IRIS-HEP

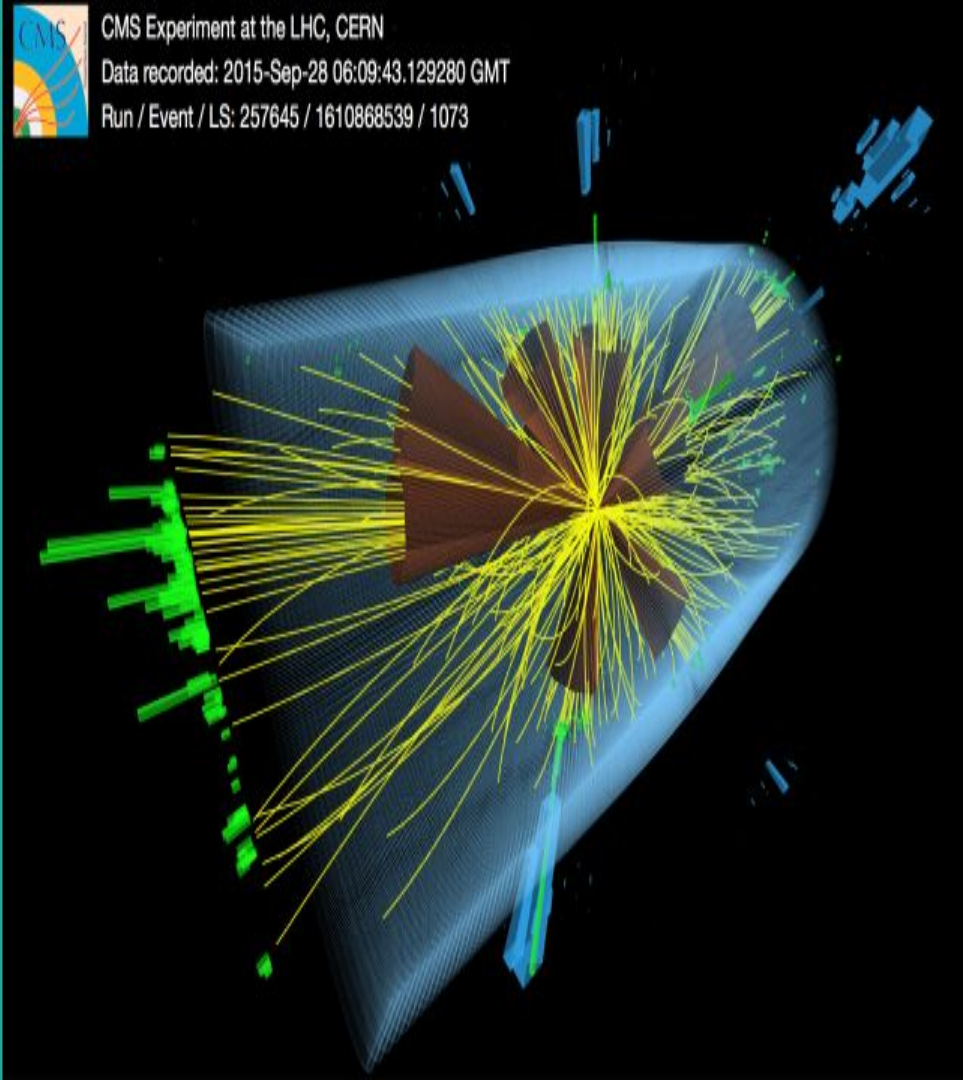
Developing C++ modules support in CMSSW and Boost

Mentors: Dr David Lange, Dr Vassil Vassilev
Student: Purva Chaudhari

Institute for Research & Innovation
in Software for High Energy Physics

What is CMSSW?

- ➢ CMS is a particle detector that is designed to see a wide range of particles and phenomena produced in high-energy collisions in the LHC.
- ➢ Many of CMS Software components (CMSSW) are hosted on Github.

