い”アクセス競合の検出

例 &mut Vec<T> とその要素への参照 &mut T

C/C++・LLVM 同様の
unsafe Rust のテスト

 kani 有界モデル検査

<https://model-checking.github.io/kani/>

Crux-mir 記号実行

<https://github.com/GaloisInc/crucible/>

Loom 並行プログラムのテスト

<https://github.com/tokio-rs/loom>

課題 Rust の所有権制約の検査

& 所有权を活かした効率化

未来への提案#1 unsafe 境界の動的検査

unsafe → safe の境界を動的検査

所有権型の保証を実行時に厳密に証明 &
所有権型が既に付いている部分は検査をスキップ

例

```
fn push(v : &mut Vec<T>) {
    v.reserve_for_push();
    unsafe { write(v.ptr.offset(v.len), a); v.len += 1; }
}
```

生ポインタ操作

所有権型 T の中身は
検査をスキップ

Vec の不变条件の
成立を動的検査

Rust 向けの検証の現在

safe Rust の検証

預言iamondベース

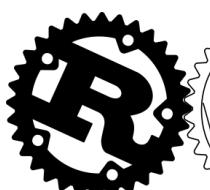


RustHorn [松下+ '20/'21]



Creusot [Denis+ '22]

純粹関数化 可変借用の操作に制限



Electrolysis [Ullrich+ '21]



Aeneas [Son+ '22]

課題 内部可変性の上手な扱い

unsafe Rust の検証



RustBelt [Jung+ POPL'18]

分離論理 Iris* [Jung+ POPL'15] で

unsafe 実装のメモリ/スレッド**安全性**を検証

例 Mutex, RwLock, Rc, Arc, ...

弱メモリ [Dang+ POPL'20]



預言diamond [松下+'22]



制限 低レベルな規則 証明が煩雑

課題 unsafe Rust の高レベル検証

未来への提案#2 高度な篩型付き Rust

Rust の所有権型 + 値に関する述語

可変借用は預言iamondで表現 借用検査は動的な条件を考慮

内部可変性の追跡には幽靈状態を利用

Nola [松下 博論'24] が意味論的基礎となる

例

```
struct Vec<T> {  
    len : uint;  cap : uint;  
    ptr : Box<[T; len, cap]>  
}
```

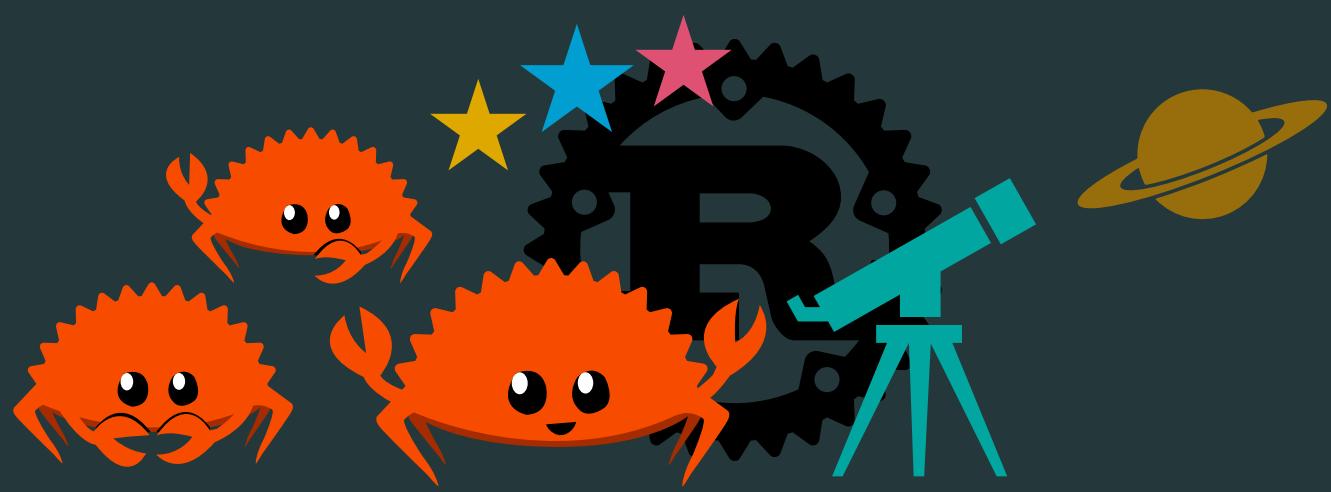
動的な値に依存

```
fn index_mut(v : &mut Vec<T>,  
             i : uint | i < v*.len) -> &mut T |  
    *res = v*.ptr[i] && 動的な条件  
    ^v = v*.ptr{i <- ^res} 預言の自動推論  
    { &mut v.ptr.offset(i) }
```

テスト・検証の現在と未来 まとめ

所有権を活かした
テスト・検証はまだまだ
探究の余地がある

最後に



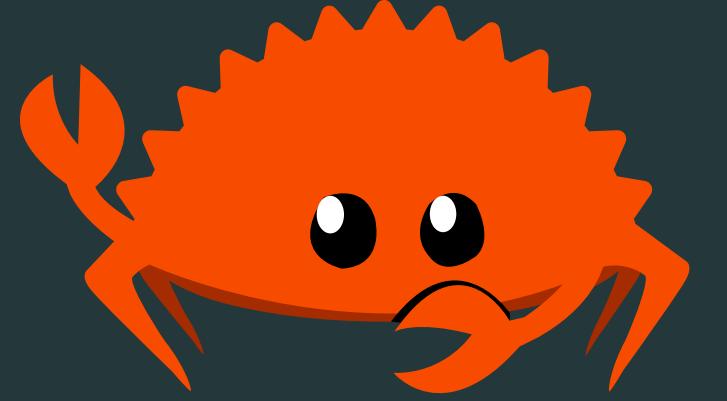
Rust は
所有権型を実用化 &
現実の開発を革新

Rust の所有権型は
借用 & unsafe & 内部可変性
の力で柔軟な操作を実現

預言の力で借用は
綺麗な関数型の世界で
表現できる

所有権を活かした
テスト・検証はまだまだ
探究の余地がある

Rust から広がる輝く未来を
切り開いていこう



Rust は人類を
ソフトウェア開発の
新時代へと導く

