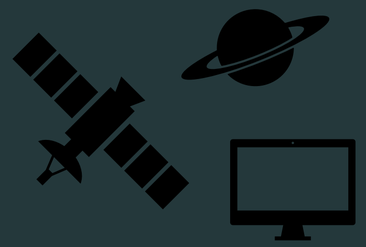
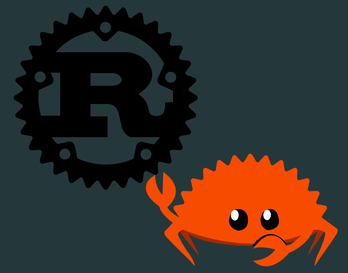


Science of Software, the Fun of Rust



Yusuke Matsushita Hakubi 15th

September 2, 2025 Hakubi Seminar

Hakubi

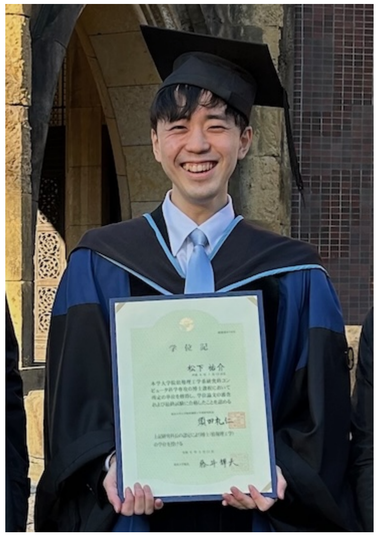
Kyoto Univ.

About me

Awarded at ACM PLDI '22

✦ **Yusuke Matsushita** 松下 祐介 *shiatsumat .github.io*

- '96 Born in Toyonaka, **Osaka**
- '09-'15 **Nada** Junior & Senior High School in Kobe
- '15-'24 **University of Tokyo**
  - Bachelor's, Master's & Ph.D.
    - Dept. of **Info. Science**, Sch. of Sci. / Dept. of **Computer Science**, Grad. School of Info. Sci. & Tech.
  - Supervisor: Naoki Kobayashi



My hobbies

✦ Music: Piano & Improvisation

- Mainly **classical**: Bach, Debussy, Chopin, …
  - Was a leader of the UTokyo Piano-no-kai
- Also, **jazz**, celtic, techno, …
  - Jazz at Zac Baran near Kumano Shrine

Gold prize at J.S.Bach concours ’22

✦ Ballroom dance

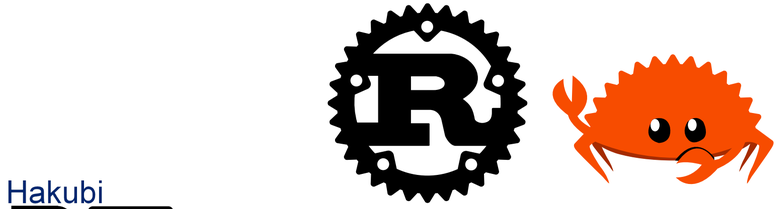
- Started during my Ph.D.

Ballroom dance in Osaka

Jazz session at Zac Baran

Me as a computer scientist

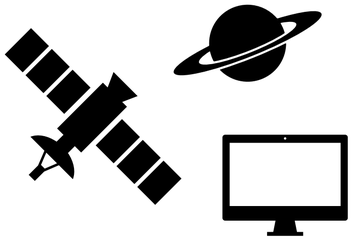
- **Studies software science**
  - Esp. the **Rust** programming language
  - ’20-’21 Intern at **MPI-SWS**, Germany
  - ’25 **Hakubi 15th** Prog.-Specific Assist. Prof.
    - **“Exploring a new age of software development springing from Rust”**
    - G.S. of **Informatics**, Igarashi & Suenaga Lab
    - ’24 JSPS PD fellow
  - ’24 & ’25 Lecturer at **IPA Security Camp**



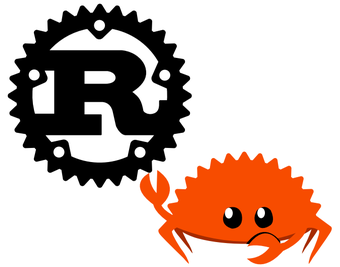
Hakubi

This talk

Science of Software



The Fun of Rust



Your questions and comments are always welcome!

Science of Software

What was ancient human life like?

- **Full of dangers & uncertainties in nature**
  - Technologies were developed to just survive
- **The stars in the sky move in a mysterious order**

Antikythera mechanism

Ancient Greek, ~150 BC

- **Oldest known analog computer**
  - Models the **solar system**
  - Could be used to **predict astronomical positions & eclipses**

Mogi Vicentini CC BY-SA 3.0

Marsyas CC BY-SA 4.0

Logg Tandy CC BY-SA 4.0

Industrial Revolution

~1760 - ~1840, ~1870 - ~1914

- **Great change in the global economy**
  - **Water & steam power → Electricity**
  - **Machines** have become vital
- **Jacquard loom 1804**
  - **Punch cards instruct complex weaving patterns**
  - Origin of **programming**

Paris Expo in 1889

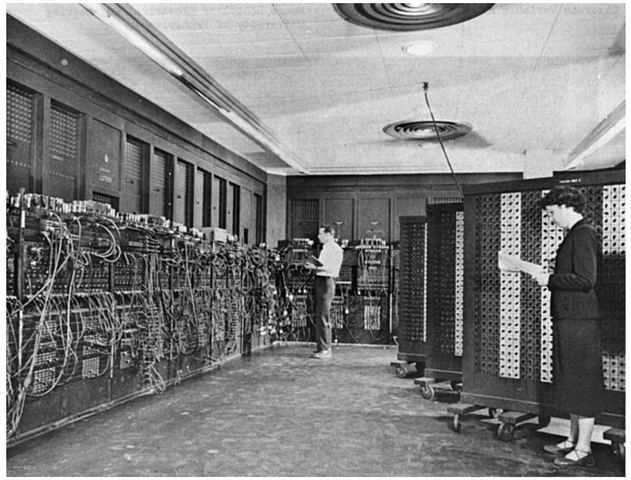
Jacquard loom

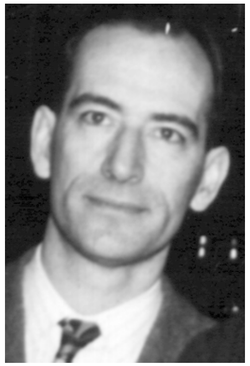
Punch for cards

Digital computers

1945 -

- **ENIAC** Electronic Numerical Integrator And Computer **1945**
  - **First general-purpose digital computer**
  - Developed at the **University** of Pennsylvania
  - Financed by the US **Army**
    - Used for research on building hydrogen bombs

