# A Lua Virtual Machine for Resource-Constrained Embedded Systems

Alex de Magalhães Machado
Prof. Antônio Augusto Fröhlich
{alex,guto}@lisha.ufsc.br

October 2010

# Summary


- An approach for HLL@ES
- Lua Profile for ES
- Case: Lua@EPOS
- Evaluation: Lua@EPOS
- Final remarks

# Summary

- An approach for HLL@ES
- Lua Profile for ES
- Case: Lua@EPOS
- Evaluation: Lua@EPOS
- Final remarks

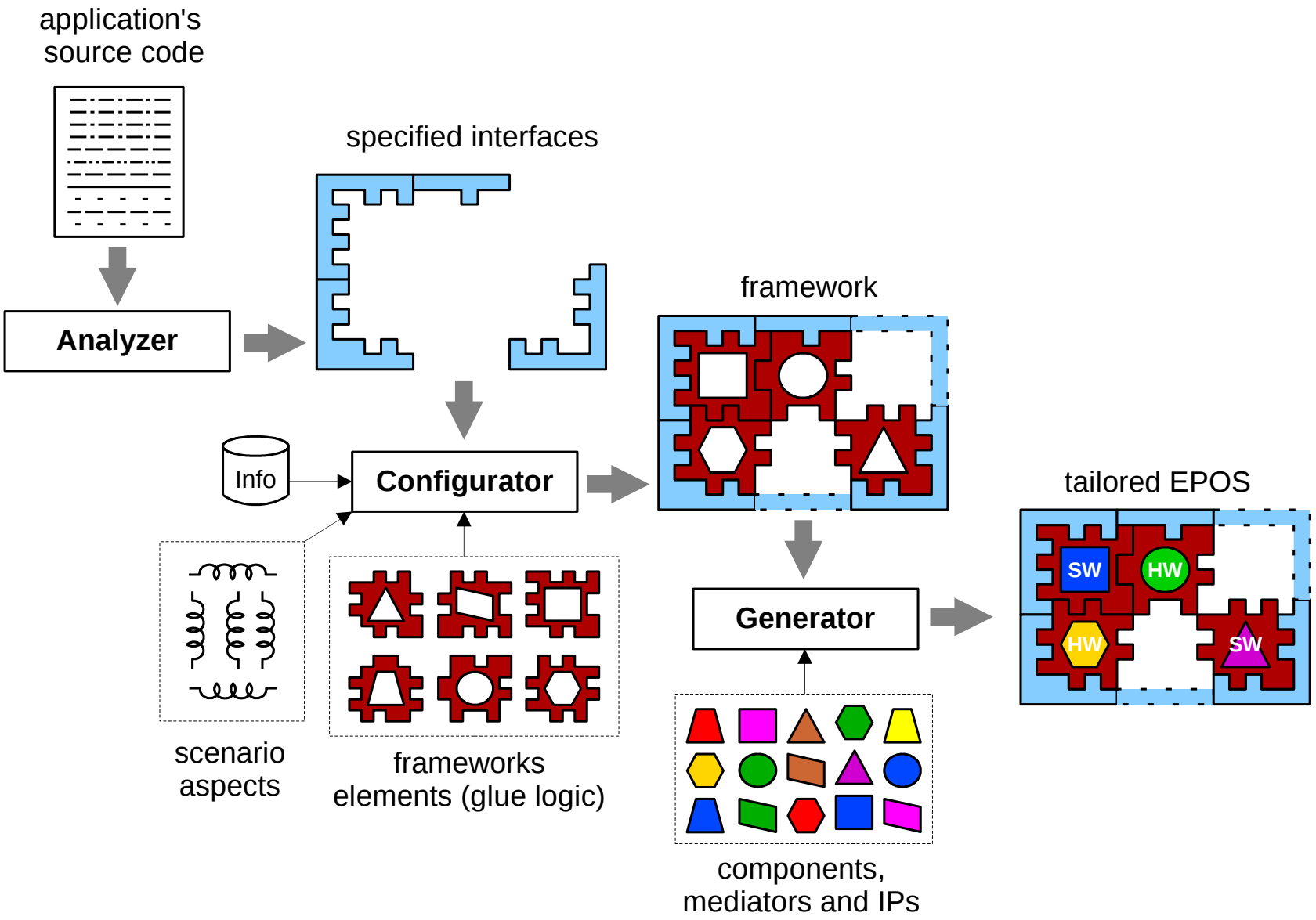
# High-level Languages for ES

- Improve application development
  - JAVA, LUA, Python
  - Plenty access to human resources
  - Cost and time reduction
- at the price of heavy run-time support systems
  - Operating system, device drivers, libraries, virtual machines and middlewares
- that seldom fit the requirements of real embedded systems
  - resources versus correctness

