# Porting of MicroPython to LEON Platforms

Damien P. George

George Robotics Limited,
Cambridge, UK

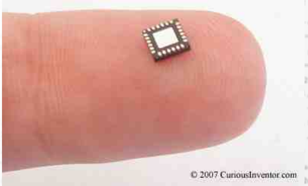
TEC-ED & TEC-SW Final Presentation Days
ESTEC, $10^{th}$ June 2016

# George Robotics Limited (UK)

- a limited company in the UK, founded in January 2014
- specialising in embedded systems
- hardware: design, manufacturing (via 3rd party), sales
- software: development and support of MicroPython code

# Motivation for MicroPython

Electronics circuits now pack an enormous amount of functionality in a tiny package.



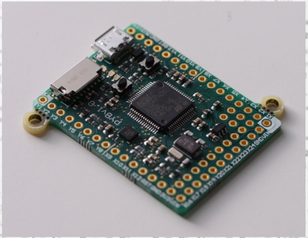

Need a way to control all these sophisticated devices.

Scripting languages enable rapid development.

**Is it possible to put Python on a microcontroller?**

Why is it hard?

- Very little memory (RAM, ROM) on a microcontroller.



# Why Python?

- High-level language with powerful features (classes, list comprehension, generators, exceptions, ...).
- Large existing community.
- Very easy to learn, powerful for advanced users: shallow but long learning curve.
- Ideal for microcontrollers: native bitwise operations, procedural code, distinction between int and float, robust exceptions.
- Lots of opportunities for optimisation (this may sound surprising, but Python is compiled).

# Why can't we use existing CPython? (or PyPy?)

- Integer operations:

  Integer object (max 30 bits): 4 words (16 bytes)

  Preallocates 257+5=262 ints $\longrightarrow$ 4k RAM!

  Could ROM them, but that's still 4k ROM.

  And each integer outside the preallocated ones would be another 16 bytes.
- Method calls:

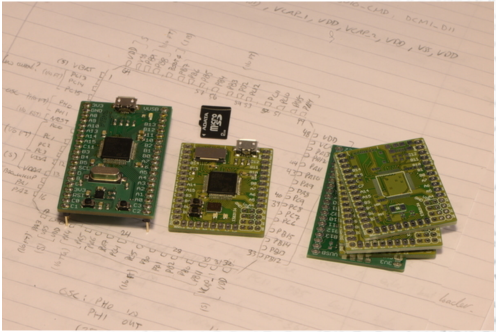
  led.on(): creates a bound-method object, 5 words (20 bytes)

  led.intensity(1000) $\longrightarrow$ 36 bytes RAM!
- For loops: require heap to allocate a range iterator.

# Crowdfunding via Kickstarter

Kickstarter, since 2009; collected so far over US$1 billion in funds

- 30th April 2013: start!
- 17th September: flashing LED with switch in bytecode Python.
- 21st October: REPL, filesystem, USB VCP and MSD on PYBv2.



1 weekend to make the video.

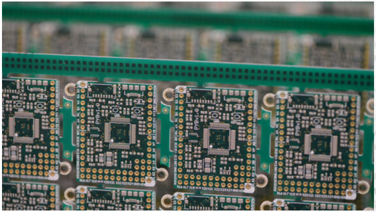
Kickstarter launched on 13 November 2013, ran for 30 days.

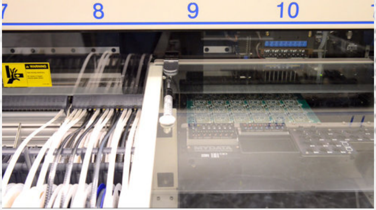
Total backers: 1,931
Total raised: £97,803

Officially finished 12 April 2015.

# Manufacturing

Jaltek Systems, Luton UK — manufactured 13,000+ boards.

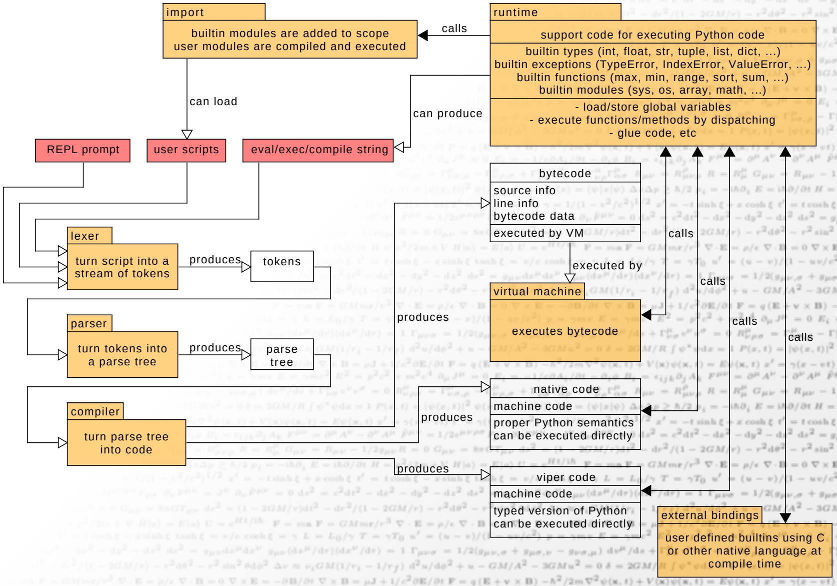


# It's all about the RAM

If you ask me **'why is it done that way?'**,
I will most likely answer: **'to minimise RAM usage'**.