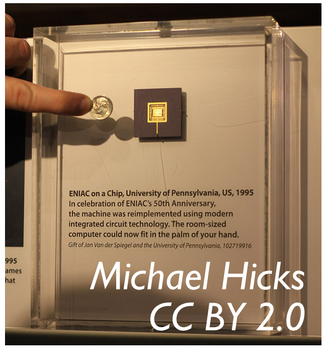




Physicist
John Mauchly

Engineer
J. Presper Eckert

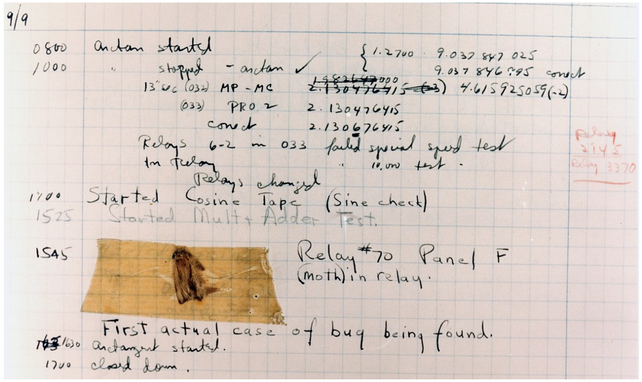
Mathematician
John von Neumann



ENIAC on a Chip, 1995

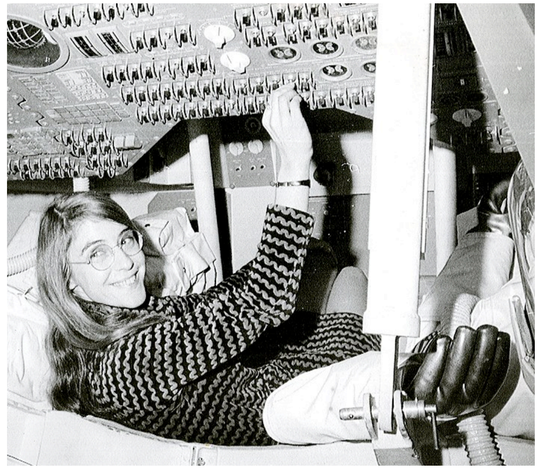
Software & Bugs

- **Software** = What instructs computers
  - To achieve **high-level goals** for humans
  - Built from programs & data
- **Bug** = Design error in software
- **Bugs can really cost**
  - Example: NASA's **Venus flyby** Mariner 1 in 1962
    - Launch failure due to a spec bug, ~$200M loss



Software science & engineering

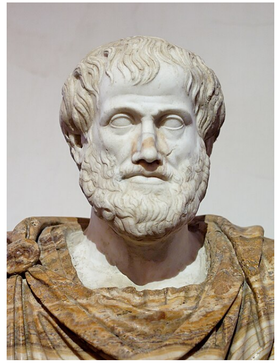
- **Solid general methodologies to develop high-quality, bug-free software**
  - M. Hamilton coined "**software engineering**"
    - She led the development of flight **software** for **NASA's Apollo 11** in the 1960s
  - **Scientific curiosity & Pragmatic usability**
  - **Programming languages**, **types**, testing & verification, fail-safety, …



Margaret Hamilton
1936-

LANGLEY FORMAL METHODS

History of logic – Philosophy & Mathematics

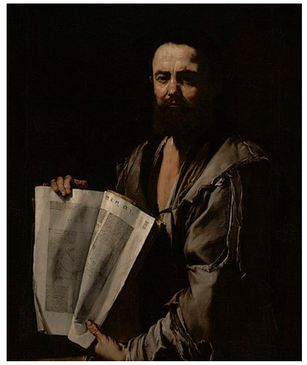


All men are mortal.
All Greeks are men.
Thus all Greeks are mortal.

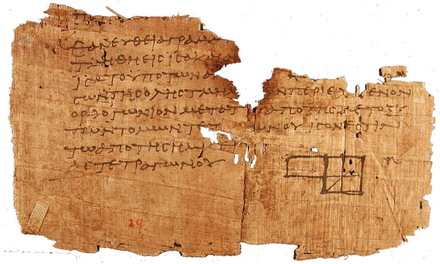
M a P, S a M; S a P

Syllogism

Aristotle 384-322BC



Euclid ~300BC

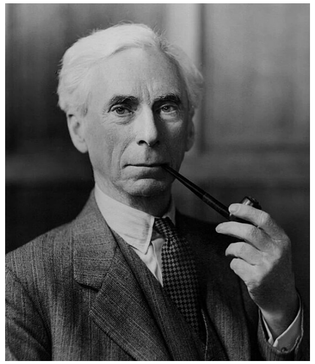


The Elements

Axioms

Gottlob Frege 1848-1925

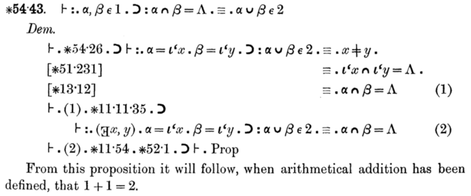
Begriffsschrift



Bertrand Russel
1872-1970

$$\{S \mid S \notin S\}$$

Russel's paradox



Principia Mathematica

Kurt Gödel
1906-1978

Gödel's 2nd incompleteness theorem

Any consistent formal system cannot prove its consistency

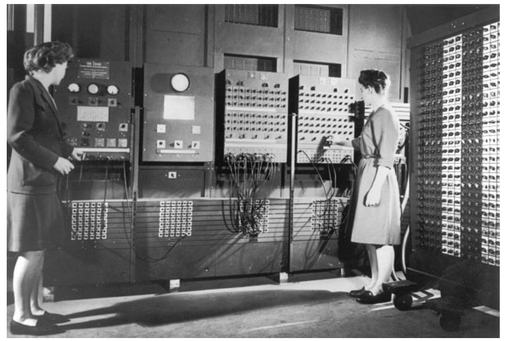
Foundational crisis of mathematics

What is the mathematical truth?

