Reaching into space
TOGETHER

MicroPython Virtual Machine
for
On Board Control Procedures

DASIA 2018

Thomas Laroche, Pierre Denis, Paul Parisis (SPACEBEL)
Damien George (George Robotics Ltd)
David Sanchez de la Llana, Thanassis Tsiodras (TEC/SWE)

Agenda

- Introduction
- Technological Choices
- Architectural Elements
- Virtual Machine Improvments
- Conclusion

Introduction

- On Board Control Procedures
- OBCPs are
  - flight procedures that provide a flexible way
    - to operate the Spacecraft
    - to extend the On Board Software functionality or to modify the behaviour of On Board Applications
- OBCPs are
  - prepared on ground
  - dynamically uploaded, even after launch, and
  - executed on board.
- OBCPs are
  - most of the time
    - written in scripting language,
    - compiled as bytecode and
    - executed in a Virtual Machine (VM)

Introduction (cont.)

On Board Operation Procedures (OBOP)

- OBOPs are kinds of macro-command typically provided by the ground operations team and uploaded on board to operate the spacecraft.
- OBOPs can also take part to the Fault Detection Isolation and Recovery (FDIR), where they detect complex failures or implement recovery procedures.
- OBOPs participate to on board autonomy when a rapid reaction is needed in spite of reduced spacecraft visibility or long propagation delay

- **On Board Application Procedures (OBAP)**

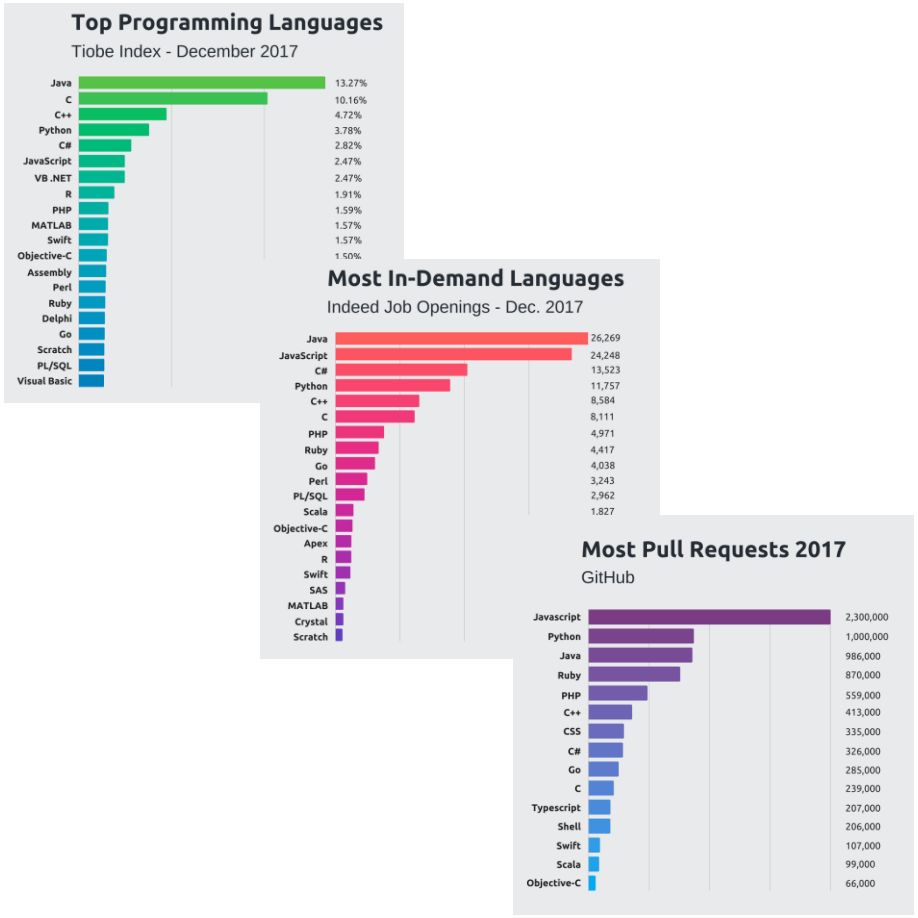
- OBAPs implement part of the basic functionality of the spacecraft.
- OBAPs may be considered as part of or as extensions to the OBSW itself.
- OBAPs are increasingly being considered, in particular for payloads.

Technological Choices

- Language
- Virtual Machine

Langage Selection

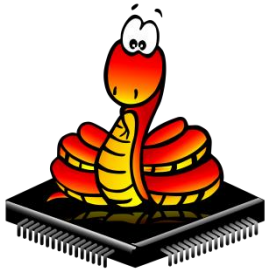
```
import obcp_engine
obcp_engine.send_tm("Hello world !")
```



- Criteria
  - Scripting Langage
  - Standard
  - Open
  - Expressive
- Alternatives
  - Java
  - Python
  - Ruby
  - Lua
- Choice
  - Python

Virtual Machine Selection

- MicroPython
  - a modern, efficient, highly portable and light-weight implementation with small memory footprint and fast execution, designed to be embedded in application and optimised to run on microcontrollers and in constrained environments



"Porting of MicroPython to LEON platforms"
David Sanchez de la Llana, Damien George,
DASIA 2016

  - Software implementation
  - Written in C
  - Complies to Python 3 programming language,
  - Includes a selection of core Python libraries
  - Ported on various microcontrollers

