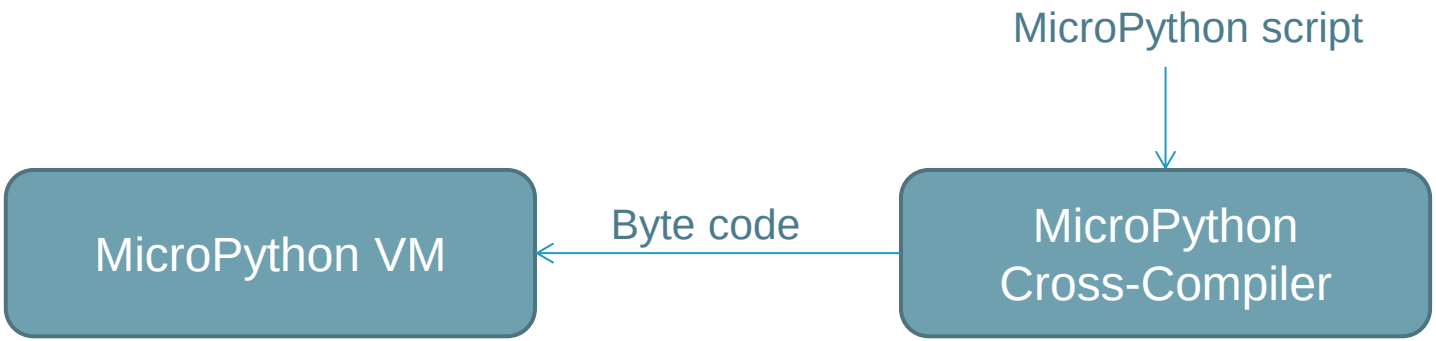
Reaching into space
TOGETHER

# QUALIFICATION OF MICROPYTHON VM AND OBCP ENGINE
# PYTHON IN SPACE

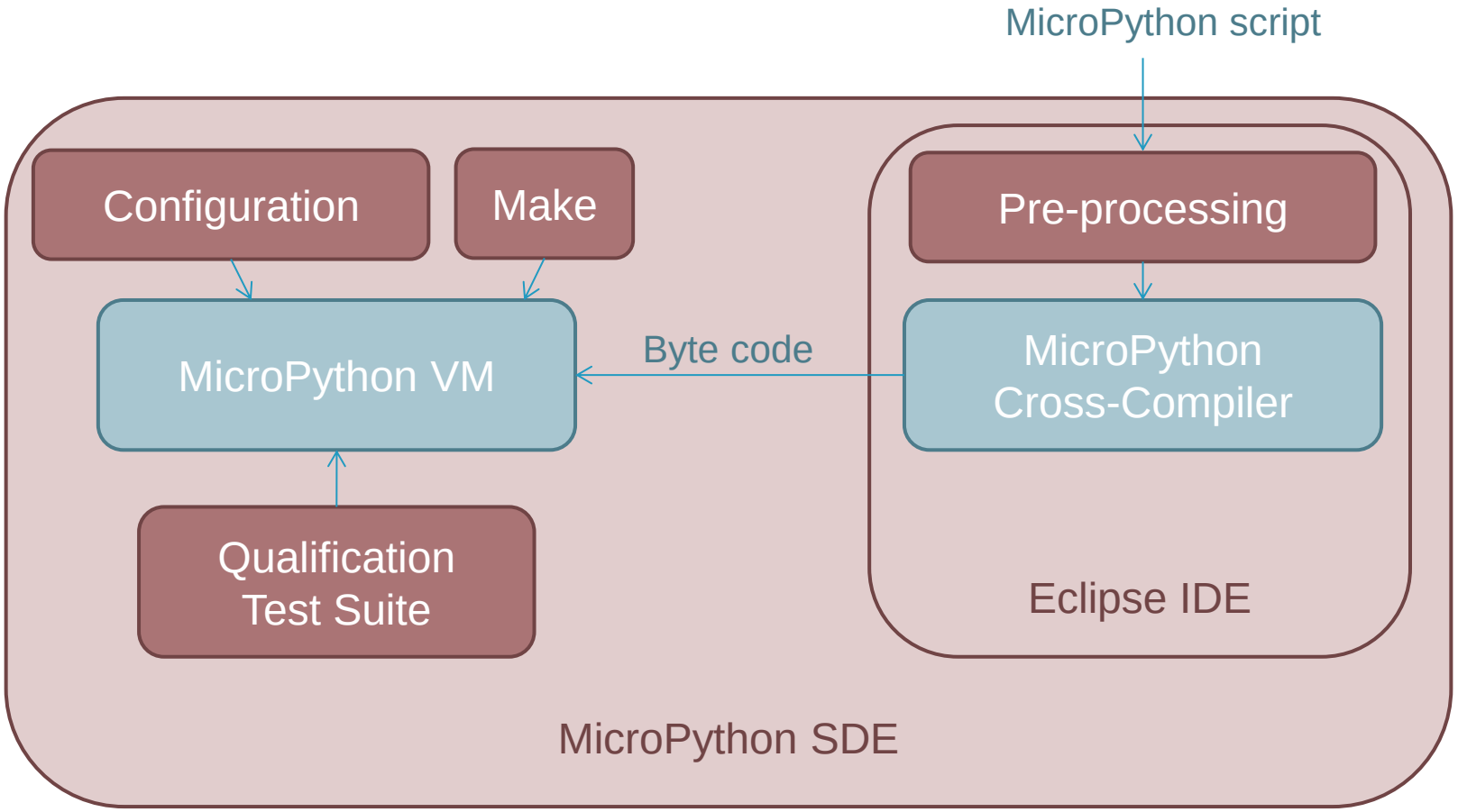
ESTEC – TEC-SW Final Presentation Day
Dec 11th 2018

