Why Go Rocks for Building a Lua Interpreter

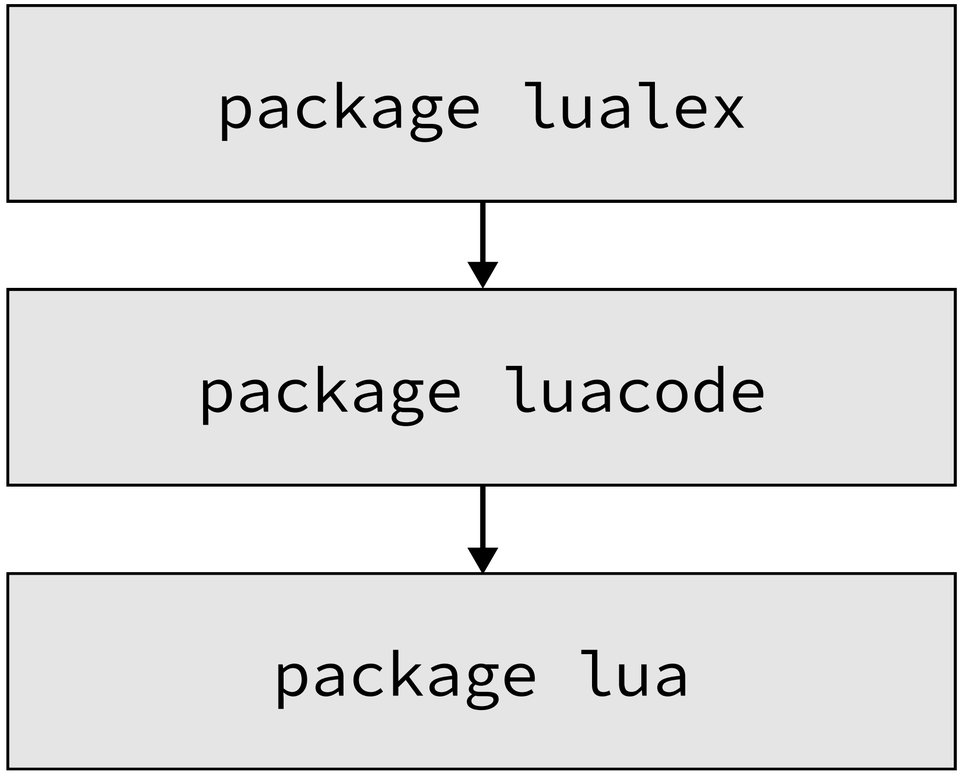
Roxy Light, August 2025

The Requirement

- Configuration language for build tool, zb
- Custom: strings need dependency information

  (like Terraform)
- General purpose and existing language.
- (Starlark not quite enough.)

Packages Rock!



1. Split tokens
2. Parse into bytecode
3. Execute bytecode

Parsing

1. Split tokens

2. Parse into bytecode

```
local x = 42
local y = x+3
```

```
“local”, “x”, “=”, “42”,
“local”, “y”, “=”, “x”, “+”, “3”
```

```
LOADI  0 42
ADDI   1 0 3
MMBINI 0 3 6 0
```

Types Rock!

```
type value interface {
  valueType() Type
}
```

nil	→	`nil`
boolean	→	`bool`
number	→	`float64` or `int64`
string	→	`struct { s string; deps []string }`
userdata	→	`any`
table	→	`[]struct{ key value; value value }`
Lua function	→	`*luacode.Prototype`
Go function	→	`func(ctx context.Context, l *State) (int, error)`

Execution... rocks?

```
for pc := 0; ; {
  instruction := currFunction.proto.Code[pc]
  switch opCode := instruction.OpCode(); opCode {
  case luacode.OpLoadI:
    // ...
    pc++
  case luacode.OpAddI:
    // ...
    pc++

  // ... more op codes ...
  }
}
```

Go Compiler rocks!

What Rocks

- Code organization
- Type system (with garbage collection!)
- Compiler

Testing!

Testing Rocks!

- Lots of tiny tests
- github.com/google/go-cmp for diff-ing parser output
- Benchmarks and pprof

Thanks!

Roxy Light

256lights.com