JONATHAN AMSTERDAM

Software Engineer, Google

Go in the New World





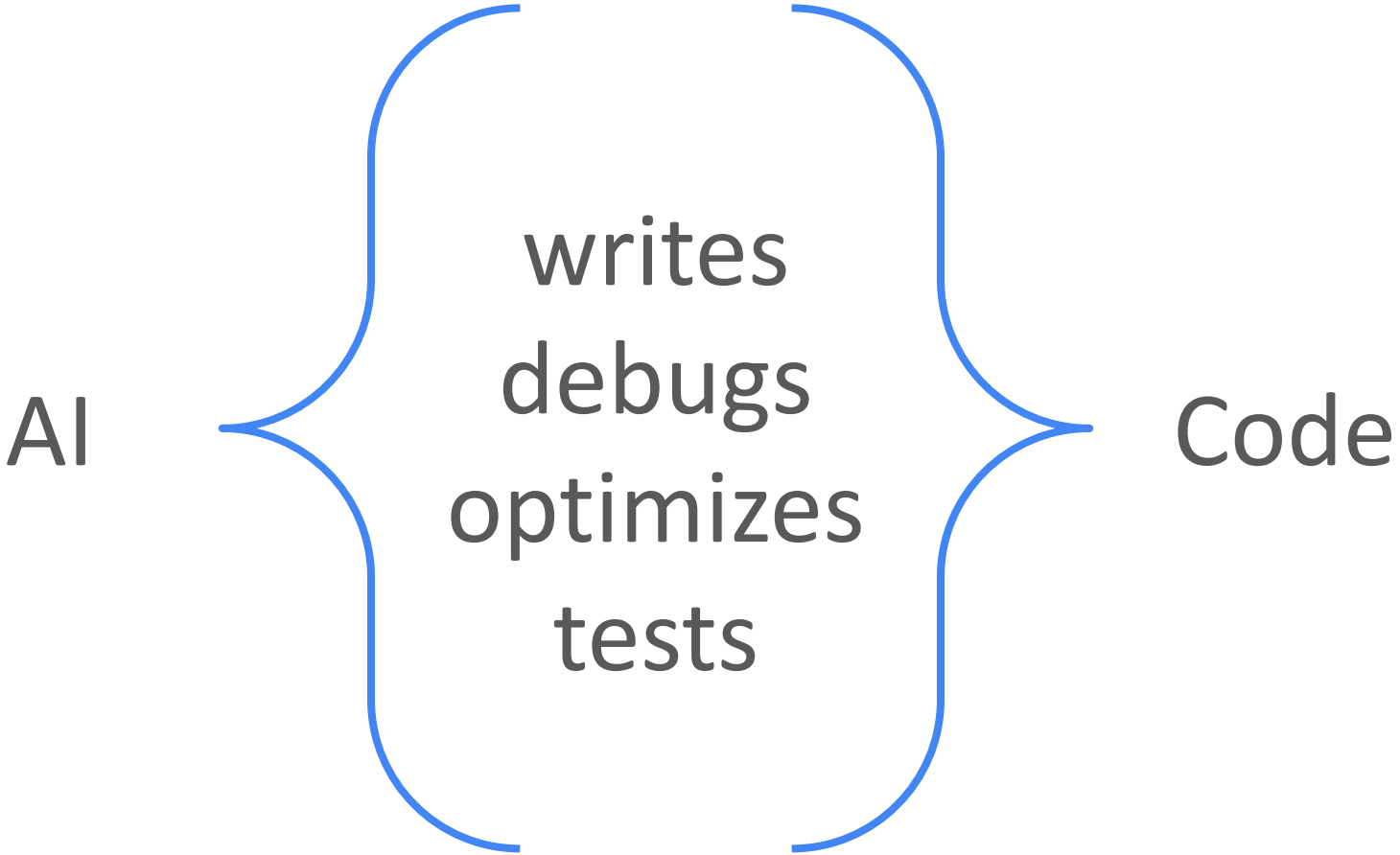
GOLAB the international conference on Go in Italy | November 1st - 3rd, 2026

OUTLINE

- What does the New World look like?
- Today’s Go in the New World
- The Verification Problem
- How Go is Evolving

What does the New World look like?



