Pontifícia Universidade Católica
do Rio de Janeiro

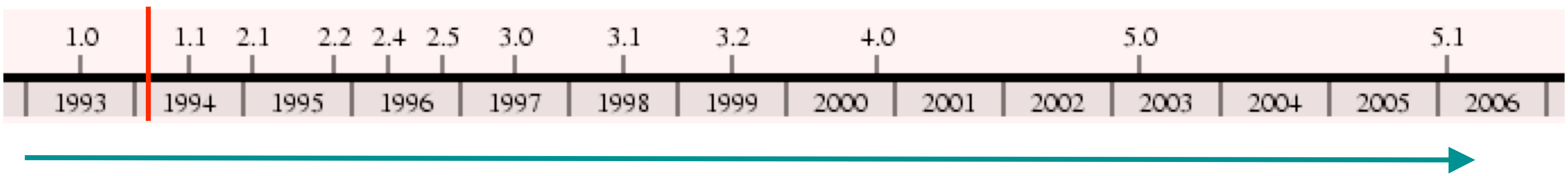
the evolution of Lua

ACM SIGPLAN
if... then
HOPL-III
SAN DIEGO 2007

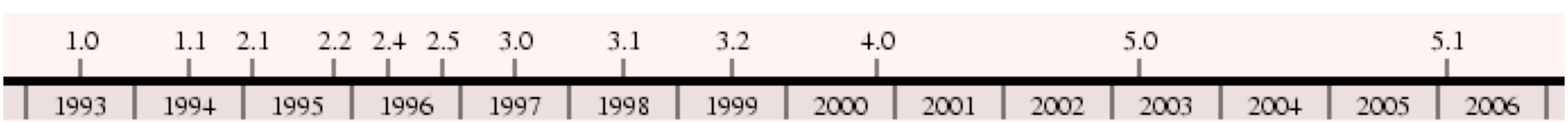
Roberto Ierusalimschy
Luiz Henrique de Figueiredo
Waldemar Celes

Outline

- brief introduction: what is Lua
- Lua's evolution

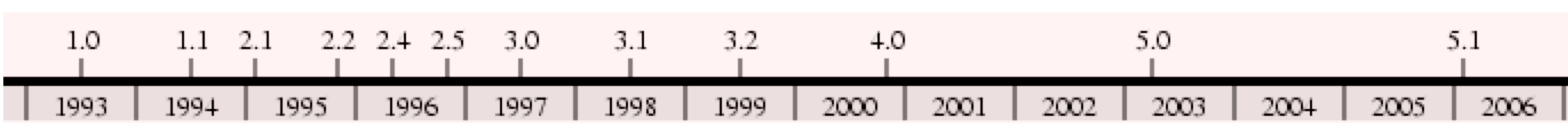


- principles we learned



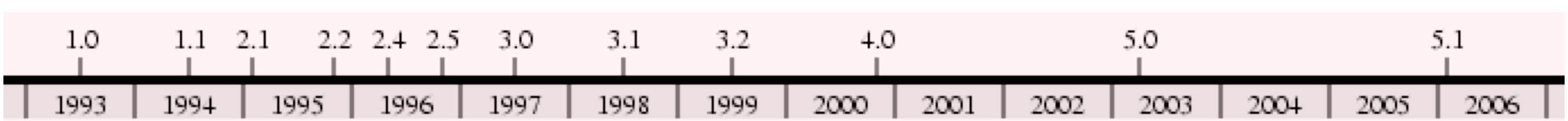
Lua is...

- a scripting language
  - interpreted (can run dynamic code)
  - dynamically typed
  - with (incremental) garbage collection
  - strong support for strings
  - also with coroutines, first-class functions with lexical scoping, proper tail calls, etc.



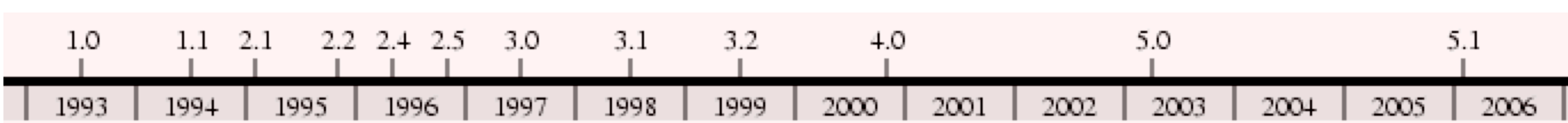
Lua is...

- a scripting language
- its main implementation
  - (at least) two other implementations
    - Lua-ML
    - Lua2IL (.Net)



Lua is...

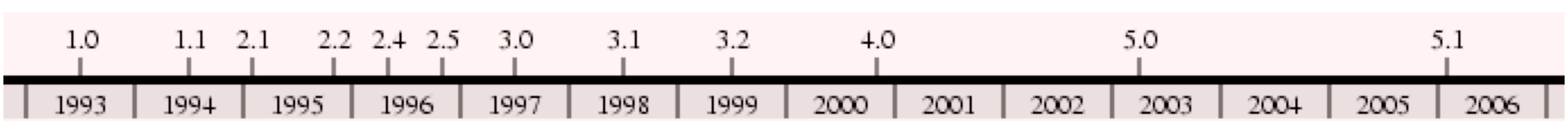
- a scripting language
- its main implementation
- an embeddable language
    - implemented as a library
    - offers a clear API for host applications
    - not only an implementation aspect!



Lua is...

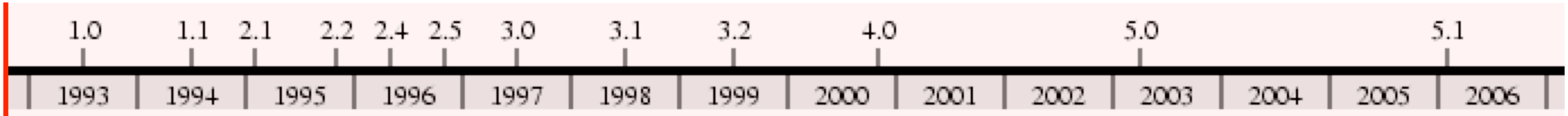
- a scripting language
- its main implementation
- an embeddable language

- embedded in a fair share of applications
  - Adobe Photoshop Lightroom, LuaTeX, nmap, wireshark, Olivetti printers, ...
  - niche in games



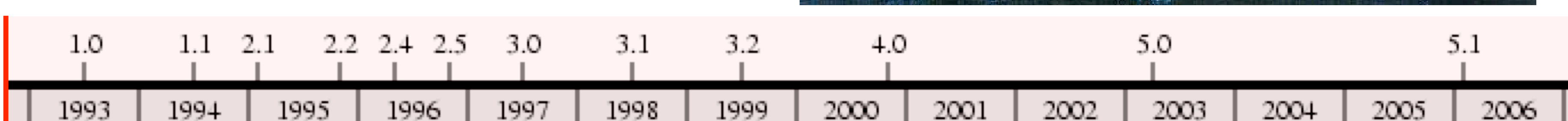
PC CD
The Sims 2
nightlife
EXPANSION PACK
PlayStation 2
A PSYCHIC ADVENTURE FROM THE MIND OF TIM SCHAFER
PSYCHONAUTS
XBOX 360
TOP SPIN 2
ESCAPE FROM MONKEY ISLAND
LUCASARTS ARCHIVE SERIES
SPELLFORCE
XBOX
NHL RIVALS 2004
SIMCITY 4
Baldur's Gate
FARCRY
UBISOFT
WORLD OF WARCRAFT
MDK 2
Out Think - Out Snipe - Out Shoot
BIOWARE
A Real Time Strategy Game from the Creators of Homeworld
IMPOSSIBLE CREATURES
relic
Microsoft
XBOX
amped
DINER DASH 2
Restaurant Rescue
Flo is back & better than ever!
GRIM FANDANGO
HOMEWORLD2
SIERRA
relic
XBOX
NFL FEVER 2004
XBOX 360
AMPED 3