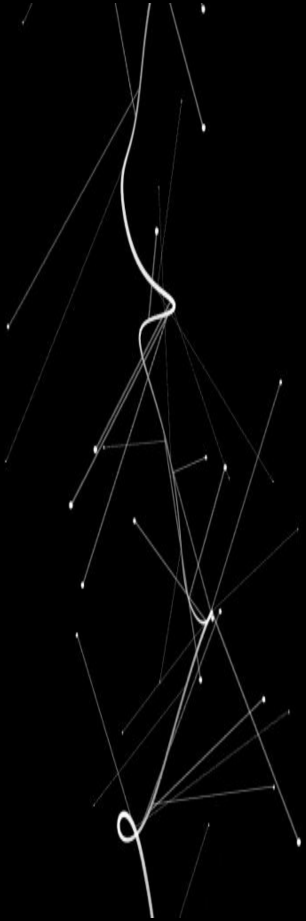

CMS Experiment at the LHC, CERN
Data recorded: 2015-Sep-28 06:09:43.129280 GMT
Run / Event / LS: 257645 / 1610868539 / 1073

About Project

Reduce Boost dependencies in CMSSW

- Reducing boost dependencies helps us create more lightweight boost clang modules for upcoming c++20.
- This also reduces the amount of headers that we need to work on to be able to use c++20 clang modules.

Previous work done

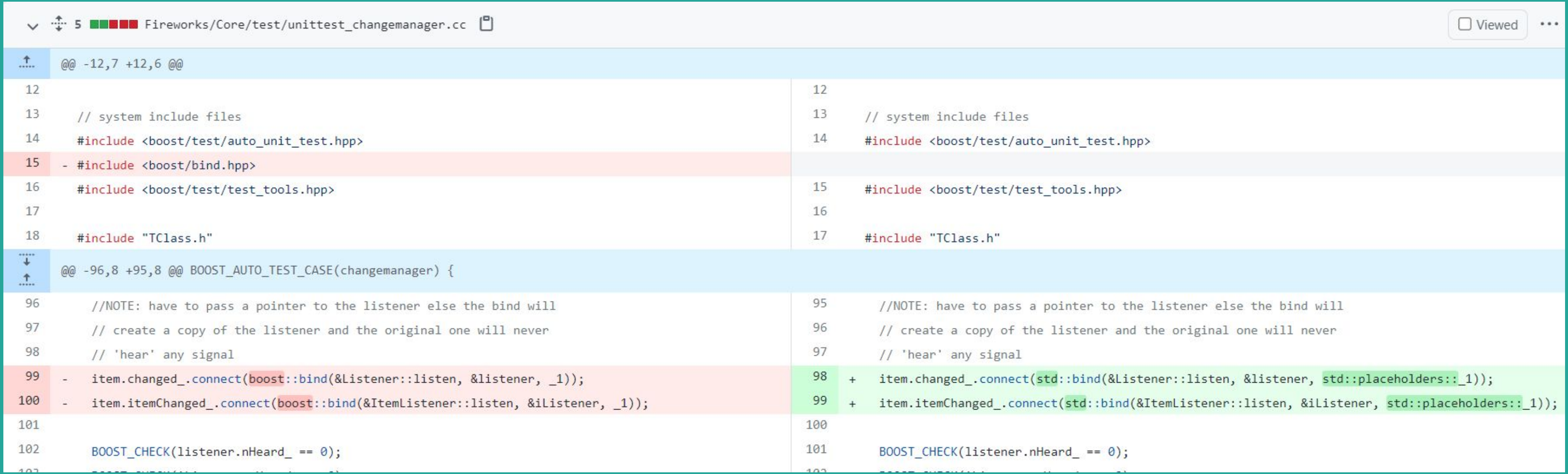


- boost::array → std::array
- boost::unordered_map → std::unordered_map
- boost::unordered_set → std::unordered_set
- boost::function → std::function
- boost::random → stl
- boost::hash → std::hash
- boost::filesystem → std::filesystem
- boost::mutex → std::mutex
- boost::variant →std::variant