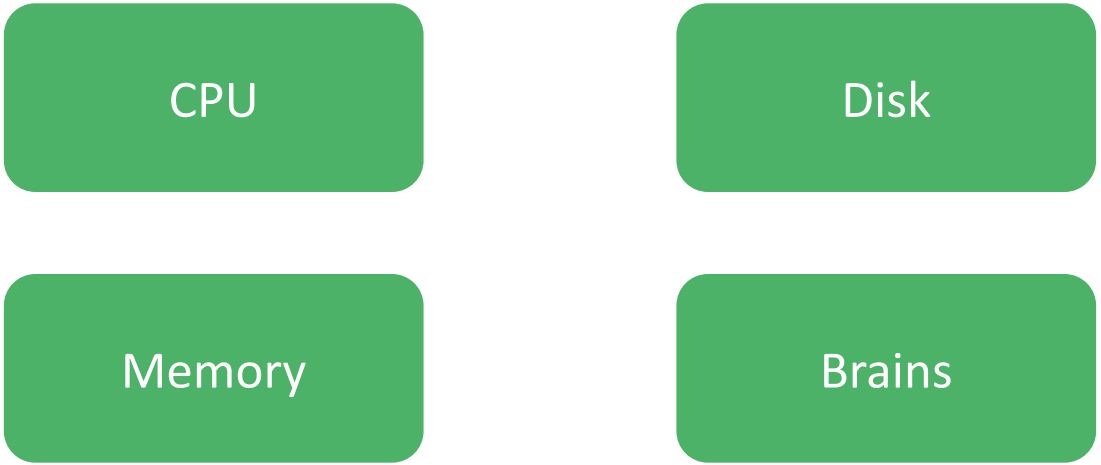

AI
writes
debugs
optimizes
tests
Code


Gopher Threads

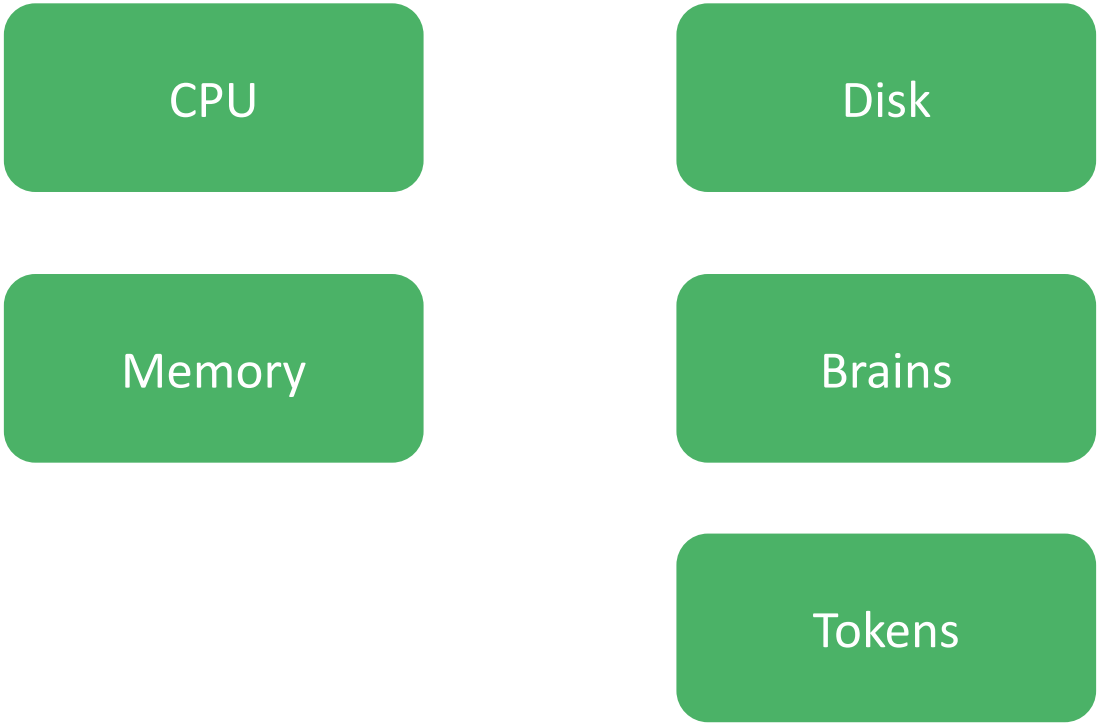

Abstract Design
+
The Real World
=


Abstract Design
+
The Real World
=
Engineering

Resources: The Old World



Resources: The New World



The Goal: Quality

Correct

Reliable

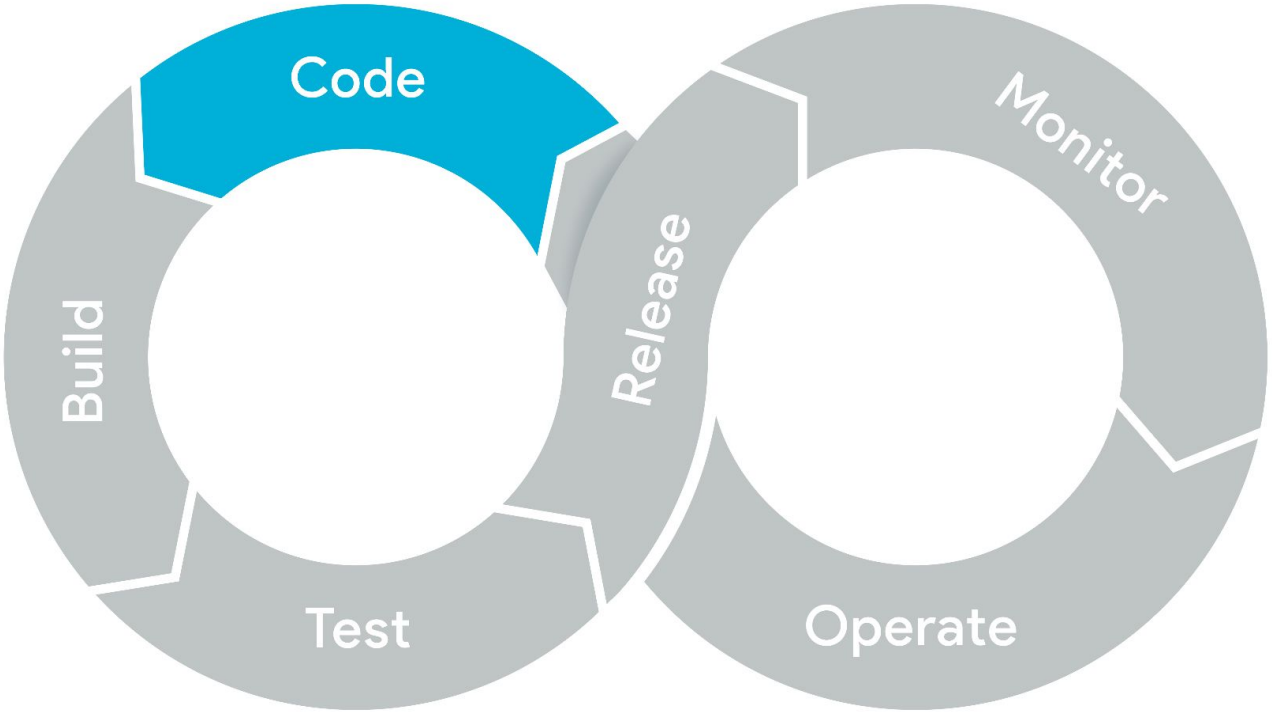
Efficient

Secure

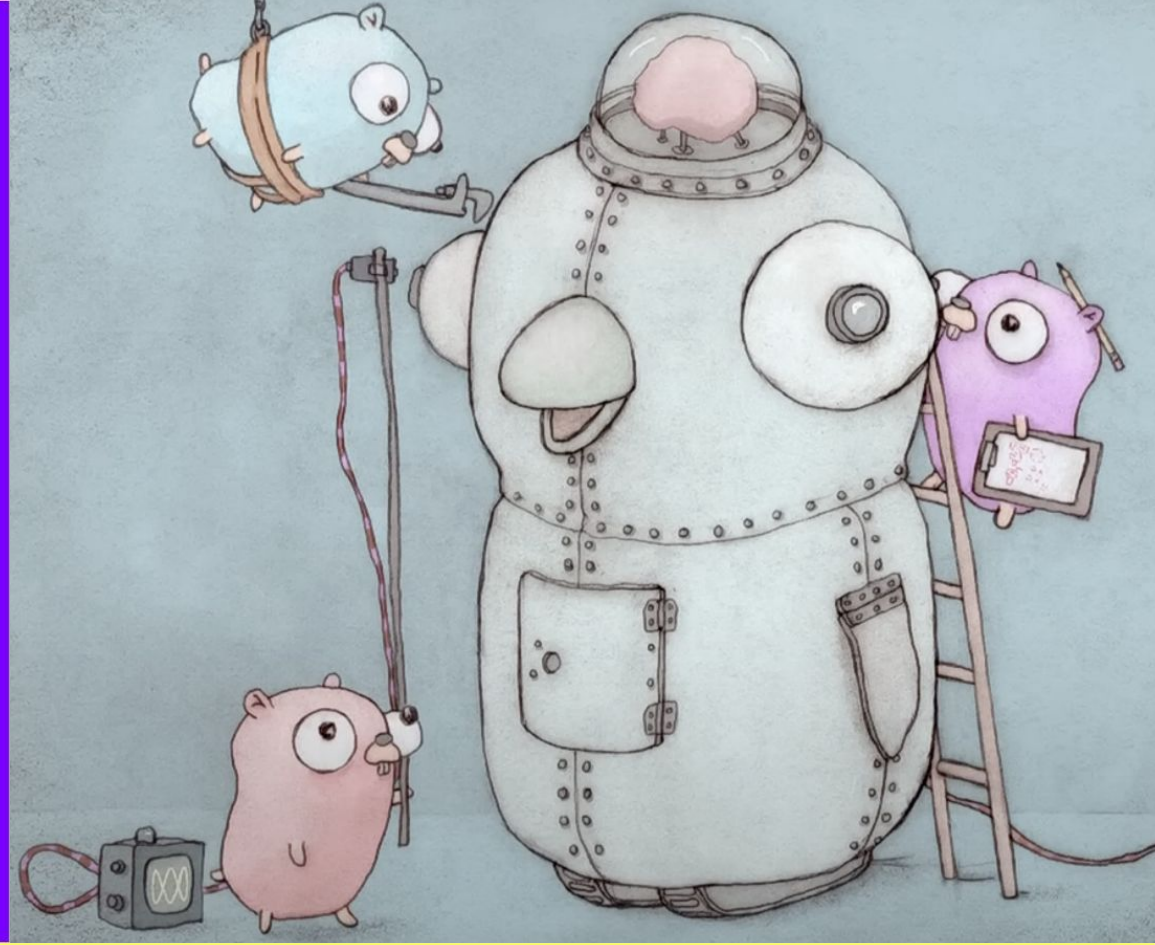
Scalable

Production-Ready

Maintainable


Code
Build
Test
Release
Monitor
Operate

TODAY'S GO IN THE NEW WORLD



Go is (still) productive

Token-efficient

Fewer round trips

Go was designed by and for people who write—and read and debug and maintain—large software systems . . . It is about **language design in the service of software engineering.**

Rob Pike
Go at Google: Language Design in the Service of Software Engineering (2012)