The Beginning



1992: Tecgraf

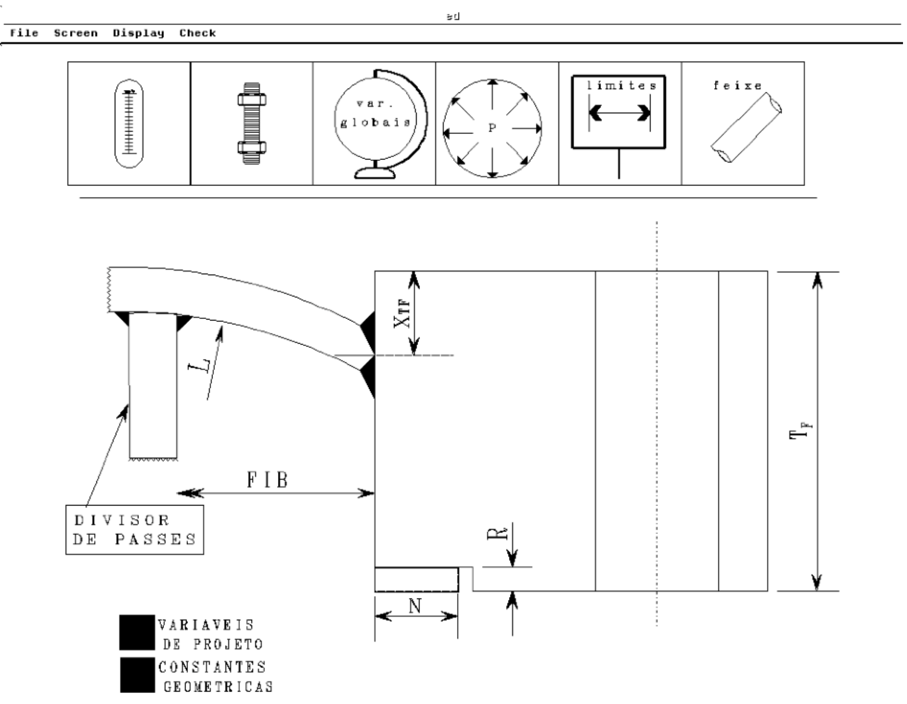
- partnership between PUC-Rio and Petrobras (the Brazilian Oil Company)



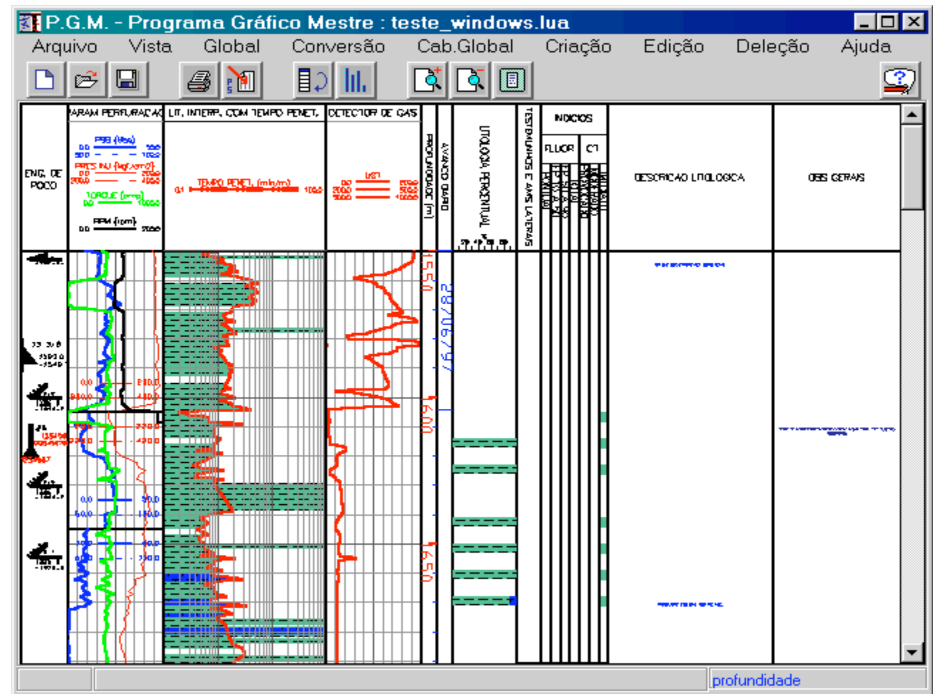
1992: Tecgraf

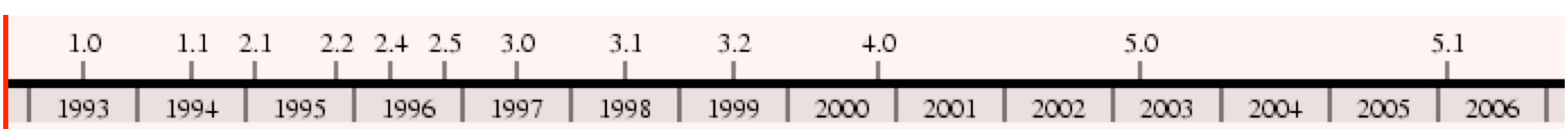
- two projects using "little languages"

DEL, for data entry



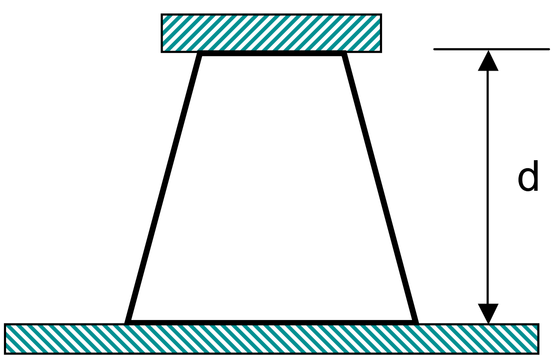
PGM, to visualize geologic profiles



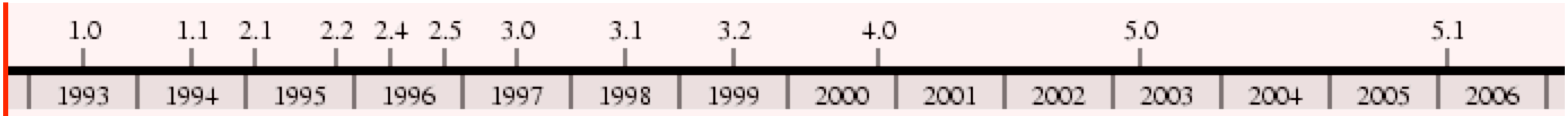


DEL
Data Entry Language

- form definition
  - parameter list
  - types and default values



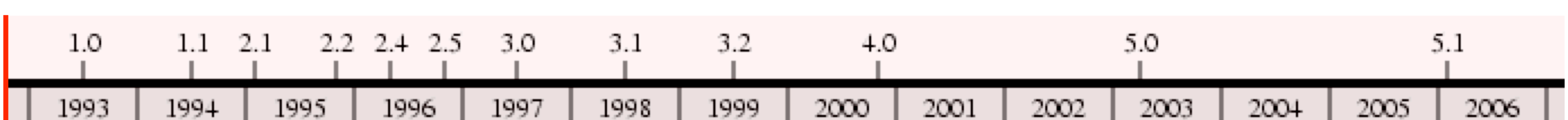

```
:e gasket "gasket properties"
mat     s              # material
d       f       0      # distance
y       f       0      # settlement stress
t       i       1      # facing type

:p gasket.d>30
gasket.d<3000
gasket.y>335.8
gasket.y<2576.8
```