SPACEBEL

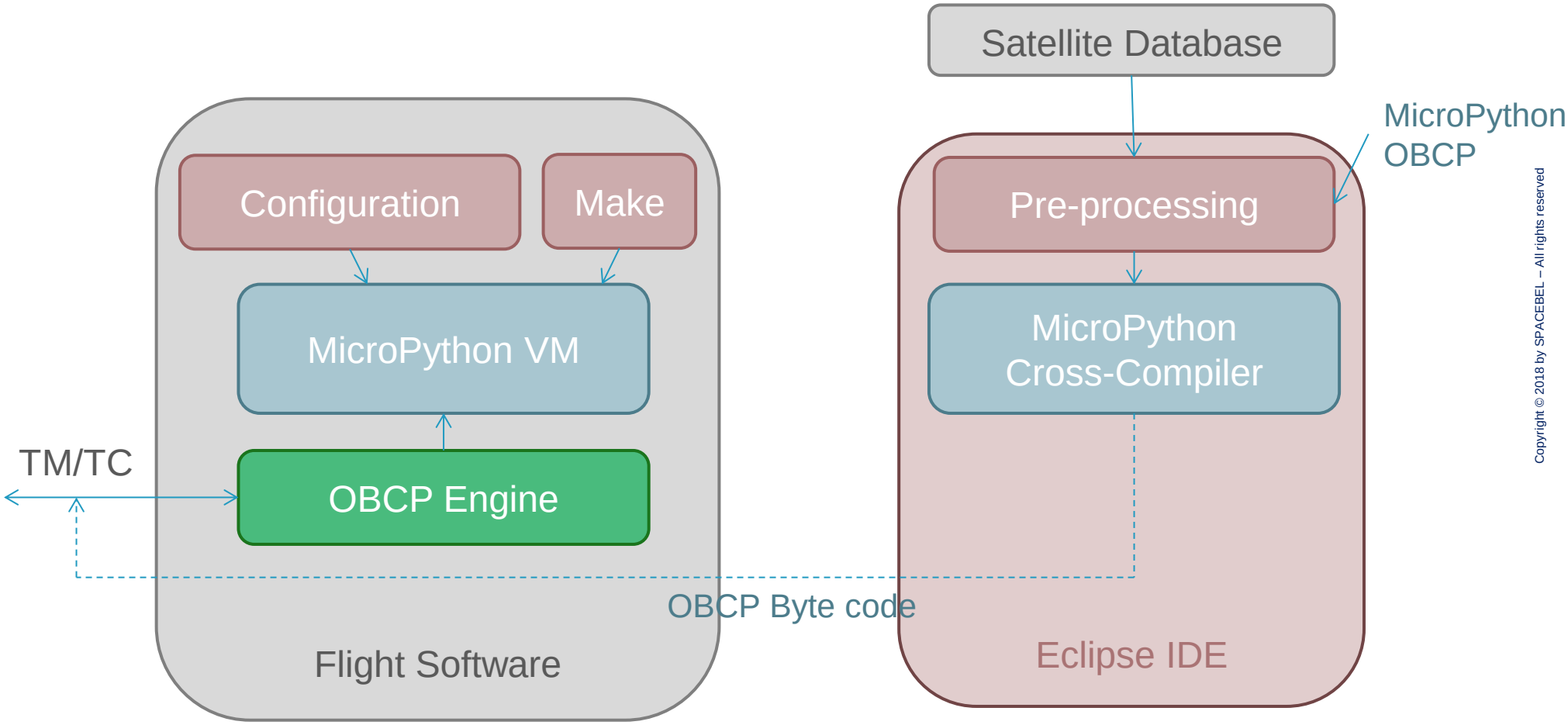
# Scope of the project



# Scope of the project



# Scope of the project

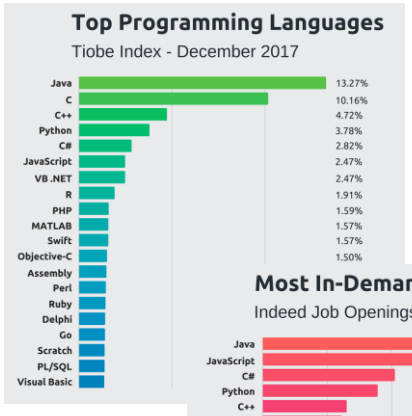


# Why a (qualified) Virtual Machine ?

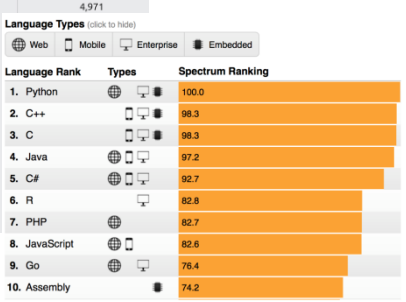
- On-Board Control Procedures (OBCP)
  - Flexible flight procedures
  - Prepared on ground and dynamically uploaded
  - Extend flight SW functionality / simplify ground procedures
  - Tight requirements on OBCP execution environment !