Features of the Go platform

Compatibility

Concurrency

Security

Production Readiness

Readability

Standard Library

Testing

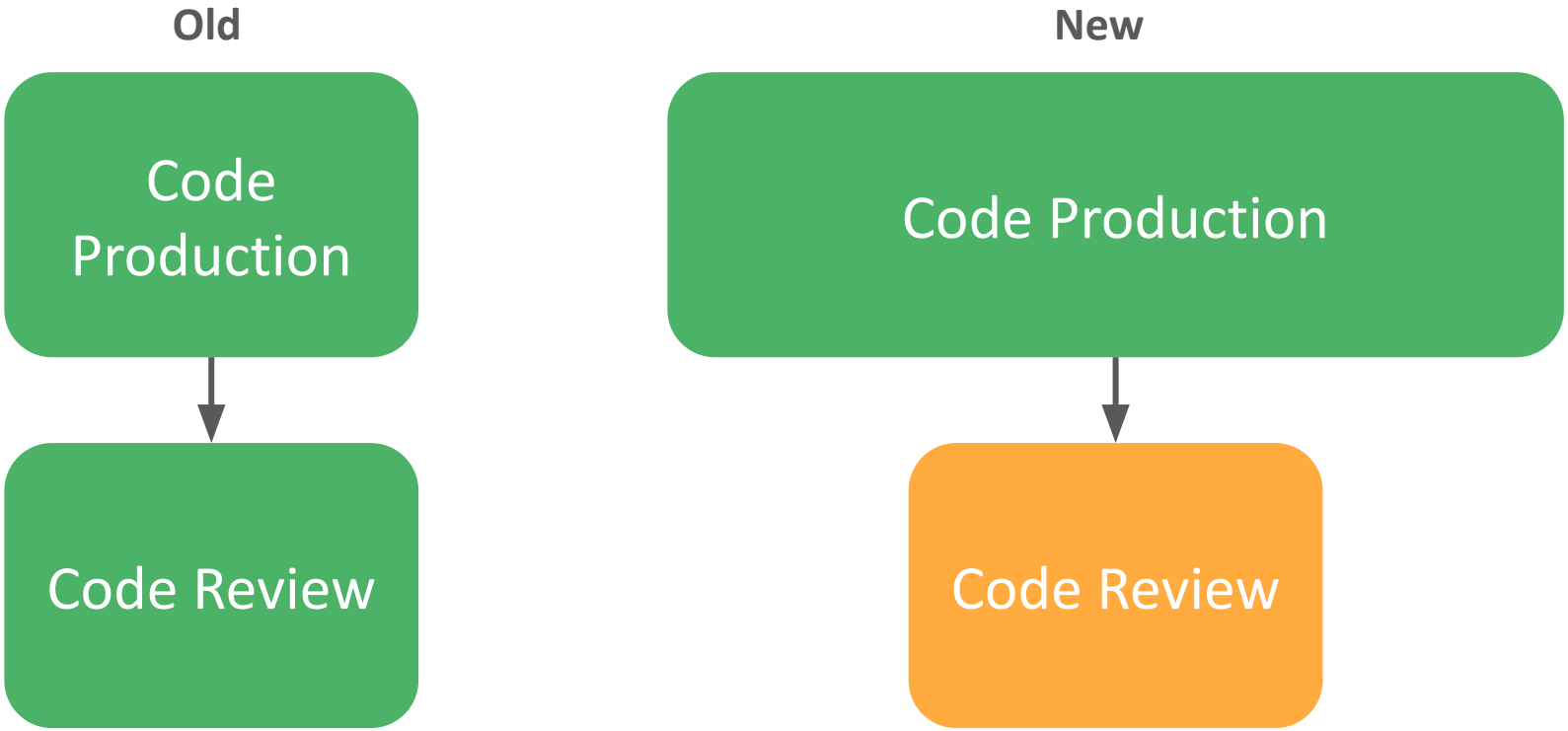
Readability

Because programs are read more often than they’re written.

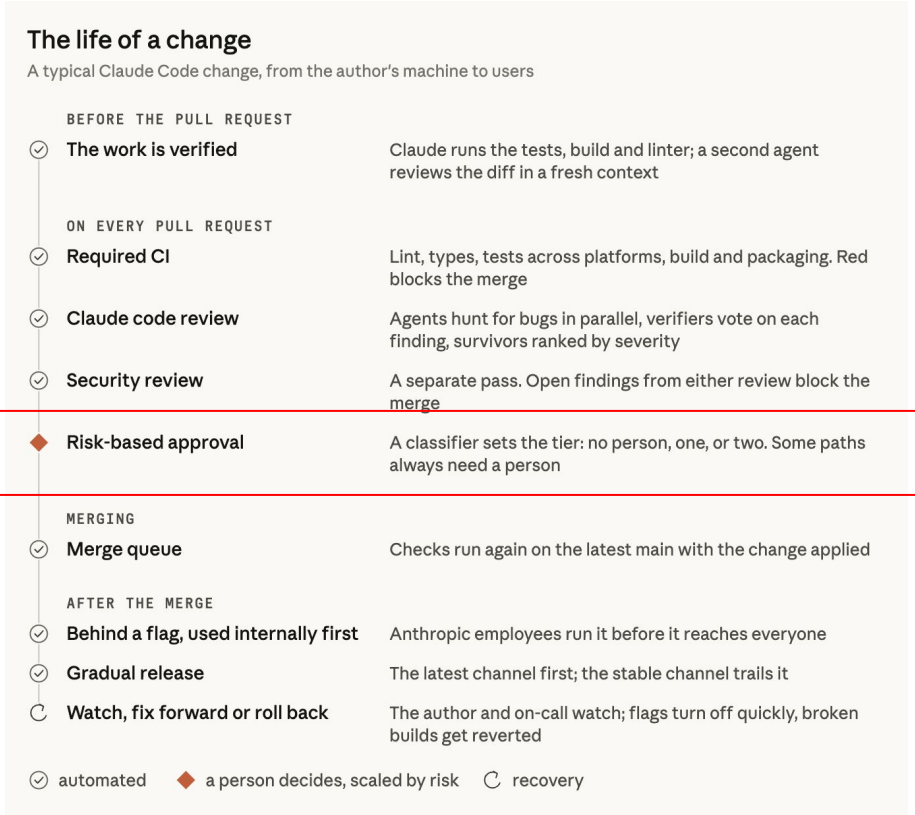
But what if they’re hardly ever read?



The code review crisis



How the Claude Code team does code review



https://claude.dev/blog/solving-code-review-at-anthropic

What makes Go readable?

