LabLua
programming language research

Pontifícia Universidade Católica
do Rio de Janeiro

# Typed Lua: An Optional Type System for Lua

Lua Workshop 2014

André Murbach Maidl
LabLua, PUC-Rio

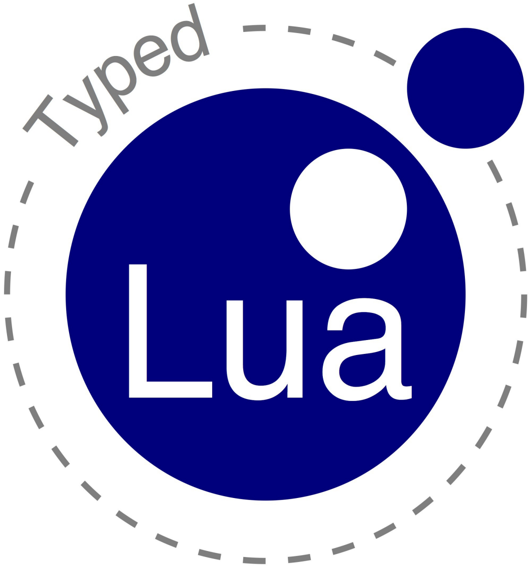
# Dynamic typing versus Static typing

- Dynamically typed languages
  - (+) Allow quick development
  - (+) Make code change easier
  - (–) May hide bugs
- Statically typed languages
  - (+) Detect many bugs
  - (+) Allow better program structure
  - (–) Force programmers to think about types

# Who never got a stack traceback?

```
lua: ./typedlua/tltype.lua:335:
attempt to index local 't' (a number value)
stack traceback:
    ./typedlua/tltype.lua:335: in function 'isUnionlist'
    ./typedlua/tltype.lua:751: in function
                               <./typedlua/tltype.lua:741>
    (...tail calls...)
    ./typedlua/tlchecker.lua:491: in function 'check_return_type'
    ./typedlua/tlchecker.lua:531: in function 'check_function'
    ./typedlua/tlchecker.lua:1238: in function 'check_exp'
    ./typedlua/tlchecker.lua:483: in function 'check_explist'
    ./typedlua/tlchecker.lua:890: in function 'check_assignment'
    ./typedlua/tlchecker.lua:1263: in function 'check_stm'
    ./typedlua/tlchecker.lua:1331: in function 'typecheck'
    ./tlc:119: in main chunk
    [C]: in ?
```

# Why not combine both?



# Typed Lua

- An optional type system for Lua
  - Allows static and dynamic typing in the same code
  - Provides optional type annotations
  - Uses local type inference
  - Does not influence run-time semantics
  - Unannotated Typed Lua code is Lua code
  - Aims to be sound
  - Rich enough to preserve some Lua idioms

# Typing Lua is challenging

- Lua is a small scripting language
  - First-class functions, extensive use of associative arrays, dynamic overloading, etc
- The primary use of Lua is as an embedded PL
  - DIY module systems
  - DIY object models

# We needed guidance on the design

- We surveyed 2598 files from 262 different projects that are on the LuaRocks repository
  - Table initialization, indexing, and iteration
  - Function declarations
  - Dynamic overloading
  - Object-Oriented programming
  - Module definitions

# A simple example

```
local function abs (n:number):number
  if n < 0 then
    return -n
  else
    return n
  end
end

local function dist (x, y)
  return abs(x – y)
end
```

# A simple example

```
local function abs (n:number):number
  if n < 0 then
    return -n
  else
    return n
  end
end

local function dist (x:any, y:any):number
  return abs(x – y)
end
```

```
any – any : any ~ number
```

# Optional type annotations

*type* ← *primarytype* ['?']

*primarytype* ← *literaltype* | *basetype*

| **nil** | **value** | **any** | **self** | *Name*

| *primarytype* '|' *primarytype*

| *functiontype*

| *tabletype*

*literaltype* ← **false** | **true** | *Number* | *String*

*basetype* ← **boolean** | **number** | **string**

*functiontype* ← *tupletype* '->' *rettype*

*tupletype* ← '(' [*typelist*] ')'

*typelist* ← *type* { ',' *type* } ['*']

*rettype* ← *type* | *unionlist* ['?']