- Interned strings, most already in ROM.
- Small integers stuffed in a pointer.
- Optimised method calls (thanks PyPy!).
- Range object is optimised (if possible).
- Python stack frames live on the C stack.
- ROM absolutely everything that can be ROMed!
- Garbage collection only (no reference counts).
- Exceptions implemented with custom `setjmp/longjmp`.

# Internals



import
builtin modules are added to scope
user modules are compiled and executed
calls
runtime
support code for executing Python code
builtin types (int, float, str, tuple, list, dict, ...)
builtin exceptions (TypeError, IndexError, ValueError, ...)
builtin functions (max, min, range, sort, sum, ...)
builtin modules (sys, os, array, math, ...)
- load/store global variables
- execute functions/methods by dispatching
- glue code, etc
can load
can produce
REPL prompt
user scripts
eval/exec/compile string
bytecode
source info
line info
bytecode data
executed by VM
calls
lexer
turn script into a stream of tokens
produces
tokens
executed by
calls
virtual machine
executes bytecode
calls
calls
parser
turn tokens into a parse tree
produces
parse tree
produces
native code
machine code
proper Python semantics
can be executed directly
compiler
turn parse tree into code
produces
produces
viper code
machine code
typed version of Python
can be executed directly
external bindings
user defined builtins using C or other native language at compile time

# Object representation

A MicroPython object is a machine word, and has 3 different forms.

Integers:

- xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxx1
- Transparent transition to arbitrary precision integers.

Strings:

- xxxxxxxx xxxxxxxx xxxxxxxx xxxxxx10
- Certain strings are not interned.

Objects:

- xxxxxxxx xxxxxxxx xxxxxxxx xxxxxx00
- A pointer to a structure.
- First element is a pointer to a type object.
- ROMable (type, tuple, dictionary, function, module, . . . ).

Work on LEON port added representation for 64-bit NaN boxing.

# GitHub and the open-source community

`https://github.com/micropython`

MicroPython is a *public* project on GitHub.

- A global coding conversation.
- Anyone can clone the code, make a fork, submit issues, make pull requests.
- MicroPython has over 2900 "stars" (top 0.02%), and more than 580 forks.
- Contributions come from many people, with many different systems.
- Leads to: more robust code and build system, more features, more supported hardware.
- Hard to balance inviting atmosphere with strict code control.

A big project needs many contributors, and open-source allows such projects to exist.

# Porting MicroPython to LEON

Small activity (ESTEC contract 4000114080) for 'MicroPython in Space'.

Objectives:

- Prototype port of MicroPython to LEON 2, on top of RTEMS 4.8
- 10 months duration, exploration of idea (no qualification)
- Special attention to the needs of Space:
  - determinism
  - concurrency
  - low use of resources (CPU, RAM)
  - interfacing to native (C/Ada) code

Deliverables:

- improved MicroPython core
- SPARC / LEON support
- RTEMS port with `rtems` module
- OBCP prototype engine
- test suite
- reports: analysis, description of system, user manual

# The port to LEON: improvements to core

- separation of VM and compiler (only VM would need to be qualified, about 200kB compiled excl. RTEMS)
- MicroPython cross compiler and persistent bytecode generation
- option for 64-bit NaN-boxing object model:
  - floats are objects: heap needed, good overall speed
  - floats are boxed: no heap (deterministic), faster FP ops, slower overall due to 64-bit copying
- tool to order static hash tables to proper hashes
- understanding of determinism:
  - execution time of VM opcodes
  - allocation of heap memory
- optimisations to eliminate heap usage in places (eg iterators)
- many bug fixes and speed optimisations (eg combining bytecodes)

# The port to LEON: LEON specifics

- support for SPARC v8 architecture
- ability to have multiple VMs running in the same address space, each with their own heap
- multitasking delegated to RTEMS (atomicity of ops guaranteed)
- synchronisation achieved with RTEMS queues
- multitasking also available via Python co-routines using "yield", good for soft real-time applications (can share heap)
- creation of the `rtems` module (queue, task, semaphore, timer)
- `datapool` module to share data (from C and Python)
- OBCP prototype engine (see later)

# The port to LEON

Heap management:

- traditional problem, it can be non-deterministic
- IBM metronome GC is too complex for qualification
- simplest solution: allocate beforehand, most functionality uses stack
- `heap_lock` and `heap_unlock` methods (exception raised on allocation, can be managed)

