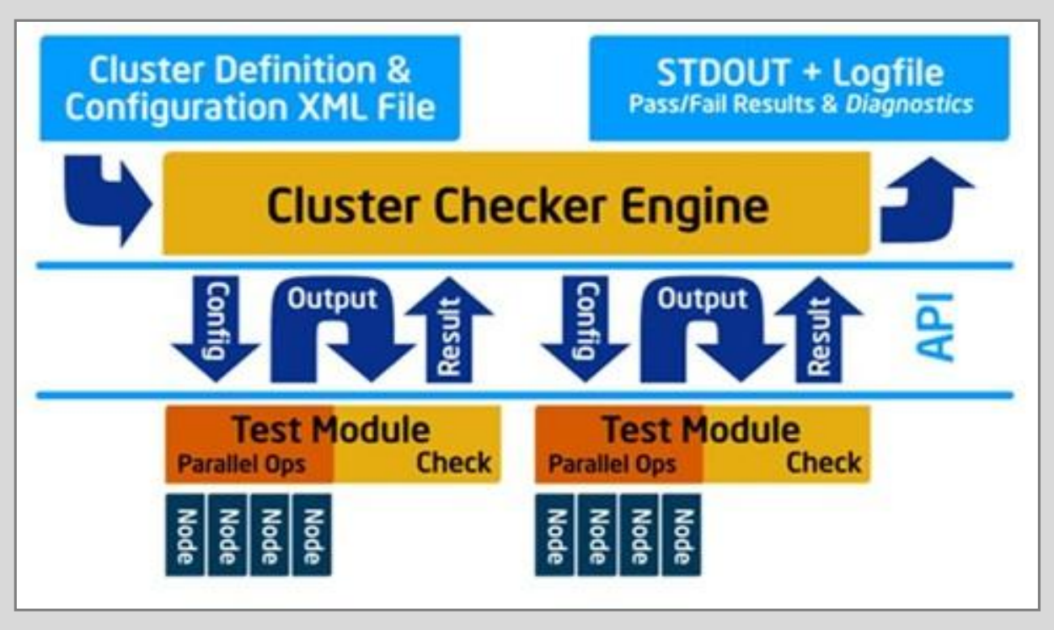
Intel® Cluster Checker Multi-Language API

Mateo Guzmán, Matias Cabral, Cesar Martinez Spessot
Argentina Software Development Center
Intel Argentina

INTRODUCTION

Intel® Cluster Checker (CLCK) is a tool that verifies the configuration, measures performance and validates compliance of the Intel® Cluster Ready Specification in Linux-based clusters. With more than 100 tests-modules CLCK performs a wide range of cluster evaluations. It assesses firmware, kernel, storage, and network settings and also stresses compute nodes and network using multiple benchmarks.
Customized checks can be created and integrated to the execution of the tool though an API provided for that purpose. However, this test modules should be written in Perl* programming language.
This work proposes a new component of CLCK called Intel® Cluster Checker Multilanguage API (CLCK_Multilang_API), which allows users and developers to write their tailored test modules in arbitrary programming languages and add them to the execution of the tool. Two example languages, Perl* and Python were used as references for this project.
The solution includes some mixing of technologies such as SWIG, native C APIs of each language, and Java Script Object Notation (JSON) .