*unionlist* ← *tupletype* | *unionlist* '|' *unionlist*

*tabletype* ← '{' [*tabletypebody*] '}'

*tabletypebody* ← [*keytype* ':'] *type* | *recordtype*

*keytype* ← *basetype* | **any**

*recordtype* ← *recordfield* { ',' *recordfield* } [',' *type*]

*recordfield* ← [**const**] *literaltype* ':' *type*

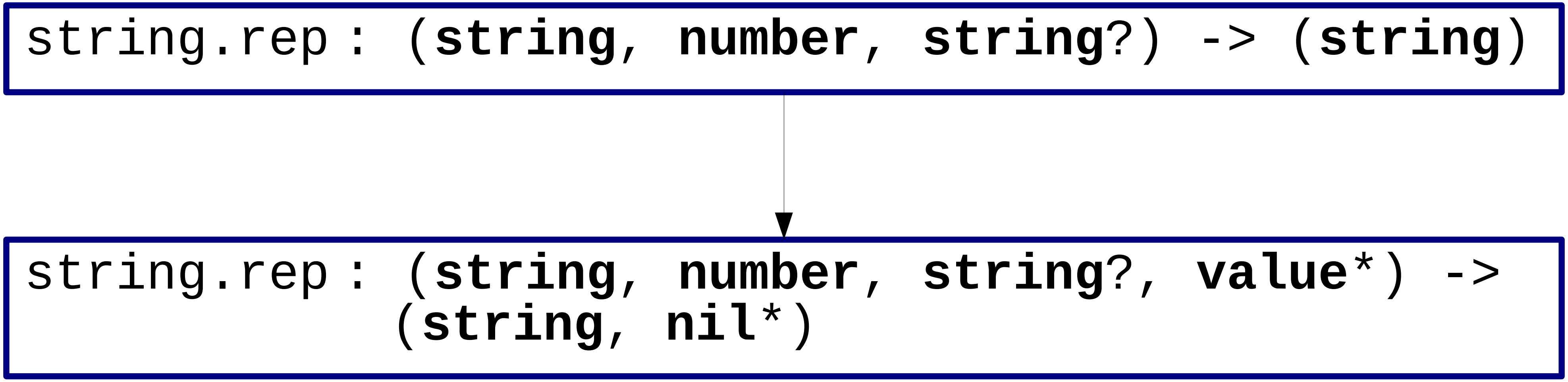
# Optional parameters

```
local function message (name:string,
                        greeting:string?)

  -- greeting : string | nil
  greeting = greeting or “Hello”

  -- greeting : string
  return greeting .. name
end
```

```
t | nil or t : t
```