What is a limit of a finite machine?

History of programming

- ✦ 1940's Physical settings
  - ▸ Plugboard wiring & switches

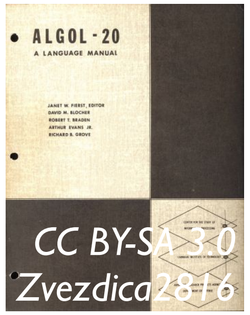


- ✦ 1947 Assembly languages
  - ▸ Transliteration of machine code

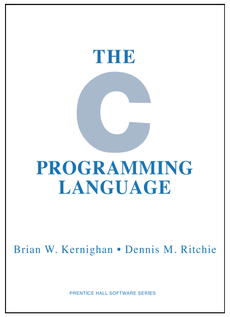
```
0013            RESETA  EQU     %00010011
0011            CTLREG  EQU     %00010001

C003 86 13      INITA   LDA A   #RESETA
C005 B7 80 04           STA A   ACIA
C008 86 11              LDA A   #CTLREG
C00A B7 80 04           STA A   ACIA

C00D 7E C0 F1           JMP     SIGNON
```

- ✦ 1952 Programming languages
  - ▸ High-level abstraction
  - ▸ Various designs explored

1958
ALGOL



1972
C



1991
Python

1995
Java

2009
Go

and so on …

Types in programming languages

- **Labels on data in computation**
  - Describes **kinds**, **attributes**, **permissions**, etc.
  - **Systematic** way to **prevent bugs** in programs

```
123 : Int
"Hello" : String
```

```
1 + 2
```


PASS

```
3
```

```
1 + "2"
```


TYPE ERROR

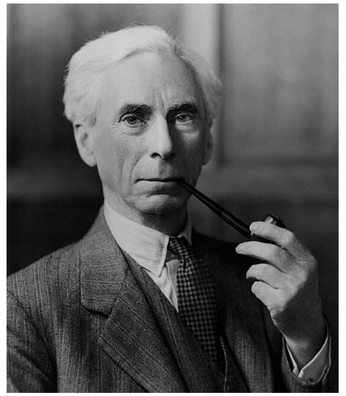
Int & String can't be added

```
0xBEEF1? "12"?
```

BUG

History of types

- **1903 Russel's work on types for logic**
  - To avoid **Russel's paradox** $\{S \mid S \notin S\}$
- **1940 Church's simply typed lambda calculus**
  - Types for a formal model of **computation**
- **1958 Types for programming languages**
  - To **prevent bugs** in software



Bertrand Russel

Alonzo Church

1. *Type declarations*

Type declarations serve to declare certain variables, or functions, to represent quantities of a given class, such as the class of integers or class of Boolean values.

Form: $\Delta \sim$ *type* (I, I, ⋯, I, I[ ], ⋯, I[,], ⋯, I[,,], ⋯) where *type* is a symbolic representative of some *type* declarator such as *integer* or *boolean*, and the I are identifiers. Throughout the program, the variables, or functions named by the identifiers I, are constrained to refer only to quantities of the type indicated by the declarator. On the other hand, all variables, or functions which are to represent other than arbitrary real numbers must be so declared.

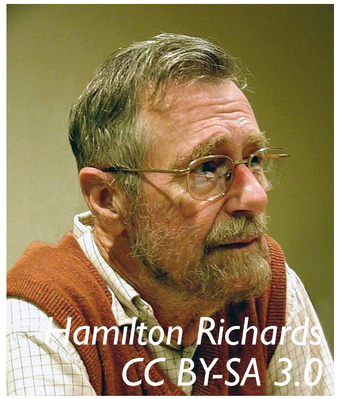
Algol 58 Report, CACM

Formal verification

- **Prove that the software satisfies the specifications**
  - Rigorously **eradicate software bugs**
  - Real-world use in **critical domains**
    - **Aerospace**, automobile, finance, security, …

- **Central topic in software science**
  - **Program logics**, especially Hoare Logic ’69
    - Dijkstra’s weakest precondition calculus ’75
  - **Mathematical logic at work**

Tony Hoare



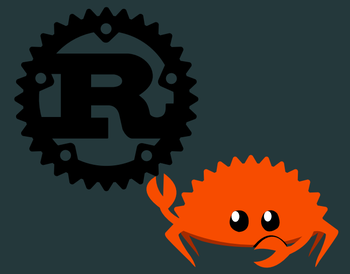
Edsger Dijkstra

Now in the Information Age

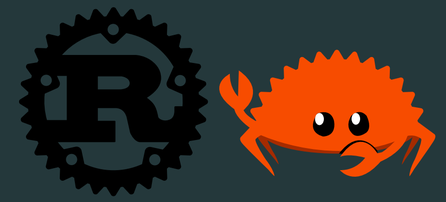
1947 - Present

- **Software science** & engineering can help shape a great future of information technology and our lives

The Fun of Rust



The Rust programming language



- Rust makes the software:
  - Performative — Low-level controls like C/C++
  - Secure — High-level safety guarantees by ownership types
  - Developed fast — Modern language features & ecosystem

Software science
is advancing the real world

Rust

A language empowering everyone
to build reliable and efficient software.