My current contributions

- ❑ boost::bind →std::bind
- ❑ boost::python → pybind11
- ❑ boost::lexical_cast → corresponding stl casting methods
- ❑ Tried boost::regex → std::regex (could not due to lower performance of std::regex)
- ❑ Checked out probable replacements methods for boost/string/algorithms

boost::bind → std::bind


5 Fireworks/Core/test/unittest_changemanager.cc

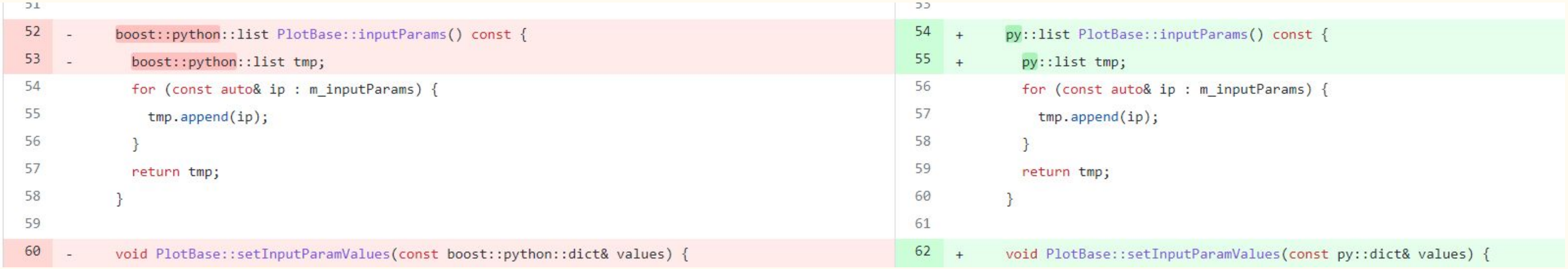
Viewed

```
@@ -12,7 +12,6 @@

  // system include files
  #include <boost/test/auto_unit_test.hpp>
- #include <boost/bind.hpp>
  #include <boost/test/test_tools.hpp>

  #include "TClass.h"
```

```
@@ -96,8 +95,8 @@ BOOST_AUTO_TEST_CASE(changemanager) {
    //NOTE: have to pass a pointer to the listener else the bind will
    // create a copy of the listener and the original one will never
    // 'hear' any signal
-   item.changed_.connect(boost::bind(&Listener::listen, &listener, _1));
-   item.itemChanged_.connect(boost::bind(&ItemListener::listen, &iListener, _1));
+   item.changed_.connect(std::bind(&Listener::listen, &listener, std::placeholders::_1));
+   item.itemChanged_.connect(std::bind(&ItemListener::listen, &iListener, std::placeholders::_1));

    BOOST_CHECK(listener.nHeard_ == 0);
```

Boost::python → pybind11

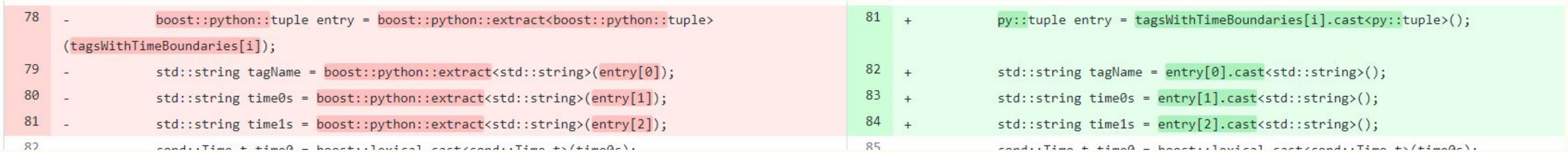
1. Boost::python::list/dict → pybind11::list/dict



```diff
-  boost::python::list PlotBase::inputParams() const {
-    boost::python::list tmp;
     for (const auto& ip : m_inputParams) {
       tmp.append(ip);
     }
     return tmp;
   }

-  void PlotBase::setInputParamValues(const boost::python::dict& values) {
```

```diff
+  py::list PlotBase::inputParams() const {
+    py::list tmp;
     for (const auto& ip : m_inputParams) {
       tmp.append(ip);
     }
     return tmp;
   }

+  void PlotBase::setInputParamValues(const py::dict& values) {
```