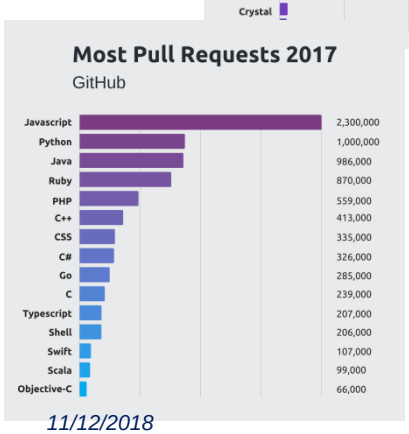
- Payload Control Software
  - High-level execution environment
  - Must provide access to minimum of HW I/O

# Why MicroPython ?

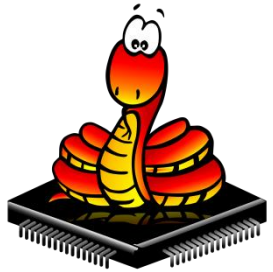







- Python is cool
  - MicroPython is available
    - Optimised for time/space
    - Python 3.4
    - Open Source
    - C implementation
    - Ported on various microcontrollers
    - George Robotics Ltd, http://micropython.org



- Prototype port for LEON2
  - “Porting of MicroPython to LEON platforms” ESA project

# MicroPython VM (qualified) features

MicroPython VM

- Python 3.4
- Object-orientation
- Exception handling
- Math functions
- list, dict, bytearray, …
- Built-in imports
- Lots of built-in functions
- Extensibility
- …

Excluded:

- External imports
- Most standard modules
- …

# VM Time/Space optimisation


Controlled memory usage:

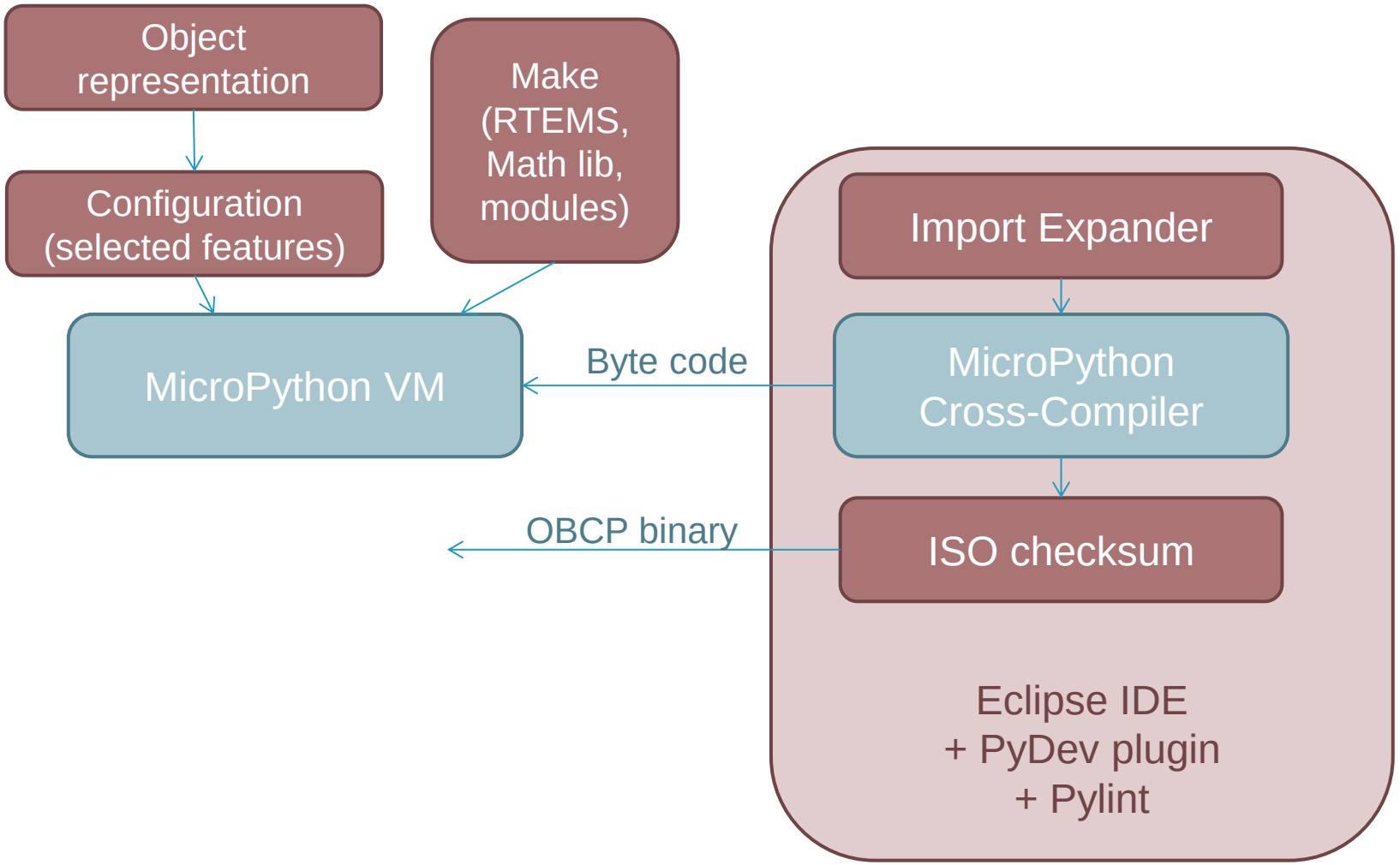
- Dedicated buffer for Python stack
- Dedicated buffer for heap
- Garbage collector
- Heap can be locked