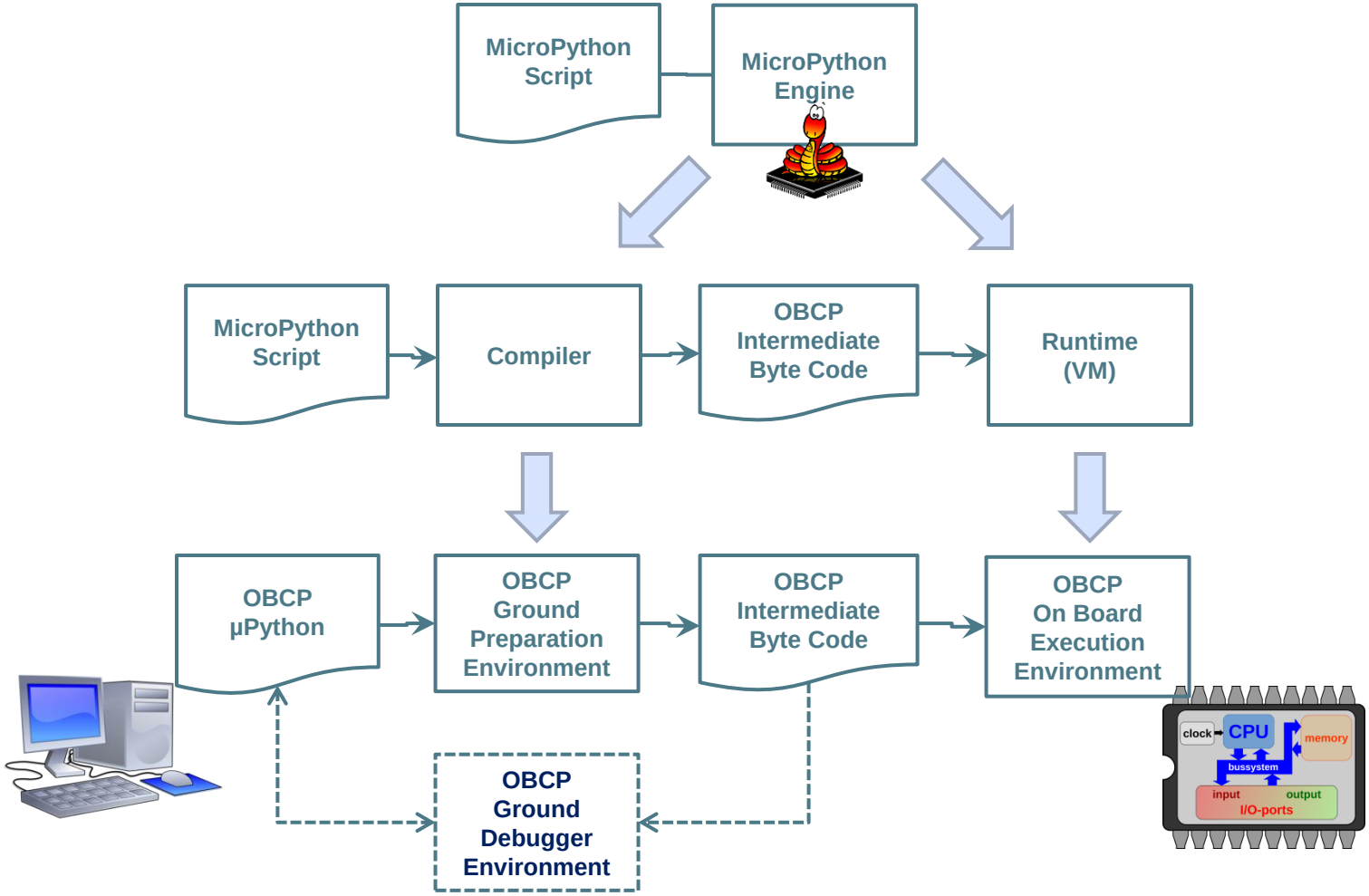
http://www.micropython.org

OBCP Solution Architecture

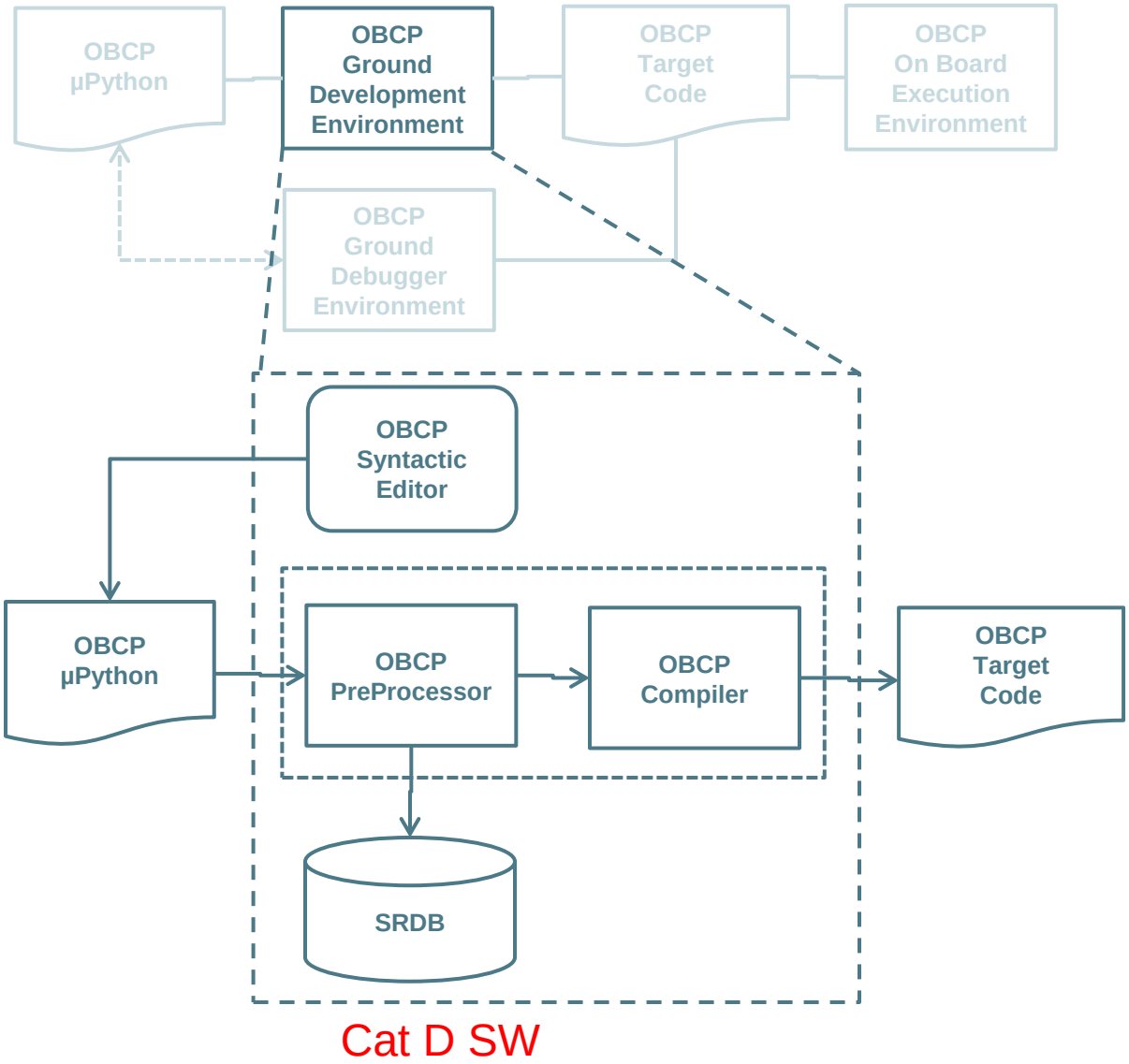
- Overall Architecture
  - Ground Development Environment
  - On Board Execution Environment

Overal Architecture

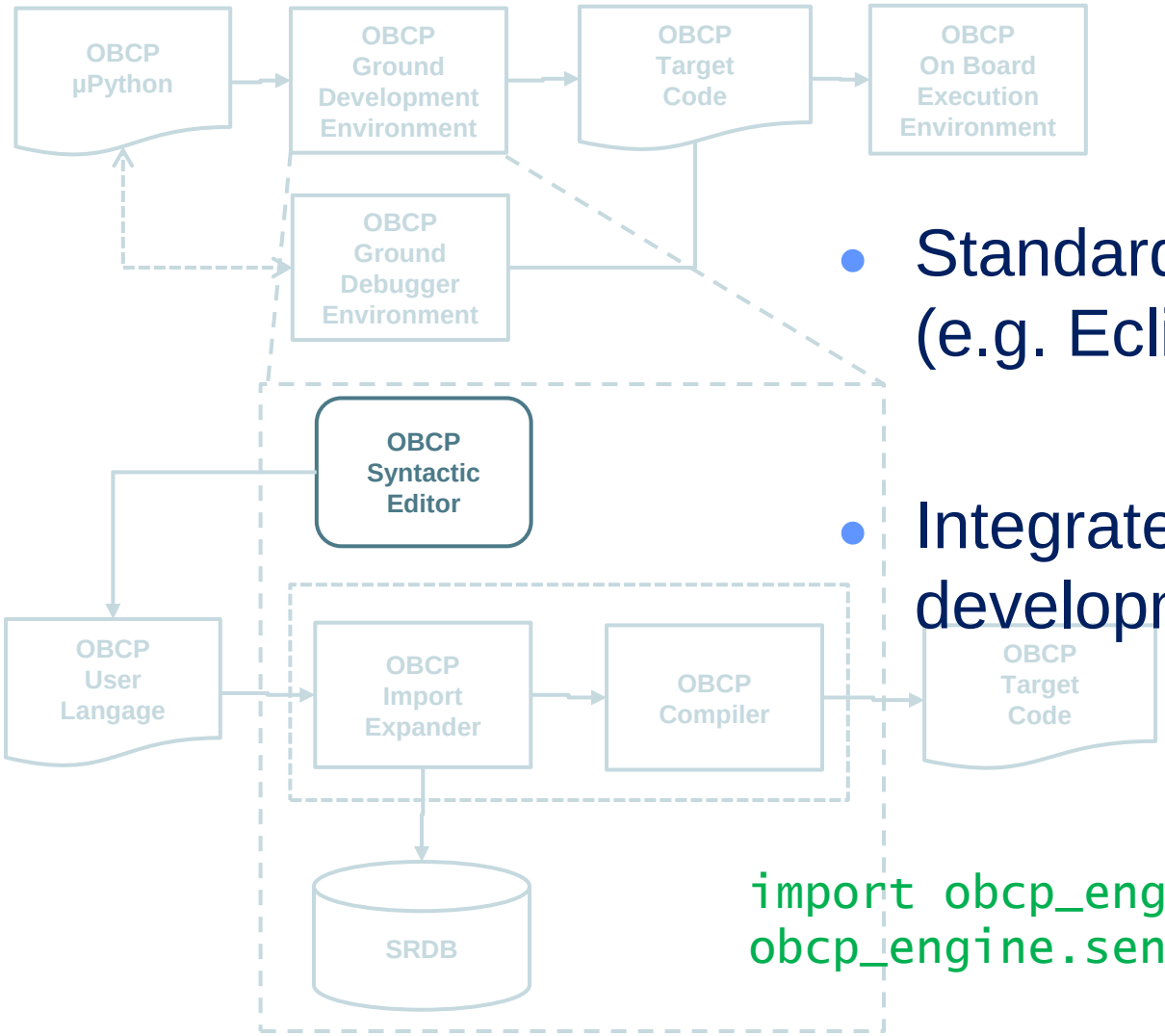
"MicroPython is a full Python Compiler and Runtime that runs on the micro-controller hardware"



