ソフトウェアの未来を科学する

Rust に誘われて

松下 祐介　京都大学・MPI-SWS

2026年9月5日　第59回情報科学若手の会

Sciencing the Future of Software

Allured by Rust

Yusuke Matsushita Kyoto University & MPI-SWS

Sept 5, 2026 59th Joho-kagaku Wakate-no-kai

About Me

- **Yusuke Matsushita** **松下 祐介**
  - July 15, 1996, Osaka
- **’15–’24 Bachelor, Master, Ph.D. at UTokyo**
  - **Computer science — Programming languages**
    - Supervised by **Prof. Naoki Kobayashi**
- **’25– Assist. Prof. at Hakubi, KyotoU**
  - Focus on **Rust**
  - ’26– **Visiting scientist** at **MPI-SWS**
    - **Germany, Prof. Derek Dreyer**

About Me

Google Scholar

SIGN IN

Yusuke Matsushita

FOLLOW

Kyoto University
Verified email at fos.kuis.kyoto-u.ac.jp - Homepage

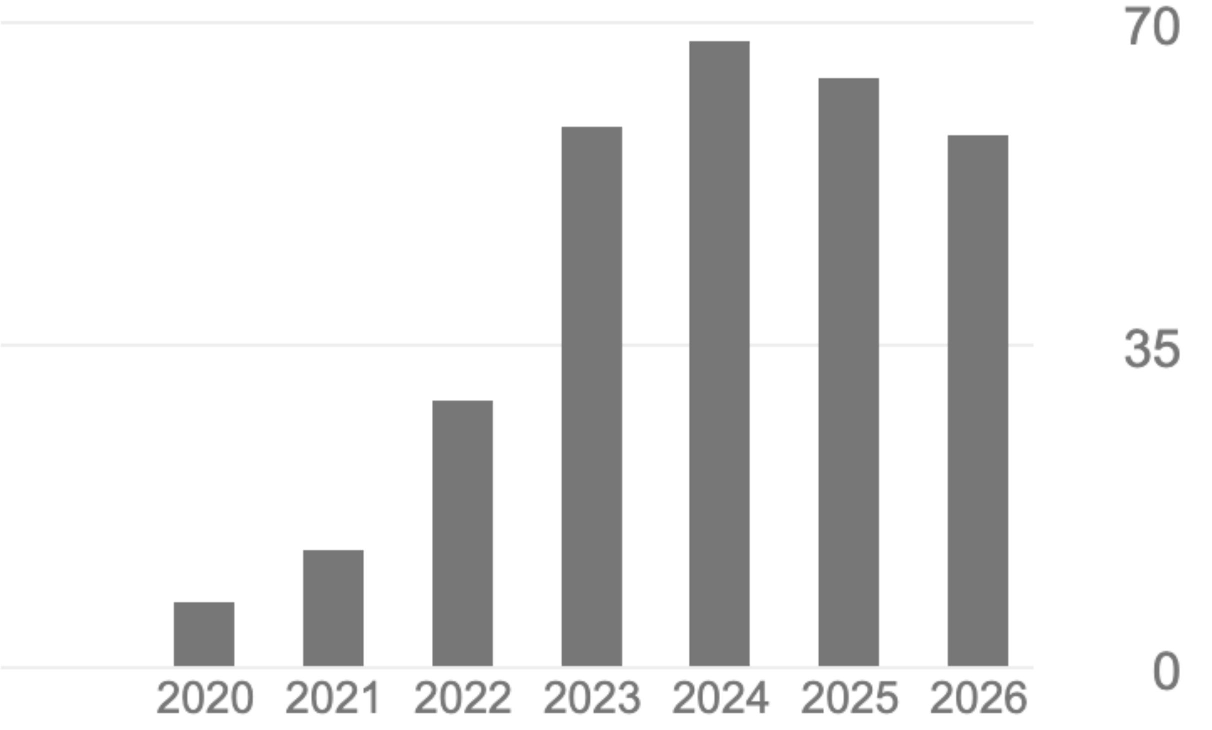
Computer Science · Programming Languages · Machine Learning

TITLE	CITED BY	YEAR
RustHorn: CHC-based Verification for Rust Programs Y Matsushita, T Tsukada, N Kobayashi ACM Transactions on Programming Languages and Systems (TOPLAS) 43 (4), 1-54	108	2021
RustHornBelt: A Semantic Foundation for Functional Verification of Rust Programs with Unsafe Code Y Matsushita, X Denis, JH Jourdan, D Dreyer Proceedings of the 43rd ACM SIGPLAN Conference on Programming Language …	94	2022
RustHorn: CHC-Based Verification for Rust Programs Y Matsushita, T Tsukada, N Kobayashi European Symposium on Programming (ESOP), 484-514	65	2020
Nola: Later-Free Ghost State for Verifying Termination in Iris Y Matsushita, T Tsukada Proceedings of the ACM on Programming Languages 9 (PLDI)	7	2025

GET MY OWN PROFILE

Cited by

	All	Since 2021
Citations	298	291
h-index	5	5
i10-index	3	3



Public access — VIEW ALL

0 articles — 6 articles

About Me

PLDI
BOULDER 2026

POPL 27

PLDI Review Committee PLDI Research Papers

Benjamin Mariano
The University of Texas at Austin, Texas, USA
United States

Guido Martínez
Microsoft Research
United States

Yusuke Matsushita
Kyoto University
Japan

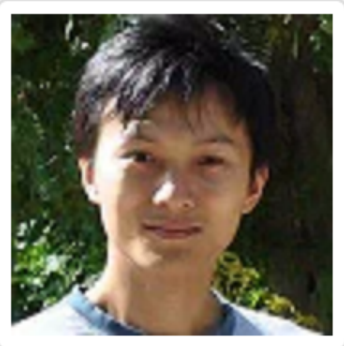
Jedidiah McClurg
Cornell University
United States

Roland Meyer
TU Braunschweig
Germany

Mae Milano
Princeton University
United States

Program Committee POPL

Yannick Zakowski
Inria
France



Yingfei Xiong
Peking University
China

Youyou Cong
Institute of Science Tokyo
Japan

Yu Feng
University of California at Santa Barbara
United States

Yu-Fang Chen
Academia Sinica
Taiwan

Yusuke Matsushita
Kyoto University
Japan

How I Became What I Am

- Elementary school — Squeak, Visual Basic
- Junior high school — HSP, C++, Programming contests, C#
- Senior high school — **Haskell, Category theory, PL basics**
- Freshman, sophomore, junior — **Rust, PL theory**
- Senior — Kobayashi Lab, **RustHorn** for my senior thesis
- Master — Internship at **MPI-SWS, RustHornBelt**
- Ph.D. → Postdoc → **Assist. Prof.** at Hakubi

When I Discovered the Idea of RustHorn

Yusuke Matsushita
@shiatsumat

Show translation

かなり理論的にも実用的にも広がりの大きい分野の先駆者になれる可能性が高まってきたし、真面目に今後の研究者人生について戦略を立てるほうがいいような気がしてきた

Translate with DeepL

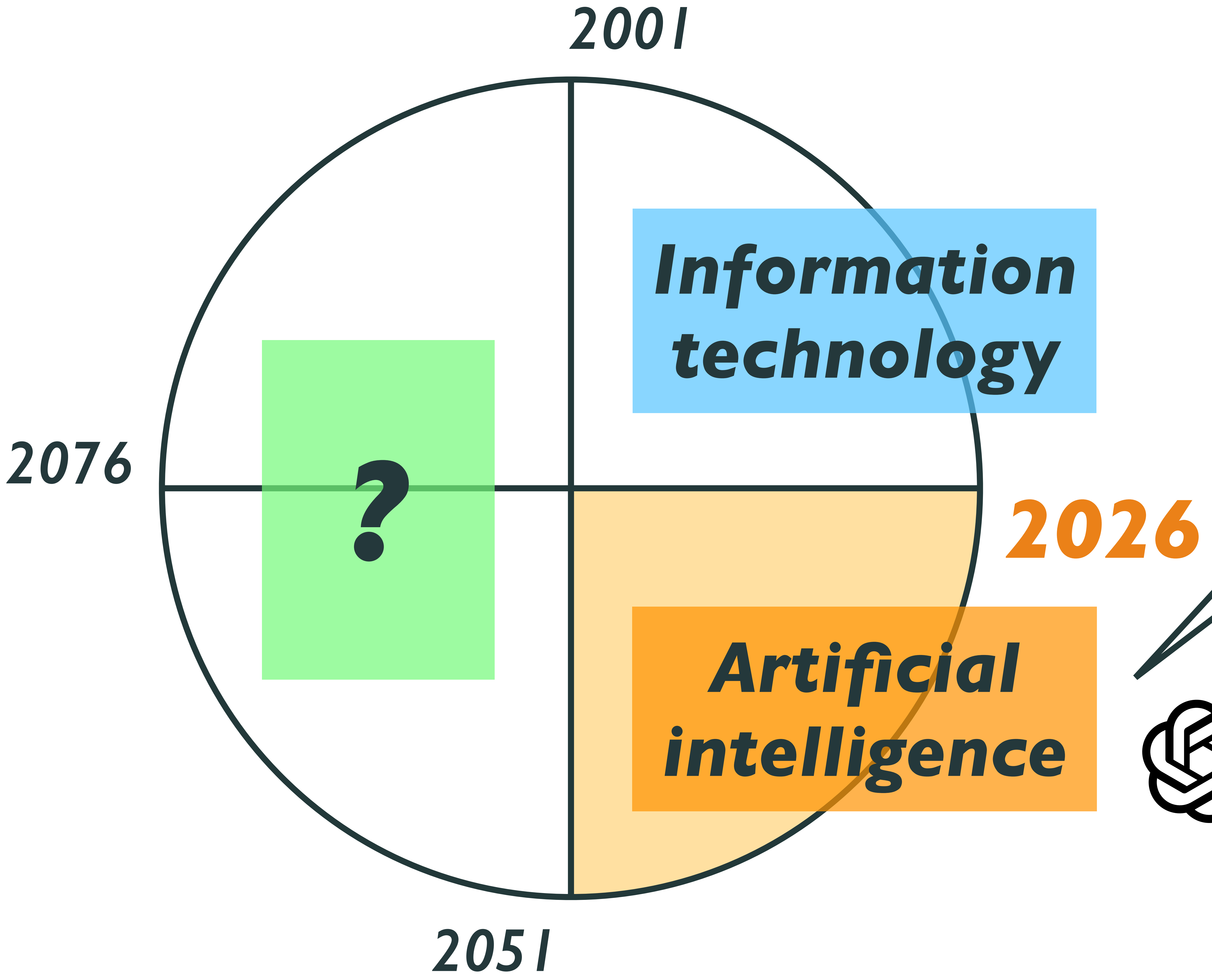
5:39 AM · Oct 18, 2018

1 11

Now I have a strong chance of becoming a pioneer in a field
with significant potential for growth both scientifically and practically;
I'm feeling that I should seriously strategize about my future career as a scientist.