GET STARTED

Version 1.83.0

Why Rust?

Performance

Rust is blazingly fast and memory-efficient: with no runtime or garbage collector, it can power performance-critical services, run on embedded devices, and easily integrate with other languages.

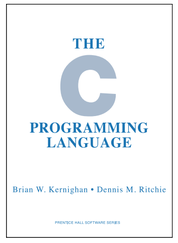
Reliability

Rust's rich type system and ownership model guarantee memory-safety and thread-safety — enabling you to eliminate many classes of bugs at compile-time.

Productivity

Rust has great documentation, a friendly compiler with useful error messages, and top-notch tooling — an integrated package manager and build tool, smart multi-editor support with auto-completion and type inspections, an auto-formatter, and more.

1972 **C**



1985 **C++**

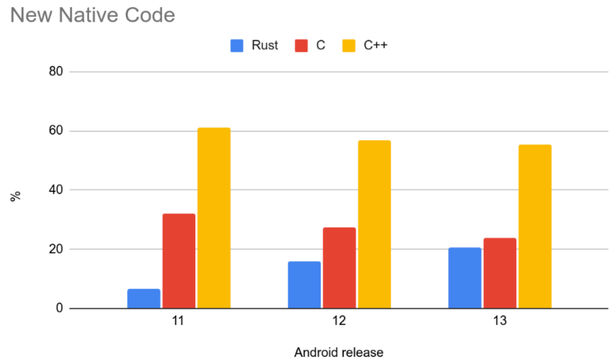
2015 **Rust**

JavaScript	33.5%	46.8%
TypeScript	31.9%	58%
Rust	29.2%	72.4%
Bash/Shell	27.4%	52.8%
Go	23.4%	56.5%

Rust has been the most admired language for 10 years

Rust in the real world

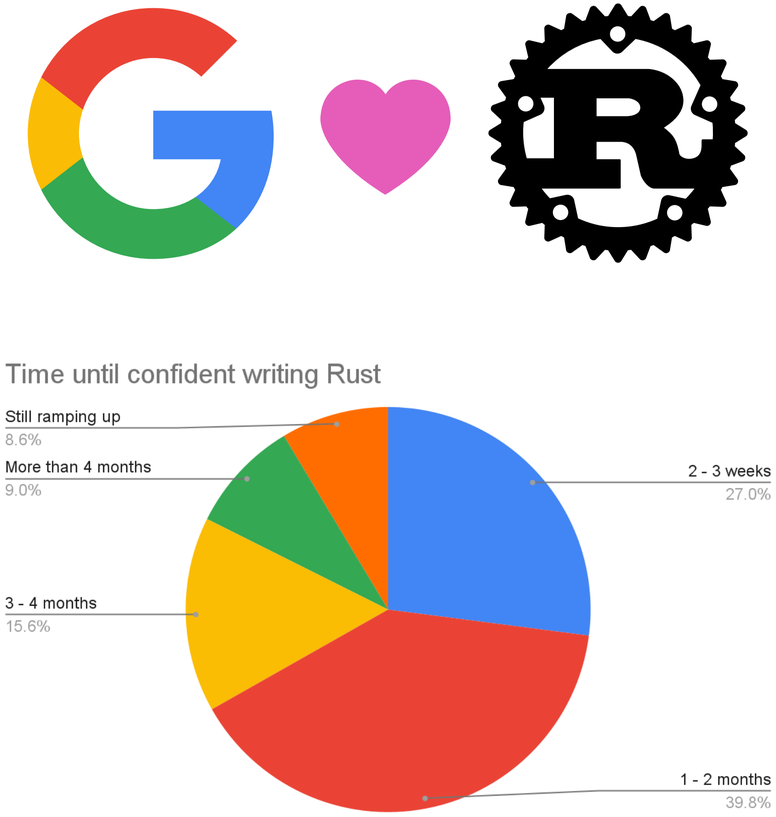
- **Rust serves a lot in real-world software**
  - **Google's Android OS**
    - Rust **reduced vulnerabilities**
  - **Mozilla's Firefox web browser**
    - Rust was born from and raised by **Mozilla**
  - **Security & performance**
    - **VMs: AWS**'s Firecracker, Google's crosvm
    - **Language engines**: Deno, Wasmer



Mozilla

Rust: Case for Googlers

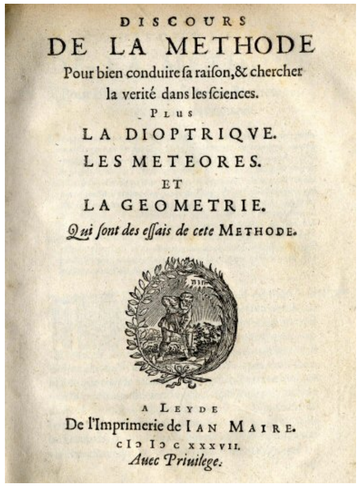
- **A survey on >1k Google software engineers in 2022**
  - ~67% got **confident** in **writing** Rust just in 2 months
  - Over 50% felt as **productive** in Rust as in other languages in 4 months
  - ~85% felt more **confident** in **correctness** in Rust than in other languages
  - Top **challenges** in **learning** Rust: Macros, **ownership/borrowing**, async



Rust, more technically

Divide the difficulties

René Descartes 1596-1650


DISCOURS
DE LA METHODE
Pour bien conduire sa raison, & chercher
la verité dans les sciences.
PLUS
LA DIOPTRIQVE.
LES METEORES.
ET
LA GEOMETRIE.
Qui sont des essais de cette METHODE.

A LEYDE
De l'Imprimerie de IAN MAIRE.
clɔ Iɔ c XXXVII.
Avec Privilege.