Ground Development Environment



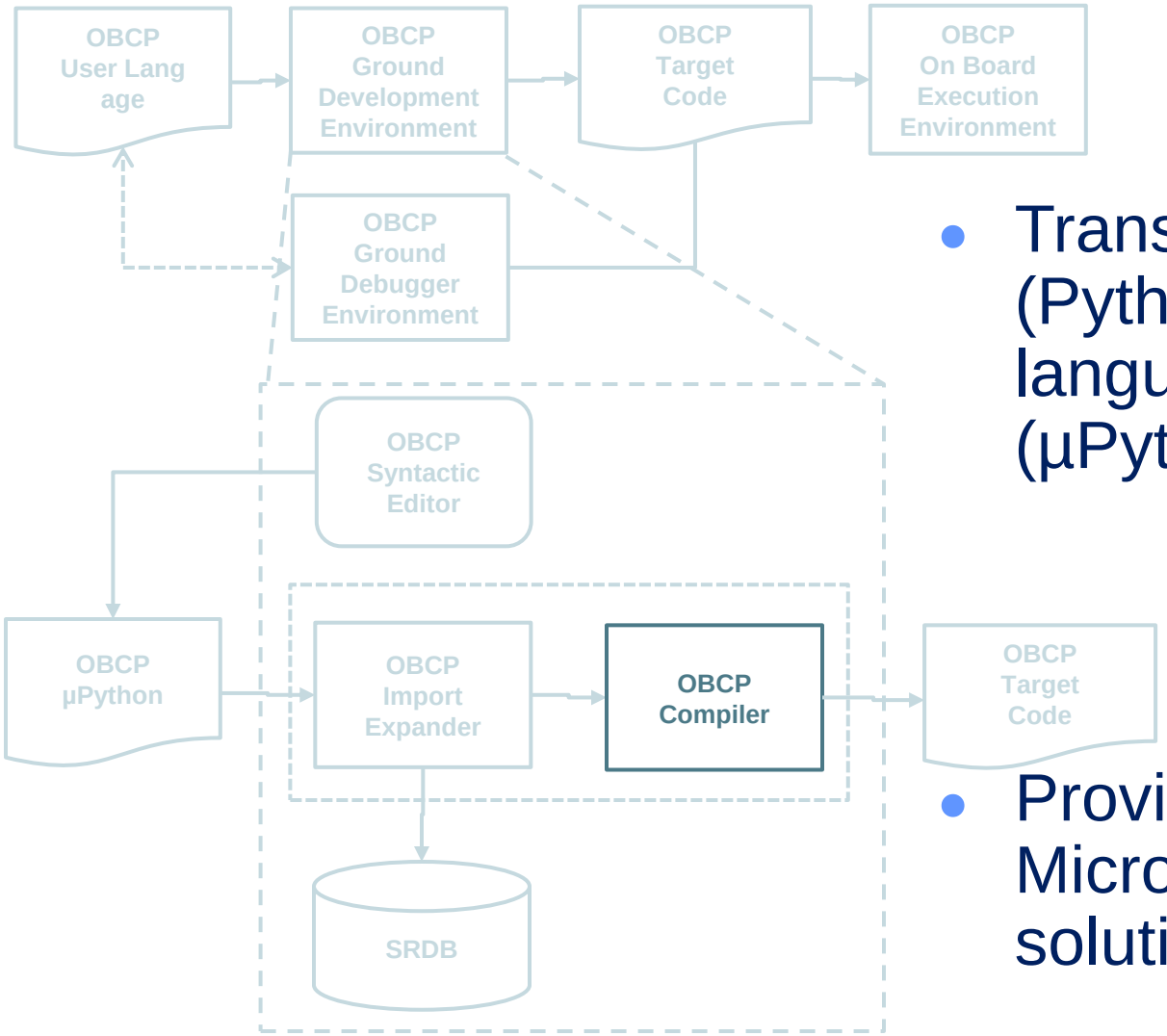
OBCP Syntactic Editor



- Standard Python Editor (e.g. Eclipse/PyDev)
- Integrated in the OBSW development environment

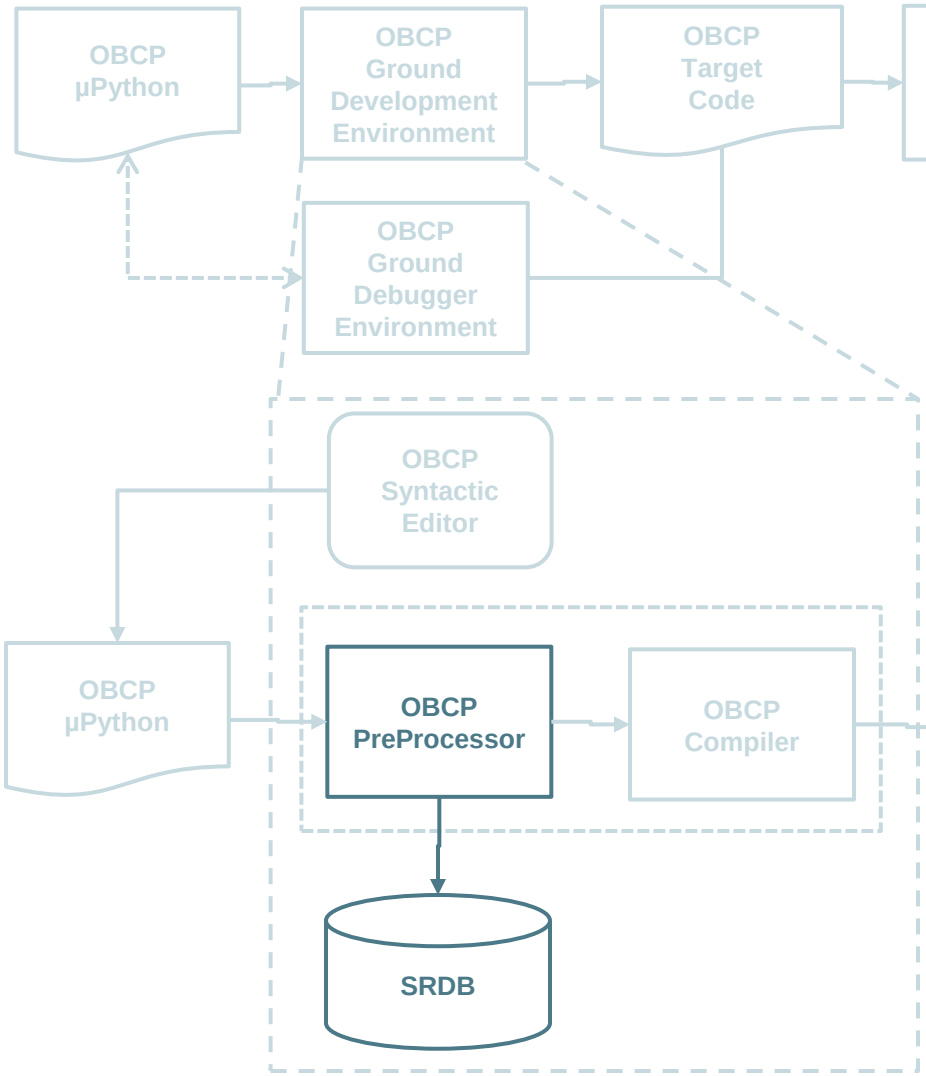
```
import obcp_engine
obcp_engine.send_tm("Hello world !")
```