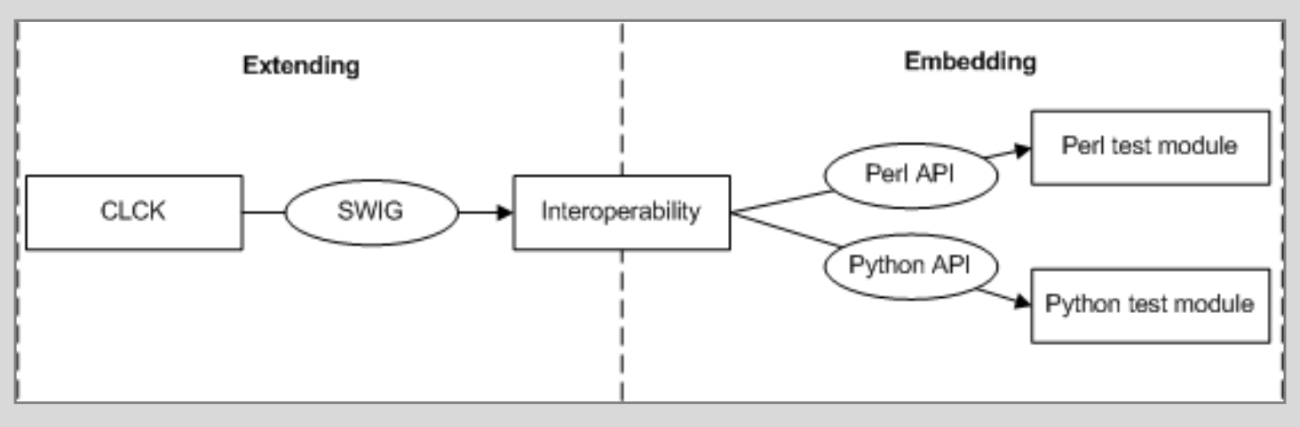
Intel® Cluster Checker Architecture

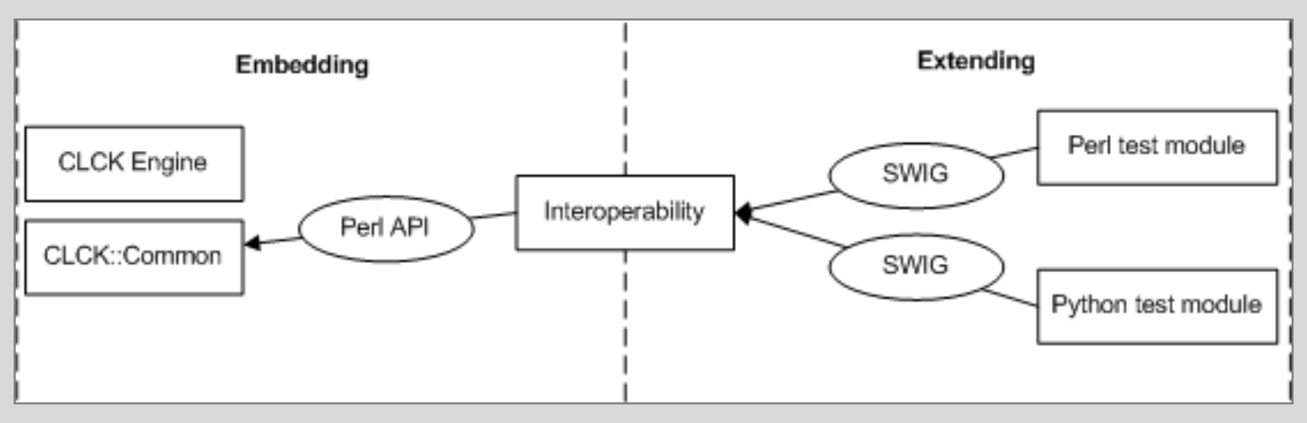


Intel® Cluster Checker consists of an engine (CLCK Engine) and a set of test modules. The engine accepts as input the list of cluster nodes and a configuration file. For each test module it gathers the information needed and controls it against a specified criteria to define if the test is successful or not

Test modules communication

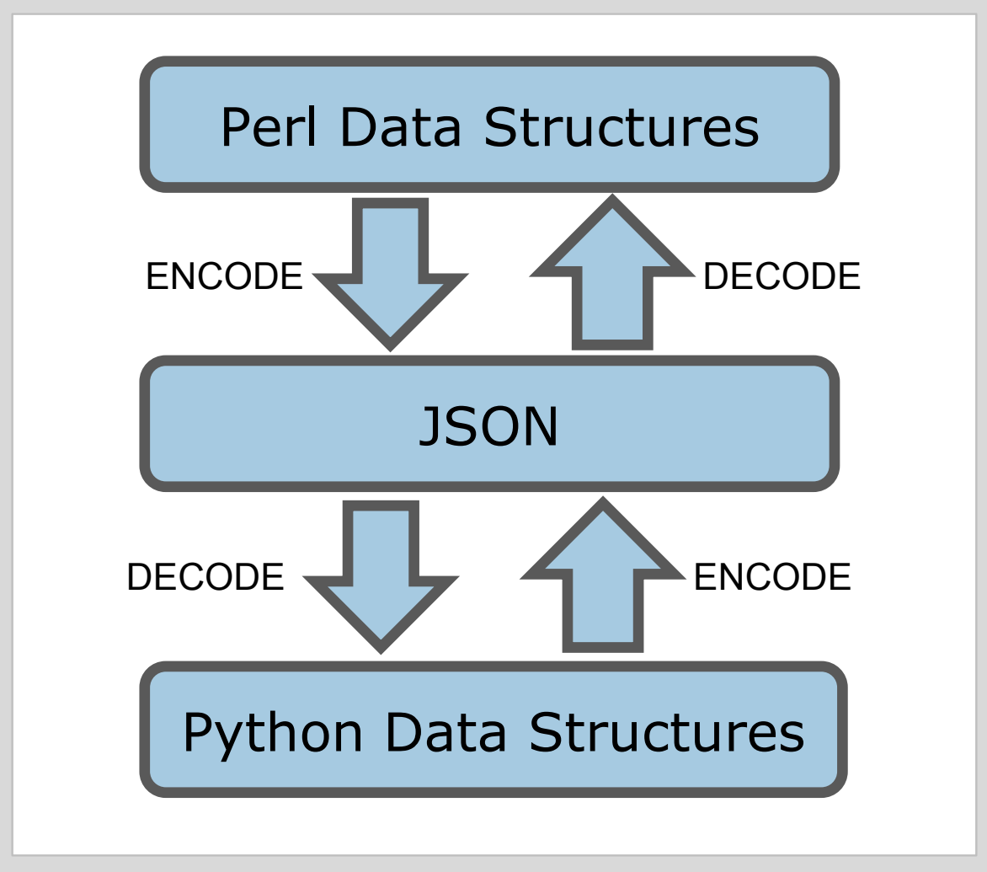
A C library "interoperability" is used as intermediate to communicate CLCK Engine (Perl*) and test modules (Perl or Python).
SWIG allows to execute C code from Perl or Python.
Perl API and Python API are used to execute test modules routines from the C library.