# Union types on input parameters

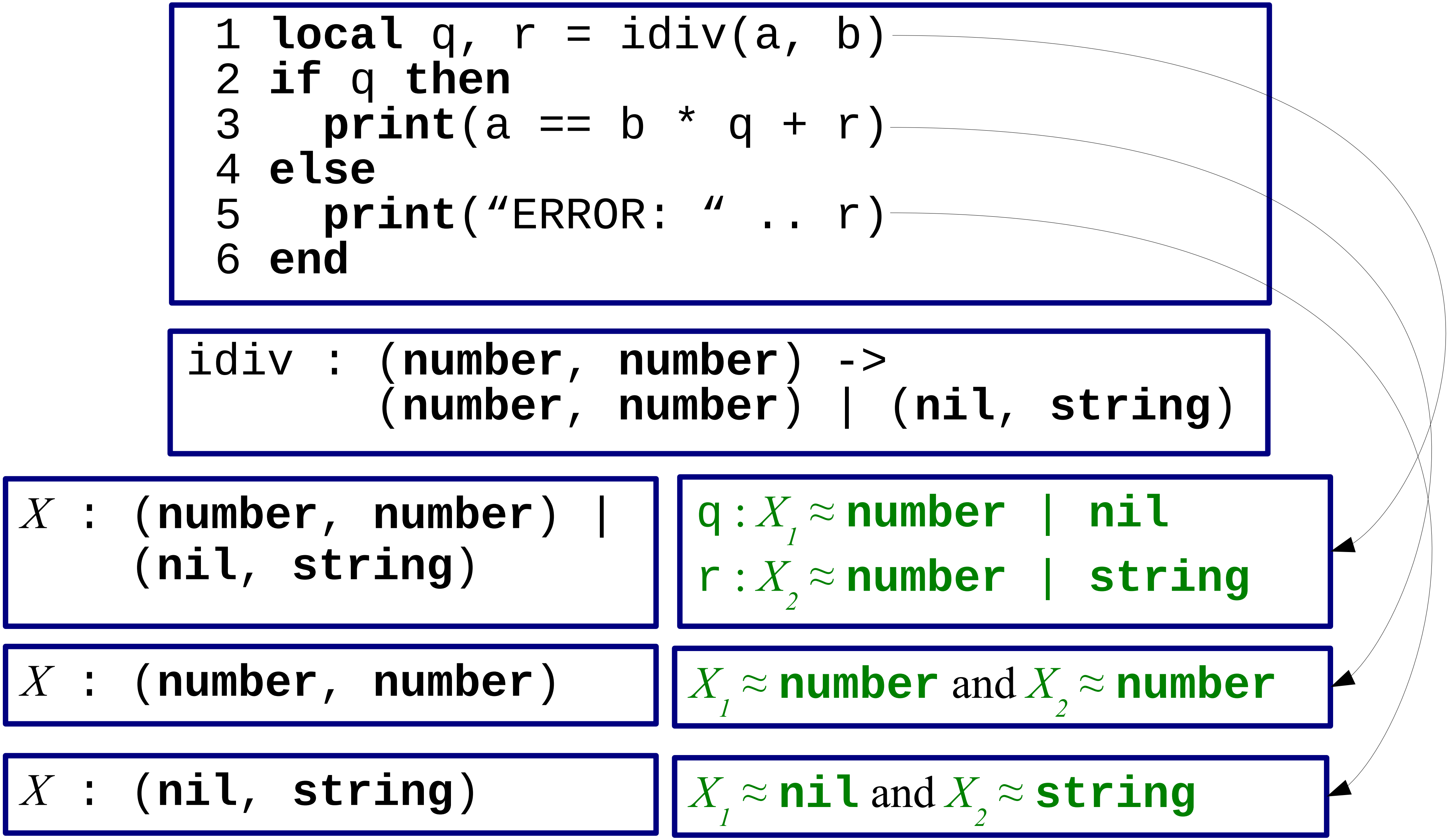
```
local function overload (s1:string,
                         s2:string|number)
  if type(s2) == “string” then
    return s1 .. s2
  else
    return string.rep(s1, s2)
  end
end
```



```
string.rep : (string, number, string?) -> (string)
```

```
string.rep : (string, number, string?, value*) ->
            (string, nil*)
```

# Union types on the return type



```
local q, r = idiv(a, b)
if q then
  print(a == b * q + r)
else
  print(“ERROR: “ .. r)
end
```

```
idiv : (number, number) ->
       (number, number) | (nil, string)
```

$X$ : `(number, number) | (nil, string)`

`q` : $X_1 \approx$ `number | nil`

`r` : $X_2 \approx$ `number | string`

$X$ : `(number, number)`

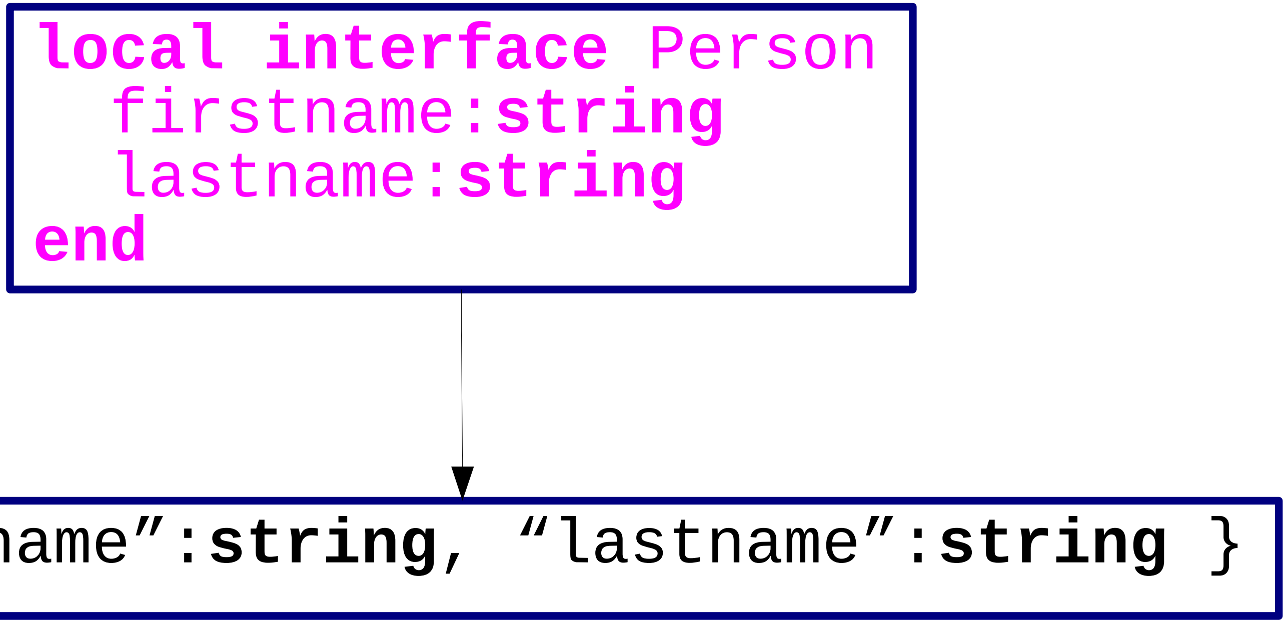
$X_1 \approx$ `number` and $X_2 \approx$ `number`