Performance:

- about 100x slower than equivalent C code (expected, similar to PC)
- implement performance critical code in C and wrap it (easy to do)
- Python is an application-level language, fast development

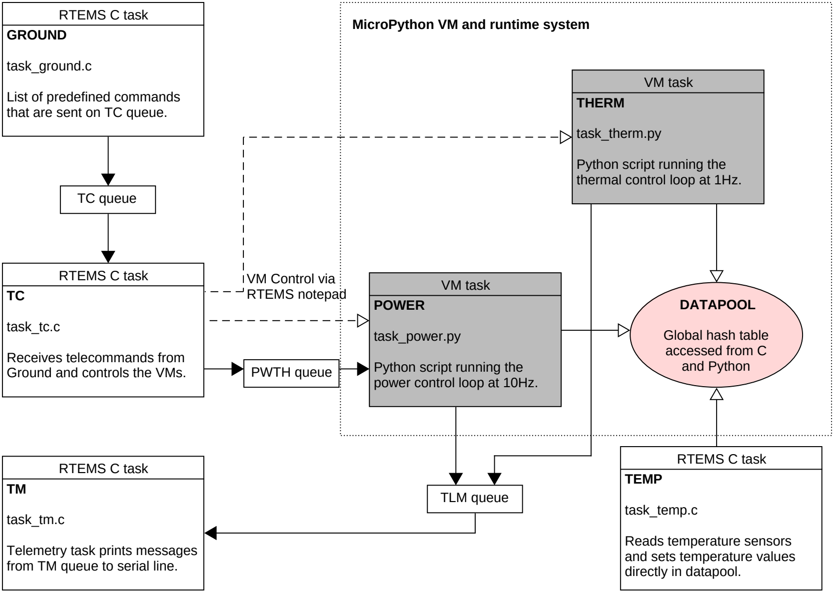
# OBCP prototype engine

A prototype On-Board Control Procedure engine, not qualified, just for demonstration purposes.

Features:

- 4 VM instances (2 idle, 2 running)
- VM interface: load, execute, pause, resume, step, stop
- simulated Ground sends precompiled bytecode
- C tasks, Python tasks running together
- communication via queues and datapool
- demonstrates calls to native code
- heap is locked after start-up

# OBCP prototype engine



# Thermal script

```
def main():
    print('THERM script started, using VM', rtems.script_id())

    # get the TM queue
    tm_q = rtems.queue.ident('TLMQ')

    # get the datapool
    dp = datapool.ident('DATAPOOL')

    # create the arrays to hold the values and thresholds
    temp_val = array.array('d', N_TEMP * [0])
    temp_thresh = array.array('d', N_TEMP * [0, 0])

    # get initial thresholds from the datapool
    dp.get_buf(K_DP_TEMP_THRESH_30, temp_thresh)

    # print initial thresholds
    for i in range(N_TEMP):
        print('THERM: initial temp_thres(%d): [%.2f, %.2f]' % (i, temp_thresh[2 * i], temp_thresh[2 * i + 1]))

    # buffer for TM messages (will be populated using struct.pack_into)
    buf = bytearray(26)

    # from now on we are deterministic
    micropython.heap_lock()

    # control loop runs at 1Hz
    while True:
        # get temperatures from the datapool
        dp.get_buf(K_DP_TEMP_VAL_30, temp_val)

        # get thresholds from the datapool
        dp.get_buf(K_DP_TEMP_THRESH_30, temp_thresh)

        # check thresholds
        for i in range(N_TEMP):
            if not temp_thresh[2 * i] <= temp_val[i] <= temp_thresh[2 * i + 1]:
                struct.pack_into('>BBddd', buf, 0, K_TM_TEMP_RANGE, i,
                    temp_thresh[2 * i], temp_thresh[2 * i + 1], temp_val[i])
                tm_q.send(buf, rtems.WAIT)

                                                                    15,0-1        73%
```

## License and availability

- MicroPython core: MIT license
- the port to LEON: "ESA Community License Type 3, permissive" (restricted to ESA states)
- for exporting outside ESA region, talk to me
- code available (mid-end June) in European Space Software Repository: https://essr.esa.int/
- documents available: Analysis and Adaptation; Test Spec and Report, Executive Summary, Final Report, User Manual

# Conclusions and Future activities

A powerful and modern language, large community, powerful tools — now available for constrained/embedded systems!

Possible applications in Space: general purpose application language for payloads, OBCP engine

Other applications:

- schools/teaching (micro:bit, pyboard, high-schools and universities)
- hobbyists and hackers
- embedded engineers, to make prototyping easier
- great potential for IoT

Continued software/hardware development:

- Python 3.5 support, and improved compatibility with CPython
- partnering with chip vendors to support their MCUs
- development of new boards

micropython.org

forum.micropython.org

github.com/micropython