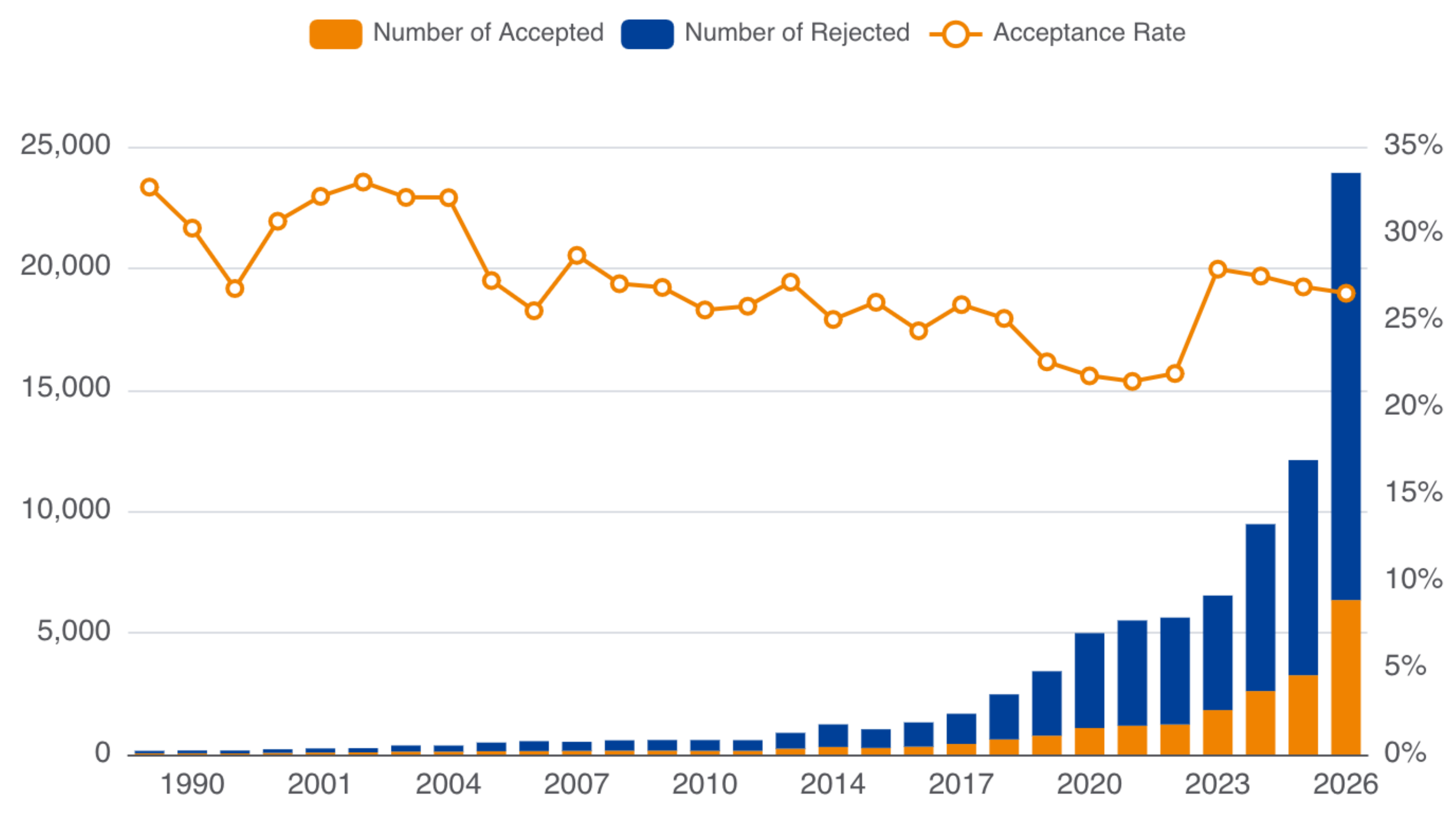
Where Are We?

The 2nd Quarter of the 21th Century

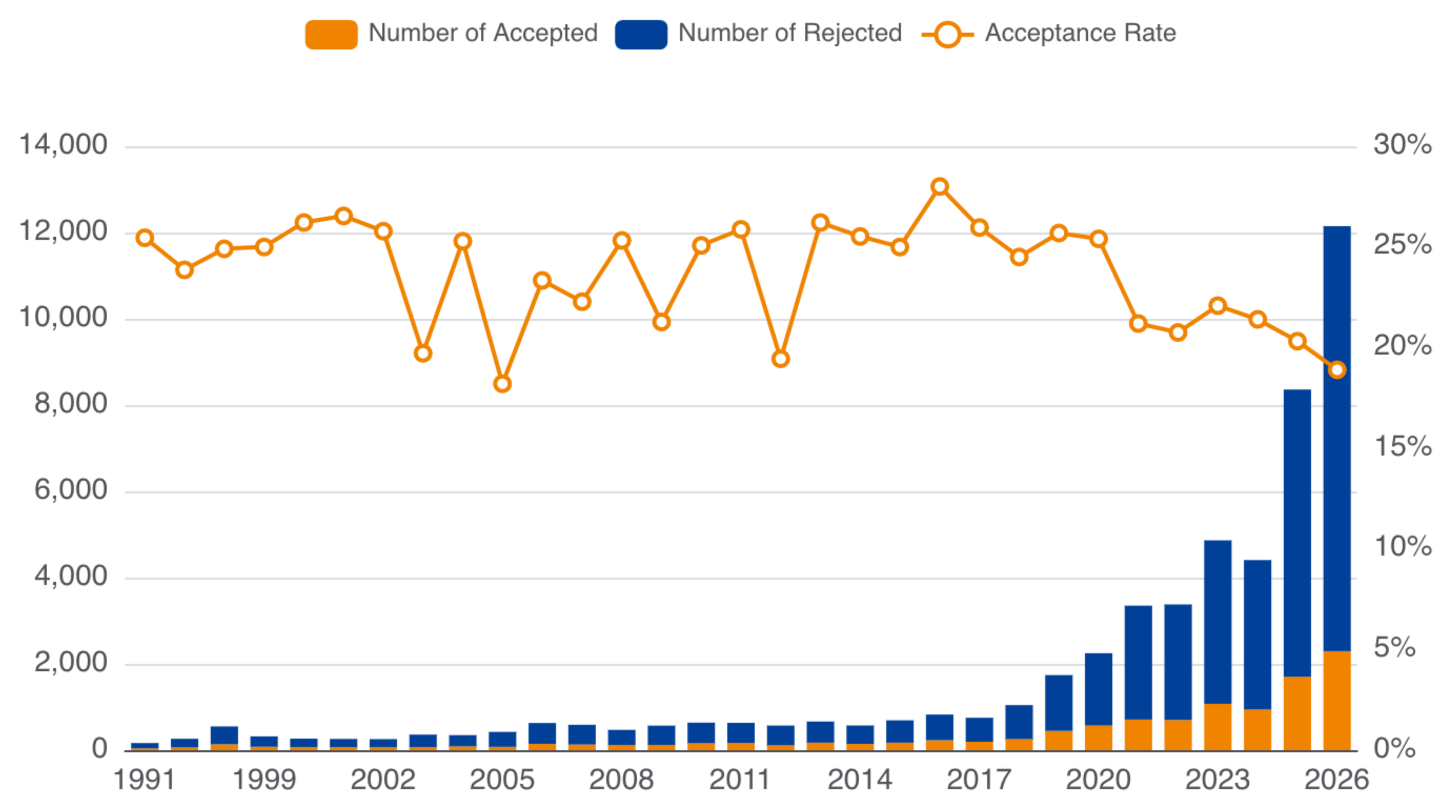


How will humans live?
How will science be?