$X$ : `(nil, string)`

$X_1 \approx$ `nil` and $X_2 \approx$ `string`

# Table types as records

```
local person:{“firstname”:string,
              “lastname”:string } =
  { firstname = “Lou”,
    lastname = “Reed” }
```



```
local interface Person
  firstname:string
  lastname:string
end
```

```
{ “firstname”:string, “lastname”:string }
```

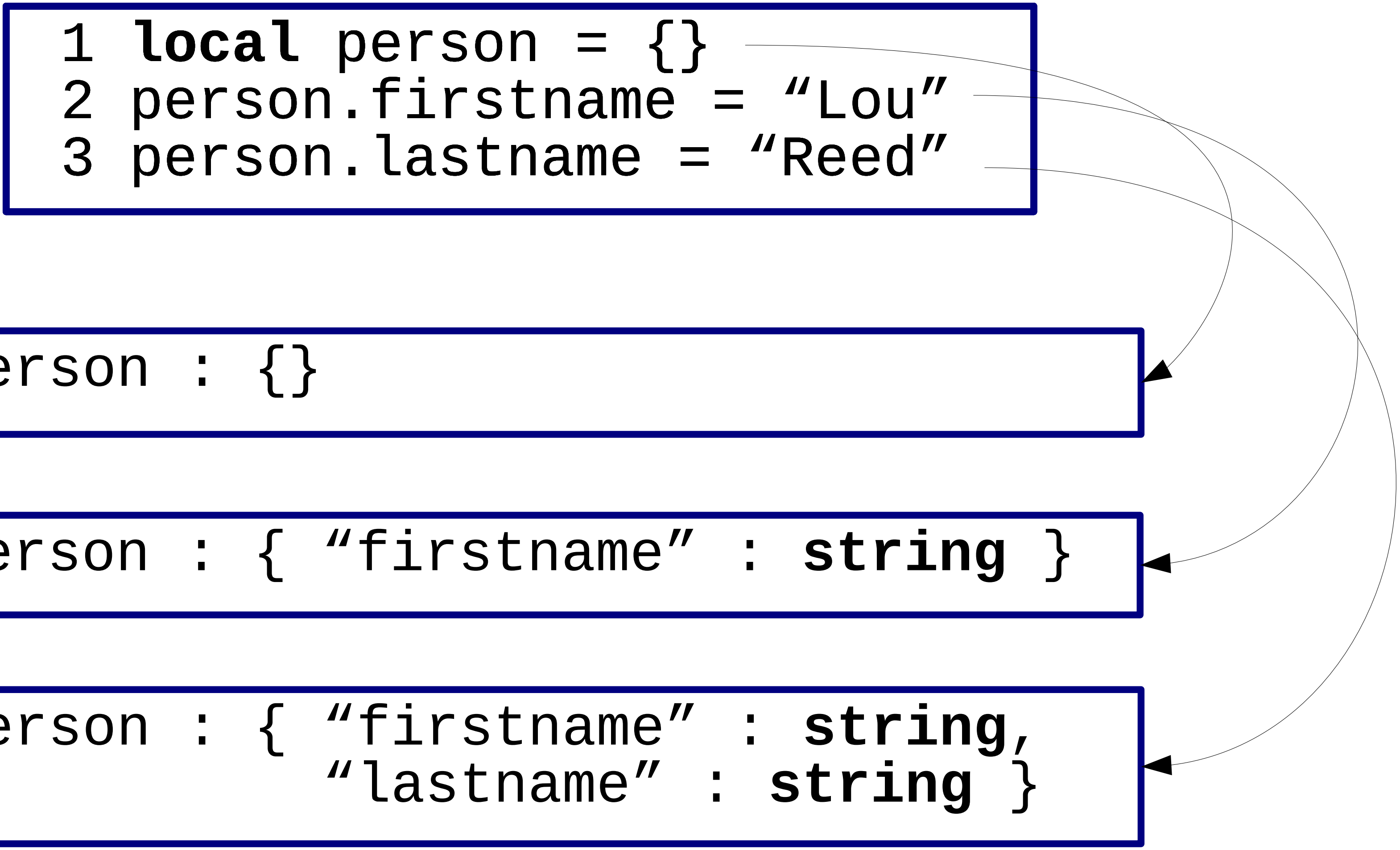
# Table types and interfaces

```
local interface Person
  firstname:string
  lastname:string
end

local function byebye (person:Person)
  return "Goodbye " .. person.firstname ..
         " " .. person.lastname
end

local user1 = { firstname = "Lou" }

local user2 = { lastname = "Reed",
                firstname = "Lou" }

print(byebye(user1))  -- compile-time error
print(byebye(user2))  -- Goodbye, Lou Reed
```