SOL
Simple Object Language

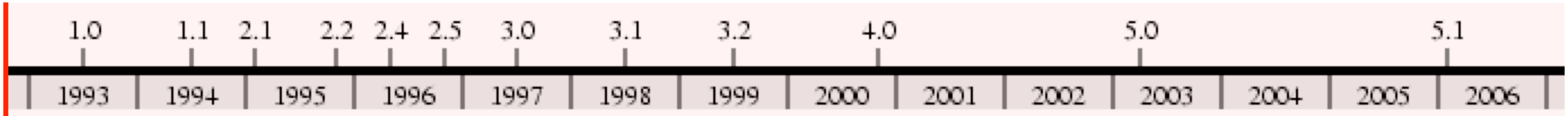
- data description language
  - not totally unlike XML
  - BibTeX-like syntax

```
type @track {x:number, y:number=23, z}

type @line {t:@track=@track{x=8}, z:number*}

-- create an object 't1', of type 'track'
t1 = @track {y=9, x=10, z="hi!"}

l = @line {t=@track{x=t1.y, y=t1.x}, z=[2,3,4]}
```



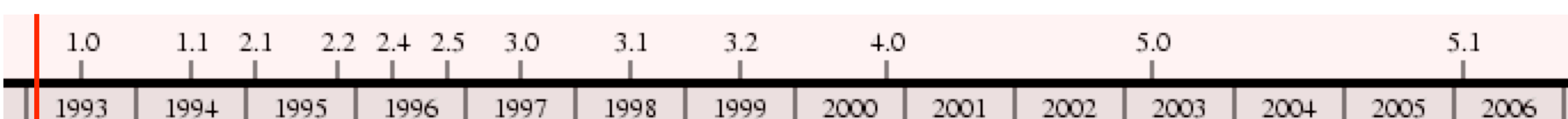
1992: Tecgraf

- two projects using "little languages"
  - DEL and PGM
- both shared several limitations
  - decision-making facilites
  - arithmetic expressions
  - abstraction mechanisms



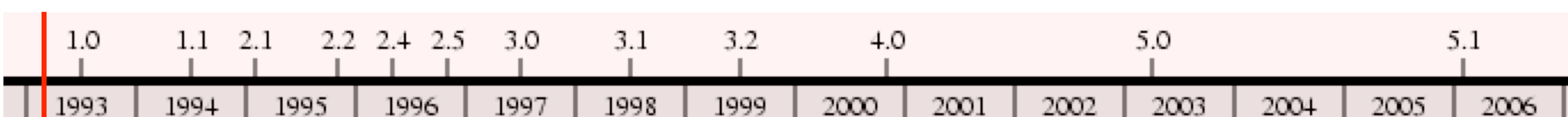
1993

- Roberto (PGM), Luiz (DEL) and Waldemar (PGM) got together to find a common solution to their common problems...

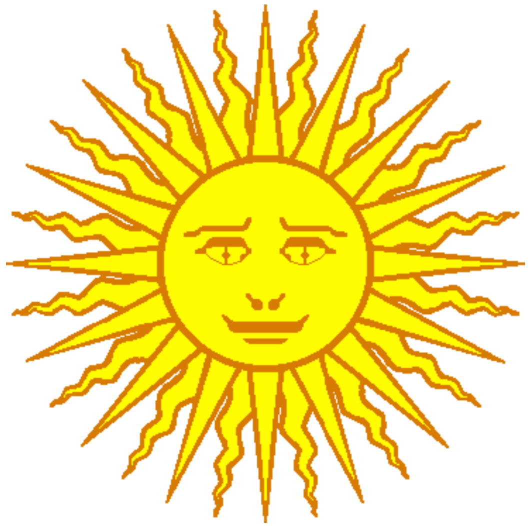


What we needed?

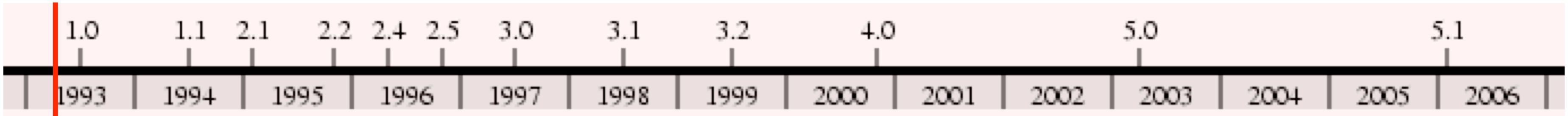
- a "generic configuration language"
- a "complete" language
- easily embeddable
- portable
  - Petrobras had a diverse array of machines
- as simple as possible
- non-intimidating syntax
  - for end users (engineers, geologists, etc.)



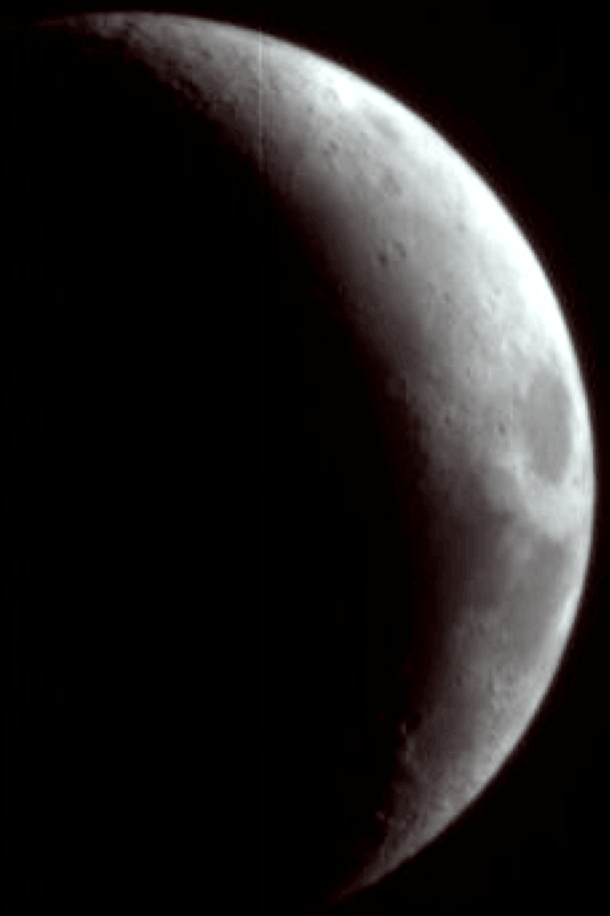
As we were giving up Sol,



a friend suggested a new name...



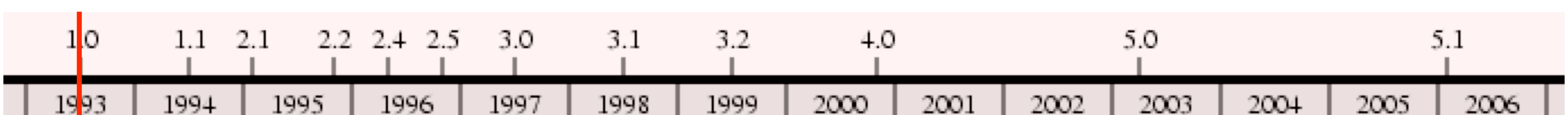
...and Lua was born



How was Lua 1.0?