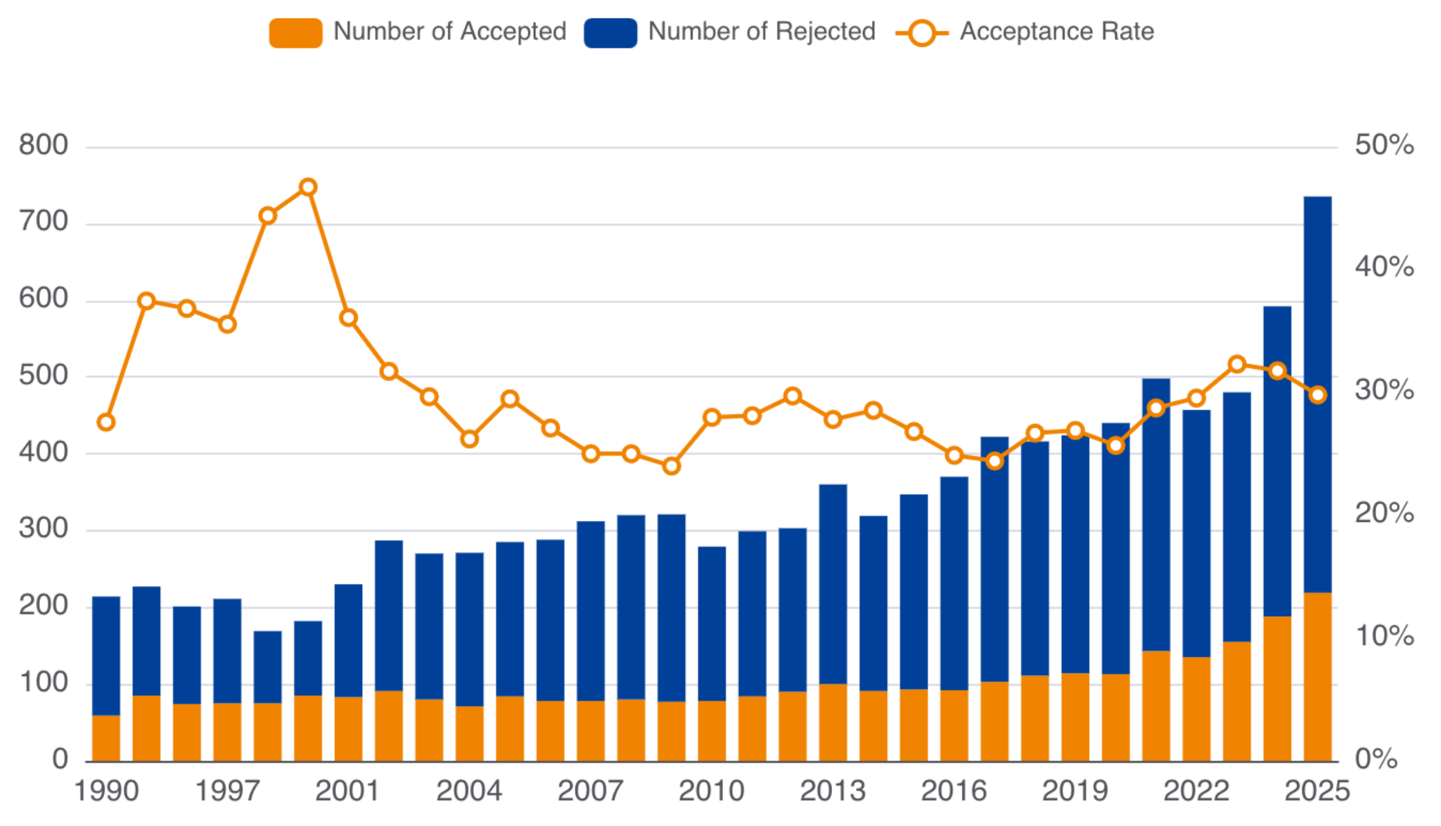
Computer Science Accelerated



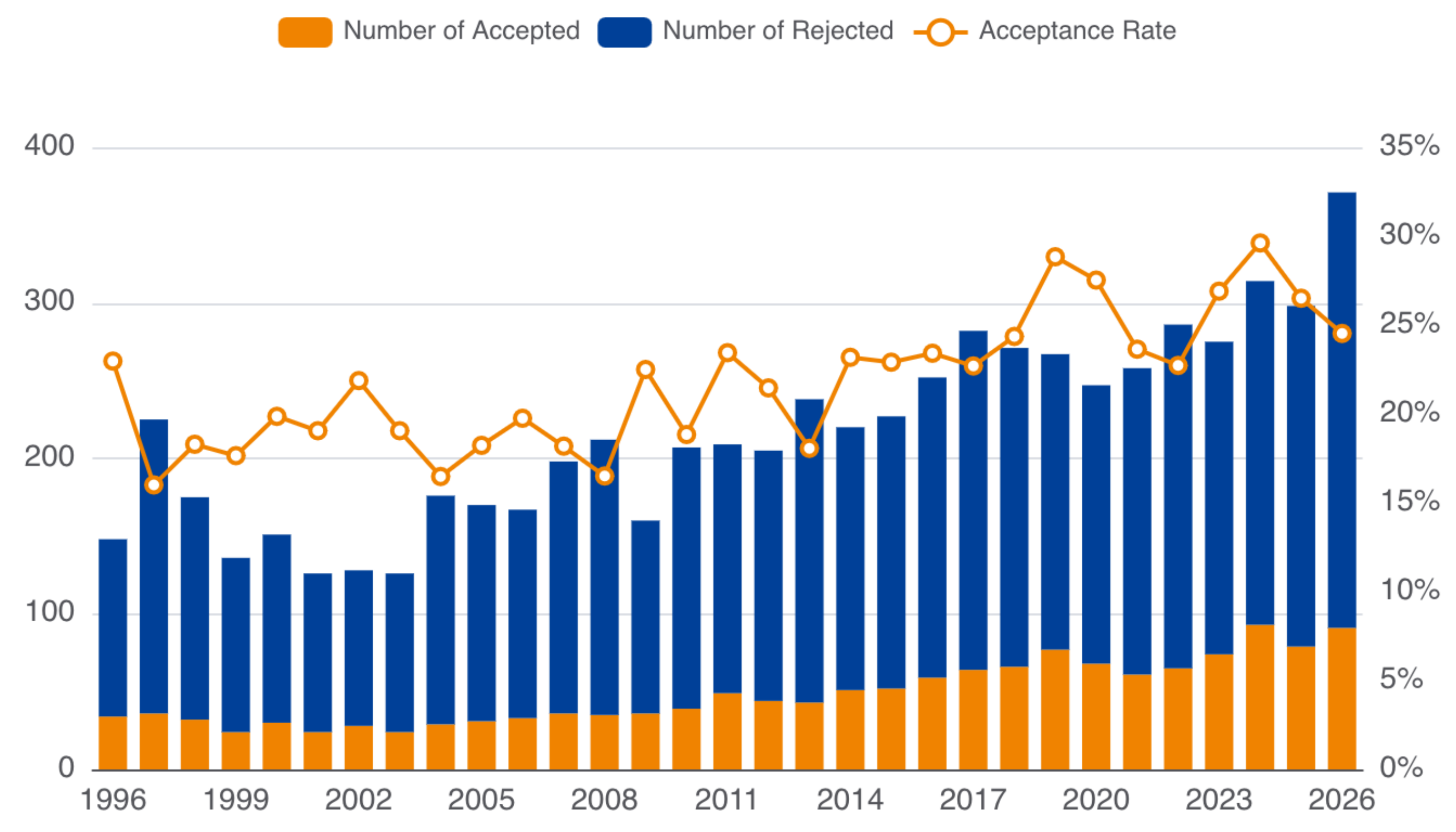
ICML



ACL



STOC



POPL

Source: csconfstats.xoveexu.com

How to Survive This Uncertainty?

Think Broad & Flexible

Learn from Other Fields & History

Reflect on Human Nature

And you can think and discuss more!

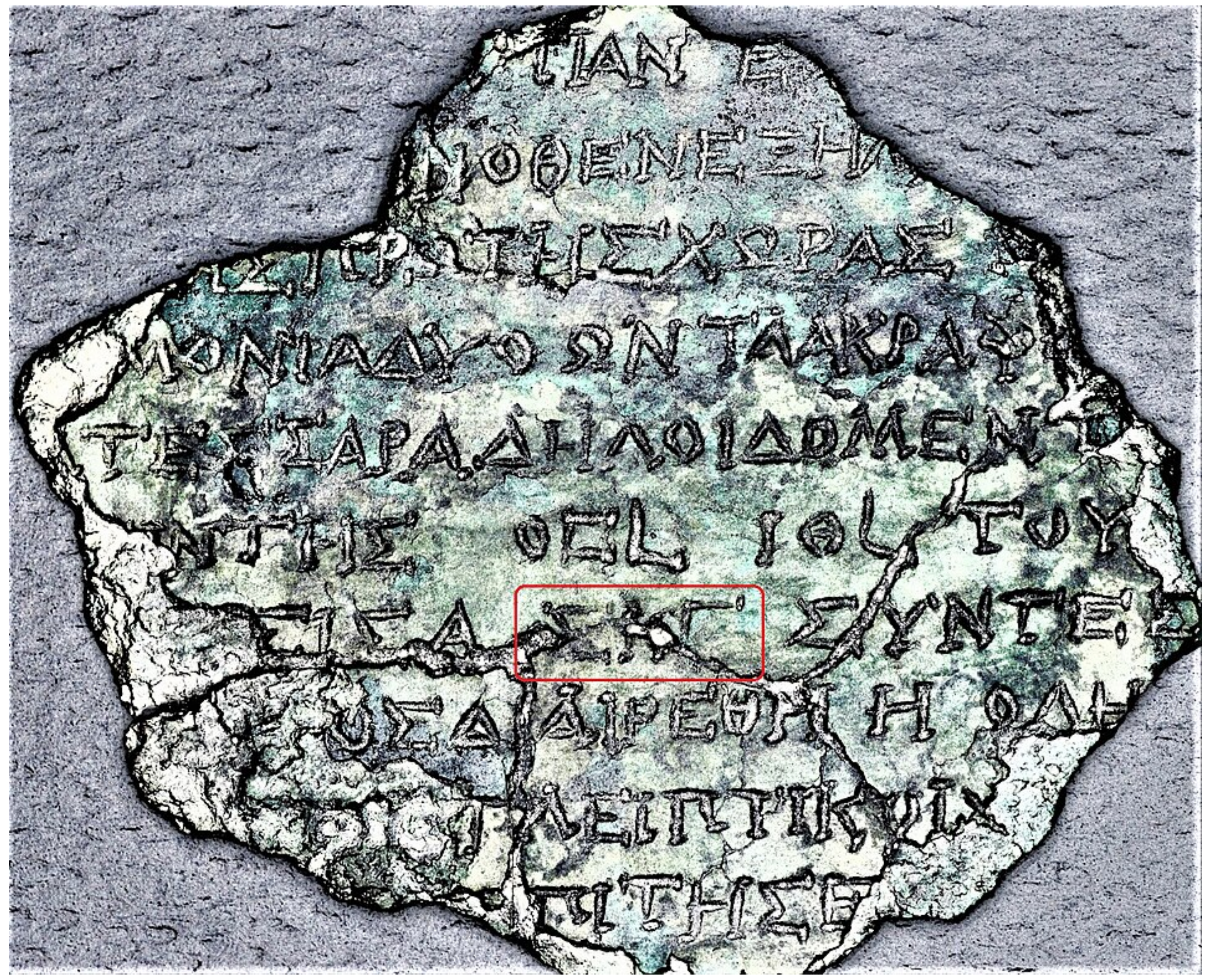
Science of Software

What Was Ancient Human Life Like?

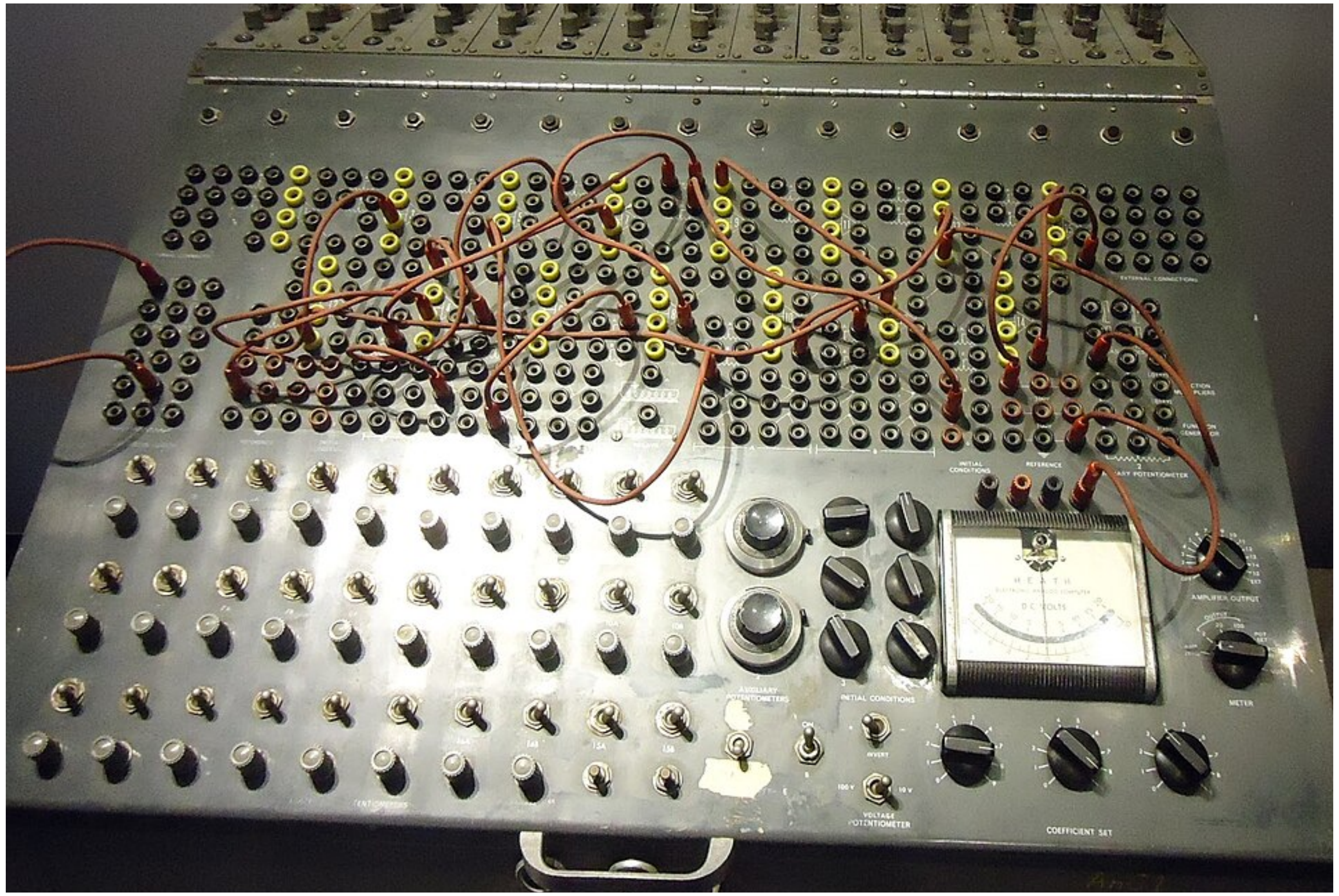
- ✦ Full of dangers & uncertainties in nature
- ✦ The stars in the sky move in a mysteriously orderly way

Oldest Computer?

- **Antikythera Mechanism** *Ancient Greek, ~150 BC*
  - Models the **solar system**
  - **Predict astronomical positions?**