OBCP Compiler



- Translates the high level (Python scripts) source language in the target (µPython) bytecode
- Provided as part of the MicroPython VM solution.

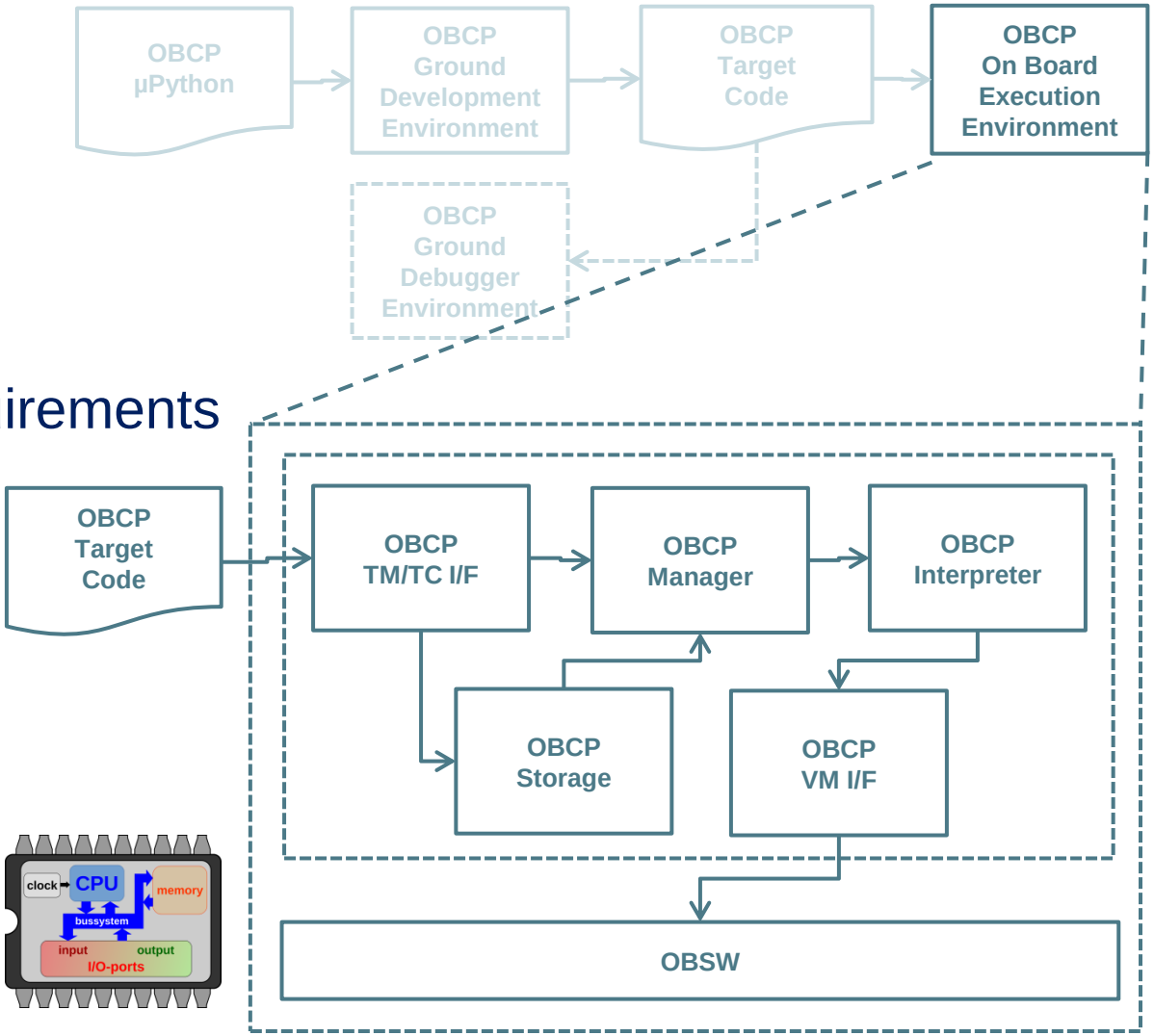
OBCP Pre Processor



- Import Files
  - Build single file from several files on ground
  - Alternative to dynamic import on board .
- Resolve Names
  - Interface to SRDB
  - Replace names by their corresponding value or adress
  - Alternative to loading (part of) the SRDB on board.