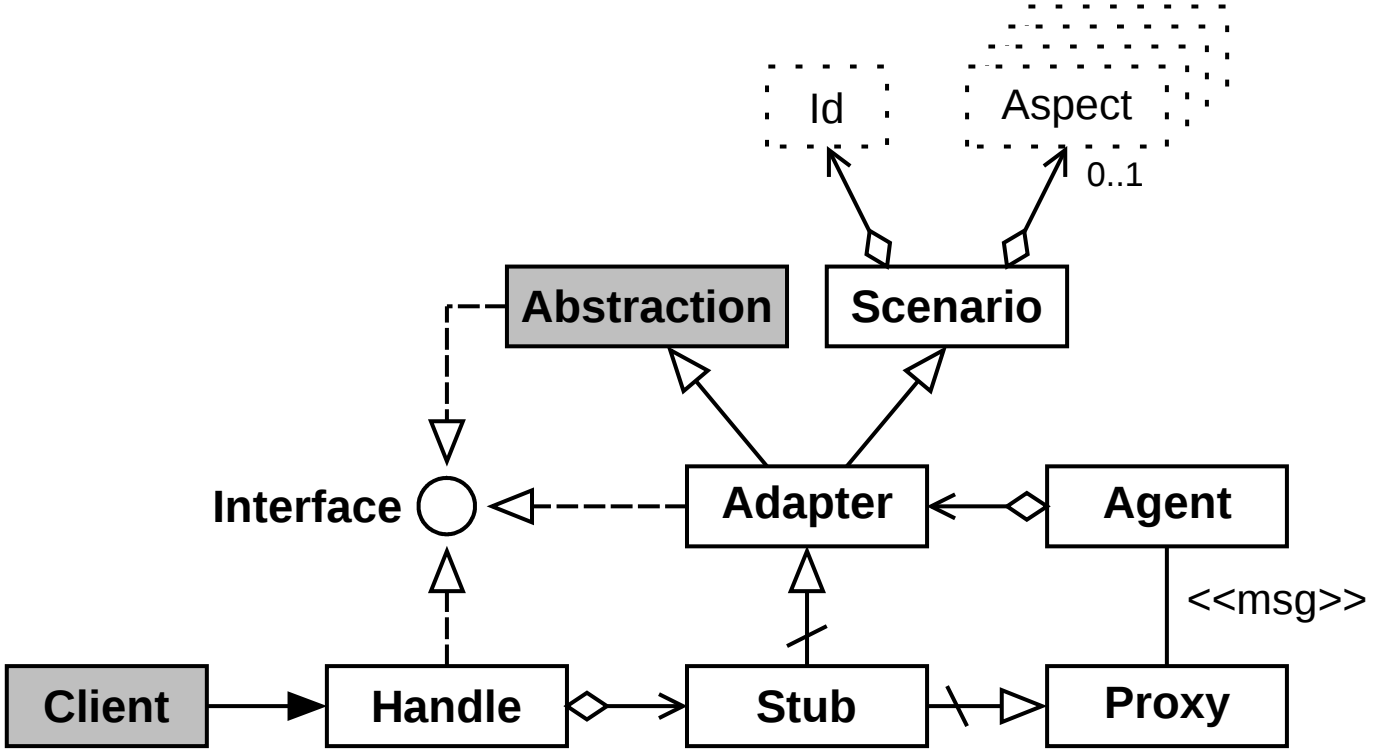
# Virtualization and ES


- The complexity of the run-time system in most high-level languages calls for virtualization
  - Virtual machines improve portability and help to preserve investments
  - They usually also impose additional load to typically resource-constrained embedded systems
- VMs usually reinvent the wheel and usually do that under a non-embedded system perspective
  - Embedded operating system usually feature adequate run-time support for most applications, turning the problem into interface adaptation along with proper language profiling

# Tailoring the Run-time System



# Adapting Interfaces



# Specifying ES Profiles for the HLL


- Embedded System

"Hardware and software which forms a component of some larger system and which is expected to function without human intervention."