2. Boost::python::list/dict → object.cast

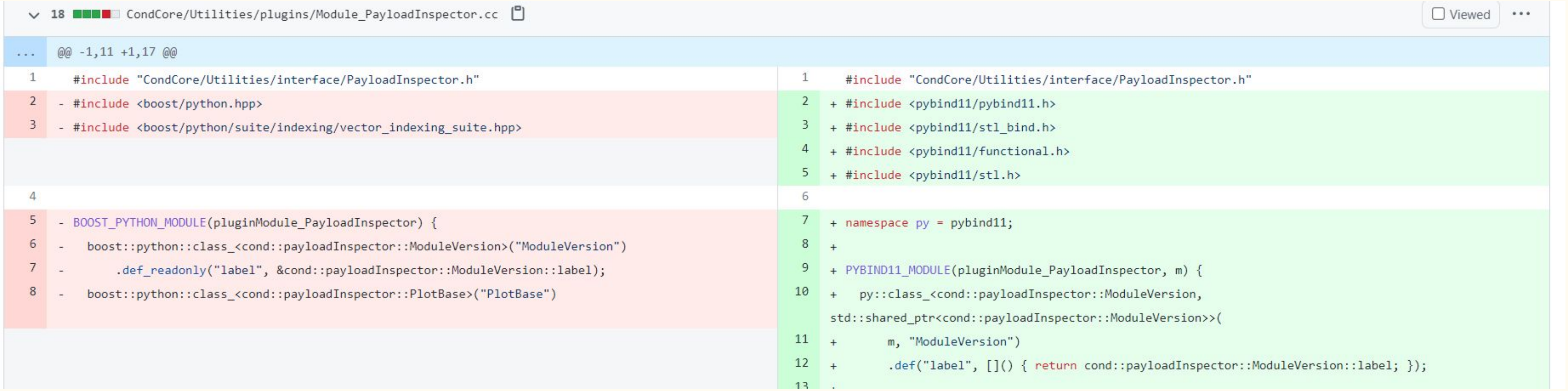


```diff
-          boost::python::tuple entry = boost::python::extract<boost::python::tuple>
(tagsWithTimeBoundaries[i]);
-          std::string tagName = boost::python::extract<std::string>(entry[0]);
-          std::string time0s = boost::python::extract<std::string>(entry[1]);
-          std::string time1s = boost::python::extract<std::string>(entry[2]);
           cond::Time_t time0 = boost::lexical_cast<cond::Time_t>(time0s);
```

```diff
+          py::tuple entry = tagsWithTimeBoundaries[i].cast<py::tuple>();

+          std::string tagName = entry[0].cast<std::string>();
+          std::string time0s = entry[1].cast<std::string>();
+          std::string time1s = entry[2].cast<std::string>();
           cond::Time_t time0 = boost::lexical_cast<cond::Time_t>(time0s);
```