On Board Execution Environment

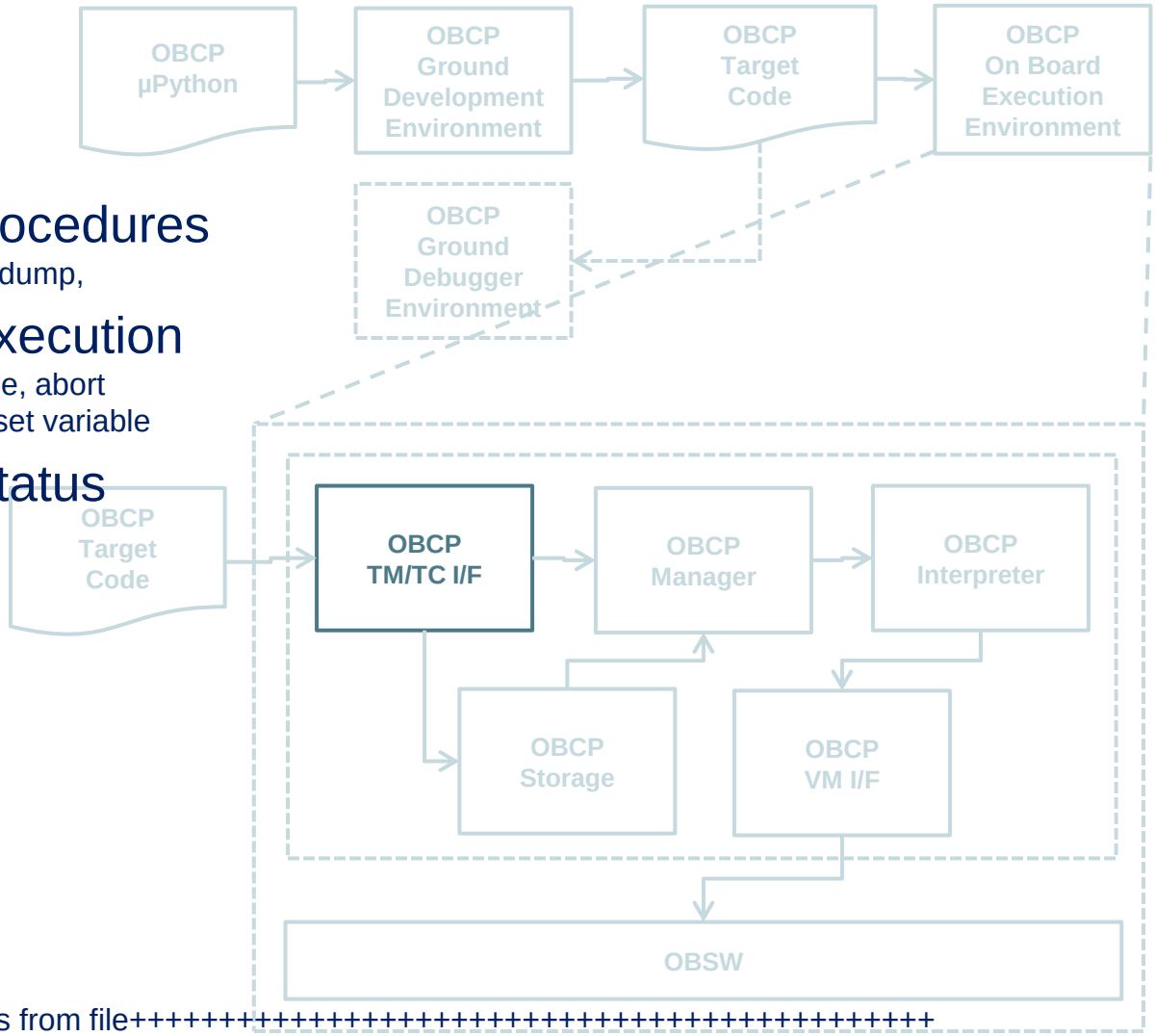
- Comply to
  - OBCP Standard
    ECSS-E-ST-70-01C
  - PUS Standard
    ECSS-E-70-41C
  - ESOC/ESA Requirements
    (from Euclid)



Cat B SW

OBCP TM/TC Interface

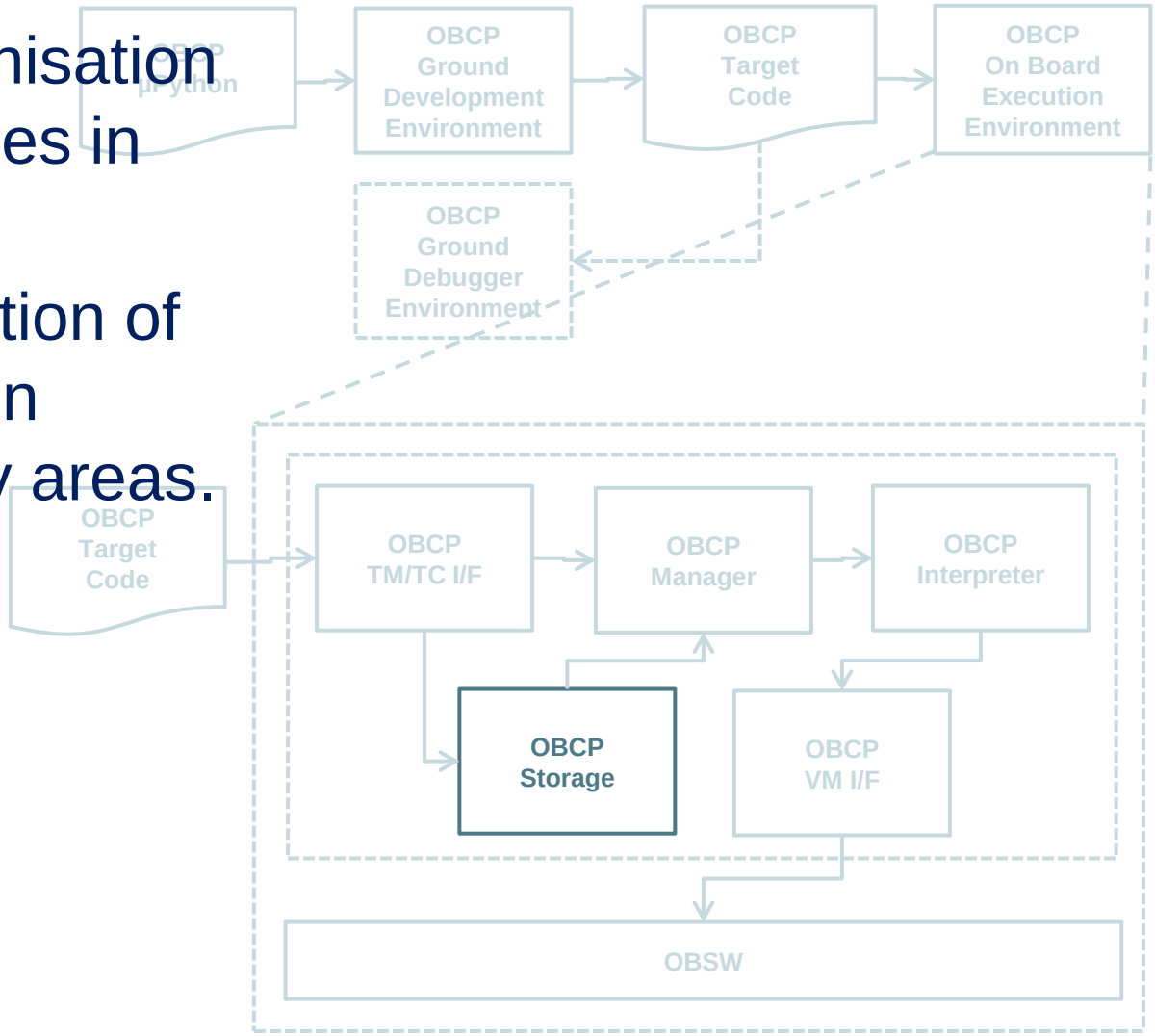
- PUS service 18
  - TC/TM
    - uploading the procedures
      load, delete, load from file, dump,
    - controlling the execution
      start, stop, suspend, resume, abort
      communicate parameters, set variable
    - monitoring the status
      report lists, …
- Interacts with
  - OBCP Storage
  - OBCP Manager