- not that different from Sol...

```
t1 = @track{x = 10.3, y = 25.9,
            title = "depth"}
```

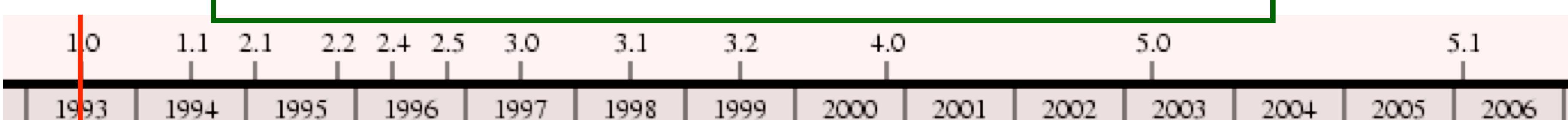


How was Lua 1.0?

- but quite different...

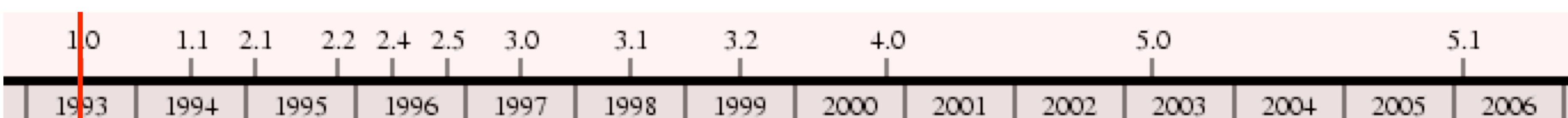
```
t1 = @track{x = 10.3, y = 25.9,
            title = "depth"}
```

```
function track (t)
  if not t.x then t.x = 0.0 end
  if type(t.x) ~= "number" then
    print("invalid 'x' value")
  end
  if type(t.y) ~= "number" then
    print("invalid 'y' value")
  end
end
```



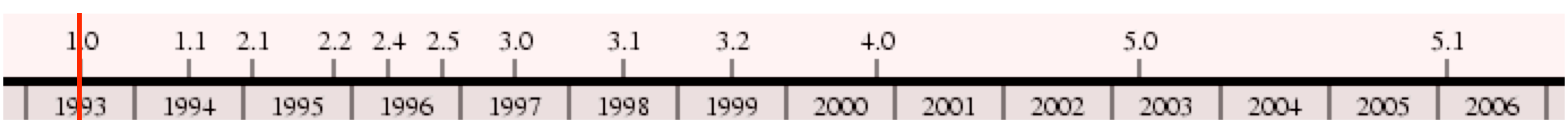
Lua 1.0

- implemented as a library
- called 1.0 a posteriori
- the simplest thing that could possibly work
- standard implementation
  - precompiler with yacc/lex
  - opcodes for a stack-based virtual machine
- less than 6000 lines of C code



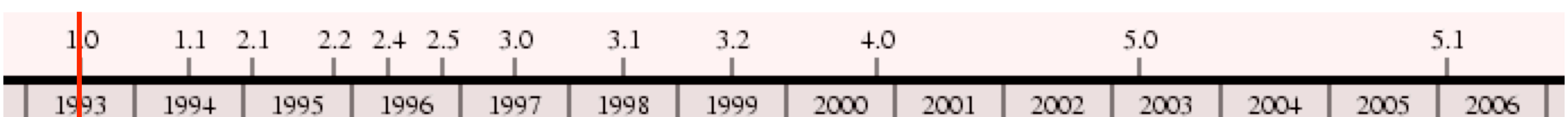
Tables in Lua 1.0

- associative arrays
- the only data structure
  - still is
  - records, lists, objects are just different constructors for tables
- sugar for records:
  - t.x for t["x"]
- primitive implementation
  - linked lists!

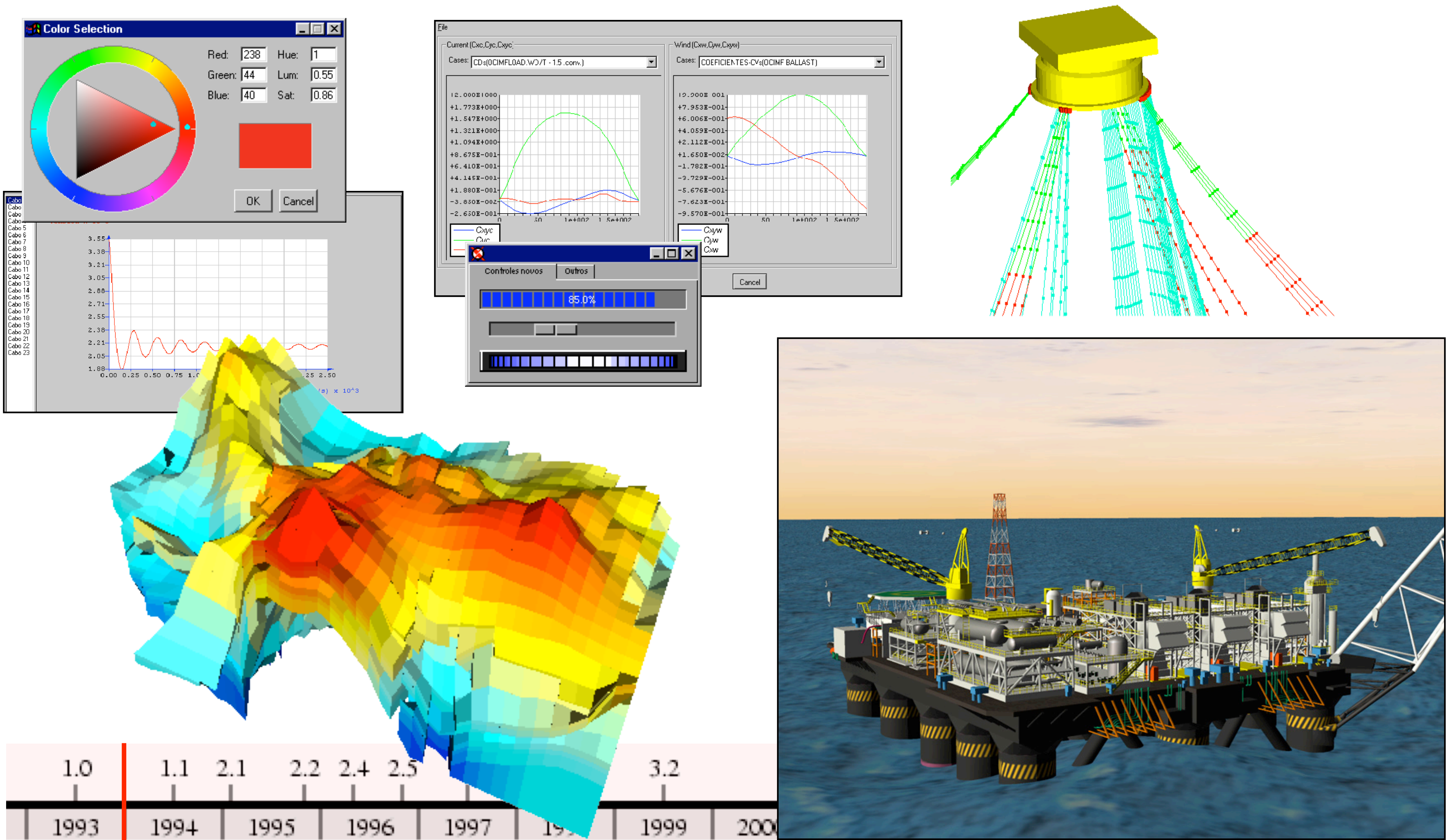


Lua 1.0

- expectations: to solve our problems with PGM and DEL
  - could be useful in other Tecgraf products
- fulfilled our expectations
  - both DEL and PGM used Lua successfully
  - PGM still in use today in oil platforms
- it was a big success in Tecgraf