Familiar syntax

Consistent formatting

Minimal context

Transparent syntax

What you see is what you get.

	Go	**Python**	**Rust**	**C++**
a = b	assignment	assignment	assignment	*anything*
f(x)	function call	*anything*	*anything*	*anything*
x - y	numeric subtraction	*anything*	*anything*	*anything*

Syntactic broccoli

The opposite of syntactic sugar.



The Go Blog

[On | No] syntactic support for error handling

Robert Griesemer
3 June 2025

One of the oldest and most persistent complaints about Go concerns the verbosity of error handling. We are all intimately (some may say painfully) familiar with this code pattern:

```
x, err := call()
if err != nil {
        // handle err
}
```

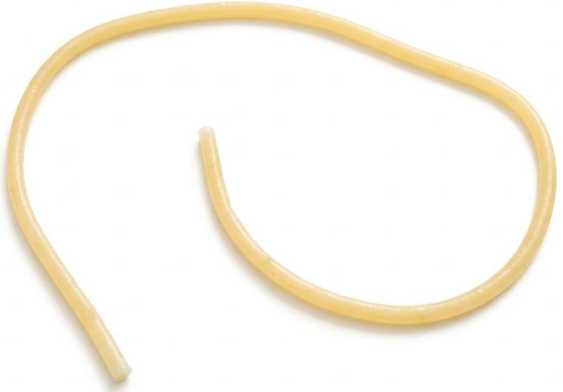
The test `if err != nil` can be so pervasive that it drowns out the rest of the code. This typically happens in programs that do a lot of API calls, and where handling errors is rudimentary and they are simply returned. Some programs end up with code that looks like this:

```
func printSum(a, b string) error {
    x, err := strconv.Atoi(a)
    if err != nil {
        return err
    }
    y, err := strconv.Atoi(b)
    if err != nil {
        return err
    }
```









The Standard Library

Focal point

Common interfaces

Idiomatic design

Risk and compliance

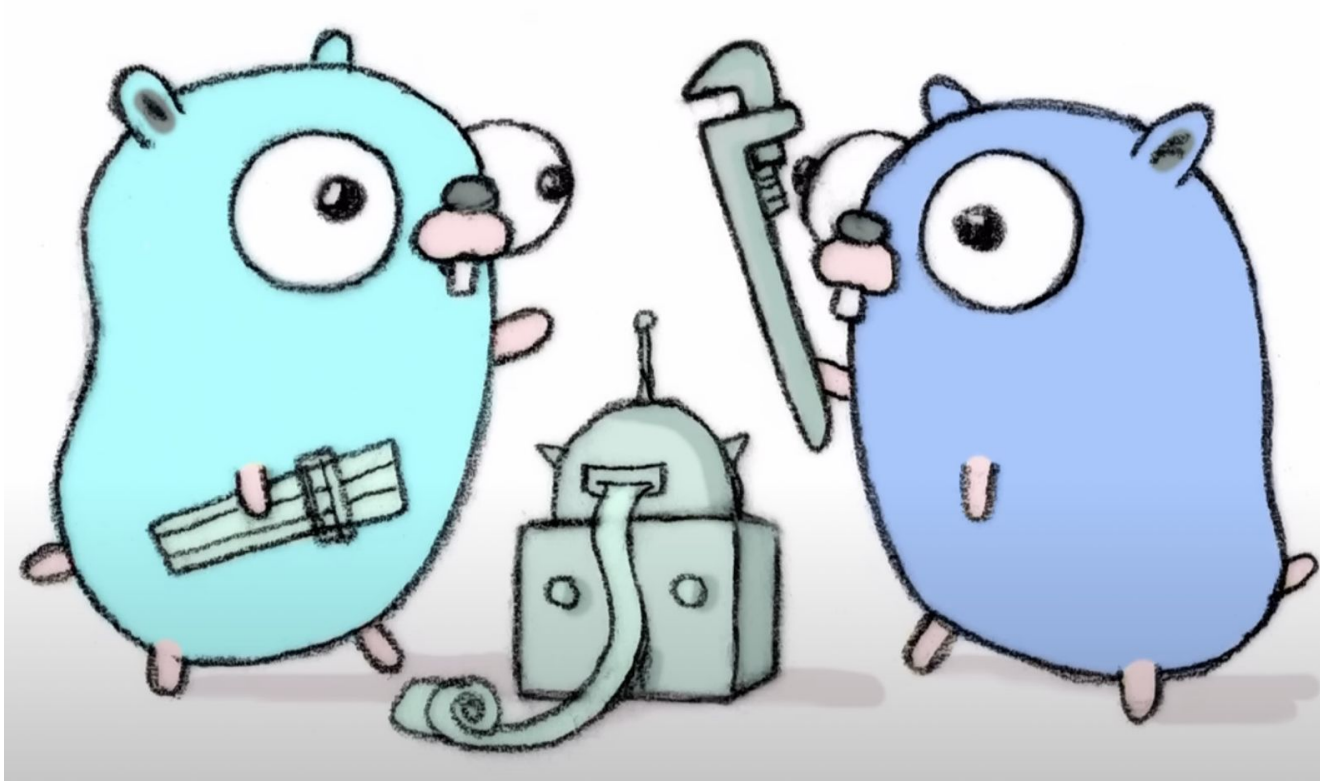


Testing

Unit test framework

Fuzz testing

synctest



Fuzz Testing

It’s not just for parsers.