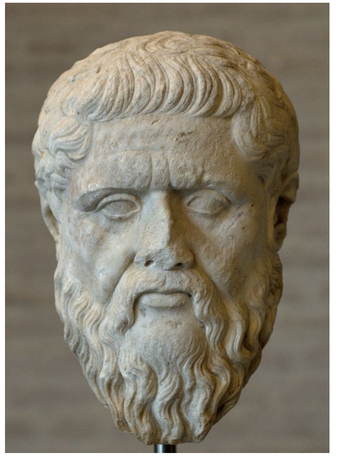
*The second: to **divide** each of the **difficulties** under examination into as **many parts** as possible, and as might be necessary for its **adequate solution**.*

Discourse on the Method, Part II

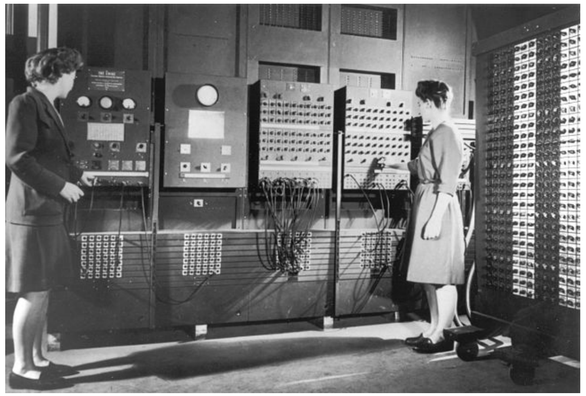
- **To overcome the difficulties, divide them**
  - Then **compose** the reasoning about the **parts**
  - **Modularity & Composability → Scalability**

Difficulties in computation

- **Ideally, everything is just persistent**
  - **Metaphysical** world of **Platonic Forms**
  - **Usual mathematics** works in this setting
- **Physical computation breaks things**
  - Inevitably deals with **mutable state**, e.g., the **memory** and the **environment**
  - **Reasoning about mutable state is hard**
    - **Naive reasoning can break** by state update



Plato
~400BC



Girard's linear logic

1987

- **Logic with non-persistent propositions**
  - Can reason about **mutable state**
  - Applications to **computer science**

Jean-Yves Girard 1947-

LINEAR LOGIC*

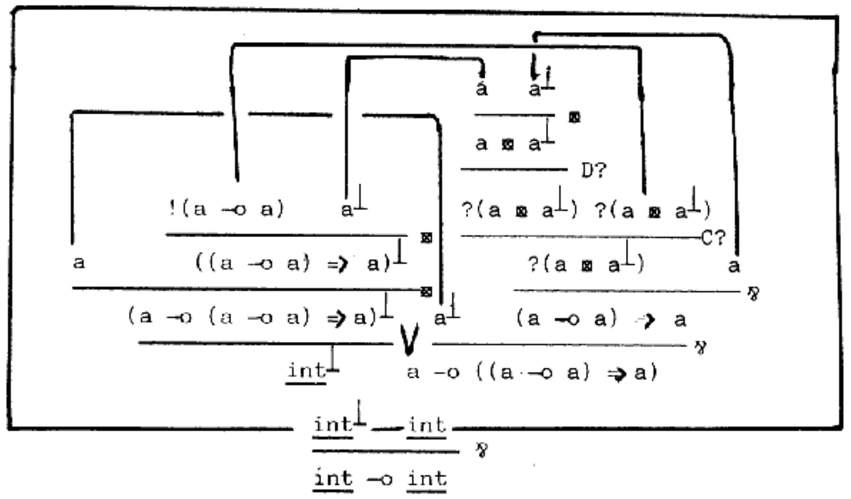
"La secrète noirceur du lait . . ."

(*Jacques Audiberti*)

$$\vdash 1 \quad \text{(axiom)}, \qquad \frac{\vdash A}{\vdash \bot, A}\,\bot,$$

$$\frac{\vdash A, C \quad \vdash B, D}{\vdash A \otimes B, C, D}\,\otimes, \qquad \frac{\vdash A, B, C}{\vdash A \parr B, C}\,\parr$$

* Because of its length and novelty this paper has not been subjected to the normal process of refereeing. The editor is prepared to share with the author any criticism that eventually will be expressed concerning this work.



From linear logic to Rust

- ✦ **Rust's ownership types originate from academic studies**
  - ▸ **Rust** is the **first** to put **ownership types** into **full practice**

Linear logic

Girard 1987

$$\frac{\vdash A, C \quad \vdash B, D}{\vdash A \otimes B, C, D}\otimes, \qquad \frac{\vdash A, B, C}{\vdash A \parr B, C}\parr$$

Linear types

Wadler 1990

Region types

Tofte & Talpin 1994

Ownership types

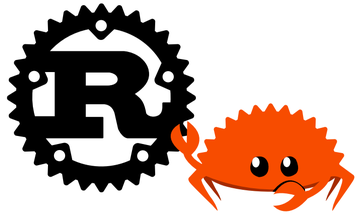
Clarke+ 1998

Cyclone

Grossman+ 2002

Borrows

Rust



Matsakis+ 2015

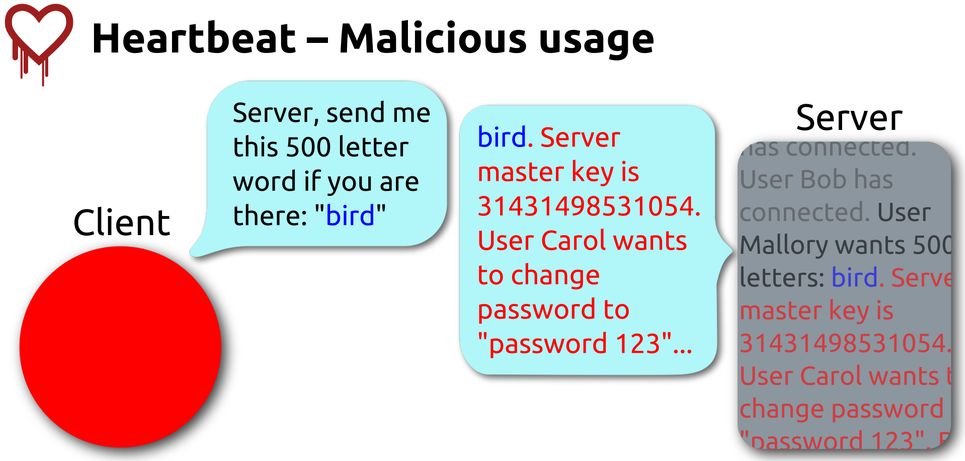
Days before Rust

Heartbleed

'12 Introduced
'14 Disclosed & fixed

Severe bug in OpenSSL, a widely used library for secure communications

Memory corruption caused by C



- **Severe vulnerabilities due to memory corruption**
  - Languages like **C/C++** are **not memory-safe!**
  - **~70% of zero-day attacks -'19 stem from memory corruption** — Google Project Zero