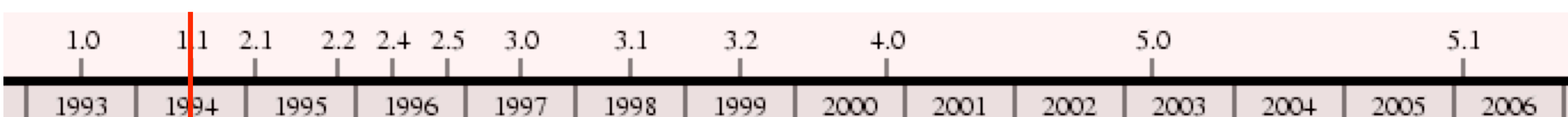


Soon, several projects at Tecgraf were using Lua



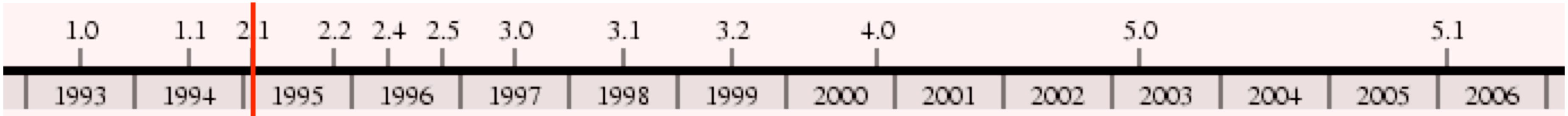
Lua 1.1

- new users brought new demands
  - several small improvements
  - mainly for performance
- reference manual
- well-defined and well-documented C API



Lua 2.1

- growing pressure for OO features
- several important changes
  - several incompatibilities!
- cleaner C API
  - no more direct references from C to Lua objects
- constructors
  - no more '@'
  - simpler syntax



Object Orientation

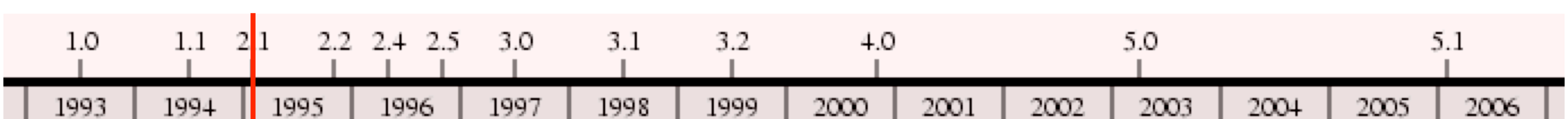
- tables + first-class functions ≈ objects
  - some (syntactical) sugar helped:

```
function a:foo (x)
  ...
end
```

```
a.foo = function (self,x)
  ...
end
```

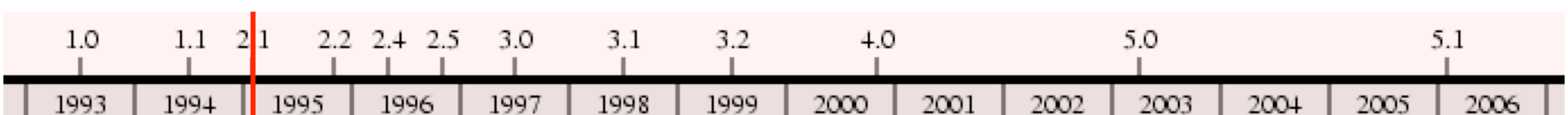
```
a:foo(x)
```

```
a.foo(a,x)
```



Fallbacks

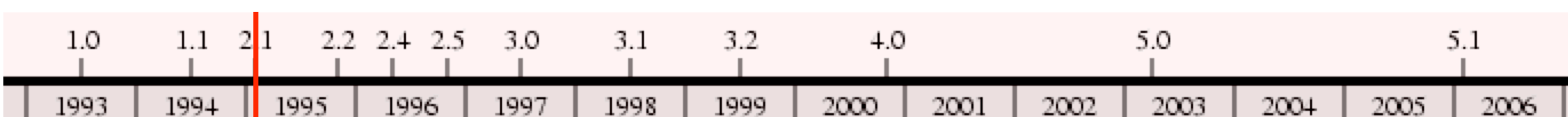
- similar to exception-handling with resumption
- delegation
  - allowed prototype-based OO
  - inspired by Self
- kind of minimum mechanism to get the label "*OO inside*"



Delegation at work

```
a = {x = 10}
b = {parent = a, y = 20}
print(b.y, b.x)    --> 20, 10
```

```
function a.foo (self)
  return self.x + self.y
end
print(b.foo(b))   --> 30
```

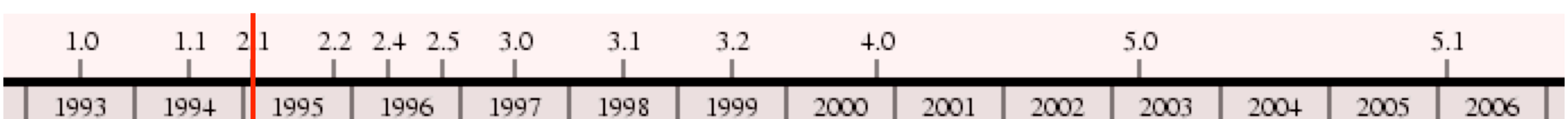


Delegation

- Lua provided only a fallback for absent indices

```
setfallback("index", inherit)
```

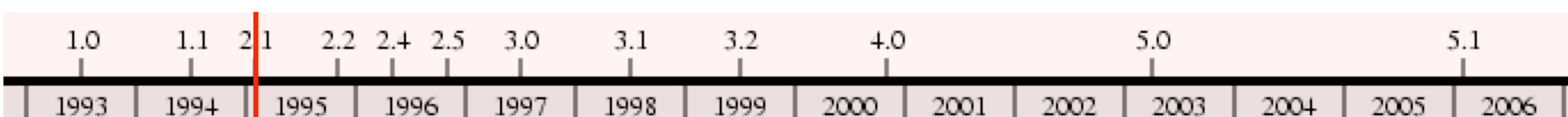
  - call function **`inherit`** when an index is absent from a table



Delegation

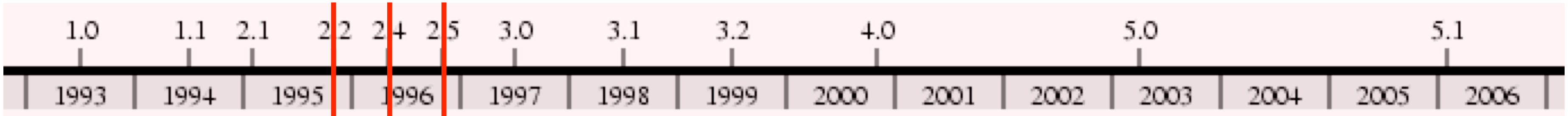
Most of the work done by the program...

```
function inherit (t, f)
  if f == "parent" then  -- avoid loops
    return nil
  end
  local p = t.parent
  if type(p) == "table" then
    return p[f]
  else
    return nil
  end
end
```



