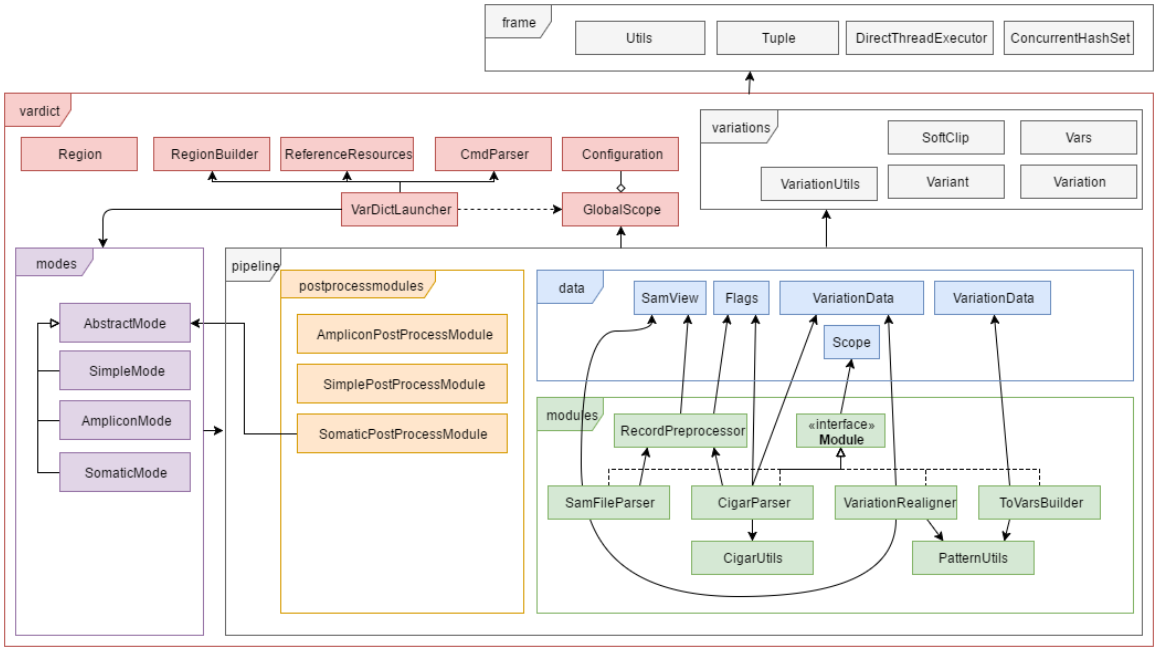
VarDict 2.0 Release note

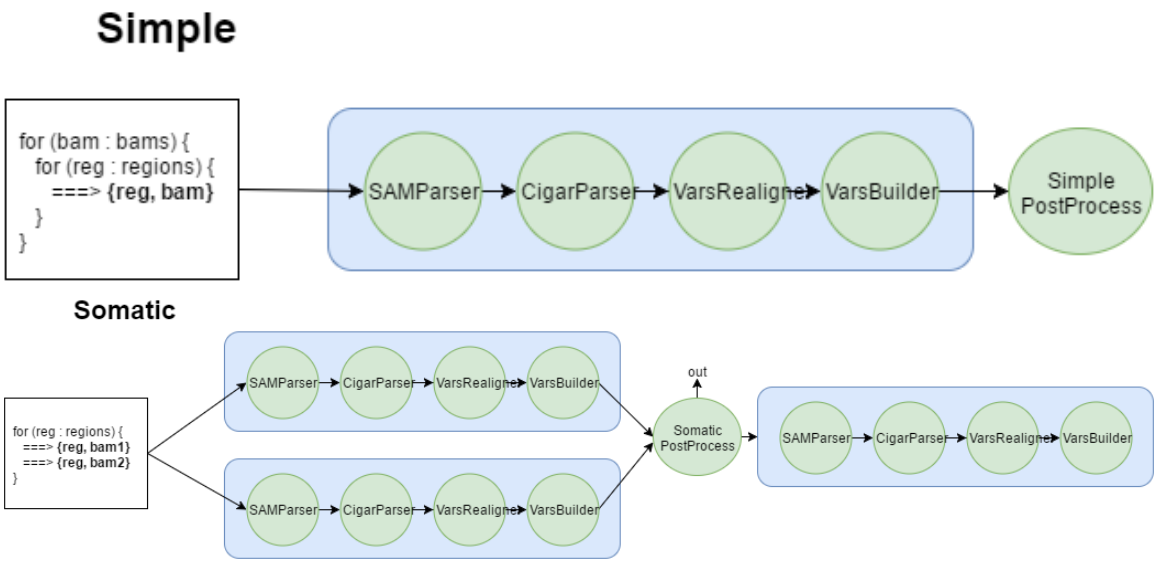
Refactoring

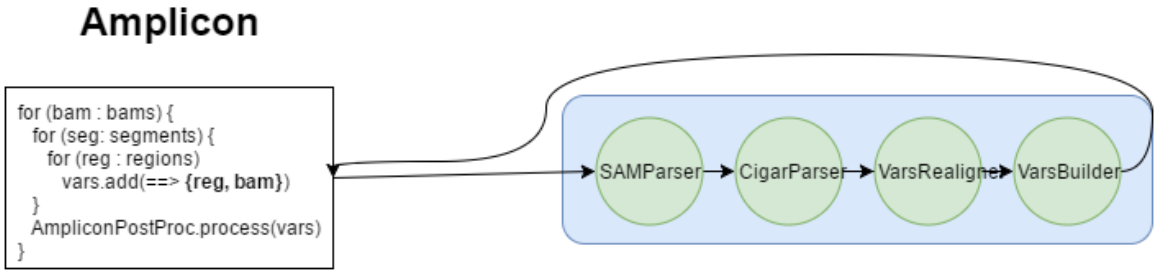
VarDictJava inherited Perl version static procedural structure. In this release huge refactoring was provided. At the top level refactoring includes following steps:

1. Scopes decomposition
2. Method decomposition
3. Data aggregation
4. Global scope extraction
5. Entities extraction
6. Data flow rerouting
7. Extraction of Modules entities
8. Pipeline organisation
9. Extraction of Modes entities



As result the single class code was decomposed into 30+ classes, organised into packages hierarchy and pipeline data flow pattern. All those structure improvements produce into flexibility of the variant caller modification, extension and testing, very good potential of functional parallelism because of pipeline structure.





Regression testing has been used for verification that VarDict still performs correctly after changes . It includes tests for simple, somatic, amplicon and somatic + amplicon in single-threaded and multi-threaded modes. We found that regression tests cover the VarDict to more than 80%. Plus basic unit tests cover 12% of methods.

Performance report shows no regression. (Performance Regressions Refactoring Report)

BAM Caching

Also, custom htsjdk version was applied in the VarDict release 2.0, it includes new cache feature for bam-files. This feature can be used in singlethread or multithread mode.

Added options

For enabling bam caching feature append the VARDICT_OPT parameter with:

-Dsamjdk.bam_caching=true

-Dsamjdk.cache_size=[value] (default value = 128, measured in MB).

The GitHub branch in EPAM`s HTSJDK fork repository is https://github.com/EpamLifeSciencesTeam/htsjdk/tree/epam-ls_BAM_cache.