Note :
- additional telecommand to upload OBCPs from file++

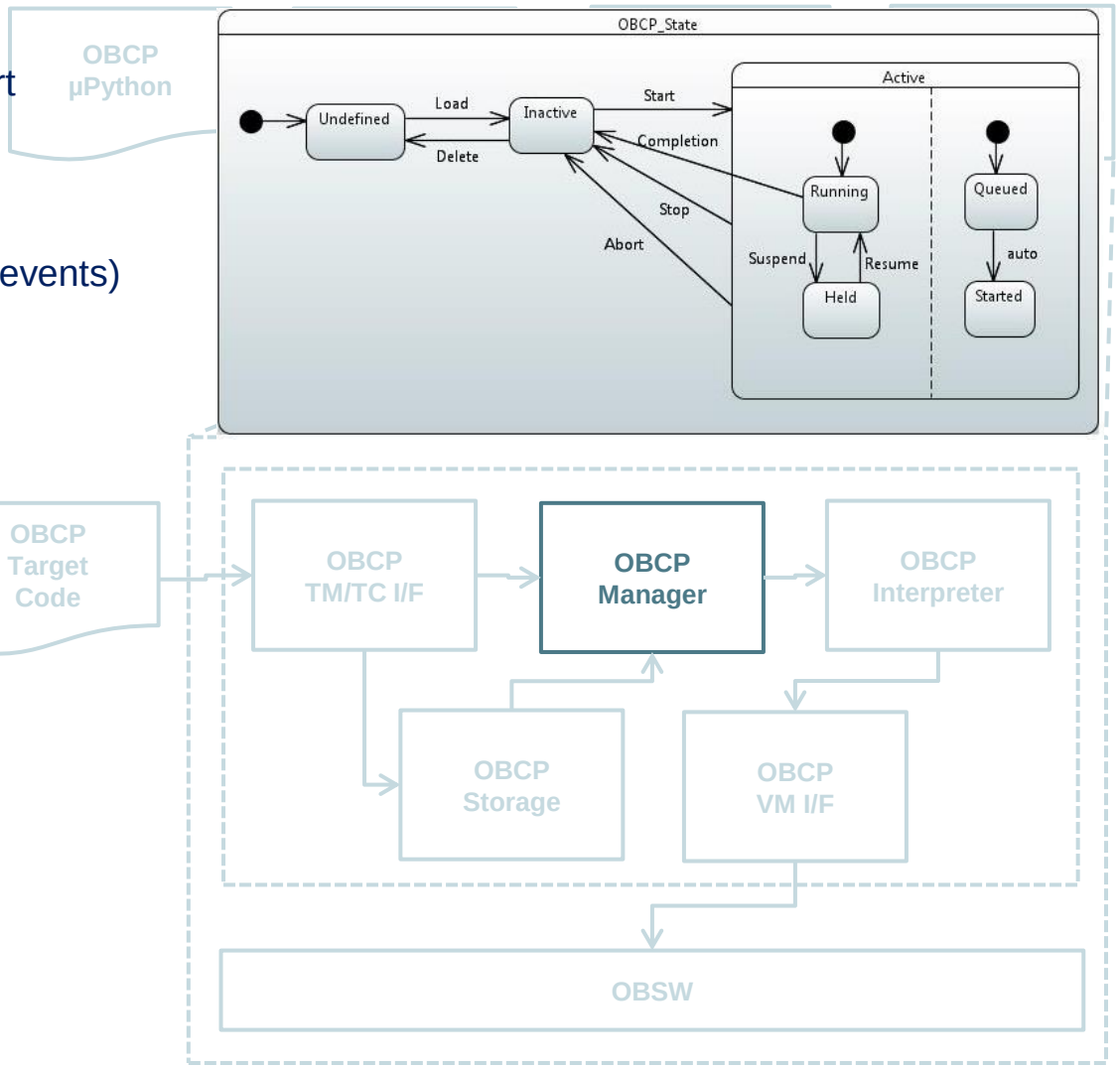
OBCP Storage

- Handle the organisation of OBCP accesses in memory
- Manage a collection of OBCPs located in different memory areas.



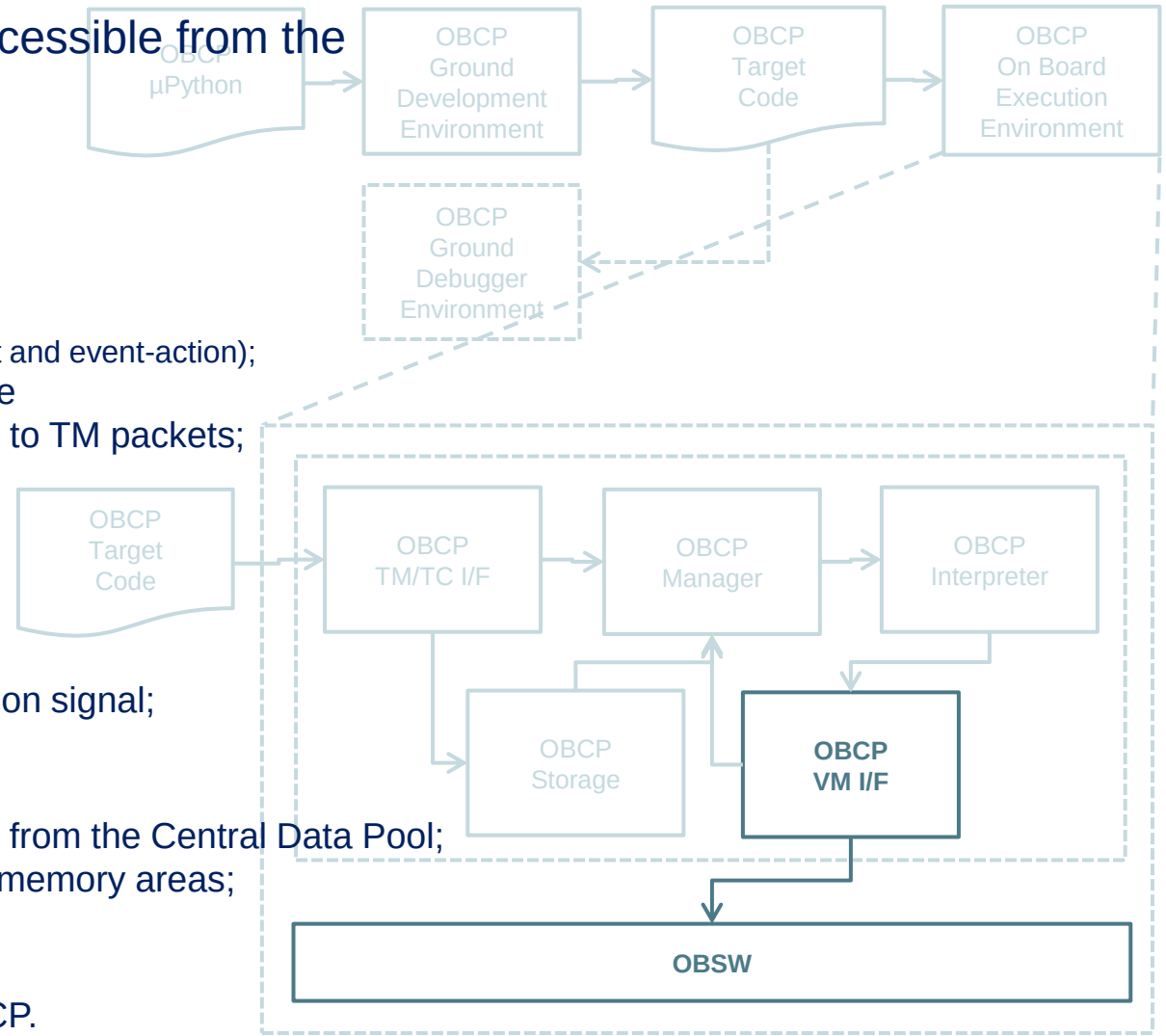
OBCP Manager

- Control
  - start/stop/suspend/resume/abort
  - step by step execution
- Monitor
  - States (databank), Transitions (events)
  - step
- Manage
  - 1 task per VM
  - State Machine
  - High and low(est) priorities
    - Low Priority: round robin
    - High Priority: queuing
- Interacts with
  - OBCP TM/TC IF
  - OBCP Storage
  - OBCP Interpreter (the VM)
- Based
  - on a hook mechanism and
  - on extension to µpython