Boost::python → pybind11

3. Boost python module → pybind11 module



18 CondCore/Utilities/plugins/Module_PayloadInspector.cc

Viewed

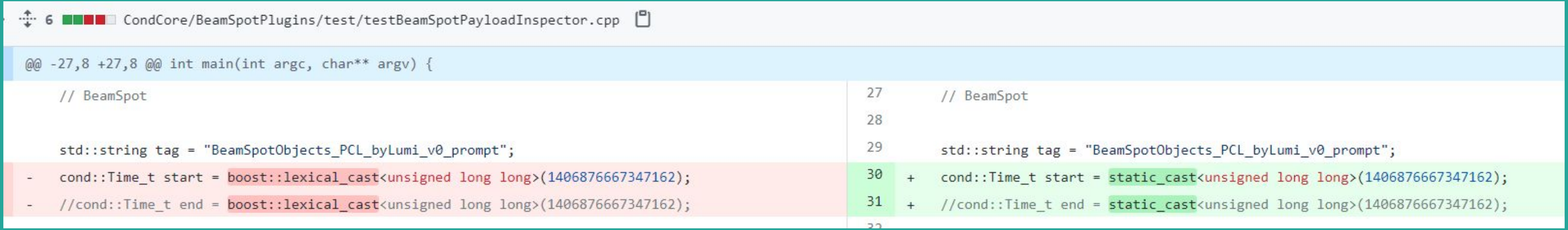
```
@@ -1,11 +1,17 @@
```

```
  #include "CondCore/Utilities/interface/PayloadInspector.h"
- #include <boost/python.hpp>
- #include <boost/python/suite/indexing/vector_indexing_suite.hpp>

- BOOST_PYTHON_MODULE(pluginModule_PayloadInspector) {
-   boost::python::class_<cond::payloadInspector::ModuleVersion>("ModuleVersion")
-       .def_readonly("label", &cond::payloadInspector::ModuleVersion::label);
-   boost::python::class_<cond::payloadInspector::PlotBase>("PlotBase")
```

```
  #include "CondCore/Utilities/interface/PayloadInspector.h"
+ #include <pybind11/pybind11.h>
+ #include <pybind11/stl_bind.h>
+ #include <pybind11/functional.h>
+ #include <pybind11/stl.h>

+ namespace py = pybind11;
+
+ PYBIND11_MODULE(pluginModule_PayloadInspector, m) {
+   py::class_<cond::payloadInspector::ModuleVersion,
std::shared_ptr<cond::payloadInspector::ModuleVersion>>(
+       m, "ModuleVersion")
+       .def("label", []() { return cond::payloadInspector::ModuleVersion::label; });
```

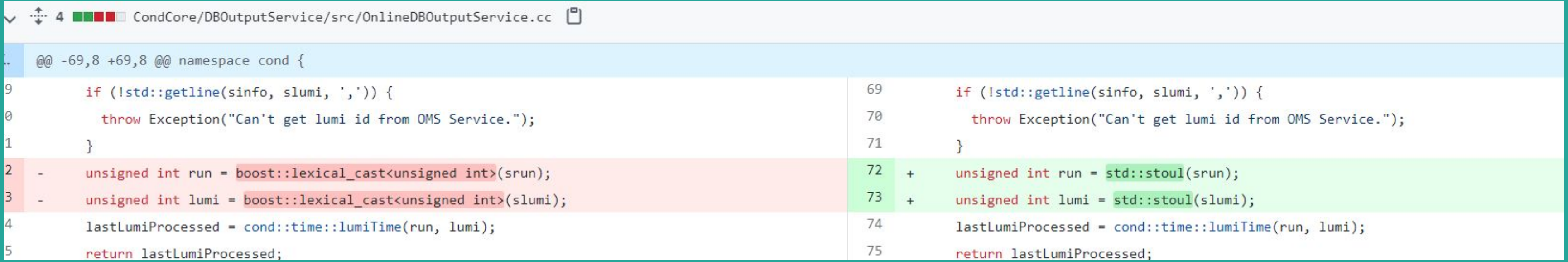
Boost::lexical_cast → stl casting

1. Upcasting/downcasting → static_cast



```
6 CondCore/BeamSpotPlugins/test/testBeamSpotPayloadInspector.cpp
@@ -27,8 +27,8 @@ int main(int argc, char** argv) {
     // BeamSpot                                                              // BeamSpot

     std::string tag = "BeamSpotObjects_PCL_byLumi_v0_prompt";                std::string tag = "BeamSpotObjects_PCL_byLumi_v0_prompt";
-    cond::Time_t start = boost::lexical_cast<unsigned long long>(1406876667347162);   +    cond::Time_t start = static_cast<unsigned long long>(1406876667347162);
-    //cond::Time_t end = boost::lexical_cast<unsigned long long>(1406876667347162);   +    //cond::Time_t end = static_cast<unsigned long long>(1406876667347162);
```