# Recursive types

```
local interface Element
  info:number
  next:Element?
end

local function insert (e:Element?,
                       v:number):Element
  return { info = v, next = e }
end
```

# Refinement of table types



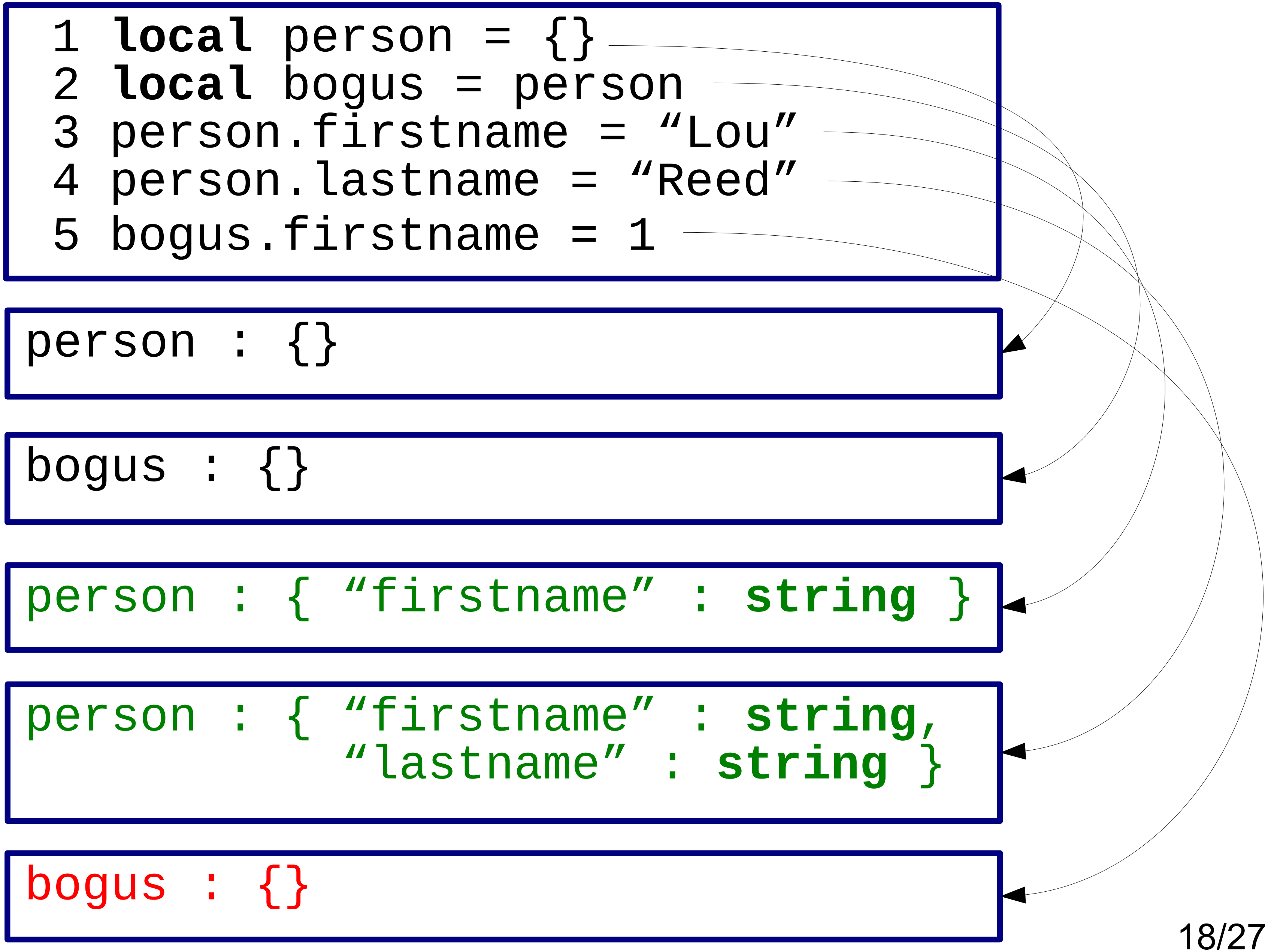
```
local person = {}
person.firstname = “Lou”
person.lastname = “Reed”
```

```
person : {}
```

```
person : { “firstname” : string }
```

```
person : { “firstname” : string,
           “lastname” : string }
```