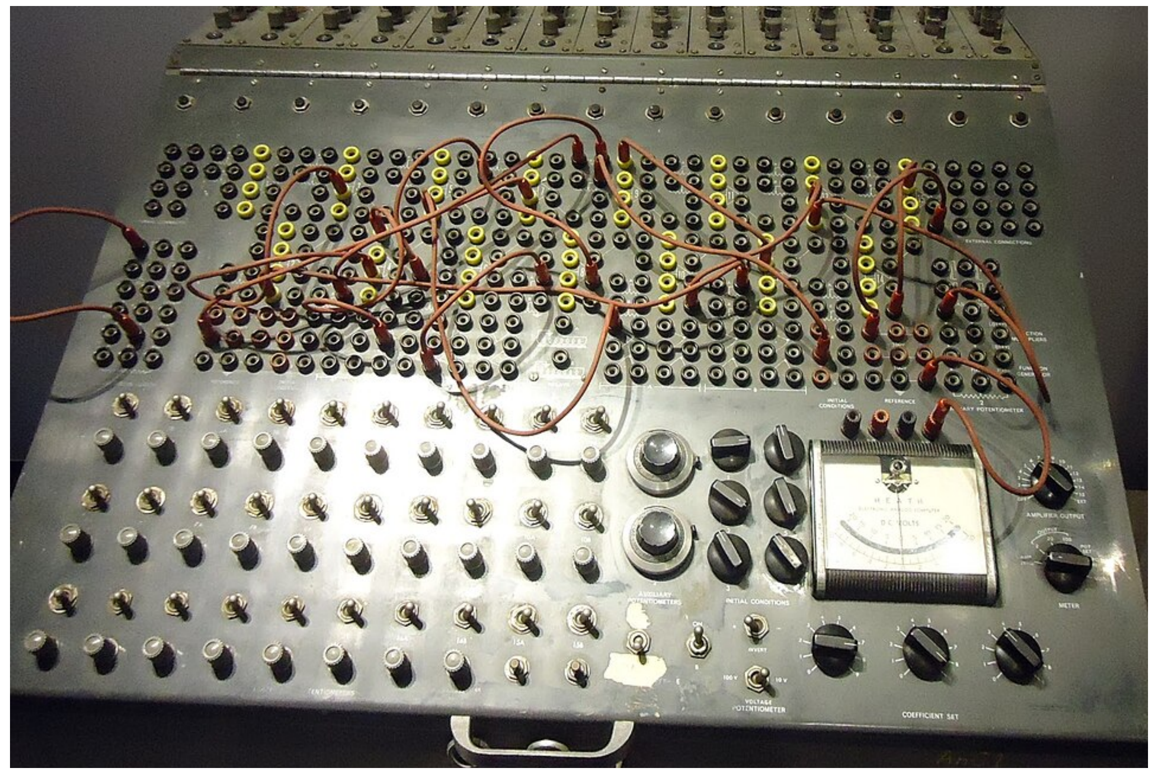


Humans – Machines



Software

Software



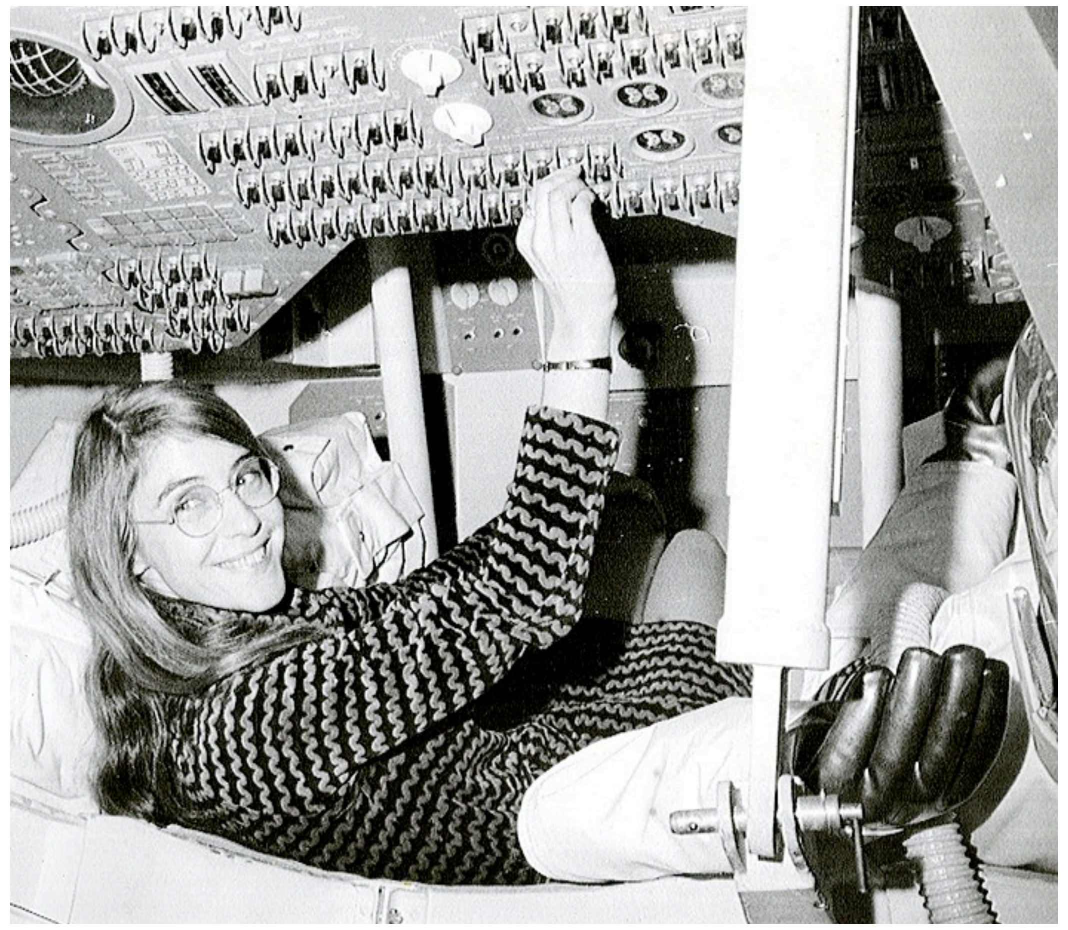
Software embodies ***human ideals***

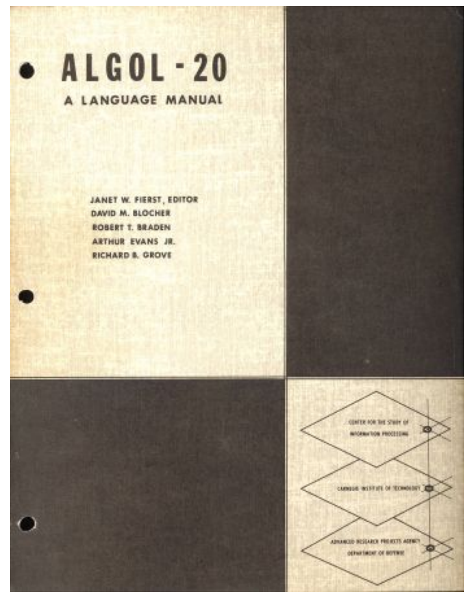
on ***finite physical machines***

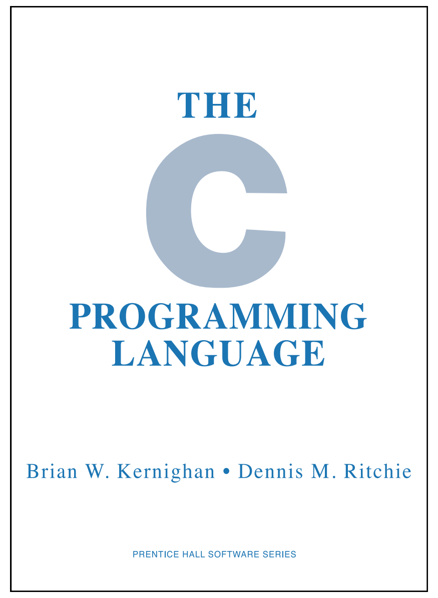
— Me, Sept 2025

Software Science & Engineering

- **Universal & pragmatic methodologies to develop high-quality, bug-free software**
  - M. Hamilton coined "**software engineering**"
  - **Science — Logic & Mathematics**
  - **Programming languages, types, …**