- Panics anywhere
- Round tripping
- Efficient vs. obvious



An example of fuzzing

Write a fuzz test for sorted.Set in set.go.
First write a set of ints using an ordinary Go map as a reference. It should have the same API.
Then fuzz using a byte slice. Interpret bytes as mutating methods.
At the end of the byte slice compare the implementations using all query methods.

testing/synctest



THE VERIFICATION PROBLEM



Formal verification

Proving that a program has a property

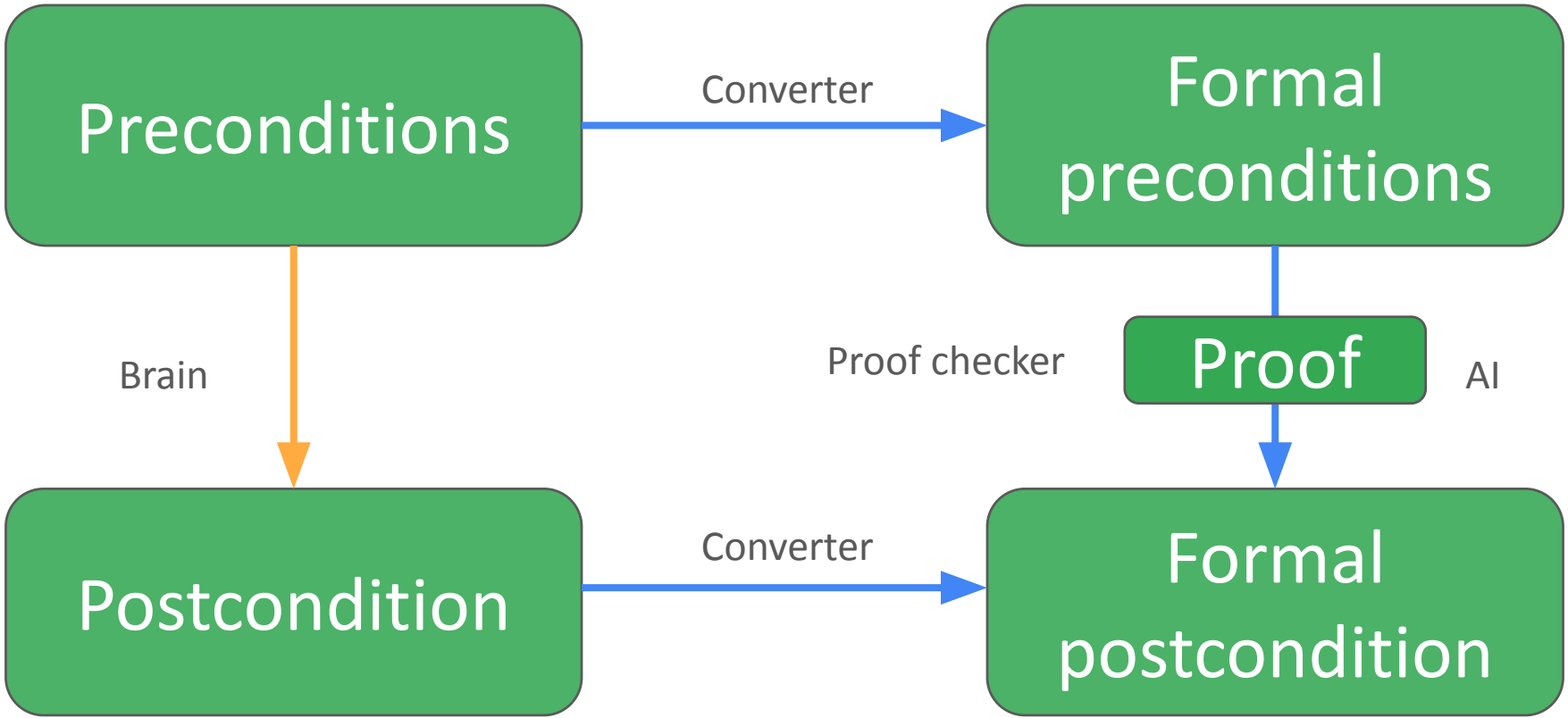
The old world

- Too much toil, not enough value

The new world

- Less work, bigger payoff

How formal verification works

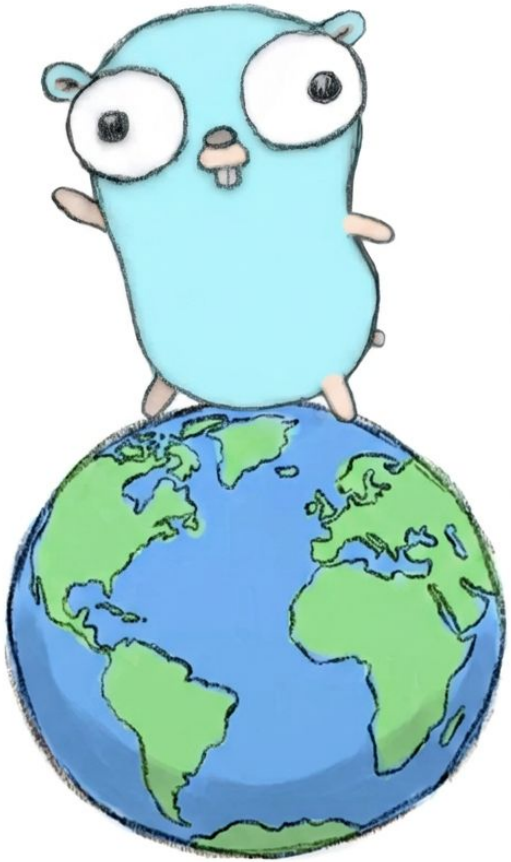


After Reinventing Mathematics, Can A.I. Eradicate Computer Bugs?

Using the same techniques that allowed A.I. technologies to crack a million-dollar math problem, researchers are building new systems that aim to solve the world's cybersecurity crisis.