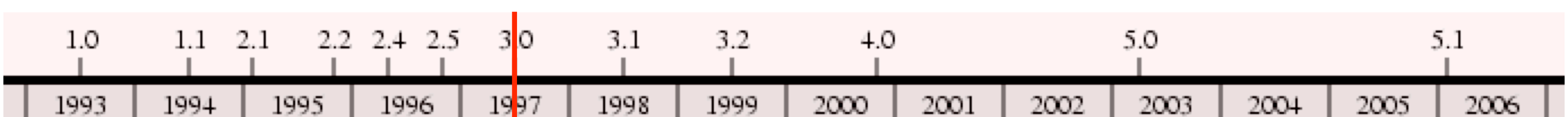
Lua 2.2 – 2.5

- external precompiler
  - faster load for large programs (metafiles)
- debug facilities
  - only basic primitives
- pattern matching



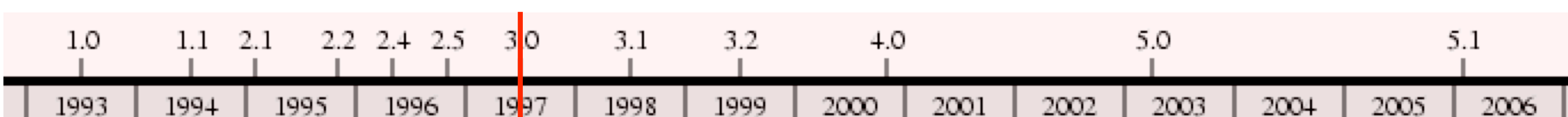
Lua 3.0

- problems with fallbacks
  - fallbacks were not built-in, but were global
  - different inheritance mechanisms from different libraries would clash
  - not a problem for small programs, without external code



Lua 3.0

- problems with fallbacks
- Lua 3.0 introduced *tag methods*
  - each object has a numerical *tag*
  - tag methods = fallbacks associated with tags
  - incompatible with previous mechanism
    - there was a "compatibility script"



Lua 3.1

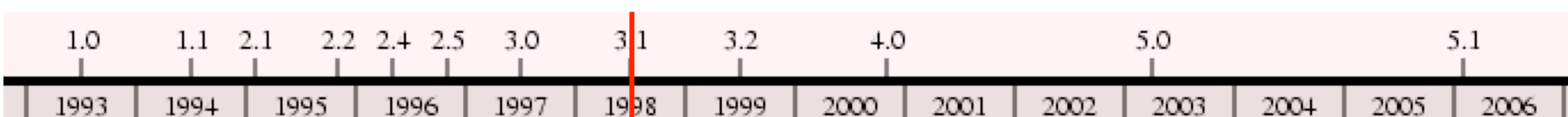
- functional features
  - syntax for anonymous, nested functions
  - since Lua 1.0, `function f ...` was sugar for `f = function ...`, except that the latter was not valid syntax!

iterators

```
foreach(t, function (k, v)
  print(k, v)
end)
```

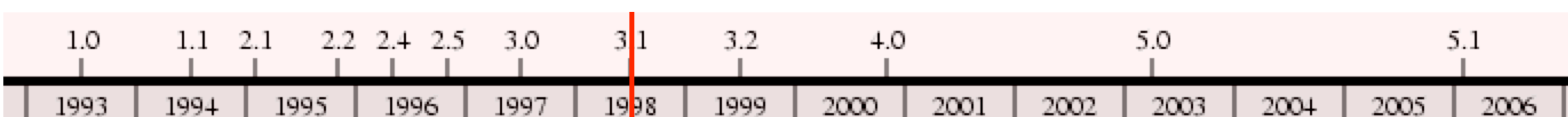
callbacks

```
button.action = function ... end
```



Lexical scoping

- functional features
- no simple and efficient way to implement lexical scoping
  - on-the-fly compilation with no intermediate representation + activation records in a stack
  - hindered earlier adoption of nested functions

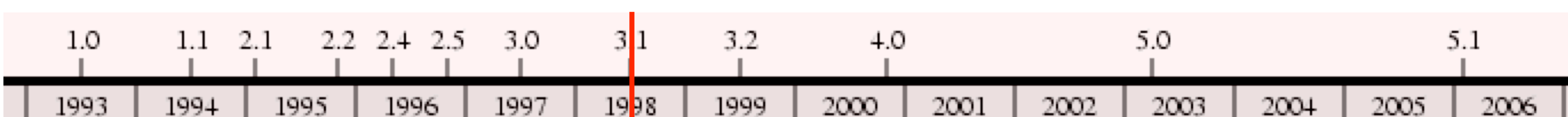


Upvalues

- "a form of proper lexical scoping"
- the frozen value of an external local variable inside a nested function
- trick somewhat similar to Java demand for `final` when building nested classes
- special syntax to avoid misunderstandings

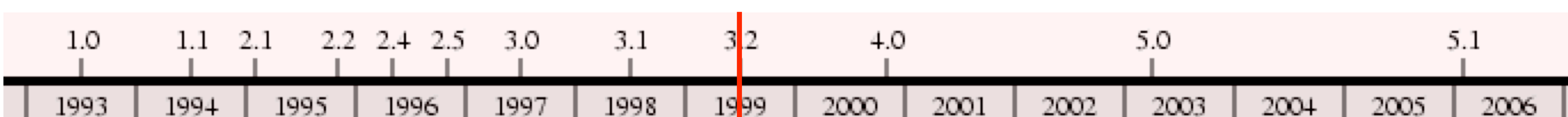
upvalue

```
function f (x)
  return function () return %x end
end
```



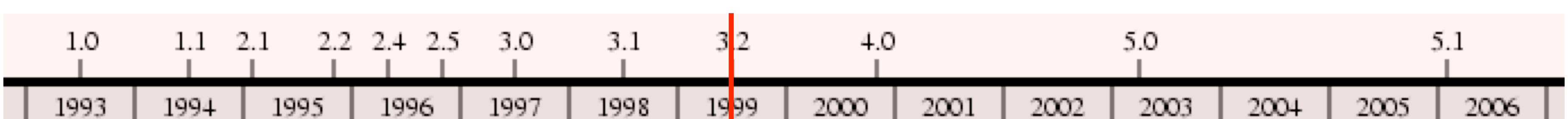
Lua 3.2

- multithreading?
  - for Web servers



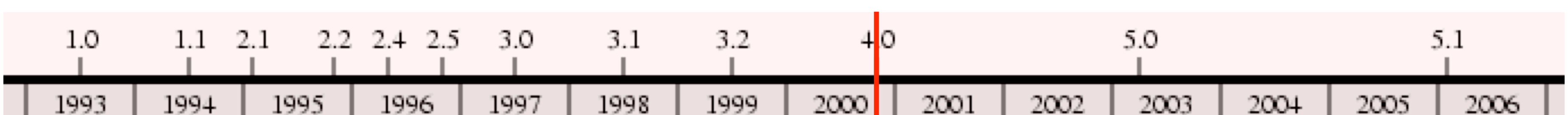
Lua 3.2

- multithreading?
- multiple "Lua processes"
  - multiple independent states in an application
  - no shared memory
- would require major change in the API
  - each function should get the state as an extra argument
  - instead, a single C global variable in the code points to the running state
  - extra API functions set the running state