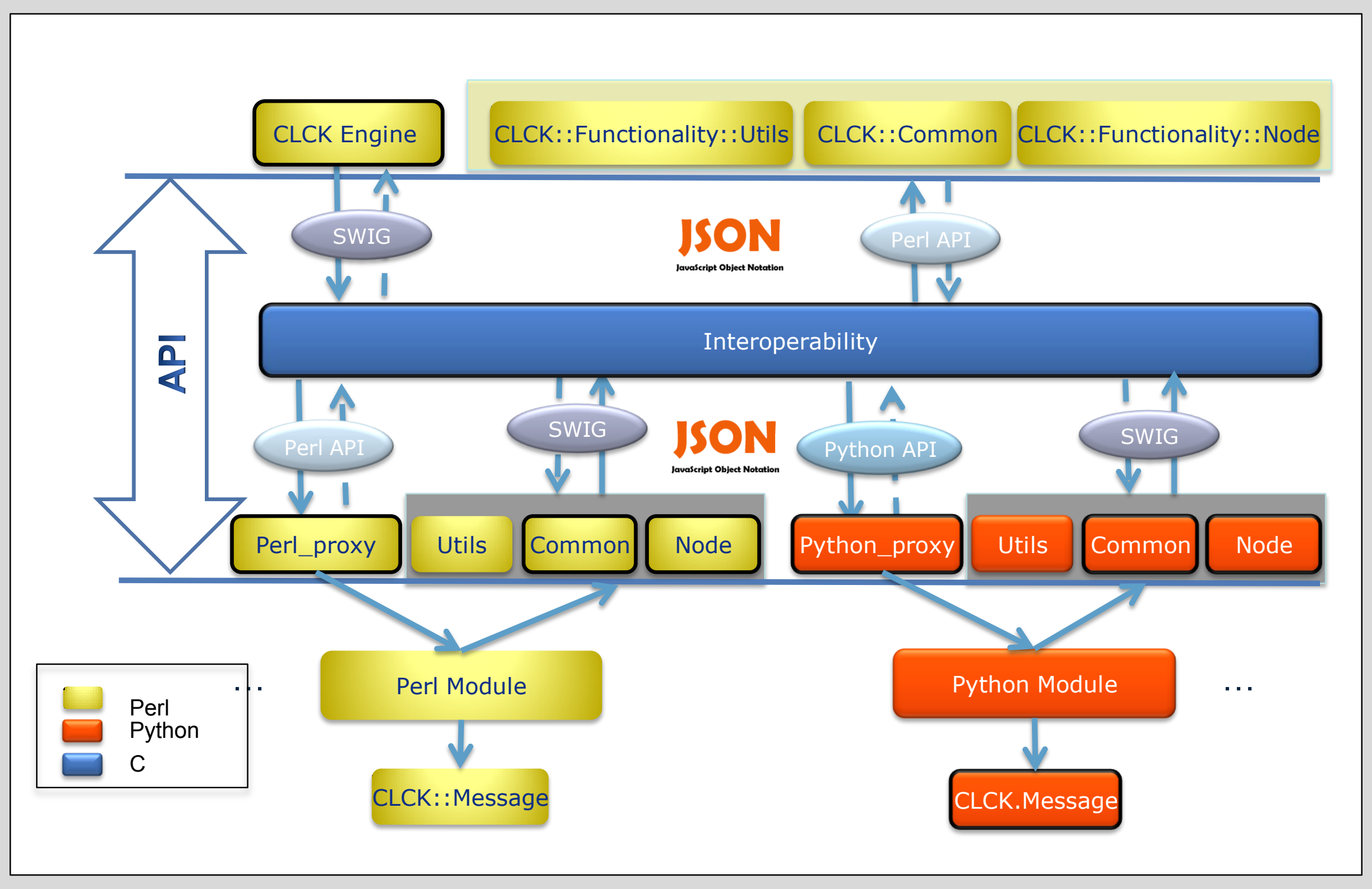
Data messaging

Data needs to be transmitted from CLCK Engine to the test modules and vice verse.
Since each language have different data types and/or treat the data types in a different manner, a mapping between them must be defined.
Java Script Object Notation (JSON) is used as data interchange format.