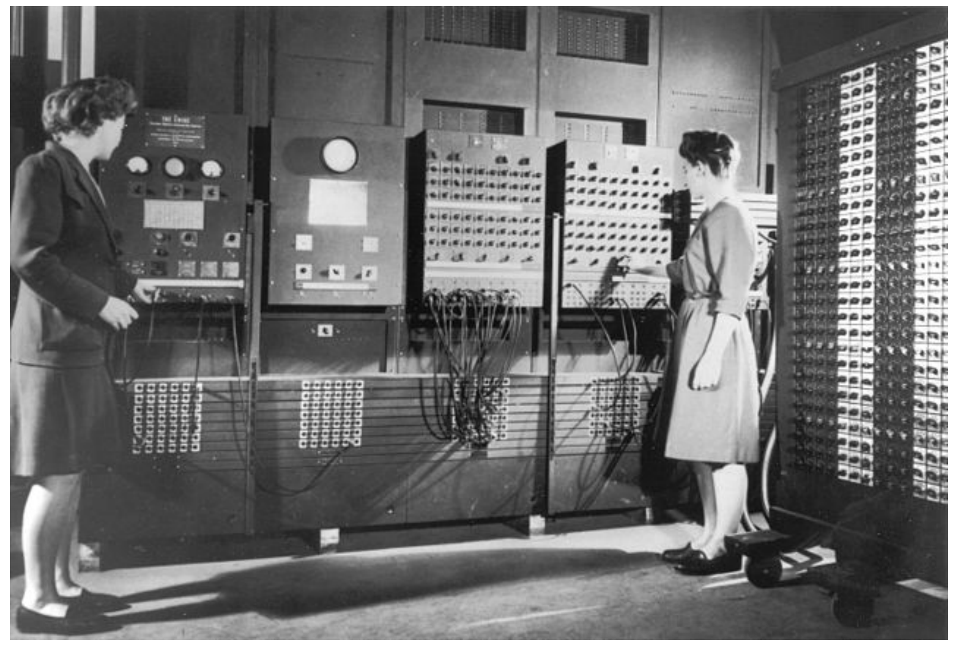






Evolution of **Programming**

- **1940's Physical settings**
  - Plugboard wiring & switches

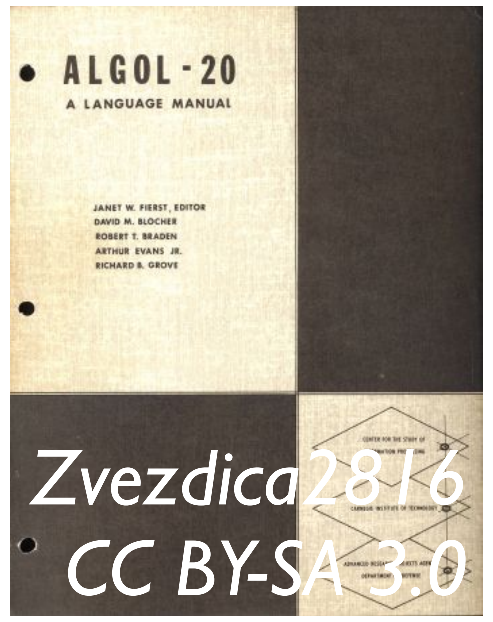


- **1947 Assembly languages**
  - Transliteration of machine code

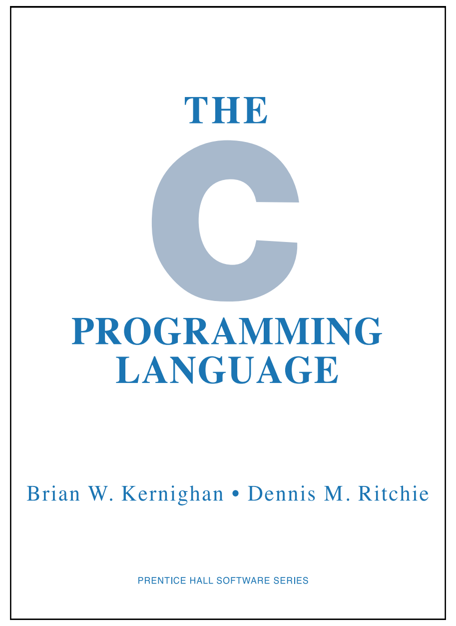
```
0013            RESETA  EQU    %00010011
0011            CTLREG  EQU    %00010001

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

History of **Types**

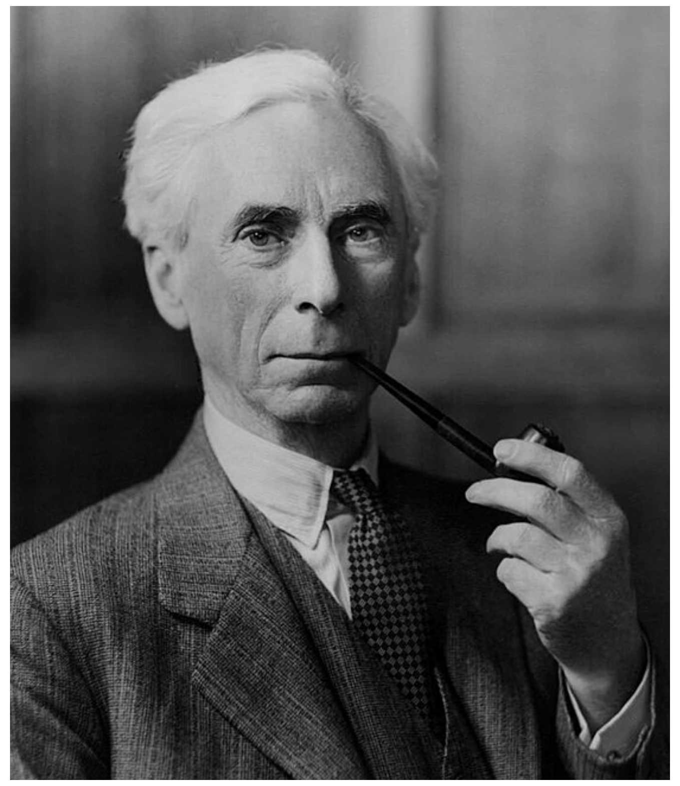
- **1903 Russel's work on types for logic**
  - To avoid **Russel's paradox** $\{S \mid S \notin S\}$
- **1940 Church's simply typed lambda calculus**
  - Types for a formal model of **computation**
- **1958 Types for programming languages**
  - To **prevent bugs** in software

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



Bertrand Russel

Alonzo Church

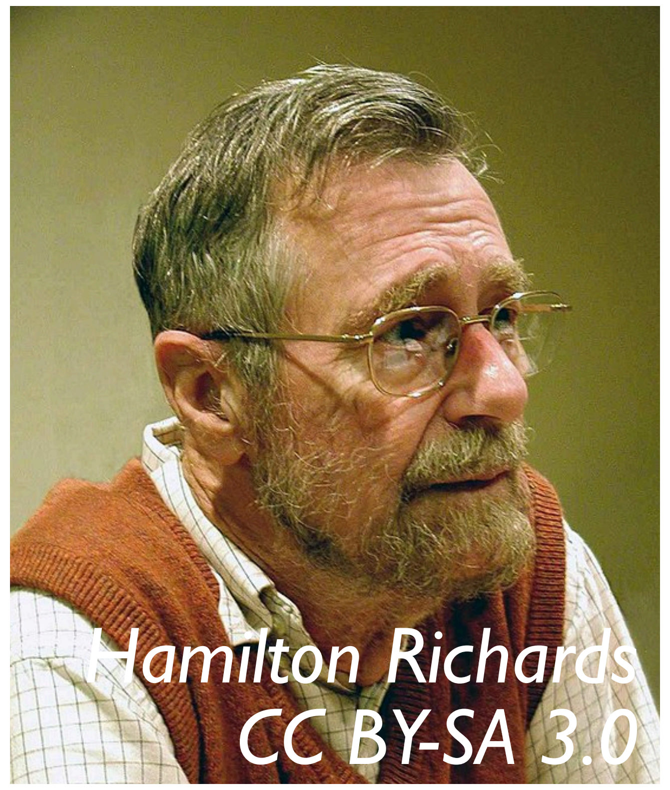
Formal **Verification**

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

LANGLEY FORMAL METHODS

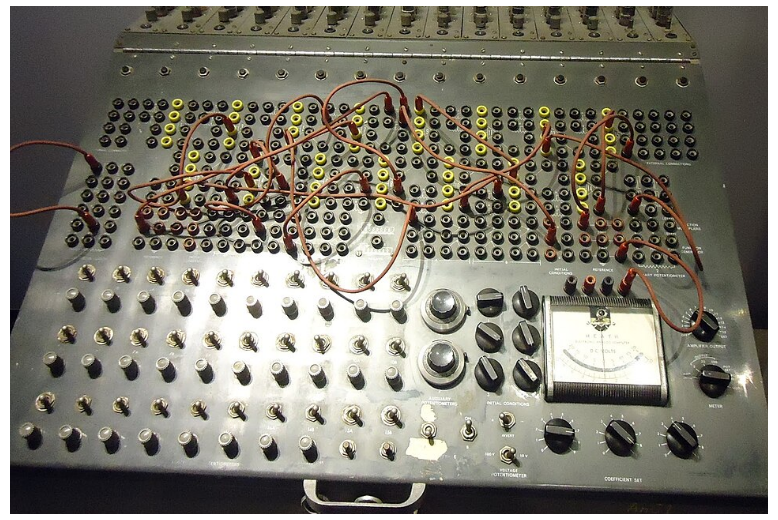
- **Program logics, esp. Hoare Logic '69**
  - Dijkstra's weakest precondition calculus '75
  - **Mathematical logic at work**

Tony Hoare



Edsger Dijkstra

Two Big Paradigms



Machines

?

Humans

Imperative

Direct control over physical computation

- ✔ *Better performance*
- ✔ *More flexibility*

Functional

High-level abstraction of compuation

- ✔ *Greater safety*
- ✔ *Easier reasoning*

Rust

The Rust Language

Rust

A language empowering everyone
to build reliable and efficient software.

GET STARTED

Version 1.83.0

Why Rust?

Performance

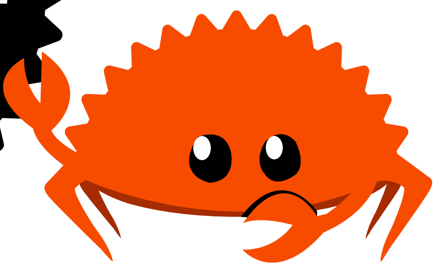
Rust is blazingly fast and memory-efficient: with no runtime or garbage collector, it can power performance-critical services, run on embedded devices, and easily integrate with other languages.

Reliability

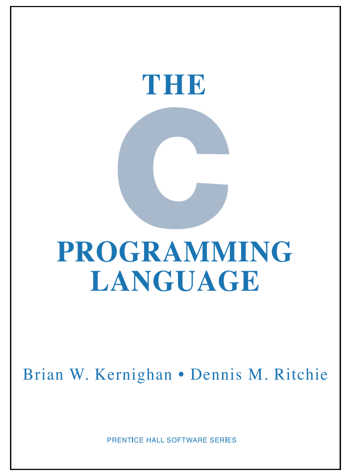
Rust's rich type system and ownership model guarantee memory-safety and thread-safety — enabling you to eliminate many classes of bugs at compile-time.

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.



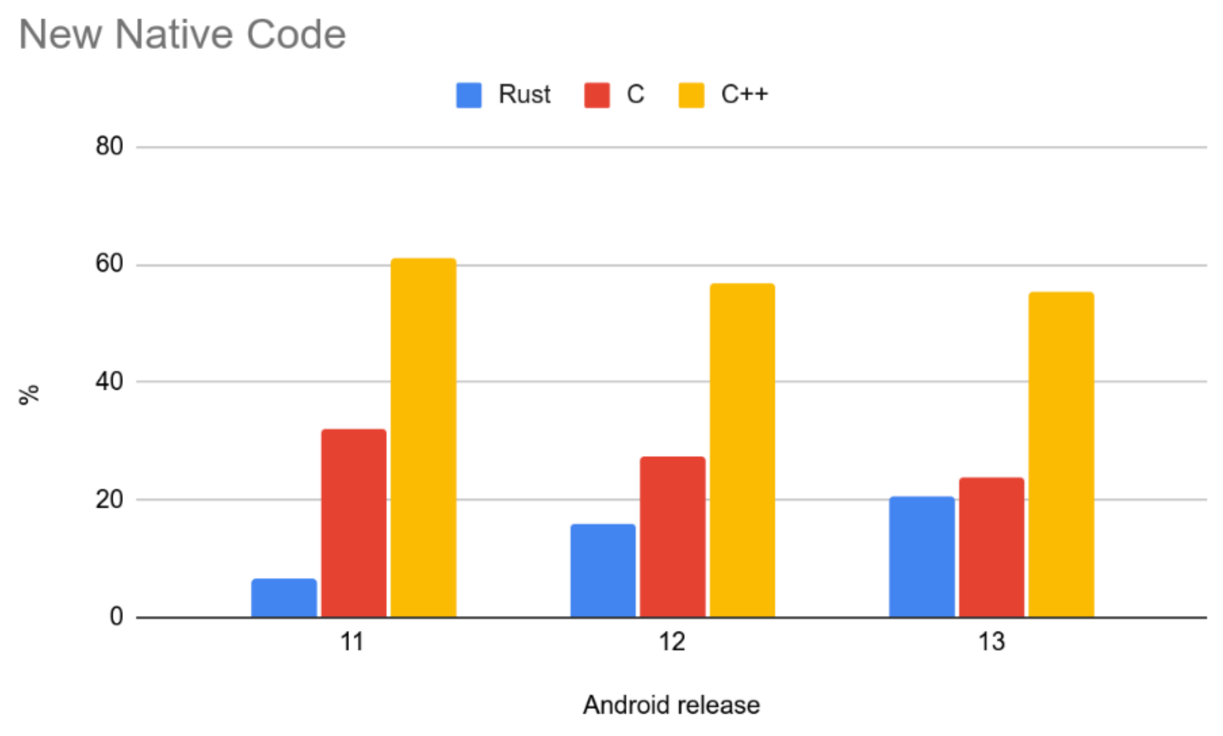
1972	1985	2015
C	C++	Rust



- Rust is imperative but more functional
  - Imperative — Low-level memory & thread controls
  - More functional — Ownership types to tame state mutation

Rust in the Real World

- 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

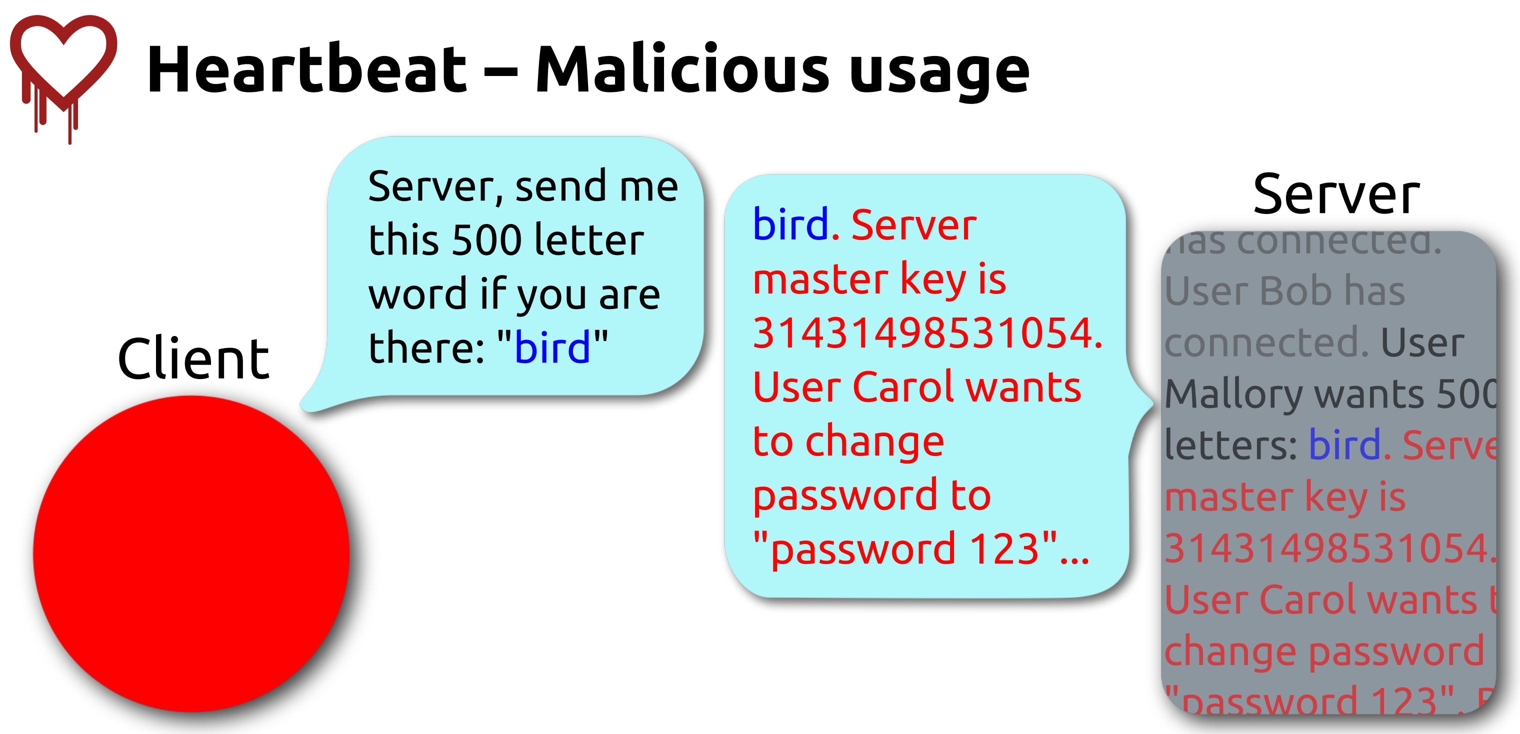
Days before Rust

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

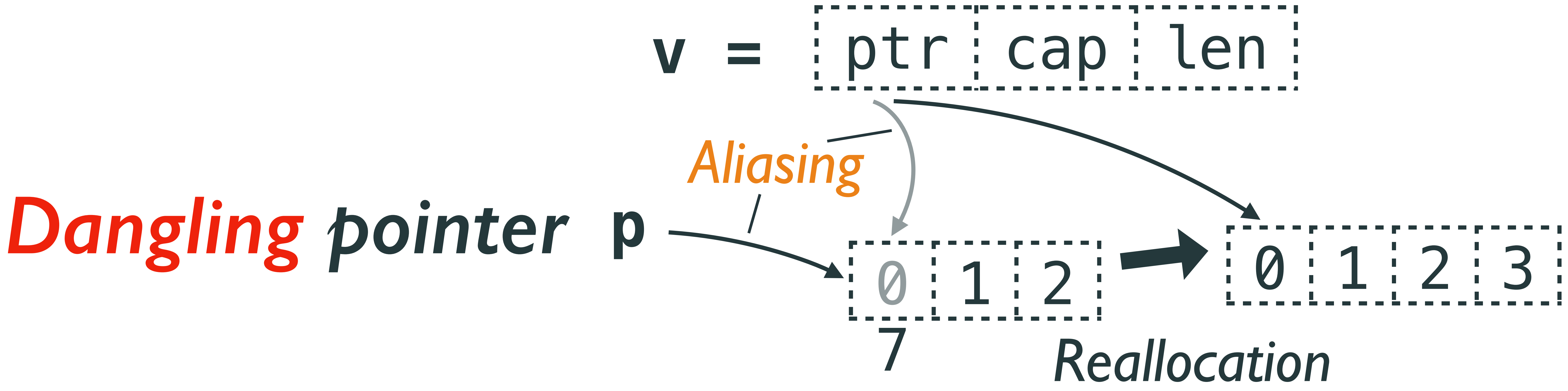
CVE	Vendor	Product	Type
CVE-2019-7286	Apple	iOS	Memory Corruption
CVE-2019-7287	Apple	iOS	Memory Corruption
CVE-2019-0676	Microsoft	Internet Explorer	Information Leak
CVE-2019-5786	Google	Chrome	Memory Corruption
CVE-2019-0808	Microsoft	Windows	Memory Corruption
CVE-2019-0797	Microsoft	Windows	[illegible]

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
```