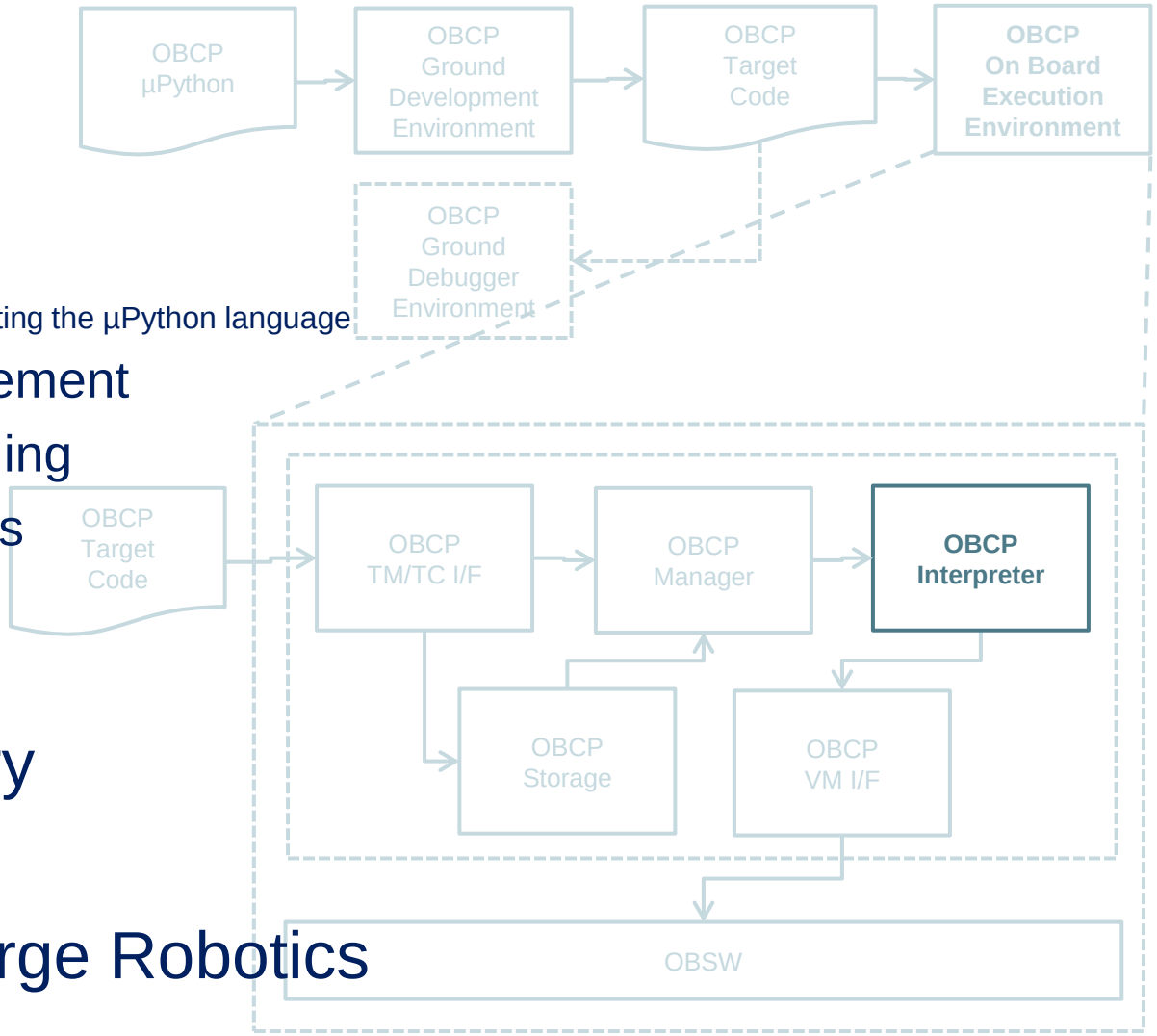
OBCP OBSW I/F

- C Extension Module Accessible from the µPython to
  - TM/TC
    - Generate TC packets
    - Generate TM packets
    - Generate Events
      (triggering the event-report and event-action);
    - Check TC acknowledge
    - Subscribe/unsubscribe to TM packets;
    - Read TM packet
  - Timing
    - Retrieve OBET;
    - Sleep
    - Wait 1Hz synchronisation signal;
  - I/O
    - Read/write parameters from the Central Data Pool;
    - Read/write authorized memory areas;
  - Control & Monitoring
    - Notify new step in OBCP.
    - Read OBCP parameters provided by ground TC;

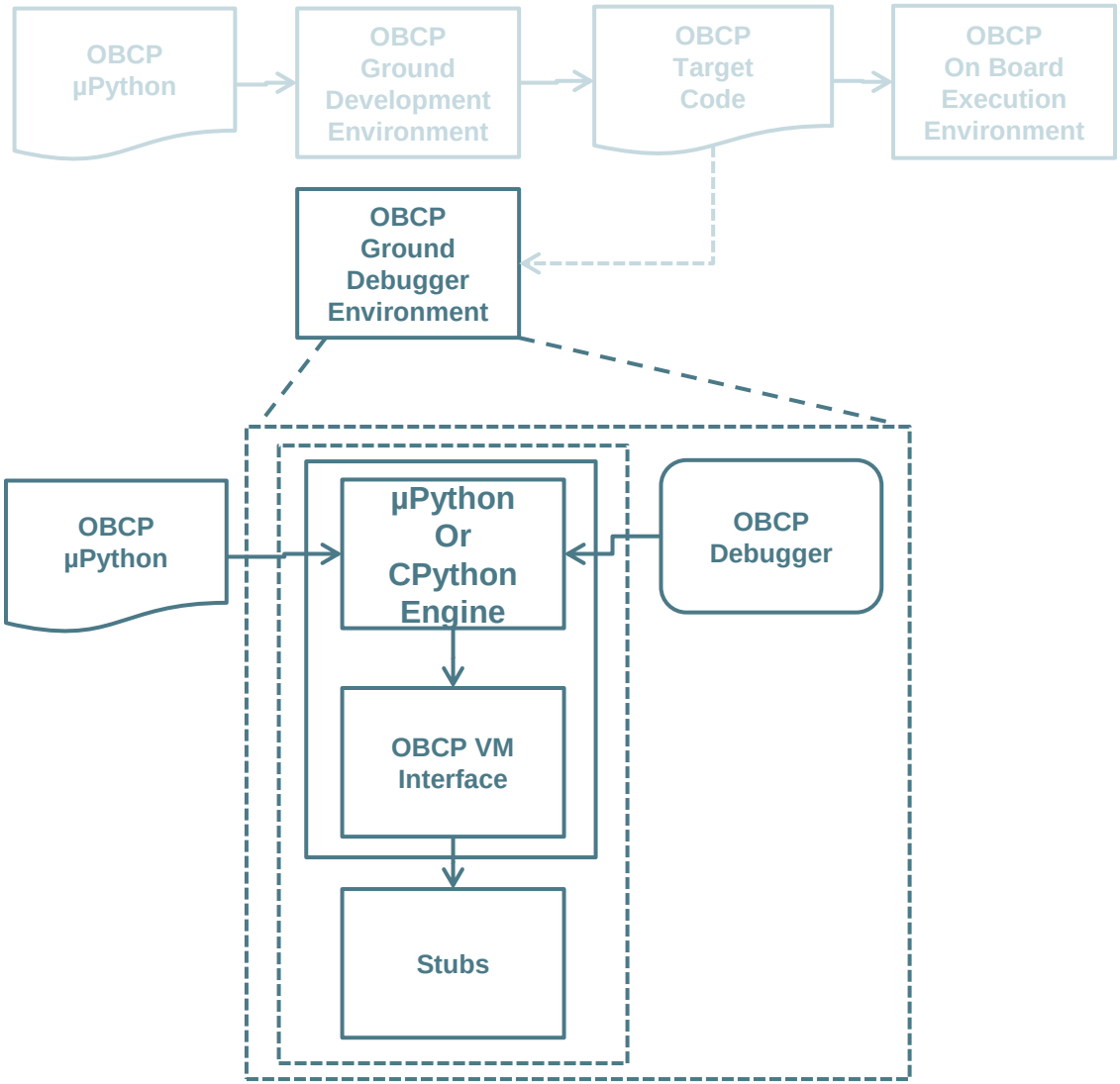


OBCP Interpreter

- MicroPython VM
  - Executor
    main loop
  - Runtime
    C types and function implementing the µPython language
    - Memory Management
    - Exception Handling
    - Built-in Functions
    - Built-in Modules
- Passive C Library
- COTS from George Robotics