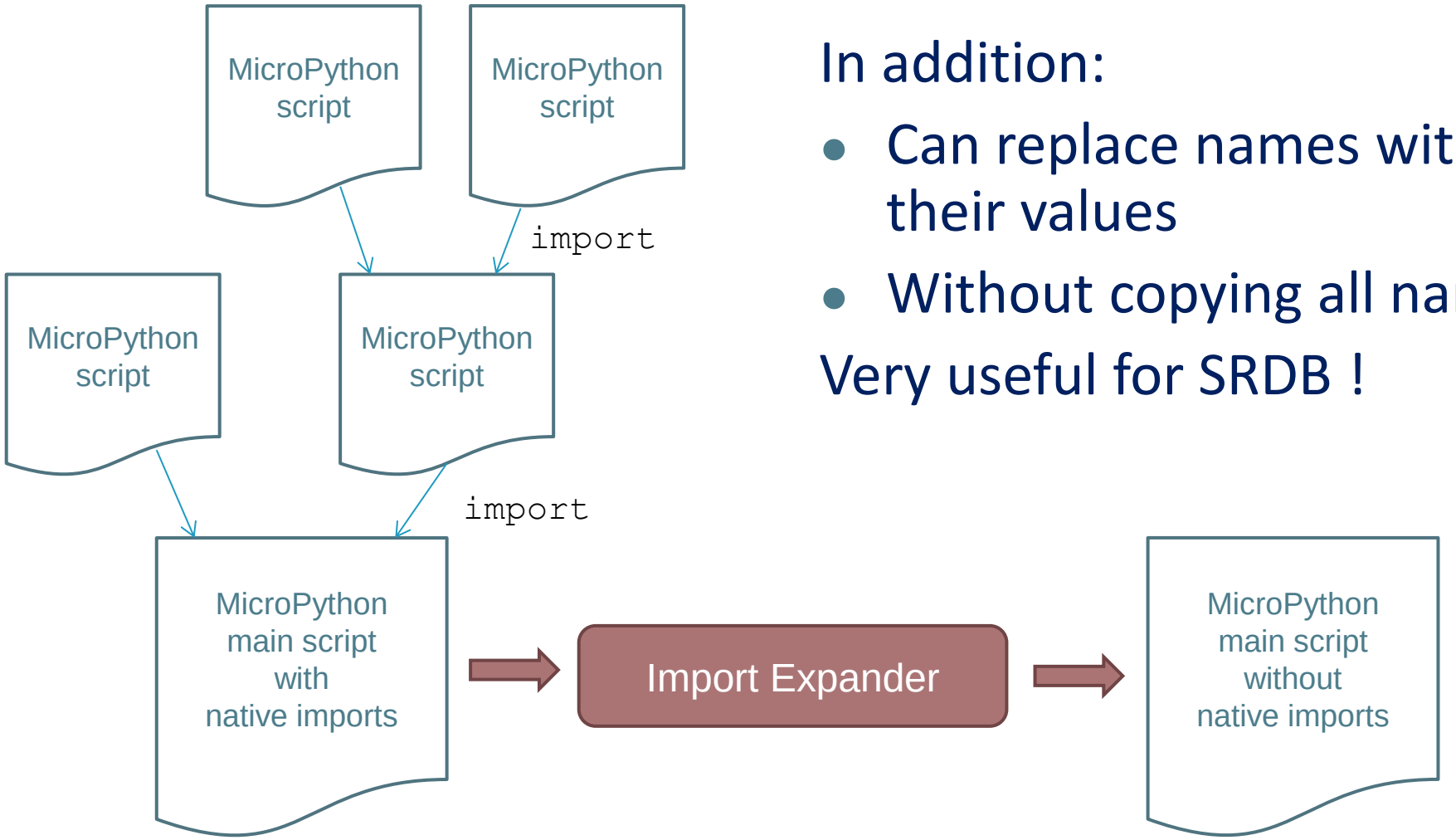
2 object representations impacting memory and heap usage:

- 32 bits (objects are 32 bits, but any float is allocated on heap)
- 64 bits NaN boxing (any object takes 64 bits)