Rust Bans Aliasing AND Mutability

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

Example

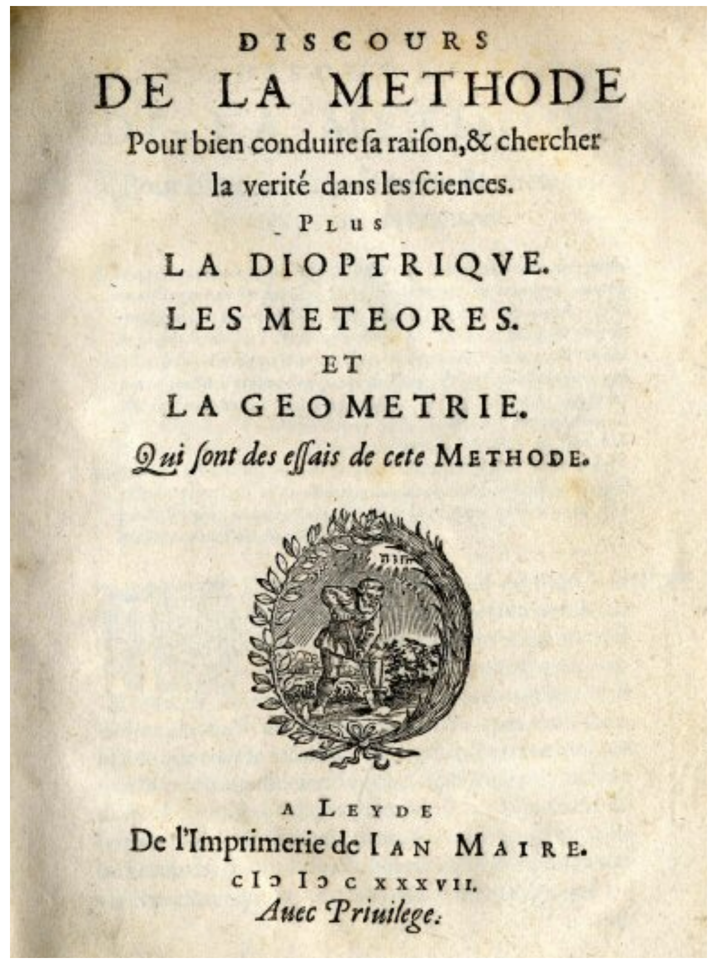
```
let mut v = vec![0, 1, 2];  let p = &mut v[0];
v.push(3);   *p = 7;  // OWNERSHIP ERROR
println!("{}", v[0]);
```

Bug detected!

```
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
```

Science behind Rust

Divide the Difficulties



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 cete METHODE.

A LEYDE
De l'Imprimerie de IAN MAIRE.
CIↄ Iↄ C XXXVII.
Auec Priuilege.

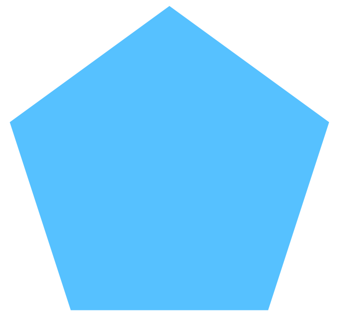
*The second: to **divide** each of the **difficulties** under examination into as **many parts** as possible, and as might be necessary for its **adequate solution**.*

Discourse on the Method, René Descartes

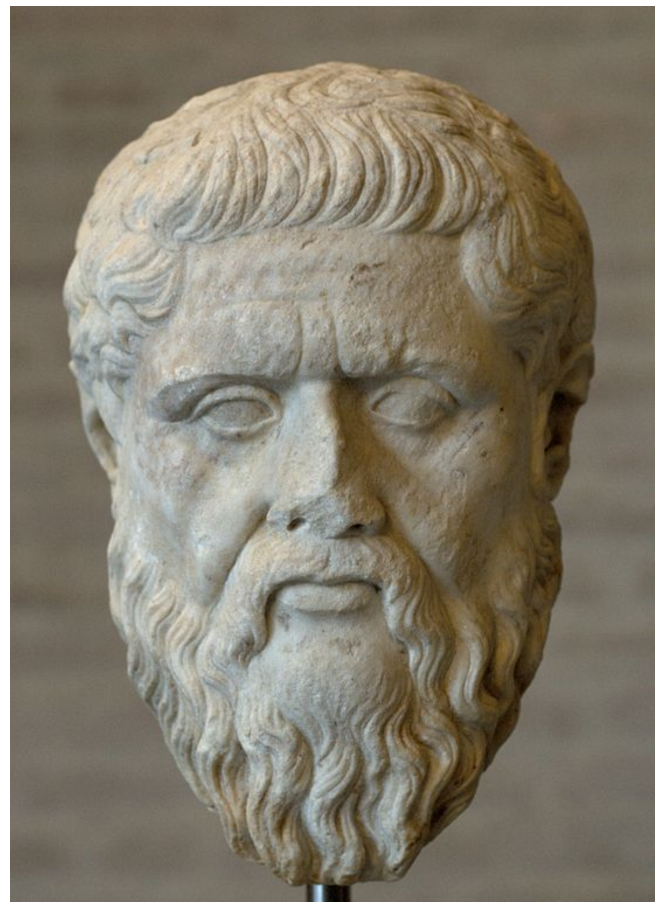
- ✦ **To overcome the difficulties, divide them**
  - ▸ **Modular & Composable → Scalable**

Human Ideals vs Physical Computation

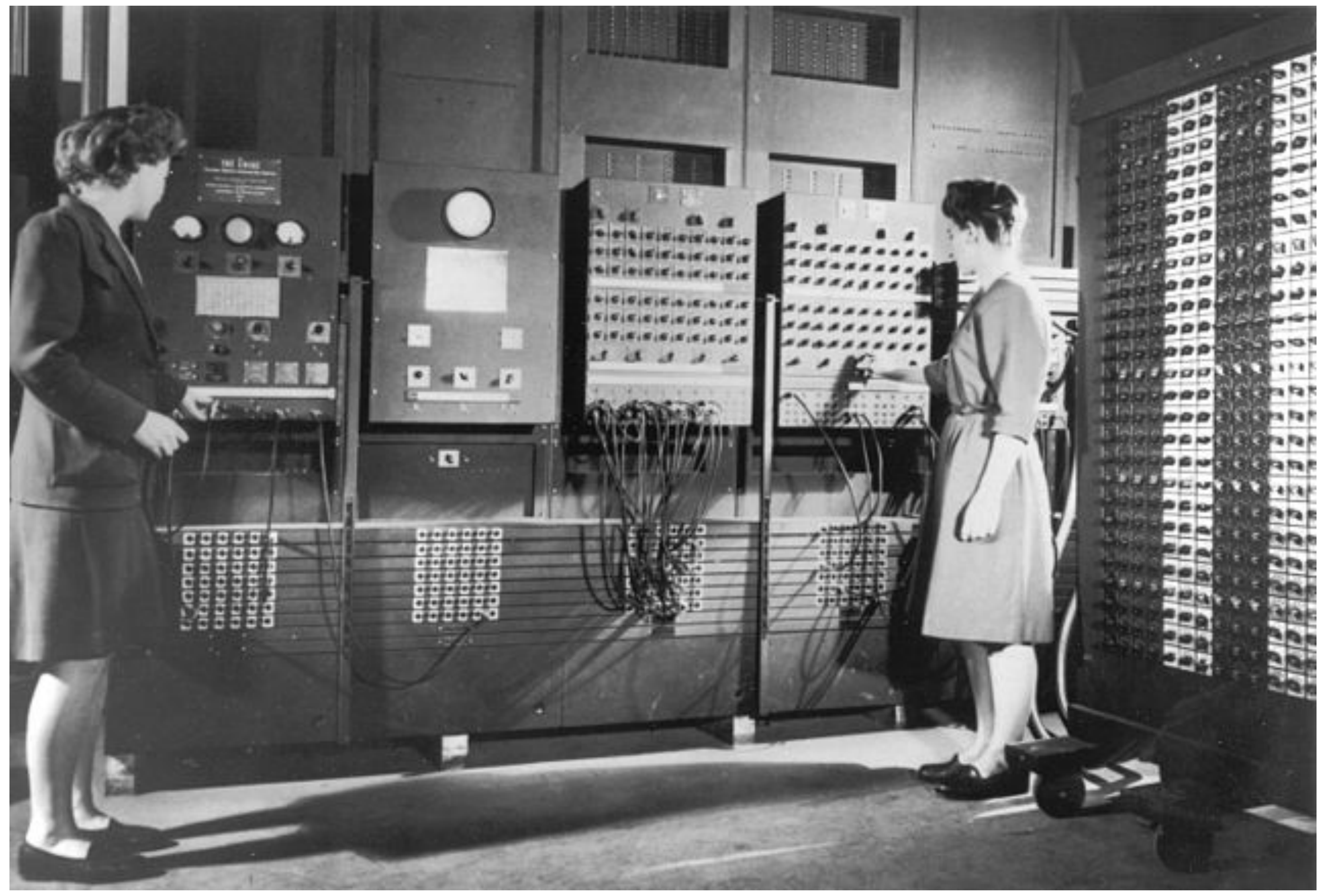
- Ideally, everything is just persistent
  - Metaphysics of Platonic Forms
  - Usual mathematics



