Compare Functional Programming: Go and Lua

Concepts of Programming Languages

By Dominik Ampletzer

Agenda

- Introduction into Lua
- Functional Programming: Go and Lua
- Exception Handling
- Lua Tabels
- Coroutines in Lua

Introduction into Lua

- Created 1993
- By Pontifical Catholic University of Rio de Janeiro in Brazil
- Since Version 5 under MIT-Licence
- Focus - Goals:
  - Portability
  - Size – Small Footprint
  - Scripting - dynamic
  - Simplicity
- Currently leading scripting language in video games

Functional Programming: Go and Lua

- In both languages
  - Functions are First Class Citizens
  - With lexical scoping
  - Multiple return values
  - Multi assignment
- They treat functions nearly the same:

```
function sequence ()
    local i = 0
    return function ()
        i ++
        return i
    end
end
```

```
func sequence() func() int  {
    i := 0
    return func() int {
        i++
        return i
    }
}
```

```
function foo(x)
    function p(y)
        print(y)
    end
    p(2*x)
end
```

```
func foo(x int) {
    b := func(y int) {
        fmt.Println(y)
    }
    b(2 * x)
 }
```

Exception Handling

Lua uses an functional approach for Exception Handling

```
local ok, err = pcall(function() <block/error> end)
if not ok then
    print(err) // error handling
end
```

Go have a build in concept for Exception Handling in it´s functions

```
ok, err := http.Get(url)
if err != nil {
    fmt.println(err) // error handling
}
```

Lua Tables

```
TableA = {
    b = "",
    New = function()
        tableNew = {}
        for k, v in pairs(TableA) do
           tableNew[k] = v
        end
        return tableNew
    end,
    foo = function(param)
        print(param.b)
    end,
}
tableB = TableA.New()
tableB.b = "HALLO WORLD"
tableB.foo(tableB)                  // "HALLO WORLD"
```

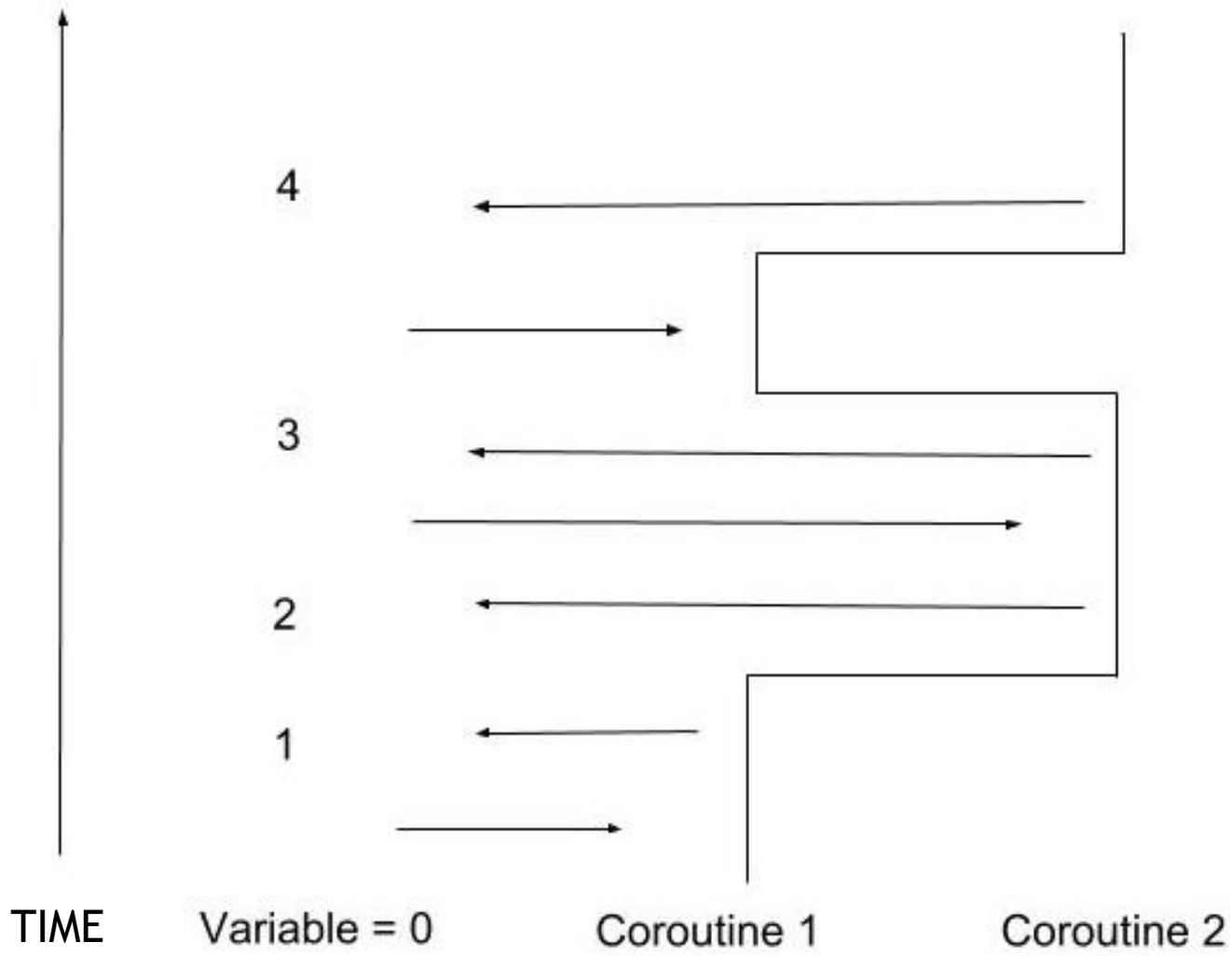
```
TableA = {
    b = "",
    mt = {},
    New = function()
        tableNew = {}
        setmetatable(tableNew, TableA.mt)   // setmetatable() came with Lua
        return tableNew
    end,
    foo = function(self)
        print(self.b)
    end,
}
 TableA.mt.__index = TableA   // allows to call functions from super.
 tableB = TableA.New()
 tableB.b = "HALLO WORLD"
 tableB:foo()  // "HALLO WORLD"
```

Coroutines

Coroutines are quite similar to threads (in the sense of multihreading), they have there own stack, own local variables, own insturction pointer, but share global variables and mostly anything else with other coroutines.

Main Difference: A Program can run only one coroutine at a time!

For serveral applications there is no need for parallel running threads. Without that requirement. Coroutines are a good alternative. They are much easier, htan threads. With coroutines there is no need to care about blocking or synchorinzation, because it is impossible that two coroutines have access to the same resource.


4
3
2
1
TIME
Variable = 0
Coroutine 1
Coroutine 2