# MicroPython SDE features



# SDE Import Expander

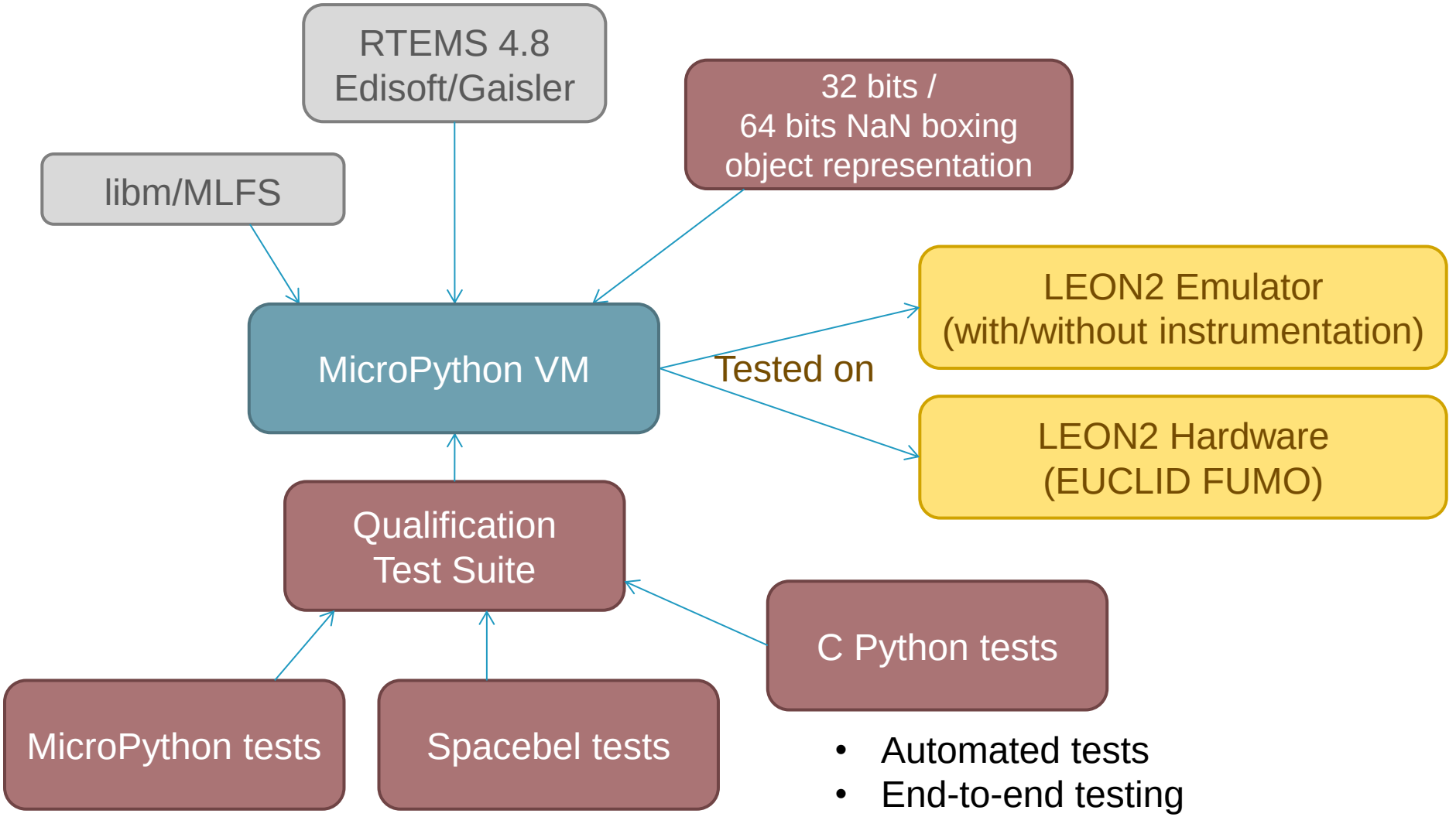



In addition:

- Can replace names with their values
- Without copying all names

Very useful for SRDB !

# VM testing



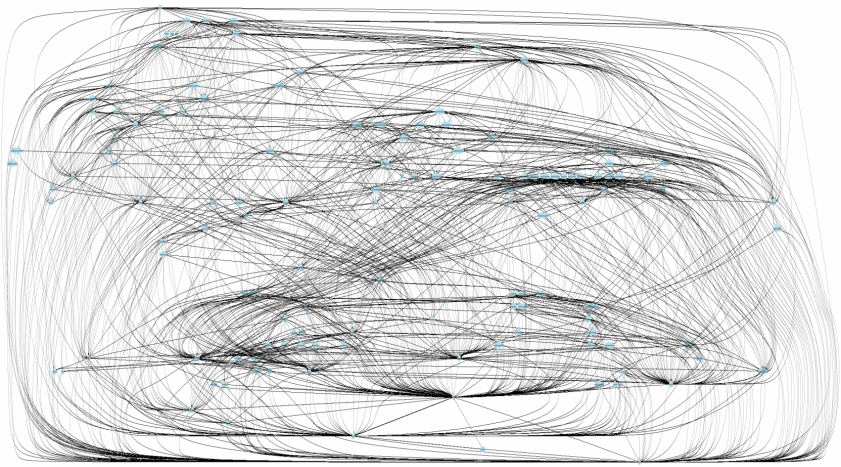

- Automated tests
- End-to-end testing
- 624 MicroPython scripts

# VM analyses

Code coverage analysis:

- Cantata
- Removal of unused code (MicroPython VM macro configuration)
- Justification of remaining non-covered code

MicroPython VM



Continuous static code analysis:

- Polyspace Bug Finder

Call graph analysis:

- Identification of direct and indirect recursive calls
- Protection of C stack usage

# VM (and related) issues

Coding style:

- Remain compatible with master branch
- No applicable coding standard and no adaptation !

MicroPython VM

Floating Points:

- NaN, -0.0
- Issues in libm
- Issues in VM
- Issues in LEON emulators
- Questions on IEEE FP standard

RTEMS Edisoft:

- Absence of trap 0x83 (window flush) required for NLR
- No raw trap handler registration (set_vector)

VM SPRs:

- 62 fixed
- 48 (minor) used as-is

# VM (and related) issues

48 minor VM SPRs used « as is »:

- 27 « not implemented » issues
- 21 « deviations » wrt C Python

MicroPython VM

## Deviations (examples):

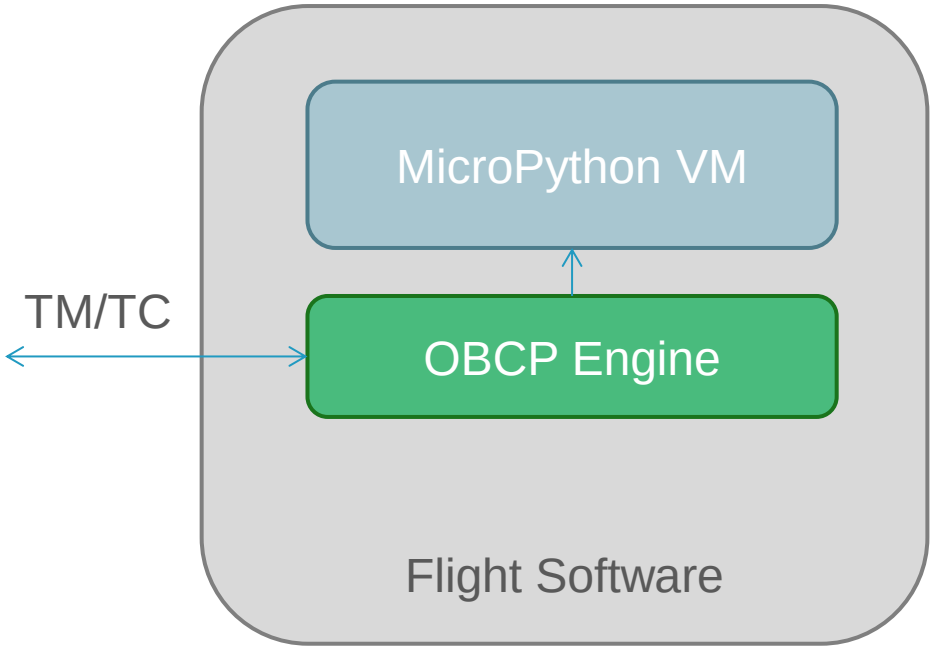
- `float('.') --> 0.0`
  - Not allowed in CPython
- `sum(['a','b','c'],'') --> 'abc'`
  - Not allowed in Cpython
- `{:f}'.format(1234.) --> '1233.999999'`
- `1.015625 == 1015625E-6 --> False`
  - Precision of 1e-13 in MicroPython via str
- Deviations with multiple inheritance
- Deviations with garbage collection

## Not implemented (examples):

- Exception raised in such case
- `locals()`
- `array.array`, `bytearray` semantics not complete (deletion, accessors)
- `bytes`, `str` semantics not complete (keywords)
- `rsplit` semantics not complete
- `__init__` method not supported for all built-in types (`Exception`…)
- List/tuple/array/bytes/subscript slices with step != 1
  - `print('abcdefghi'[0:9:2])`

# OBCP Engine features

- New Flight SW component
- PUS 18 TM/TC interface




- Additional MicroPython module: obcp_engine
- Control/monitoring of VM
- Dynamic aspects (multi-tasking)
- Fault containment
- Highly configurable

# obcp_engine module

```
import obcp_engine

obcp_engine.step(1)
obcp_id = obcp_engine.get_obcp_id()

obcp_engine.step(2)
time_before = obcp_engine.get_time()
obcp_engine.sleep(5.0)
obcp_engine.wait_1hz_sync()
time_after = obcp_engine.get_time()
delay = time_after - time_before
```