$1 + 2 = 3$

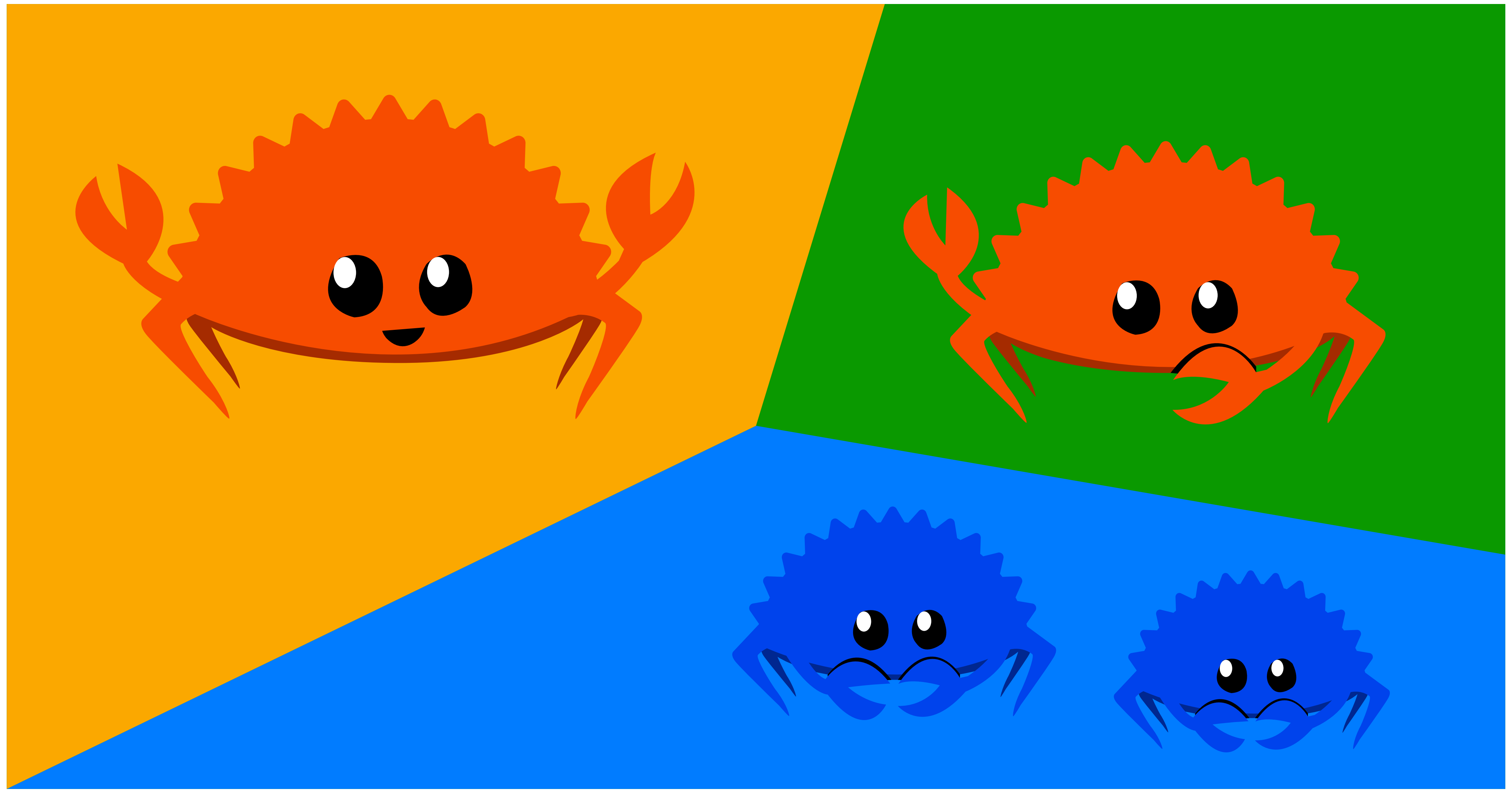


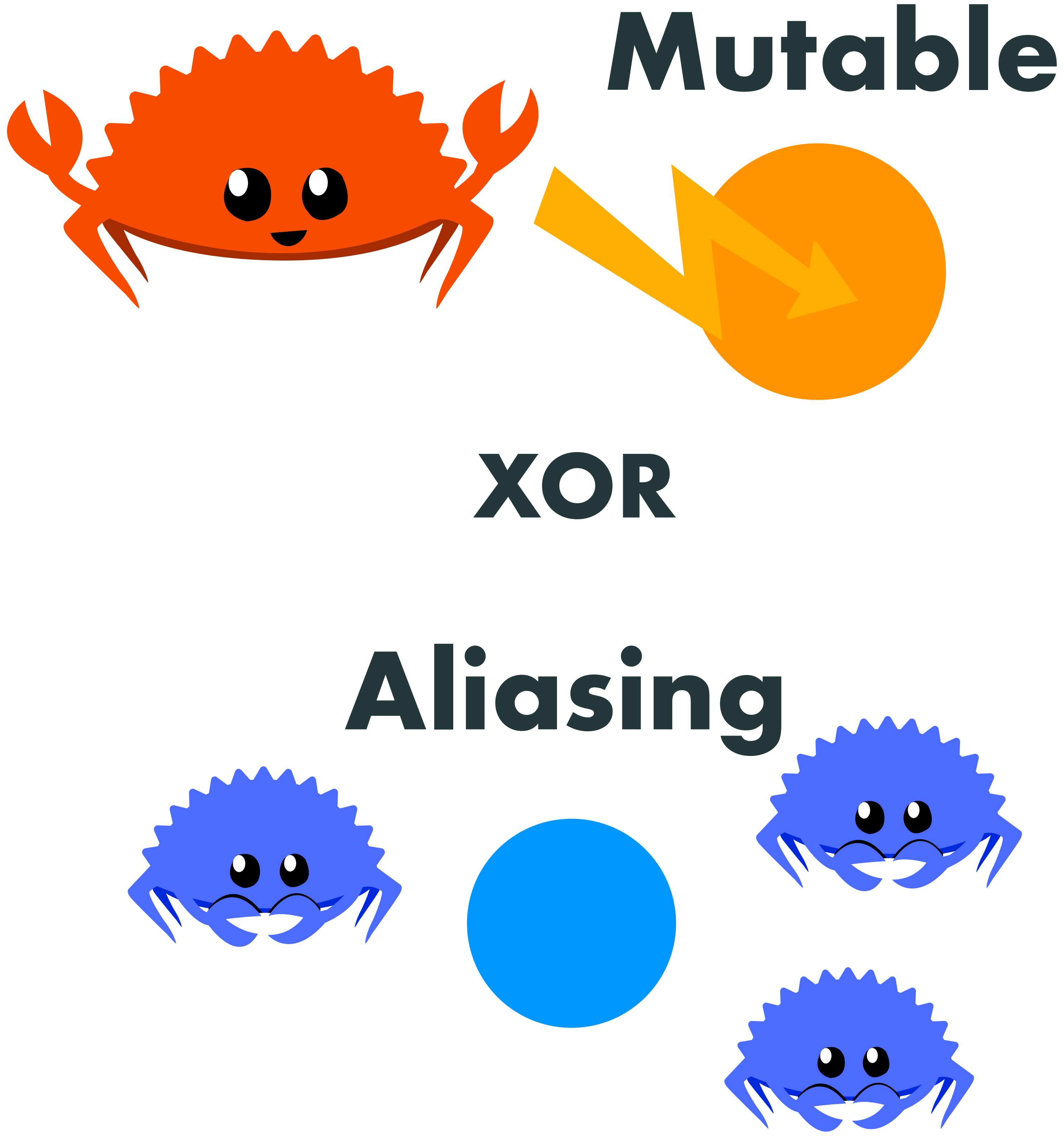
- Physical computation breaks things
  - Mutable state — Memory, environment, etc.
  - Naive reasoning can break by state update



Rust's Ownership Types

Separation by Ownership





→ ***No dangerous interference between computations***

From Logic to Rust

- Linear logic inspired Rust’s ownership types

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

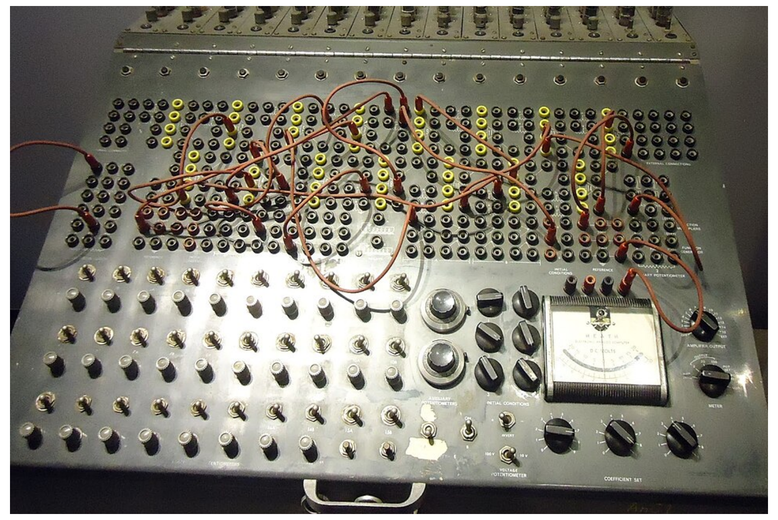
Ownership types
Clarke+ 1998

Cyclone
Grossman+ 2002
Borrows

Rust
Matsakis+ 2015

My Research

Big Goal of Software Science



Machines

?

Humans

Imperative

Direct control over physical computation

- ✔ *Better performance*
- ✔ *More flexibility*

Functional

High-level abstraction of compuation

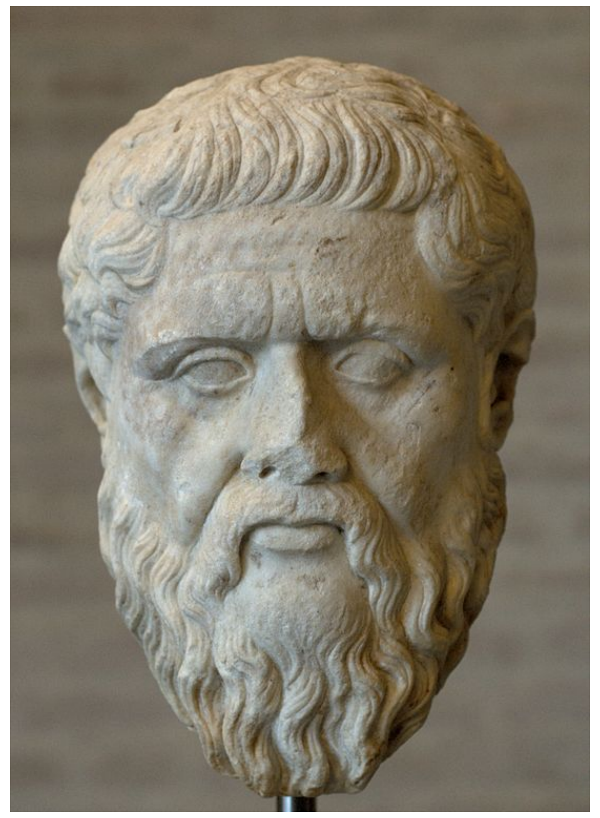
- ✔ *Greater safety*
- ✔ *Easier reasoning*