[Foldoc]

- Run-time requirements
  - Dynamic module loading?
  - Remote method invocation?
  - Graphical interface?
  - Disk I/O?
  - Localization?
  - Environment variables?

# Summary


- An approach for HLL@ES
- Lua Profile for ES
- Case: Lua@EPOS
- Evaluation: Lua@EPOS
- Final remarks

# JAVA Lessons


- Remove all complex language features
  - Resulting in something that is not so JAVA as usually advertised
- Post-process bytecode
  - Impact portability
  - Manually handle resource allocation
- Exceeding resource demands
  - Does not fit in a 8-bit, 32 Kbytes MCU
- Language Profiles
  - One language, several deployment scenarios
  - Define features for specific application classes
  - Rational between functionality and overhead

# Lua

- Fastest in the realm of interpreted dynamic languages
- Simple, but not simplistic
  - Powerful syntax
  - Consistent and intuitive semantics
- Lightweight, small footprint
- Small, yet flexible C API
- Fits into
  - 128K ROM
  - 64K RAM

# LVM

- Middleware level virtual machine written in C
- Register-based *bytecode* interpretation
  - Closely resembles an actual hardware
- Lua programs are compiled at run-time
- The C API is divided into two parts:
  - The Lua core
  - The Lua auxiliary library

# Lua Profile for ES

| Basic library | Package and Coroutine libraries | String library | Mathematical library | I/O library | OS and Table library | Debug library |
|---|---|---|---|---|---|---|
| assert , error | ~~module~~ | byte | abs | io.close | clock | debug |
| collectgarbage | ~~require~~ | char | acos , asin | io.flush | date | getfenv |
| ~~dofile~~ , ~~loadfile~~ | ~~cpath~~ | dump | atan , atan2 | ~~io.input~~ | difftime | gethook |
| getfenv | ~~loaded~~ | find | ceil , floor | ~~io.lines~~ | ~~execute~~ | getinfo |
| getmetatable | ~~loaders~~ | format | cos , sin | ~~io.open~~ | exit | getlocal |
| ipairs , pairs | ~~loadlib~~ | gmatch | cosh , sinh | io.output | ~~getenv~~ | getmetatable |
| load , loadstring | ~~path~~ | gsub | deg | ~~io.popen~~ | ~~remove~~ | getregistry |
| next | ~~preload~~ | len | exp | ~~io.read~~ | ~~rename~~ | getupvalue |
| pcall , xpcall | ~~seeall~~ | lower , upper | fmod | ~~io.tmpfile~~ | ~~setlocale~~ | fenv |
| print | | match | frexp , ldexp | ~~io.type~~ | time | sethook |
| rawequal | create | rep | log , log10 | io.write | ~~tmpname~~ | setlocal |
| rawget, rawset | resume | reverse | max , min | ~~file:close~~ | | setmetatable |
| select | running | sub | modf | ~~file:flush~~ | concat | setupvalue |
| setfenv | status | | pow | ~~file:lines~~ | insert | traceback |
| setmetatable | wrap | | rad | ~~file:read~~ | maxn | |
| tonumber | yield | | random | ~~file:seek~~ | remove | |
| tostring | | | randomseed | ~~file:setvbuf~~ | sort | |
| type | | | sqrt | ~~file:write~~ | | |
| unpack | | | tan , tanh | | | |

# Summary

- An approach for HLL@ES
- Lua Profile for ES
- Case: Lua@EPOS
- Evaluation: Lua@EPOS
- Final remarks

# LVM@EPOS

- EPOS opens the LVM and sends the Lua program to be compiled
- The LVM creates the Lua State and compiles the Lua program
- EPOS then tells LVM to run the application
- They communicate through the C API
- LVM uses EPOS to allocate the memory it needs
- Currently, EPOS runs one Lua program at a time
- Limited I/O
  - Just the necessary facilities for this proof-of-concept

# LVM and EPOS Utilities


- Lua Libraries were implemented using EPOS utilities
- Memory and String handling
- Time and Date manipulation and formatting
- Mathematical functions
  - Some embedded systems do not support floating-point
  - We inform the virtual machine whether the hardware supports it or not