Step is observable (in HK)
Possible to stop/suspend on a step

OBCP id determined when loading

Returns OBET as float

« OS » task wait versus synchro with HW signal

# obcp_engine module

```
arg_buffer = bytearray(4)
obcp_engine.read_obcp_arguments(arg_buffer)

import sdb
obcp_engine.write_db_parameter(sdb.DB_PARAM5,arg_buffer)
x = obcp_engine.read_db_parameter(sdb.DB_PARAM6)

mem_buf = bytearray(20)
obcp_engine.read_memory(sdb.MEM_EEPROM,0,mem_buf)
obc_engine.write_memory(sdb.MEM_RAM,1024,mem_buf)

obcp_engine.send_tm(sdb.GROUND,128,2,mem_buf)
obcp_engine.generate_event(sdb.NULL,sdb.EVT_EXAMPLE1)
```

Arguments can be provided when OBCP is started or when it runs

Data Bank access (using SRDB)

Memory access

TM generation

# obcp_engine module

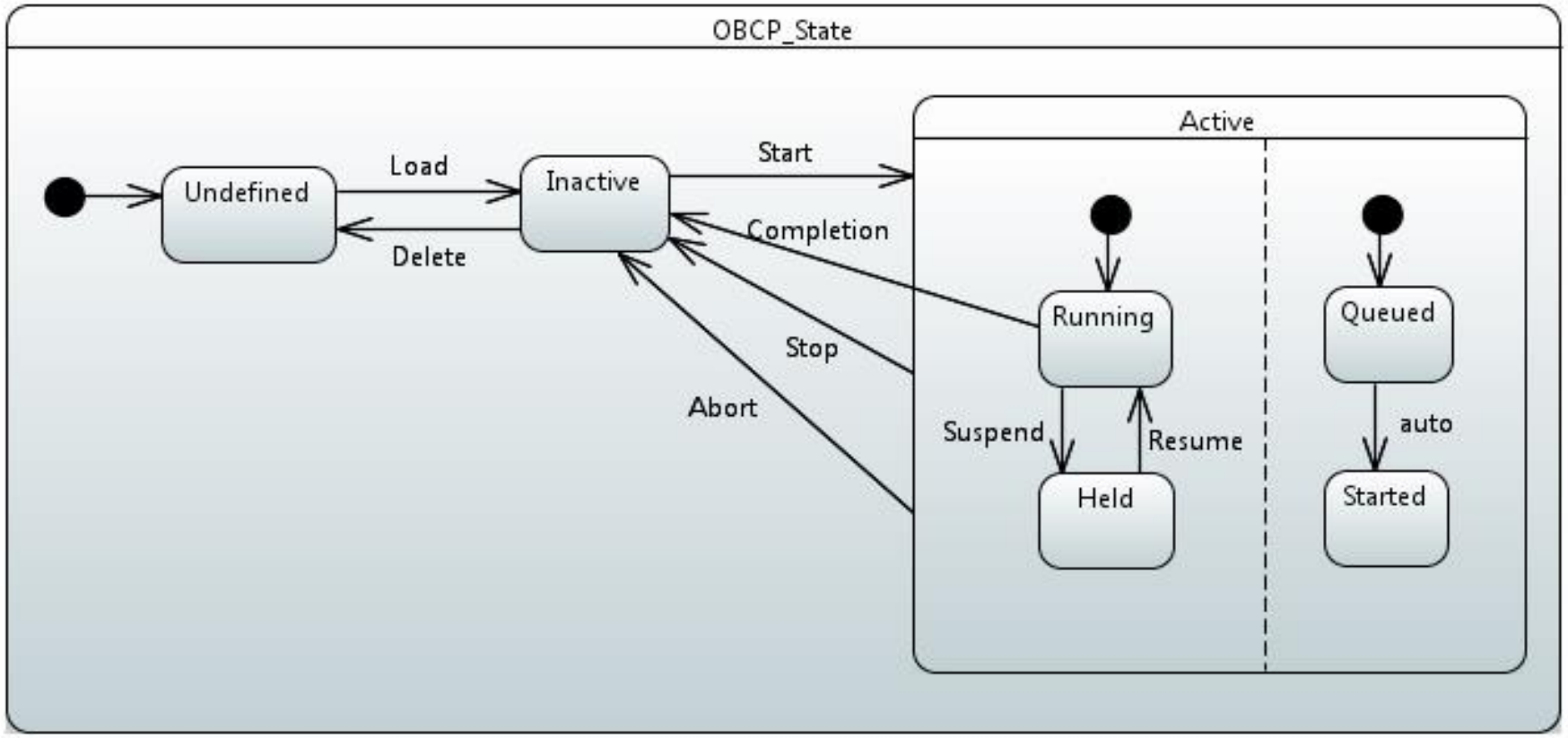
```
tc = bytearray(25)
obcp_engine.format_tc(tc,sdb.HK,3,1,True,True)
tc_id = obcp_engine.send_tc(tc)

acc = obcp_engine.check_tc_acceptance_acknowledge(tc_id)
compl = obcp_engine.check_tc_completion_acknowledge(tc_id)

tm = bytearray(40)
obcp_engine.subscribe_tm(sdb.HK,3,25,5)
while obcp_engine.receive_message(tm) == 0:
    pass
# process tm here
obcp_engine.unsubscribe_tm()
```

Separate functions to format and to send a TC

TC checks (-1 returned if no ack)