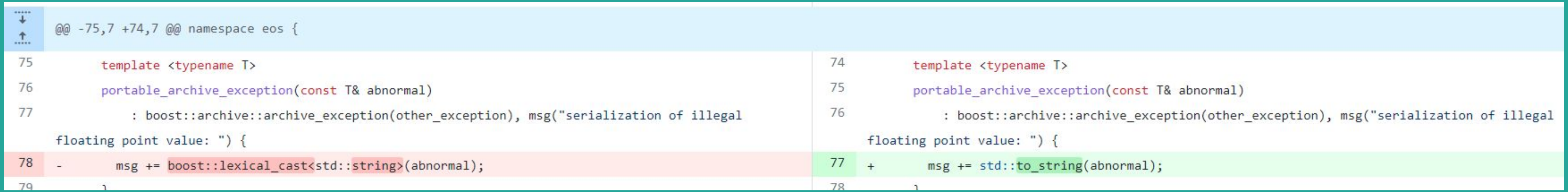
2. From string to int/double/unsigned long long → std::stoi/ std::stod/ std::stoull



```
4 CondCore/DBOutputService/src/OnlineDBOutputService.cc
@@ -69,8 +69,8 @@ namespace cond {
       if (!std::getline(sinfo, slumi, ',')) {                              if (!std::getline(sinfo, slumi, ',')) {
         throw Exception("Can't get lumi id from OMS Service.");              throw Exception("Can't get lumi id from OMS Service.");
       }                                                                    }
-      unsigned int run = boost::lexical_cast<unsigned int>(srun);      +   unsigned int run = std::stoul(srun);
-      unsigned int lumi = boost::lexical_cast<unsigned int>(slumi);    +   unsigned int lumi = std::stoul(slumi);
       lastLumiProcessed = cond::time::lumiTime(run, lumi);                 lastLumiProcessed = cond::time::lumiTime(run, lumi);
       return lastLumiProcessed;                                            return lastLumiProcessed;
```

Boost::lexical_cast → stl casting

3. From int/ unsigned/ template to string → std::to_string



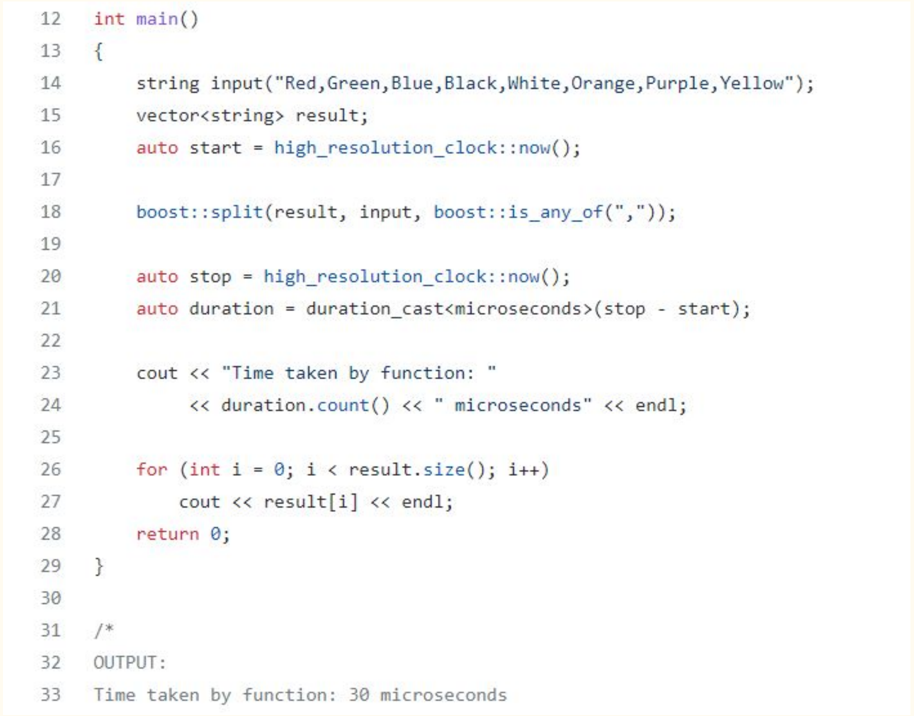
```
@@ -75,7 +74,7 @@ namespace eos {
```

```
        template <typename T>
        portable_archive_exception(const T& abnormal)
            : boost::archive::archive_exception(other_exception), msg("serialization of illegal
floating point value: ") {
-           msg += boost::lexical_cast<std::string>(abnormal);
```