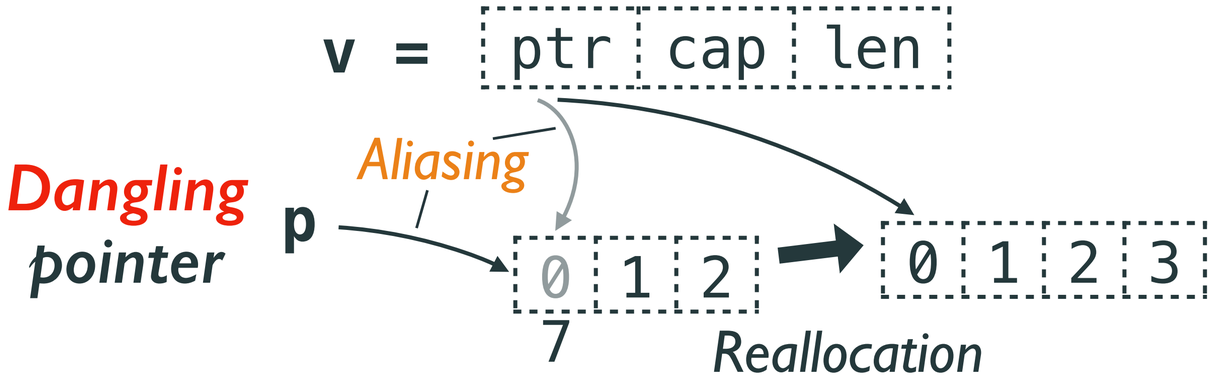
CVE	Vendor	Product	Type
CVE-2019-7286	Apple	iOS	Memory Corruption
CVE-2019-7287	Apple	iOS	Memory Corruption
CVE-2019-0676	Microsoft	Internet Explorer	Information Leak
CVE-2019-5786	Google	Chrome	Memory Corruption
CVE-2019-0808	Microsoft	Windows	Memory Corruption
CVE-2019-0797	Microsoft	Windows	Mem

Aliasing AND Mutability is dangerous

- **Aliased** mutable state is the core source of bugs

Example A **dangling pointer** caused by a **data race**

```
vector<int> v { 0, 1, 2 };  int *p = &v[0];
v.push_back(3);   *p = 7;  // Data race!
printf("%d\n", v[0]); // Prints 0, not 7
```



*Even worse, this can cause a **buffer overflow**, **leaking secrets**!*

Rust bans Aliasing AND Mutability

- Rust's ownership types ban "Aliasing AND Mutability"

Example

```
let mut v = vec![0, 1, 2];  let p = &mut v[0];
v.push(3);   *p = 7;  // OWNERSHIP ERROR
println!("{}", v[0]);
```

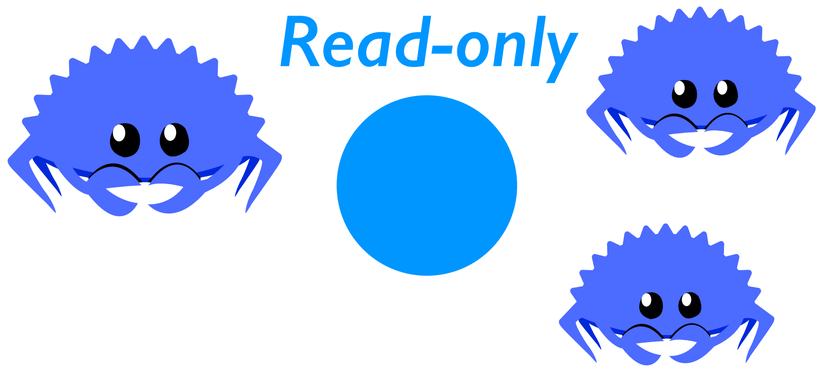
```
error[E0499]: cannot borrow `v` as mutable more than once at a time
  | ...;  let p = &mut v[0];
  |                   - first mutable borrow occurs here
  | v.push(3);   *p = 7;  // Data race!
  | ^           ------ first borrow later used here
  | second mutable borrow occurs here
```

Bug detected!

Ownership for Aliasing XOR Mutability

- Each resource is either aliased or mutable, never both
- Managed by ownership = access permission