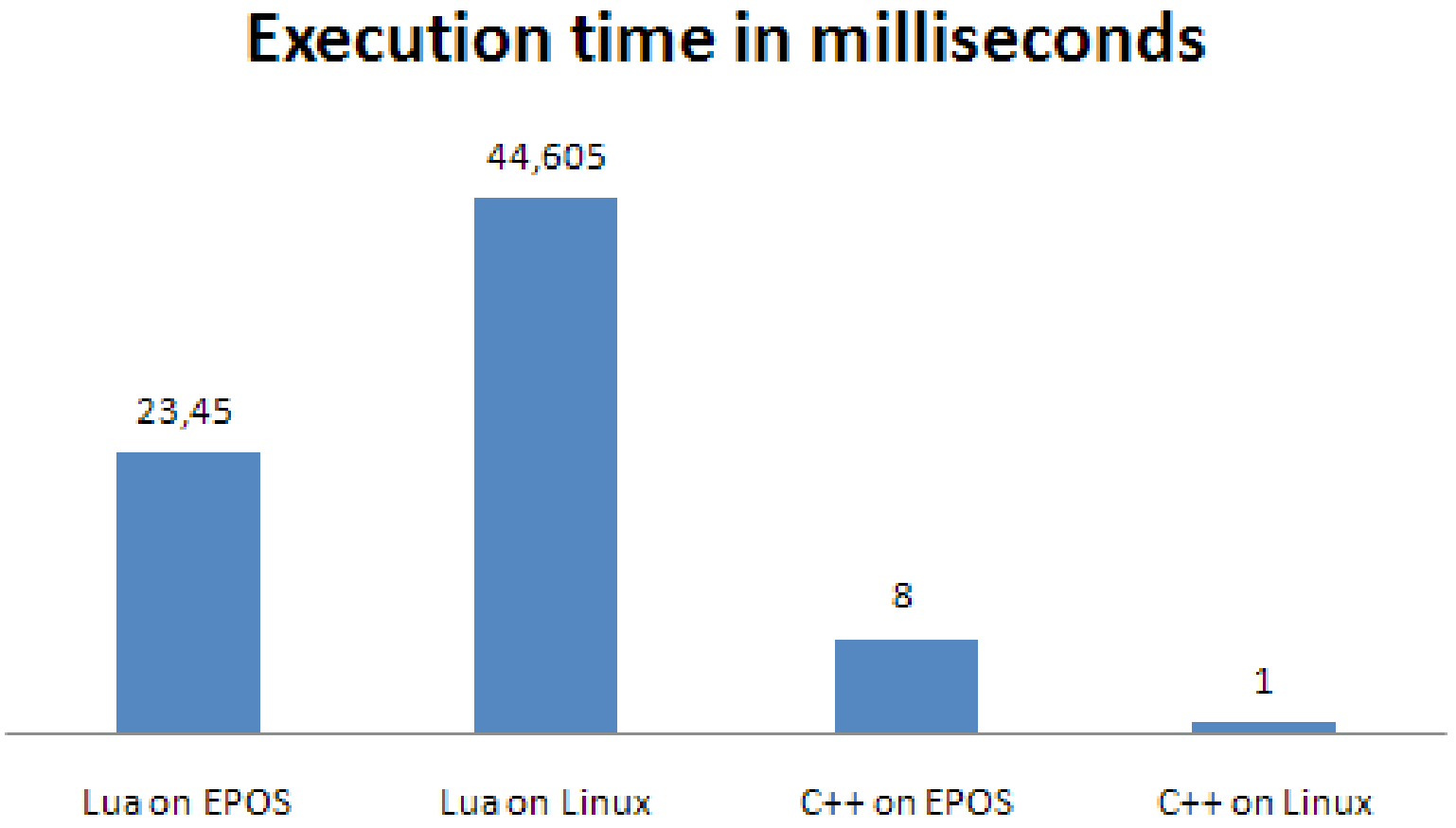
# Summary


- An approach for HLL@ES
- Lua Profile for ES
- Case: Lua@EPOS
- Evaluation: Lua@EPOS
- Final remarks