TM access through subscription (similar to ground procedures)

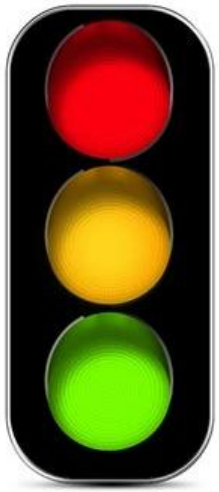
# OBCP states




- High priority OBCP: 1 single VM (with queueing)
- Low priority OBCP: 10 available VM (without queue)

# Dynamic aspects

- 1 VM == 1 RTEMS task
- VM tasks have lowest priority in Flight SW
- 1 VM task with « high priority »
- 10 VM tasks with same « low priority »
  - RTEMS time-slicing to share the available CPU



- VM execution = 1  unique call
- VM control using « hook » mechanism
  - suspend/resume OBCP
  - stop/abort OBCP (with exception)
  - housekeeping (VM memory usage…)

# Fault containment

- CPU usage cannot be limited

→ VM must run at lowest priority

- Controlled access to Flight SW resources
  - Data Bank
  - Memory

- Python stack limited and checked
- Python heap limited and checked
- Lockable heap (no garbage collection)
- C stack cannot be limited (recursivity)
  - but checked in all recursivity cases

→ MicroPython Exception raised in case of memory error

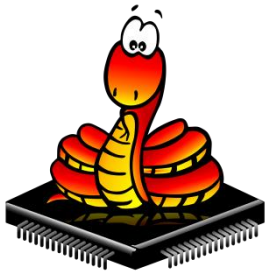
- FP exception/trap cannot be prevented

→ MicroPython Exception is raised in context of VM

# Conclusions

3 available products:

- MicroPython VM + cross-compiler
  - George Robotics Ltd (UK)
  - Open source (MIT license)
    http://micropython.org/
    http://www.georgerobotics.com/



- MicroPython « SDE » + qualification su
  - Including MicroPython VM for LEON
  - ESA Community License (ECL-3)
    https://essr.esa.int/

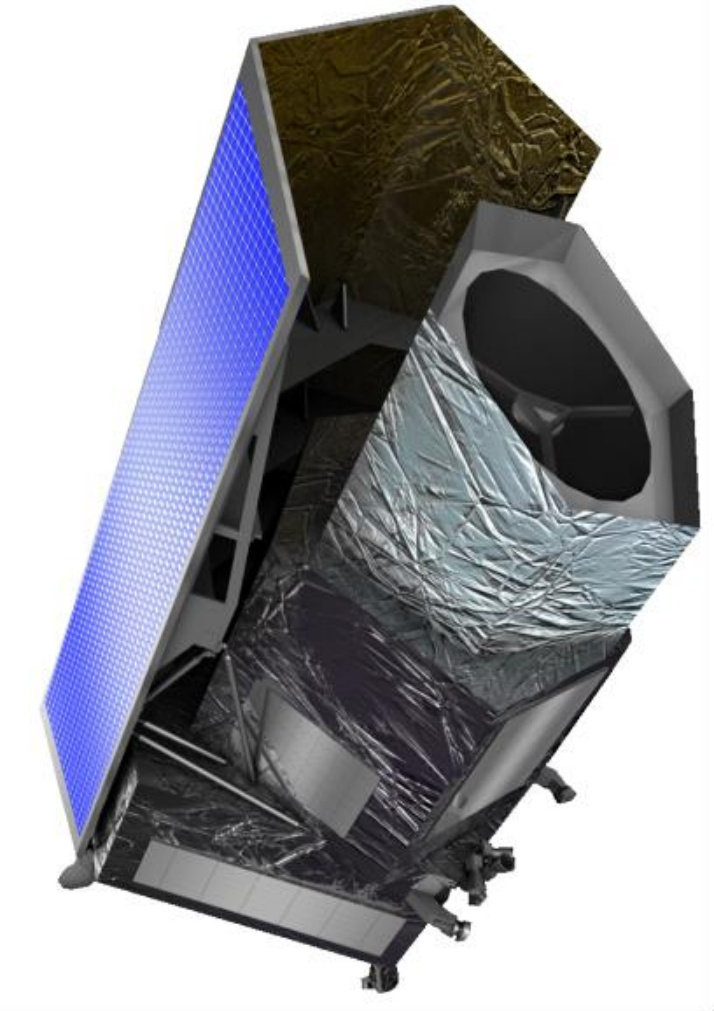
- OBCP Engine based on MicroPython
  - Spacebel IPR
    https://www.spacebel.be/

# Conclusions

Qualification status:

- As per ECSS standards
  - ECSS-E-ST-40C for development
  - ECSS-E-ST-70-01C for OBCP functionality
  - ECSS-E-70-41A for TM/TC PUS 18 service
    (limited gap towards PUS C)
- As per criticality B software (mission critical)
- Full set of documentation and analyses
- Automated test suite executed on LEON2 emulator and LEON2 hardware (EUCLID FUMO)

# Future



ESA EUCLID mission

…

What about your project ?

Reaching into space
TOGETHER

# QUESTIONS ?



SPACEBEL