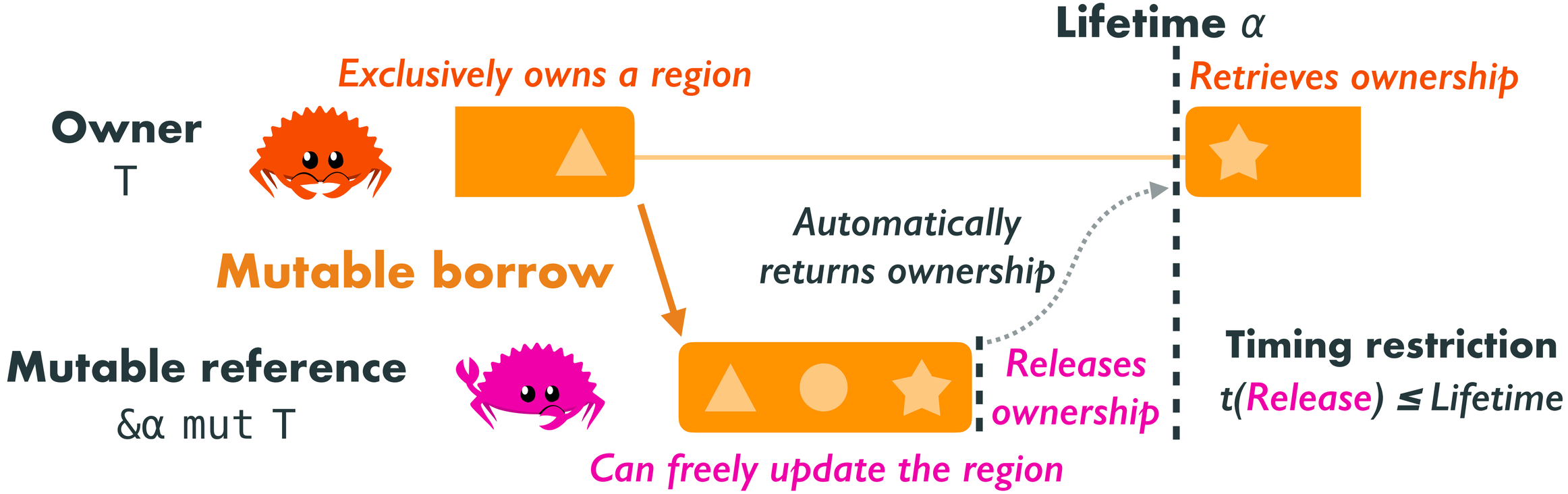
Aliased & Immutable



Mutable & Exclusive

Rust's key idea: Borrowing

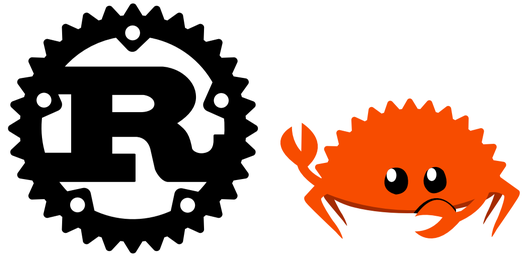
- To borrow ownership temporarily
  - **No direct communications** needed when releasing ownership



My research on Rust

Formal verification of Rust programs

- ✦ Significant in the real world
  - ▸ As **Rust** is widely used for **foundational software**
  - ▸ Operating systems, servers, crypto, …
- ✦ Can take advantage of Rust's ownership types



∀ ∃

Microsoft Research

Practical, High-Performance Verification in Rust

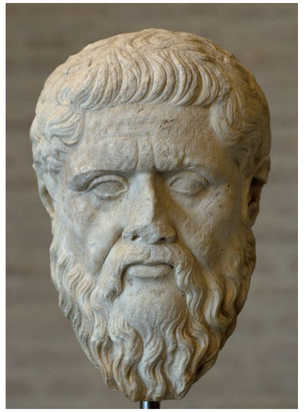
Rust Verification Workshop

Fifth Rust Verification Workshop (RW2025)

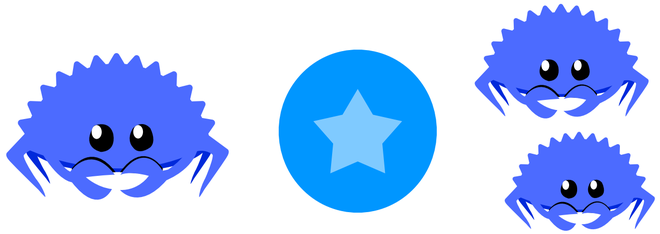
May 5-6, 2025
co-located with ETAPS 2025, Hamilton, Canada

Basic idea: Rust into mathematical models

- ✦ **Turn Rust programs into mathematical models**
  - ▸ Leverage **Rust's ownership types**
  - ▸ To the "**metaphysical**" world where **all is persistent**

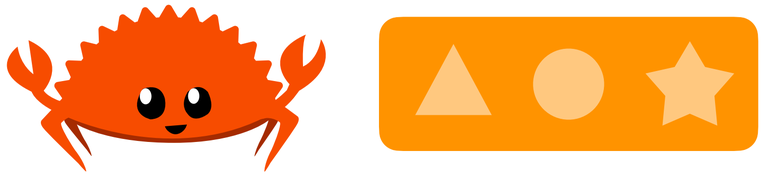


Aliased & Immutable



Just an immutable value

Fully own



Can track the value change

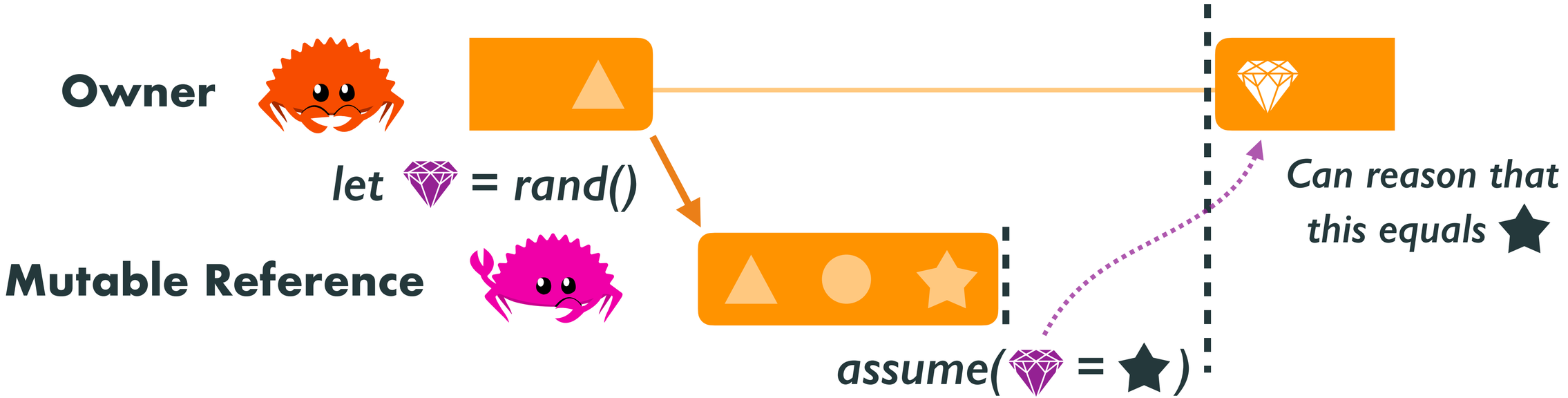
Challenge: Mutable borrows

- ✦ **Mutable borrows are hard for functional reasoning**
  - ▸ How to "**tell**" the **lender** the **result** of the **borrower's mutation?**