The New York Times, 29 October 2026

HOW GO IS EVOLVING



Language and library additions

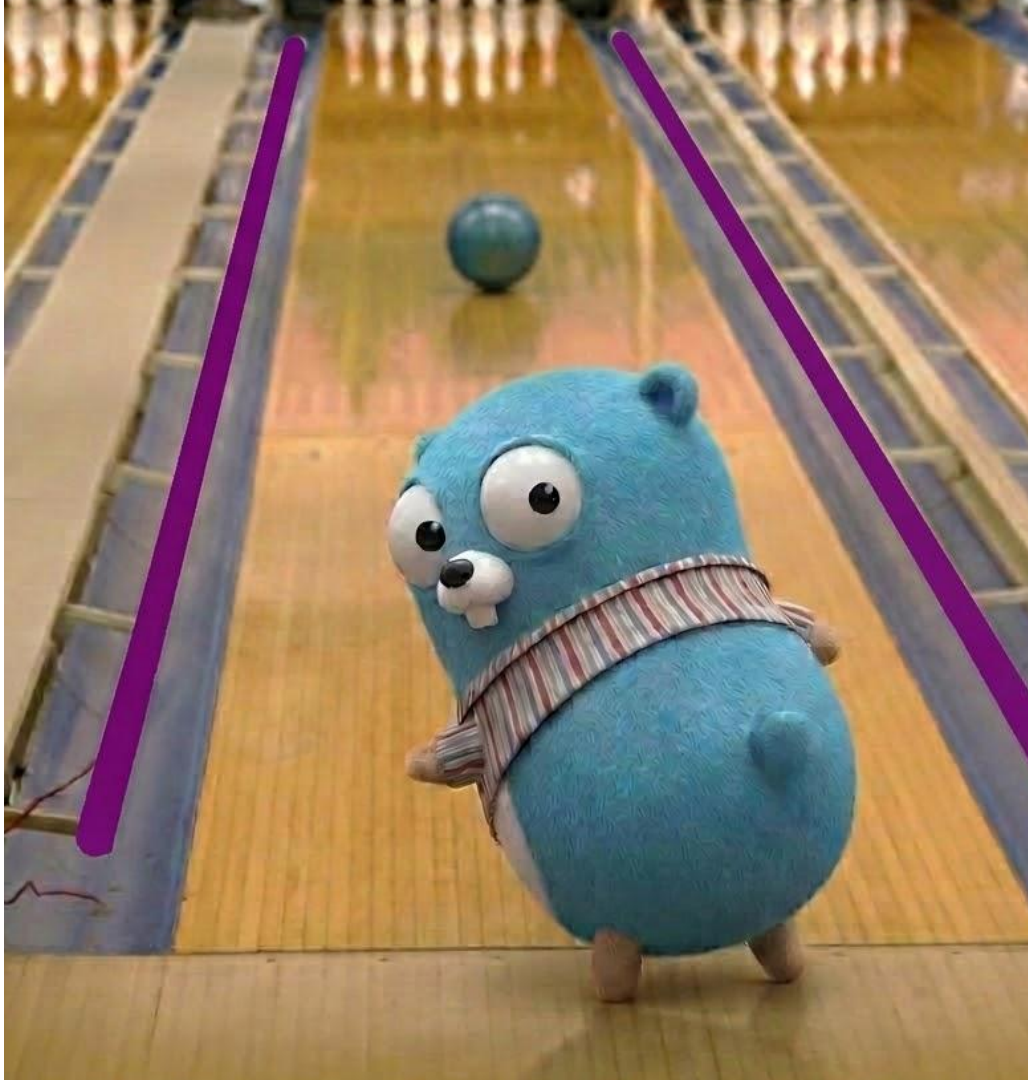
1.26	runtime/secret	Security
	Green Tea GC	Performance
	crypto packages	Security
1.27	SIMD	Performance
	Generic methods	Syntax
	encoding/json/v2	Reliability and Security
	Goroutine leak detection	Production Readiness

Improving the development loop with bumpers

Better tool output

Errors with fixes

Traces as friction logs



Shifting invariants left

Expose the underlying invariants of the codebase

- nilability
- mutability
- locking
- races

checklocks

Static analyzer for mutex and atomic errors

```
type Account struct {
    mu sync.Mutex
    // +checklocks:mu
    balance int
}
```

```
file.go:7:4: invalid field access, mu (&({param:a}.mu))
must be locked when accessing balance
```

Example 1: Nilness checking

Nils today: runtime error (sometimes)

```
func f(x int) int {
    var p *int
    if x >= 0 {
        p = &x
    }
    return *p
}
```

runtime error

Example 1: Nilness checking

A simple “this is never nil” type won’t work in Go.

```
func f(x int) int {
    var p nonnil *int = &x
    ...
}
```

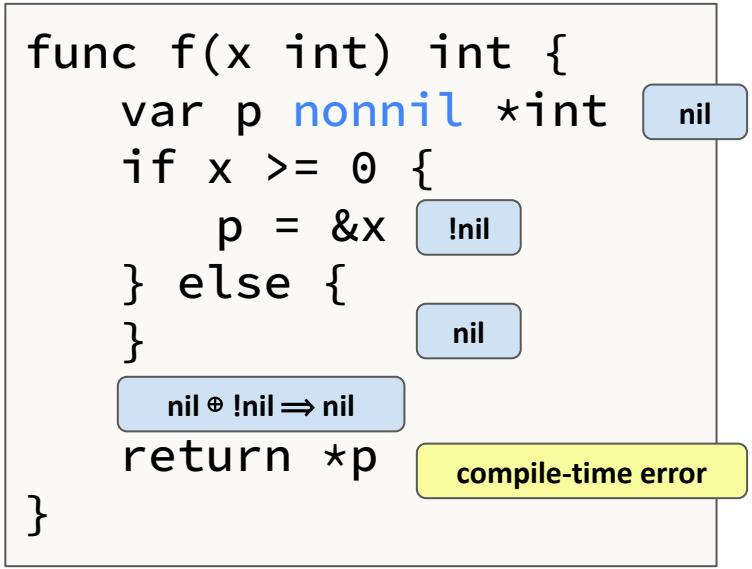
Sometimes you can avoid the zero value ...

```
type Thing struct {
    p nonnil *int
}
```

but sometimes you can’t.