My Work — Unify Imperative & Functional

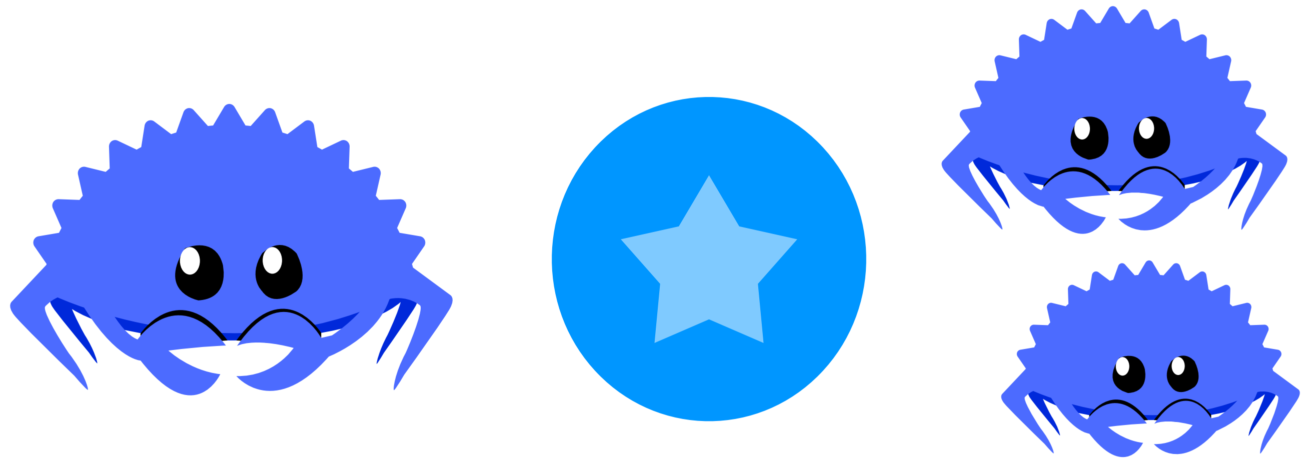
- ✦ **RustHorn** ESOP '20 / TOPLAS '21
  - ▸ **Rust** can be modeled as **functional** by **prophecies**
- ✦ **RustHornBelt** PLDI '22
  - ▸ Proving **RustHorn sound modularly** in **logic**
- ✦ **Pure Borrow** PLDI '26
  - ▸ **Linear Haskell + Rust-style borrows,** with **purity**

Basic Idea: Rust into Functional Models

- Turn Rust programs into functional models
  - Leveraging **Rust's ownership types**
  - Into the mathematical world where **all is persistent**

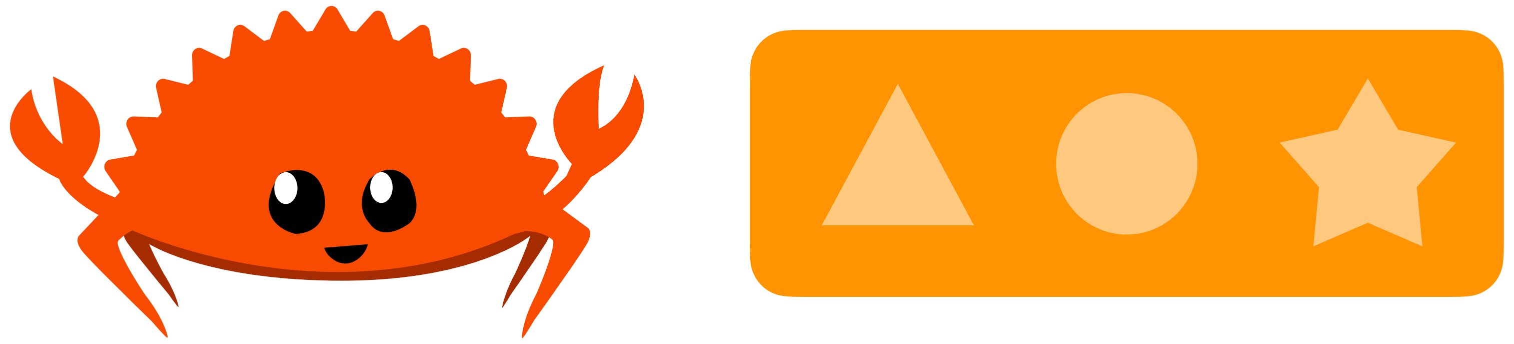


Aliased & Immutable



Just an immutable value

Fully owned



Can track the value change

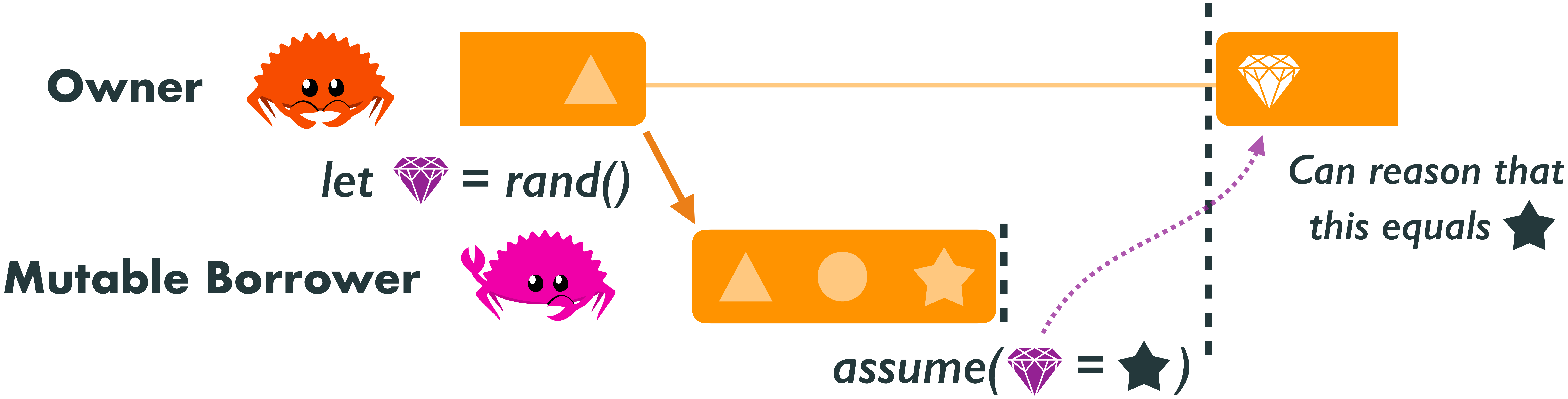
Challenge: Rust's Mutable Borrows

- 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**
    - Here, the **result** of the **borrower's mutation**



Example of RustHorn's Translation

Rust → ML

```
fn take_max<α>
    (ma: &α mut i32, mb: &α mut i32)
     -> &α mut i32 {
  if *ma >= *mb { ma } else { mb }
}

fn test(mut a: i32, mut b: i32) {
  *take_max(&mut a, &mut b) += 7;
  assert!((a - b).abs() >= 7);
}
```

```
let take_max (a, a') (b, b') =
  if a >= b then
       (assume(b' = b); (a, a'))
  else (assume(a' = a); (b, b'))

let test a b =
  let a' = rand() in let b' = rand() in
  let (r, r') =
    take_max (a, a') (b, b') in
  assume(r' = r + 7);
  assert(abs (a' - b') >= 7);
```

RustHorn's Idea in Diverse Contexts

- RustHorn: Fully automated Rust verifier
  - Find invariants with CHC solvers [Bjørner+ '15]
- Creusot: Automated Rust verifier [Denis+ '22]
- RusSOL: Rust program synthesizer [Fiala+ '23]
- Thrust: Rust verifier with refinement types [Ogawa+ '25]
- Verus: New automated Rust verifier [Lattuada+ '23/'24]
  - Initiated at **Microsoft Research**, highly practical

I’m Happy

Google Scholar   SIGN IN

Yusuke Matsushita

FOLLOW

Kyoto University
Verified email at fos.kuis.kyoto-u.ac.jp - Homepage

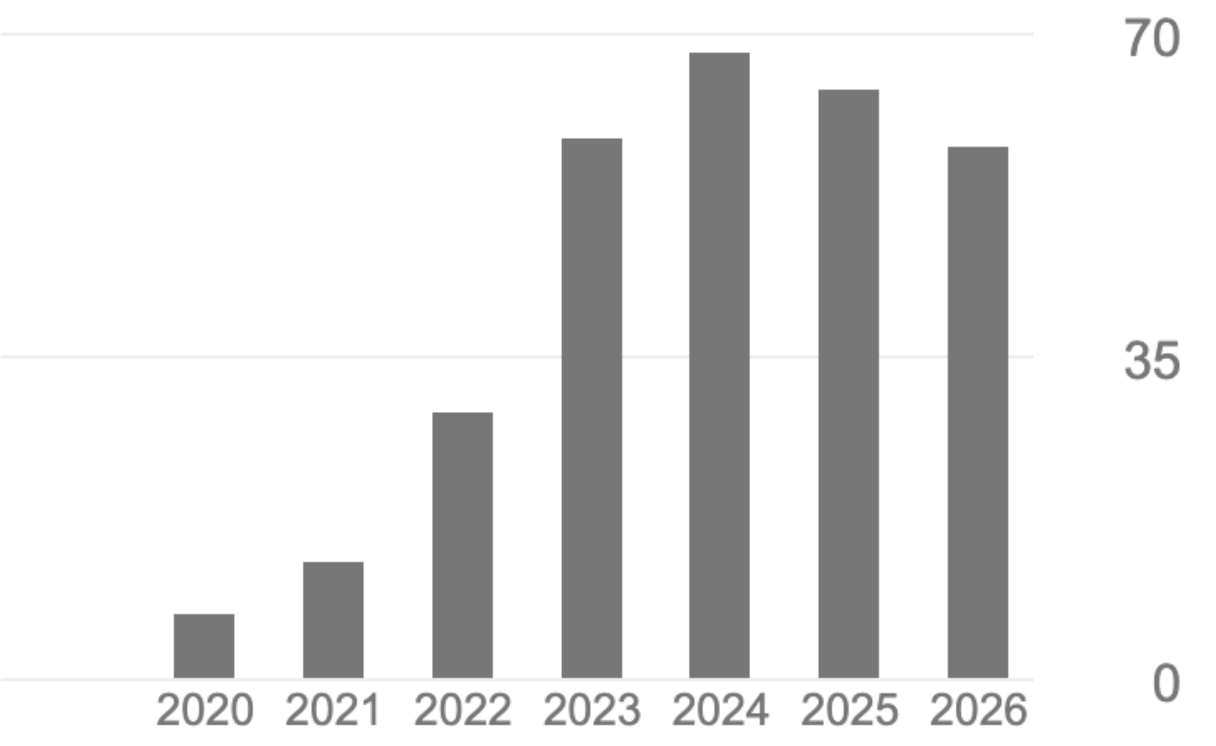
Computer Science   Programming Languages   Machine Learning

TITLE	CITED BY	YEAR
RustHorn: CHC-based Verification for Rust Programs Y Matsushita, T Tsukada, N Kobayashi ACM Transactions on Programming Languages and Systems (TOPLAS) 43 (4), 1-54	108	2021
RustHornBelt: A Semantic Foundation for Functional Verification of Rust Programs with Unsafe Code Y Matsushita, X Denis, JH Jourdan, D Dreyer Proceedings of the 43rd ACM SIGPLAN Conference on Programming Language …	94	2022
RustHorn: CHC-Based Verification for Rust Programs Y Matsushita, T Tsukada, N Kobayashi European Symposium on Programming (ESOP), 484-514	65	2020
Nola: Later-Free Ghost State for Verifying Termination in Iris Y Matsushita, T Tsukada Proceedings of the ACM on Programming Languages 9 (PLDI)	7	2025

GET MY OWN PROFILE

Cited by

	All	Since 2021
Citations	298	291
h-index	5	5
i10-index	3	3



Public access   VIEW ALL

0 articles   6 articles

Why Software Science & Engineering?

- **AIs are drastically changing software development**
  - Humans write less code, more high-level instructions
- **Writing code is easy; Achieving high quality is hard**
  - Systematic management of the software quality is crucial
- **Formal methods can really help**
  - Very strong safeguards, robust to human/AI errors
  - Writing proofs is becoming easy thanks to AIs + advances in SwS/E

Why Science?

- **Humans' intellectual curiosity is valuable**
  - We should not give up thinking
- **Keep distance from capitalism & money games**
  - It can be hard to stay sane amid a rapid flow
- **Engage in intellectual exploration rooted in ancient times**



Thank you!

Best of luck for your future

ありがとうございました

皆さんの未来に幸あれ