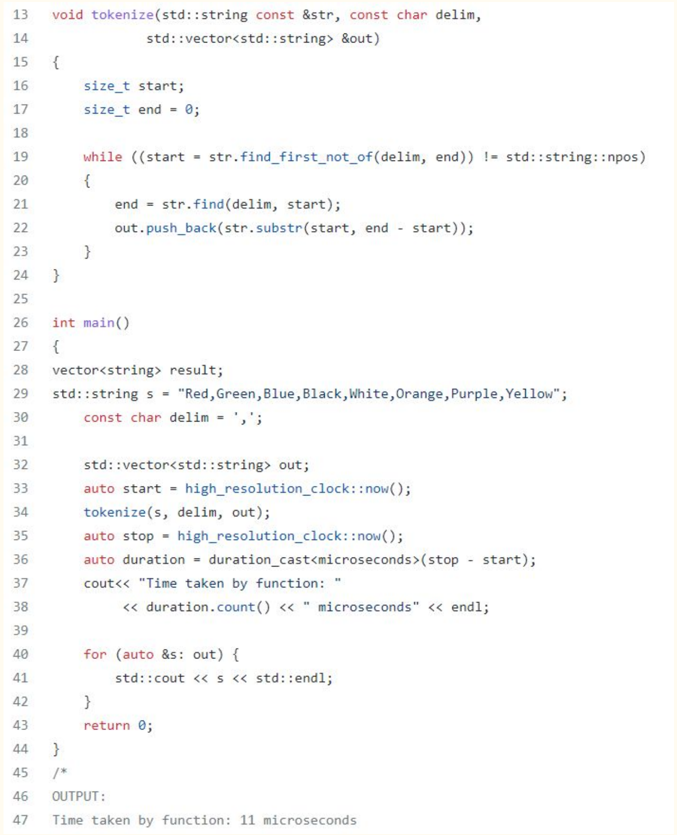
```
        template <typename T>
        portable_archive_exception(const T& abnormal)
            : boost::archive::archive_exception(other_exception), msg("serialization of illegal
floating point value: ") {
+           msg += std::to_string(abnormal);
```

Probable approach to eliminate boost algorithms

1. boost::algorithm::split



```cpp
int main()
{
    string input("Red,Green,Blue,Black,White,Orange,Purple,Yellow");
    vector<string> result;
    auto start = high_resolution_clock::now();

    boost::split(result, input, boost::is_any_of(","));

    auto stop = high_resolution_clock::now();
    auto duration = duration_cast<microseconds>(stop - start);

    cout << "Time taken by function: "
         << duration.count() << " microseconds" << endl;

    for (int i = 0; i < result.size(); i++)
        cout << result[i] << endl;
    return 0;
}

/*
OUTPUT:
Time taken by function: 30 microseconds
```



```cpp
void tokenize(std::string const &str, const char delim,
            std::vector<std::string> &out)
{
    size_t start;
    size_t end = 0;

    while ((start = str.find_first_not_of(delim, end)) != std::string::npos)
    {
        end = str.find(delim, start);
        out.push_back(str.substr(start, end - start));
    }
}

int main()
{
vector<string> result;
std::string s = "Red,Green,Blue,Black,White,Orange,Purple,Yellow";
    const char delim = ',';

    std::vector<std::string> out;
    auto start = high_resolution_clock::now();
    tokenize(s, delim, out);
    auto stop = high_resolution_clock::now();
    auto duration = duration_cast<microseconds>(stop - start);
    cout<< "Time taken by function: "
         << duration.count() << " microseconds" << endl;

    for (auto &s: out) {
        std::cout << s << std::endl;
    }
    return 0;
}
/*
OUTPUT:
Time taken by function: 11 microseconds
```

Existing boost dependencies

- boost::regex → std::regex has low performance
- boost::format →std::format c++20.
- boost::posix → std::chrono can be a probable approach
- boost::serialization
- boost::iterator
- boost::spirit
- boost::algorithm

Thank You