# Refinement of table types



```
local person = {}
local bogus = person
person.firstname = “Lou”
person.lastname = “Reed”
bogus.firstname = 1
```

```
person : {}
```

```
bogus : {}
```

```
person : { “firstname” : string }
```

```
person : { “firstname” : string,
           “lastname” : string }
```

```
bogus : {}
```

# Modules

```
local mymath = {}

local RADIANS_PER_DEGREE = 3.14 / 180.0

mymath.deg = function (x:number)
  return x / RADIANS_PER_DEGREE
end

mymath.rad = function (x:number)
  return x * RADIANS_PER_DEGREE
end

mymath.pow = function (x:number, y:number)
  return x ^ y
end

return mymath
```

# Modules

```
local m = require “mymath”
print(m.pow(2, 3))       -- 8
print(m.pow(2, “foo”)) -- compile-time error
```

```
m : {
  “deg” : (number) -> (number),
  “rad” : (number) -> (number),
  “pow” : (number, number) -> (number)
}
```

# Object-Oriented Programming

```
local Shape = { x = 0, y = 0 }

const function Shape:new (x:number,
                          y:number):self
  local s = setmetatable({},
                         { __index = self })
   s.x = x
   s.y = y
   return s
end

const function Shape:move (dx:number,
                           dy:number)
  self.x = self.x + dx
  self.y = self.y + dy
end

return Shape
```

# Object-Oriented Programming

```
interface Shape
  x, y:number
  const new:(self, number, number) -> (self)
  const move:(self, number, number) -> ()
end
```

```
local Shape = require “shape”
local shape1 = Shape:new(0, 5)
local shape2:Shape = Shape:new(10, 10)
```

# Description files

```
userdata md5_context
  __tostring : (self) -> (string)
  clone : (self) -> (self)
  digest : (self|string, value) -> (string)
  new : () -> (self)
  reset : (self) -> (self)
  update : (self, string*) -> (self)
  version : string
end

__tostring : (md5_context) -> (string)
clone : (md5_context) -> (md5_context)
digest : (md5_context|string, value) -> (string)
new : () -> (md5_context)
reset : (md5_context) -> (md5_context)
update : (md5_context, string*) -> (md5_context)
version : string
```

# Description files

```lua
local m = require "md5"
local x = m.new()
local y = x:clone()
local z = m.clone("foo") -- compile-time error
print(x:digest() == m.digest(x))
```

# Lua Standard Libraries

- Could use only static types on 71%
- Had to rely on the dynamic type on 29%
- Could type most of its members
  - string, math, bit32, io, and os
- Polymorphism may help to type its members
  - table
- Could not type most of its members
  - base, and coroutine

# Limitations and Future Work

- Lua 5.3
- ZeroBrane Studio
- Polymorphic functions and tables
- Operator overloading
- Coroutines
- Soundness proof

# Thank you!

- Questions?
- For more information:
  - https://github.com/andremm/typedlua
  - amaidl@inf.puc-rio.br