Solution: My work, RustHorn

Matsushita+
ESOP'20 / TOPLAS'21

- **Model mutable borrows by prophecy**
  - **Prophecy** [Abadi & Lamport '88]: Fetches info about the **future**
    - Specifically, the **result** of the **borrower's mutation**



Power of RustHorn's approach

- ✦ **Prophecy can naturally model various operations**
  - ▸ By resolving **prophecies** just as **ownership** is **released**

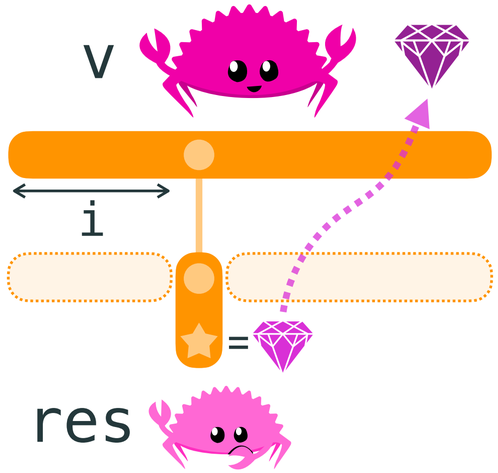
Example

```
fn push<α,T>(v : &α mut Vec<T>, a : T)
    ensures  ^v = *v ++ a
```

Resolve the prophecy 

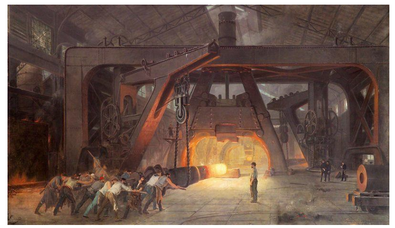
```
fn index_mut<α,T>(v : &α mut Vec<T>, i : uint) -> &α mut T
    requires i < v.len()
    ensures  *res = v[i]  &&  ^v = *v { i := ^res }
```

Partially resolve the prophecy 



RustHorn's approach at work

- ✦ **Demonstrated in diverse contexts**
  - **Fully automated** Rust verifier **RustHorn** [**Matsushita+** '20/21]
    - Automatically **find invariants** with CHC solvers [Bjørner+ '15]
  - **Semi-automated** Rust verifier Creusot [Denis+ '22]
    - Used for verifying a **SAT solver** written in Rust [Skotåm '22]
  - Rust **program synthesizer** RusSOL [Fiala+ '23]
    - Synthesizes to satisfy **RustHorn-style specs & ownership constraints**
  - **Refinement-type** based verifier Thrust [Ogawa+ '25]

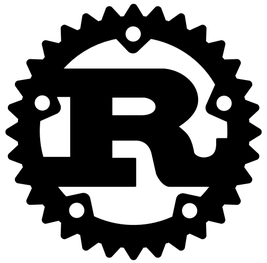


Foundational verification of Rust

- ✦ **Metatheory on Rust's core features**
  - ▸ **Ownership types** + Extension by **Rust APIs**
  - ▸ **RustBelt** [Jung+ '18] — **Foundation** for Rust
    - **Memory & thread safety** of Rust's ownership types
    - Modeling **types** in **separation logic** [O'Hearn+ '01], a variant of **linear logic**
  - ▸ **RustHornBelt** [Matsushita+ '22] — **Foundation** for **RustHorn**
    - Extended **RustBelt** to prove **RustHorn**-style reasoning **sound**
    - Modeled **nestable prophecies** in **separation logic**

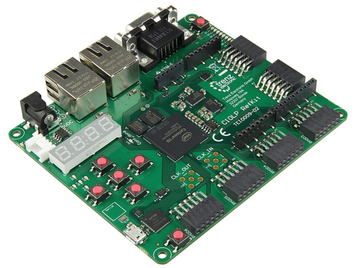
Future research topics

- **Beyond Rust's ownership types**
  - **Practically verify** more **low-level** code
  - Reason about **shared mutable state**
  - Prove **contextual refinement & equivalence**
    - **Compare** program **behaviors** under **Rust's types**

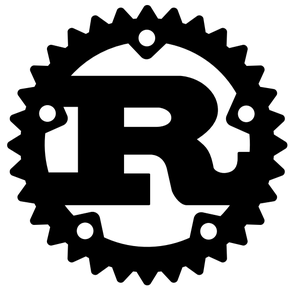


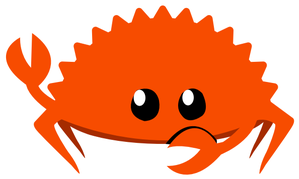
- **Different angles**
  - **Synthesize** programs & proofs with **AI**
  - Apply to **hardware & quantum computing**



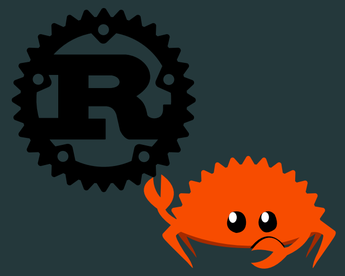



Creating the new age with Rust





∀∃



Software science & **Rust**
can take us to
a great future of
information technology