OBCP Ground Debug Env



Cat D SW

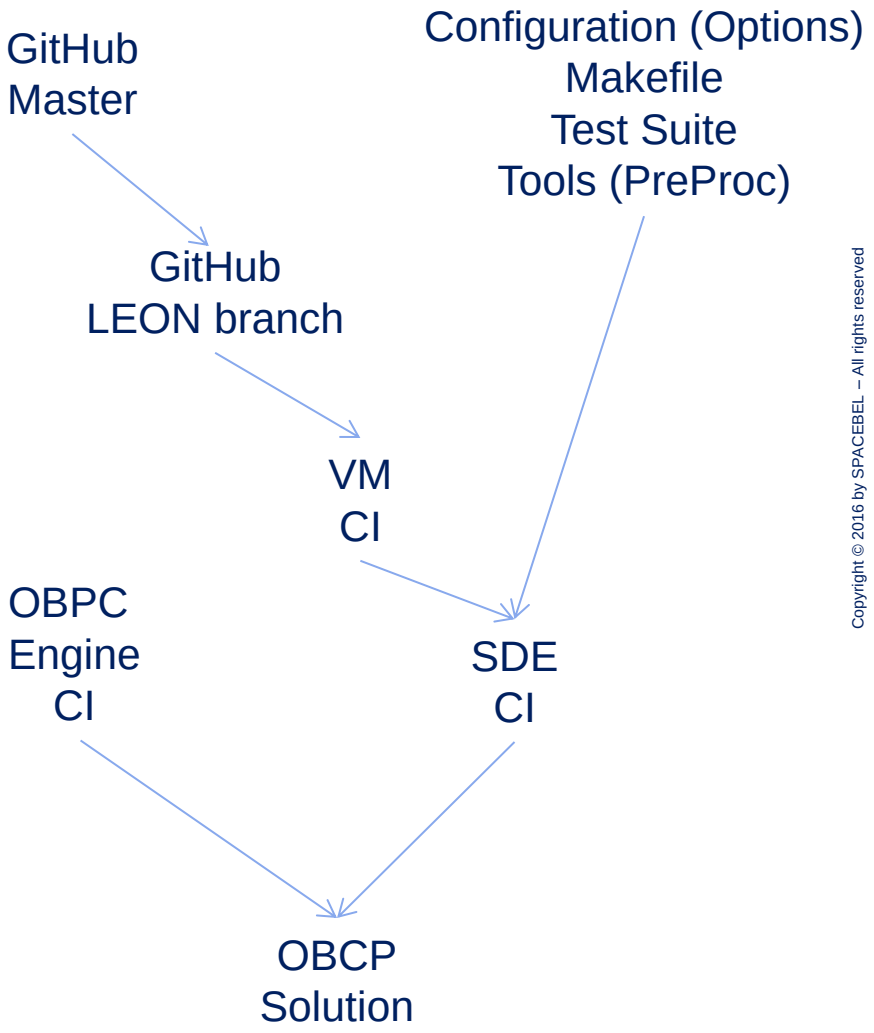
Micro Python Virtual Machine

- Configuration Control
- Techical Improvments
- Fault Containment
- Code Analysis
- Test Approach

VM Configuration Control

- Stick to the maximum extent to the master branch
  To benefit from the Open Source approach and from the usage and implicit testing by a large community

- Agreed changes (improvements or bug fixes) that have been implemented in the MicroPython github master branch are ported on the MicroPython VM for LEON branch. Dedicated improvements or bug fixes may only concern this specific branch. At key points in the project, the corresponding code is extracted and placed under configuration of the qualification project.



VM Code Analysis

- **Static Code Analysis** on µPython VM code
  - Identification of potential threats and bugs.
  - Fixing or proper justification of all discovered issues.

- **Complete Call Graph** of µPython VM code
  - Identification of all cycles.
  - Verification of the C stack usage for each cycle, to prevent memory corruption
  - (in case the remaining C stack is too low, the running OBCP is aborted).

- ….

VM Technical Improvments

- **Splitting of Stack**
  C / RTEMS stack and Python stack
- **Improvment of Memory Usage Monitoring**
  Heap, C stack and Python stack
- **Mastering of Recursion**
  Identification of recursion in C functions, whether direct and indirect, followed by rewriting to eliminate or control (C stack verification) such recursion
- **Addition of Configuration Macros**
  in order to reduce the code to the minimum needed by qualified MicroPython features
- **Addition of Entry-Point Function**
  to start execution and report details of potential uncaught exception in a dedicated C structure (for observability)
- **Handling of Python Exception**
  Python exception handling implemented with NLR mechanism relying on LEON software “flush register” trap 0x83, implemented in standard RTEMS, was removed from qualified ESA RTEMS and therefore has to be reintroduced in the MicroPython VM
- **Improvment of Memory Usage for 64bits NaN Boxing**
  exploits the unused bits of the floating-point representation of Not-A-Number in order to identify whether the object is an integer, a floating-point value, a pointer or an interned string. A key advantage of this representation lies in the fact that it does not require any Python heap allocation for floating-point numbers

VM Fault Containment

- Avoid propagation of errors stemming from an OBCP or from the VM
  - CPU Usage
    - The VMs executes in Low Priority Tasks
  - C Stack Usage
    - The VM dynamically checks that its C stack usage remains below the allocated range
    - In case of problem detected by the VM (in recursive function), the OBCP execution is stopped with an exception.
  - µPython Stack and Heap Usage
    - memory required for the MicroPython OBCP stack or heap is allocated in a dedicated memory area
    - In case of overflow detected by the VM (at allocation), the OBCP execution is stopped with an exception.
  - Floating Point Errors
    - Traps cannot be avoided.
    - In case of trap in the context of a VM, the OBCP execution is stopped with an exception
  - Interface with the OBSW
    - Check of arguments.
    - Restricted access to OBSW resources (memory, data pool)

VM Testing

- **Exhaustive end to end tests**
  - All relevant tests defined in the frame of the
    - MicroPython VM master branch and
    - MicroPython VM for LEON branch;
  - C Python standard test suite for the core Python language (math functions, list, dictionary, etc.);
  - Additional tests for improvement of the code coverage
    - Particular effort on µPython exception handling code

- **Outcomes**
  - 100% statement coverage (with justification of non covered code)
  - 621 test scripts
  - 56 bug fixes
  - 48 minor deviations or limitations wrt Cpython 3.4 documented

Conclusion

- **Two Reusable Components**
  can be used together or separately

  - **OBCP Engine**
    - For PF or PL
    - PUS Standard; OBCP Standard, ESA Ops requirements
    - Standard ECSS documentation

  - **VM Virtual Machine**
    - Python 3.4 Standard,
    - Robustness Improvments
    - Qualification Test Suite
    - Standard ECSS documentation

- Intended for flight on Euclid

Property Rights

- The OBCP Engine is developed by Spacebel. It is IPR of Spacebel S.A.
- The MicroPython VM is developed by George Robotics. It is IPR of George Robotics LTD. It is made available under the MIT Open Source license.
- The porting on the LEON and the RTEMS has been funded by ESA. It is IPR of George Robotics LTD. The code and documentation are distributed under ESA Community License type 3, permissive.

Thank you for your attention