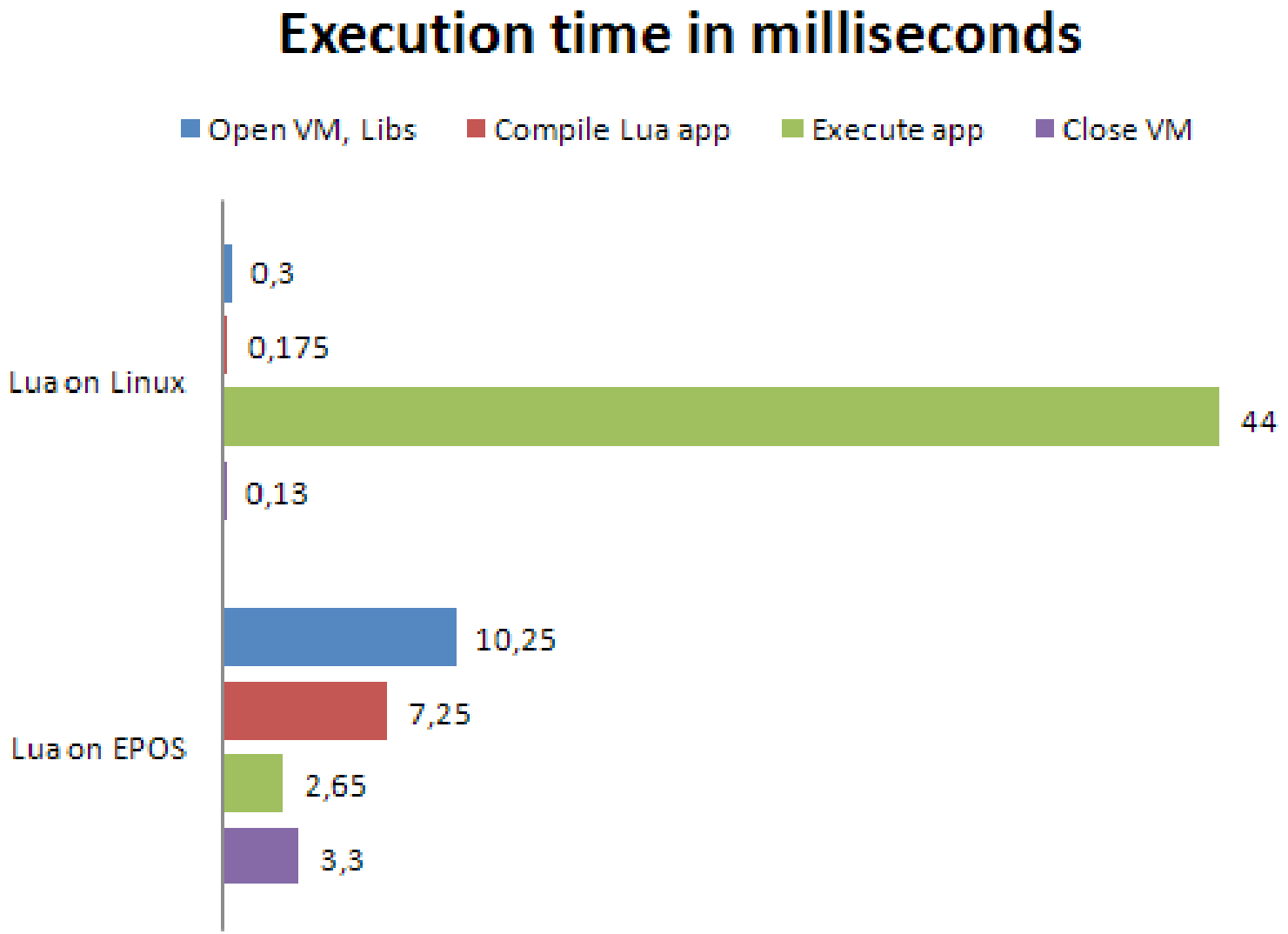
# Evaluation

- Overhead and performance
  - Additional resources to support LVM on EPOS
- Comparison with LVM on LINUX
- Simple test application that calculates the 24th Fibonacci number
  - Versions in Lua and C++ (EPOS native language)
  - Tested on OpenEPOS 1.0 and on Ubuntu 9.04 in the same x86-based hardware platform
  - Times obtained at I/O ports with an oscilloscope