Example 1: Nilness checking

Typestate analysis



Example 2: Unions

“This type is an A or a B, and nothing else.”

An Optional[T] is either a T or it’s nothing.

A tree node is either an internal node or a leaf.

Unions in Go today: interfaces

```
type Node[T any] interface {
    isNode()
}

type Internal[T any] struct { left, right Node[T]}

func (*Internal[T]) isNode() {}

type Leaf[T any] struct { value T }

func (*Leaf[T]) isNode() {}
```

Design 1: unions are interfaces

```
type Node[T any] interface { Internal[T] | Leaf[T] }

type Internal[T any] struct { left, right Node[T]}

type Leaf[T any] struct { value T }
```

Design 2: unions are like structs

```
type Node[T any] union {
    Internal struct { Left, Right Node[T] }
    Leaf     T
}
```

```
var root Node[int]
root.Internal.Left = Node[int]{Leaf: 2}
root.Internal.Right := Node[int]{Leaf: 5}
fmt.Println(root.Internal.Left.Leaf)
```

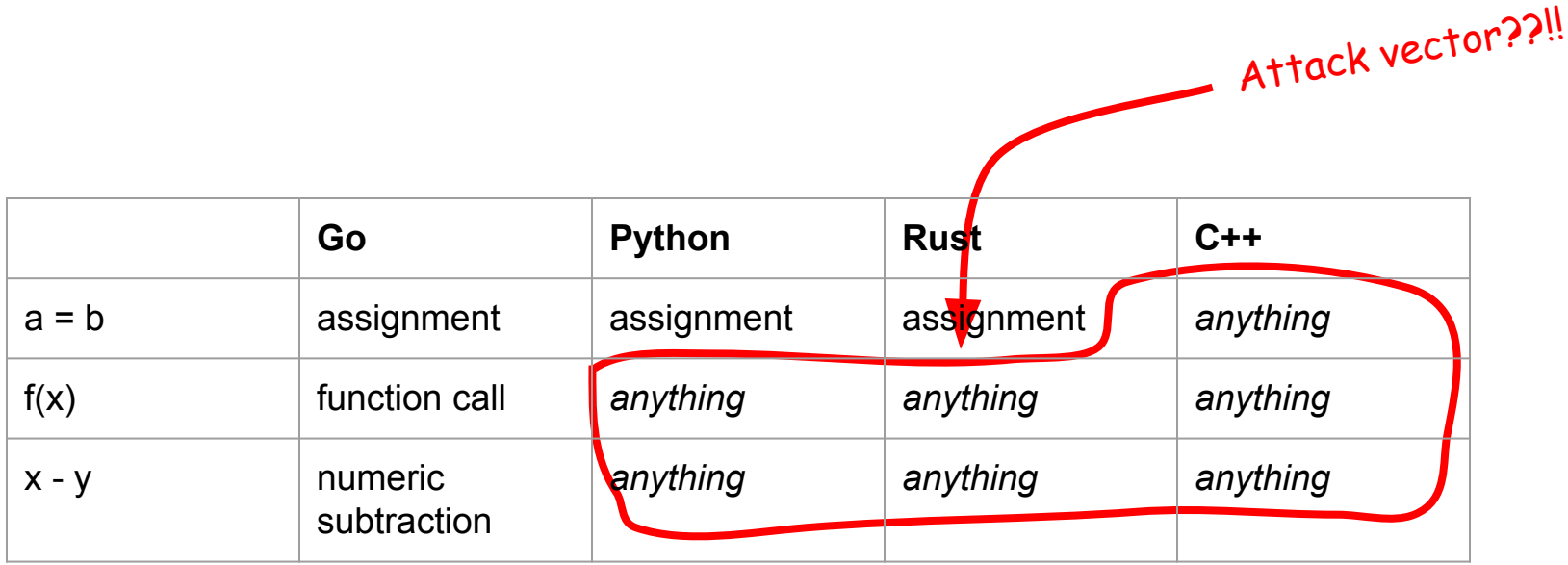

Thank you!
Go
Next '24
GO
Android
GopherCon
Korea 2024



Thank you



Operator overloading



	Go	Python	Rust	C++
a = b	assignment	assignment	assignment	*anything*
f(x)	function call	*anything*	*anything*	*anything*
x - y	numeric subtraction	*anything*	*anything*	*anything*

Compatibility

The Go 1 promise

Module versioning

> “As always, the release **maintains the Go 1 promise of compatibility**. We expect almost all Go programs to continue to compile and run as before.”
>
> -Go Release Notes

Reliability

Static type system

Memory safety

Near-realtime, self-tuning GC

Goroutine leak detector

Security

Memory safety

Crypto libs

govulncheck

go.sum + sumdb

Comprehensive standard library

Production Readiness

Single binary

Small footprint

Profiles

Execution tracing

Runtime stats

Intuitive Concurrency