Intel® Cluster Checker Multilanguage API

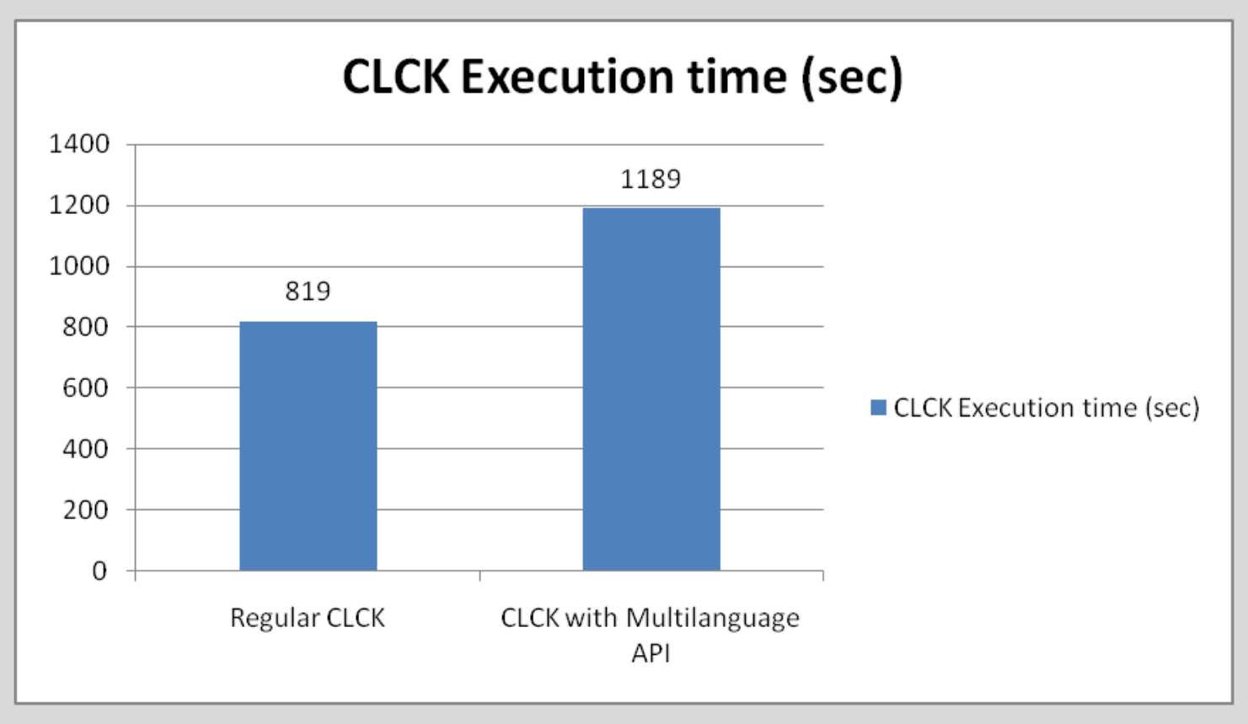
When CLCK Engine needs to call a test-module routine, encodes the parameters (JSON) and calls the functionality provided by Interoperability C Library (ICL) exposed by SWIG. A proxy module was created for each language to encapsulate the JSON encoding / decoding and dynamically call the required test module´s routine .



Three new modules (Utils, Common, Node) were created for each language in order to work as a façade of the original modules. They provide the "Utils", "Common" and "Node" functionality encapsulating the encoding/decoding process.
Finally a Message module was created for each language which contains the status of the test-modules to be returned to the CLCK Engine.

Performance Analysis

The solution introduces an overhead in the execution of the tool mainly caused by the data structures conversion which is done twice for each call to a test module.
This overhead increments the execution time by 1.45 times compared with the regular CLCK execution. This figure is the result of executing multiple times a test module with the current version of the tool and using the new API.



CONCLUSIONS

The solution proves that it is possible to write test modules in different programming languages and integrate them to the execution of Intel® Cluster Checker. For simplicity matters, Python and Perl* where used as reference, however, the architecture allows adding support for new languages with little effort.
Although some performance is lost due to the test-modules communication and data structures conversion, the flexibility improvement is worth this loss of performance. This API can be implemented as an optional feature of the tool, not affecting performance unless the user requires to use it.
The research will continue adding support for new languages and reducing the execution time.

AKNOWLEDGEMENTS

Thanks to the Argentina Cluster Engineering Team (Intel) for giving the opportunity and space for this research project.