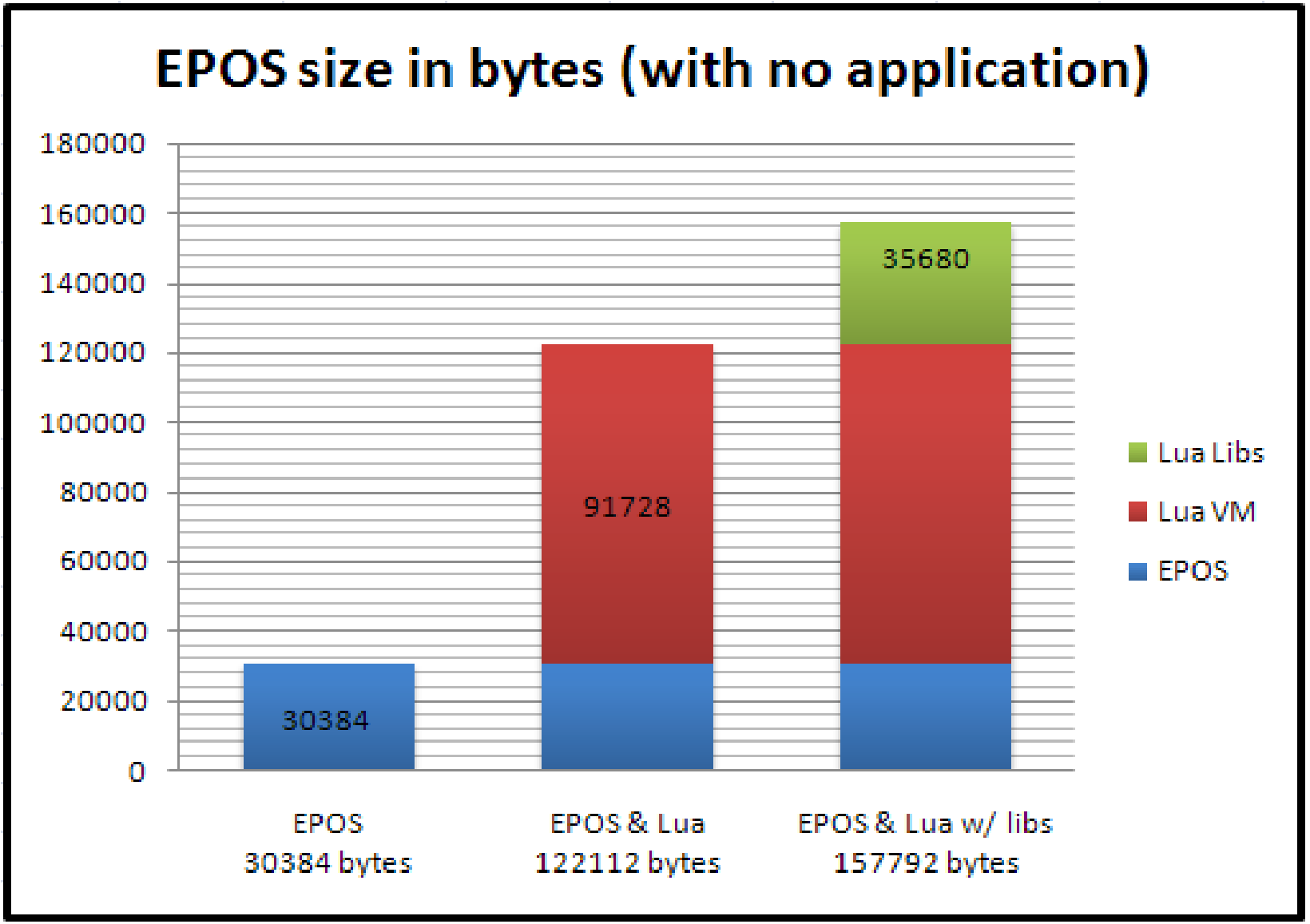
# Performance Evaluation



# Performance Evaluation



# Footprint Evaluation



# Summary

- An approach for HLL@ES
- Lua Profile for ES
- Case: Lua@EPOS
- Evaluation: Lua@EPOS
- Final remarks

# Final Remarks

- Lua for EPOS
  - Improves application development
    - Portability and flexibility for embedded systems
  - Reduces time-to-market
- LVM @ EPOS
  - Reduces the gap between high-level languages and embedded systems
- Impact on performance and footprint still prevents real deployment
  - LVM redesign might reach the 8-bit / 32 Kbytes frontier