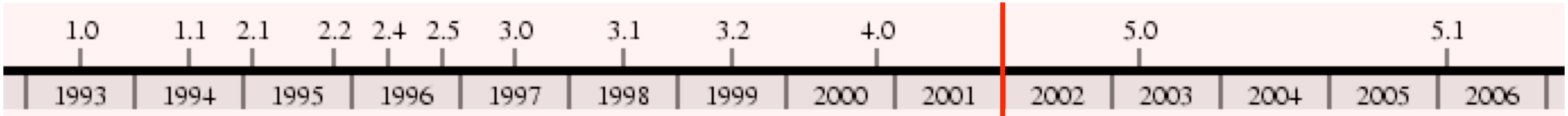
Lua 4.0

- major change in the API
  - all functions got a new parameter (the state)
  - no more C global variables in the code
  - libraries should not use C globals, too
  - concurrent C threads can each has its own state
- we took the opportunity and made several other improvements in the API
  - stack oriented



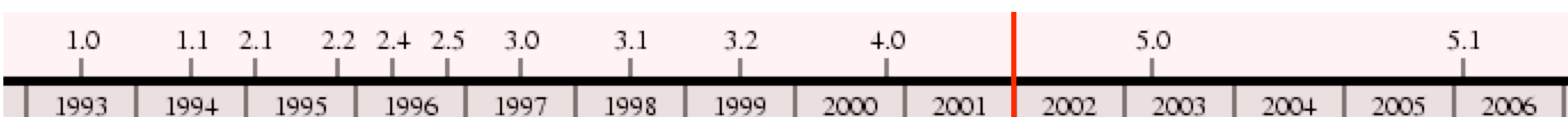
Plans for Lua 4.1

- multithreading?
  - multiple characters in games



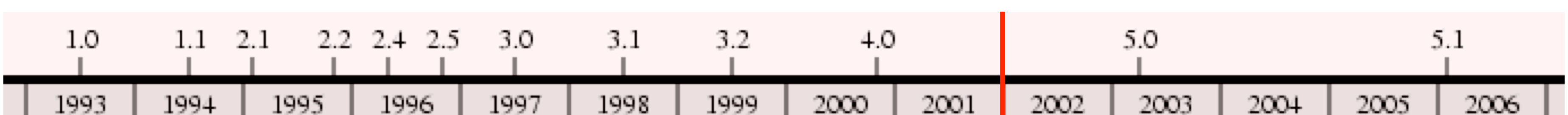
Plans for Lua 4.1

- multithreading?
- problems with multithreading
  - (preemption + shared memory)
  - not portable
  - no one can write correct programs when `a=a+1` is non deterministic
  - core mechanisms originally proposed for OS programming
  - almost impossible to debug



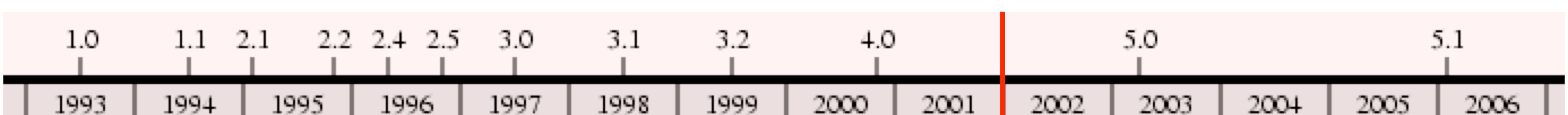
Plans for Lua 4.1

- multithreading?
- coroutines!
  - portable implementation
  - deterministic semantics
  - coroutines + scheduler =
    non-preemptive multithreading
  - could be used as a basis for multithreading for those that really wanted it



Plans for Lua 4.1

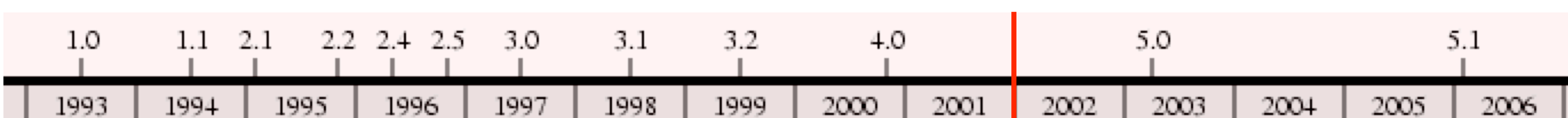
- new algorithm for upvales
  - allowed "true" lexical scoping!
- new algorithm for tables
  - store array part in an actual array
- new register-based virtual machine
- tags replaced by metatables
  - regular tables that store metamethods (old tag methods) for the object



Plans for Lua 4.1

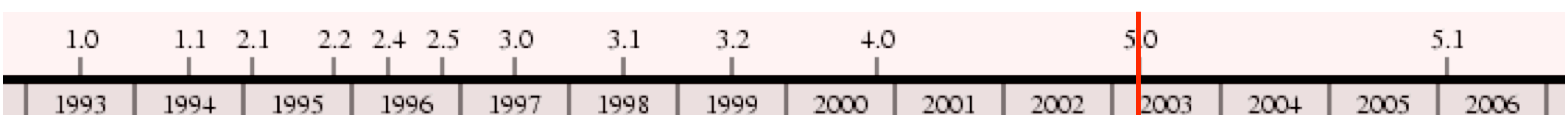
- new algorithm for upvales
  - allowed "true" lexical scoping!
- new algorithm for tables
  - store array part in an actual array
- new register-based virtual machine
- tags replaced by metatables
  - regular tables that store *metamethods* (old *tag methods*) for the object

Too much for a minor version...



Lua 5.0

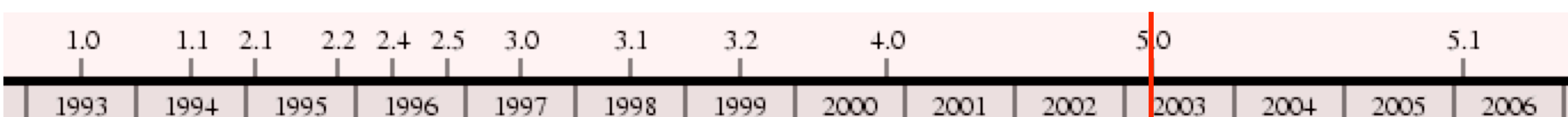
- coroutines
- lexical scoping
- metatables
- boolean type, weak tables, proper tail calls, ...
- module system
  - incompatibility



Modules

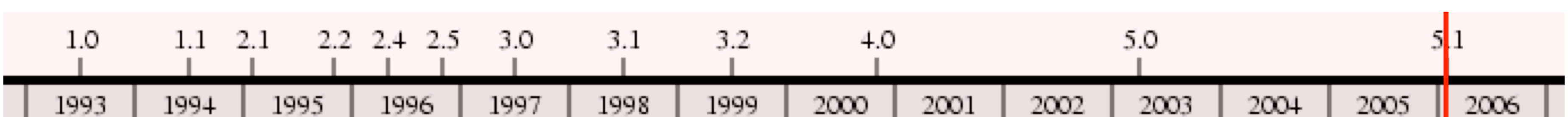
- tables as modules
  - `math.sin` (`sin` entry in table `math`)
- actually not a mechanism, but a policy
  - possible since Lua 1.0, but Lua itself did not use it
- several facilities for free

`local m = mod`	local renaming
`local foo = mod.foo`	unqualified import
`mod.submod.foo(...)`	submodules

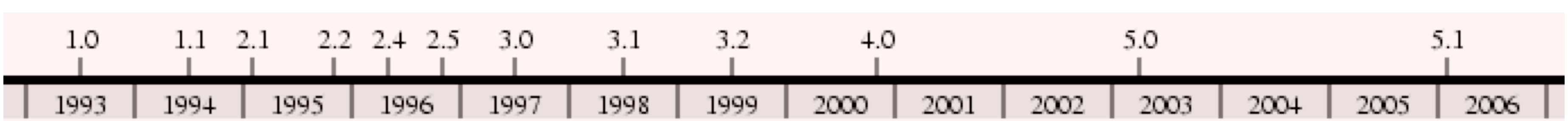


Lua 5.1

- incremental garbage collector
  - demand from games
- better support for modules
  - more policies
  - functions to help following "good practice"
- support for dynamic libraries
  - not portable!
  - the mother of all (non-portable) libraries
  - this support cannot be dynamically loaded!

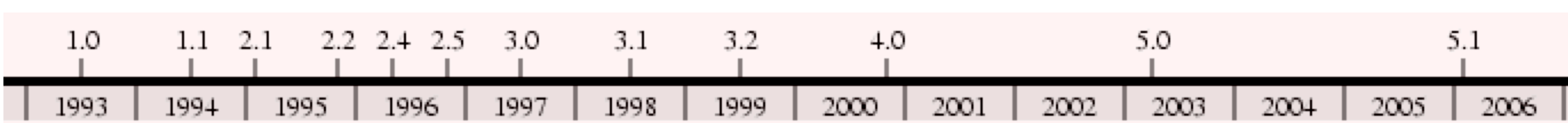


Principles we learned



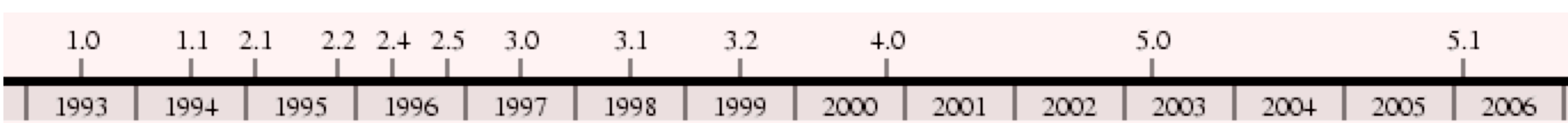
Principles we learned

- it is much easier to add a missing feature than to remove an excessive one
  - nevertheless, we have removed several features
- it is very hard to anticipate all implications of a new feature
  - clash with future features



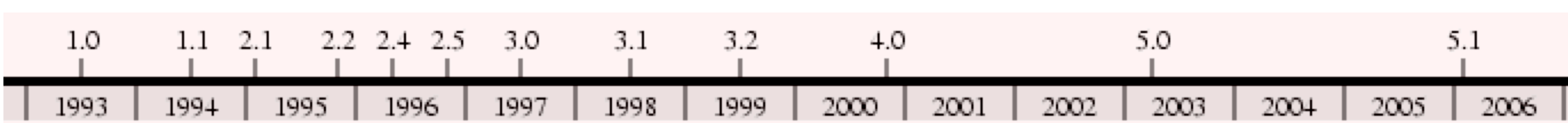
Principles we learned

- "Mechanisms instead of policies"
  - effective way to avoid tough decisions
  - type definitions in Lua 1.0
  - delegation in Lua 2.1
  - coroutines
  - did not work with modules...



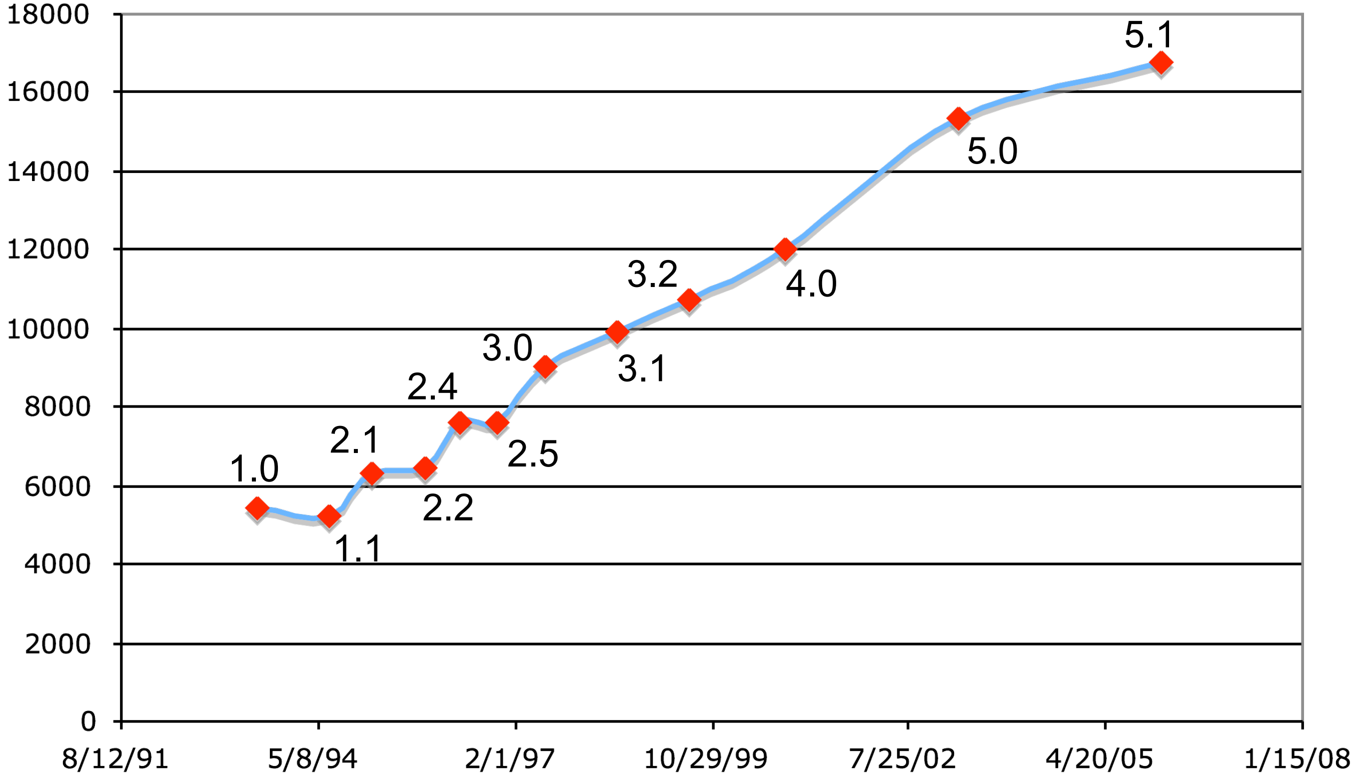
Principles we learned

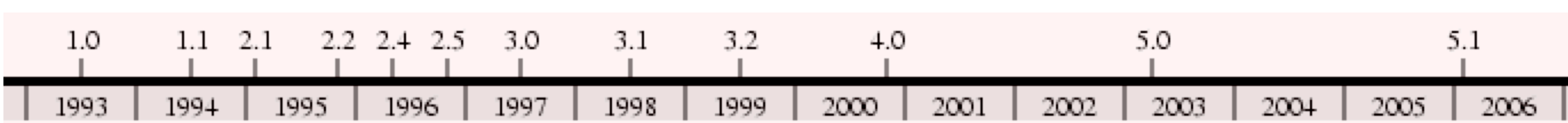
- emphasis on embedding
- portability
  - development for a single and very well documented platform: ANSI C
- keep it simple
  - ?



Growth in lines of code

- a proxy for complexity...





the evolution of Lua

ACM SIGPLAN
if . . . then
HOPL-III
SAN DIEGO 2007

Pontifícia Universidade Católica do Rio de Janeiro

Roberto Ierusalimschy
Luiz Henrique de Figueiredo
Waldemar Celes