# CppInterOp: Advancing Interactive C++ for High Energy Physics

Aaron Jomy
Mentored by : Dr Vassil Vassilev (CERN/Princeton)
Dr Wim Lavrijsen (LBNL)

# BRIEF INTRO

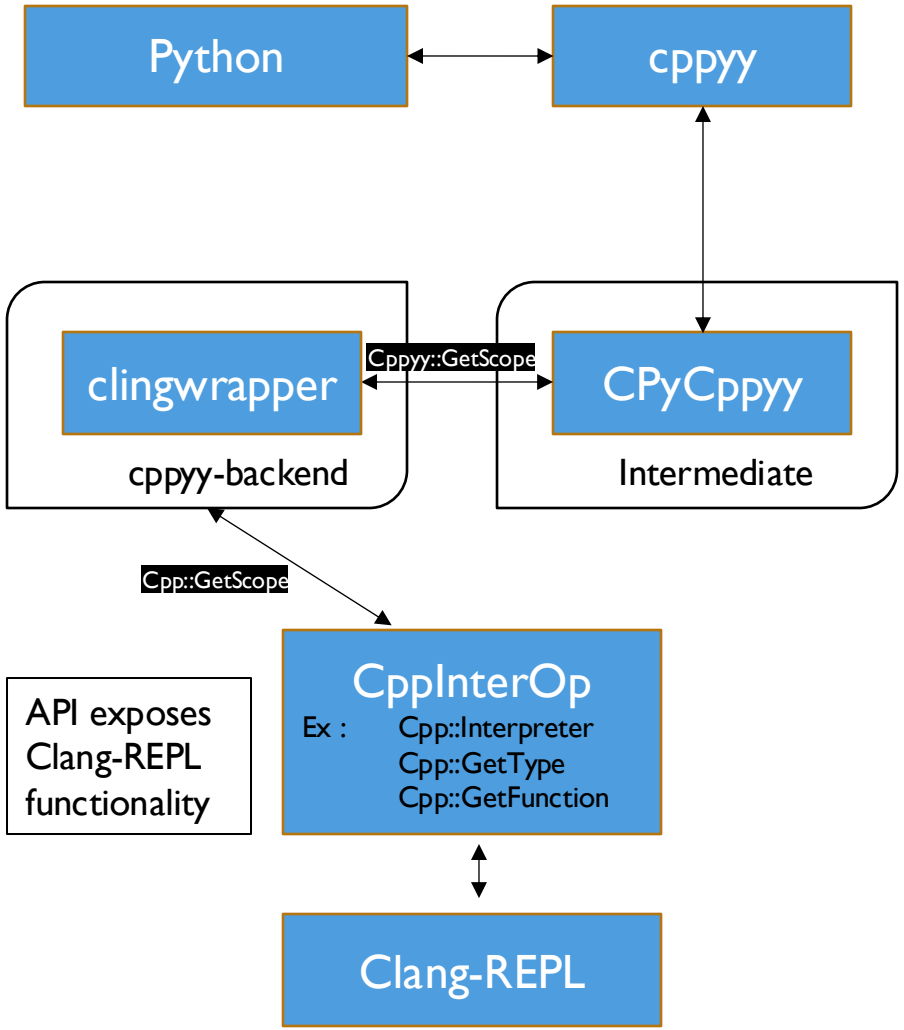
- Cppyy
  An automatic C++ - Python runtime bindings generator which provides the user with the C++ feature set
- Cling
  An interactive C++ interpreter, built on LLVM and Clang
  Used in Cppyy's(upstream) backend
- Clang-REPL
  A generalization of Cling in LLVM - supports interactive programming for C++ in a read-evaluate-print-loop (REPL) style

cppyy

LLVM
COMPILER INFRASTRUCTURE

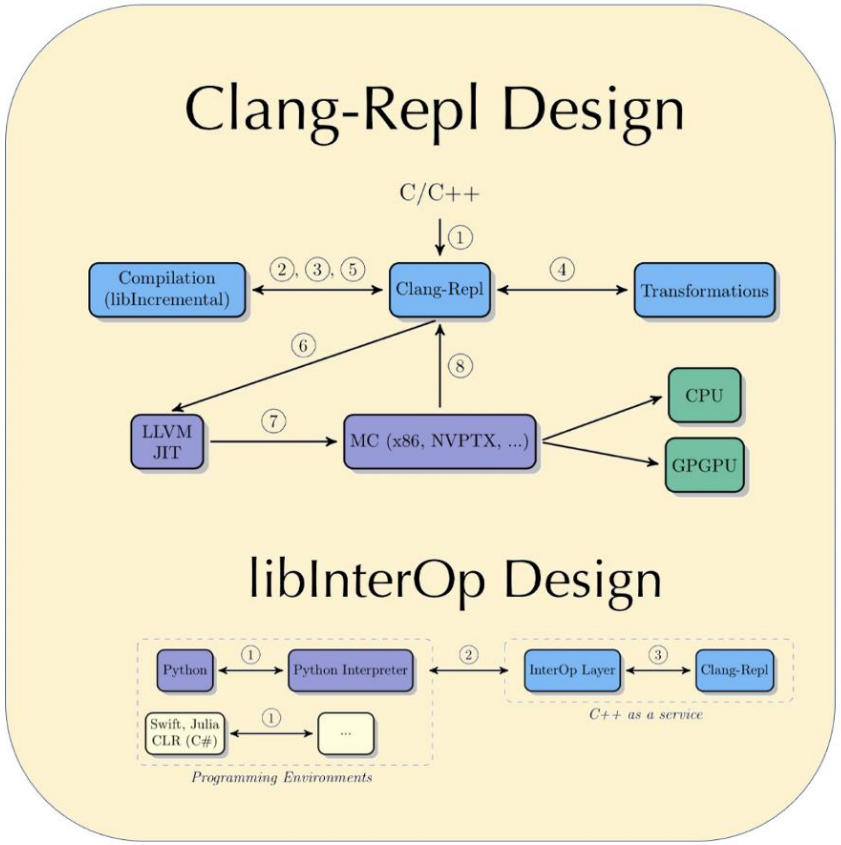
# COMPILER RESEARCH FORKS

- CppInterOp allows Cppyy to use LLVM's Clang-REPL as a runtime compiler
- This avoids the string parsing logic used with the current Cling based cppyy-backend
- Opens up more C++ features that can be used by Cppyy users
- Lower dependencies leads to performance improvement
- CppInterOp unit tests verify the API that is used for proxy creation, lookups, function reflection, etc



# CPPINTEROP

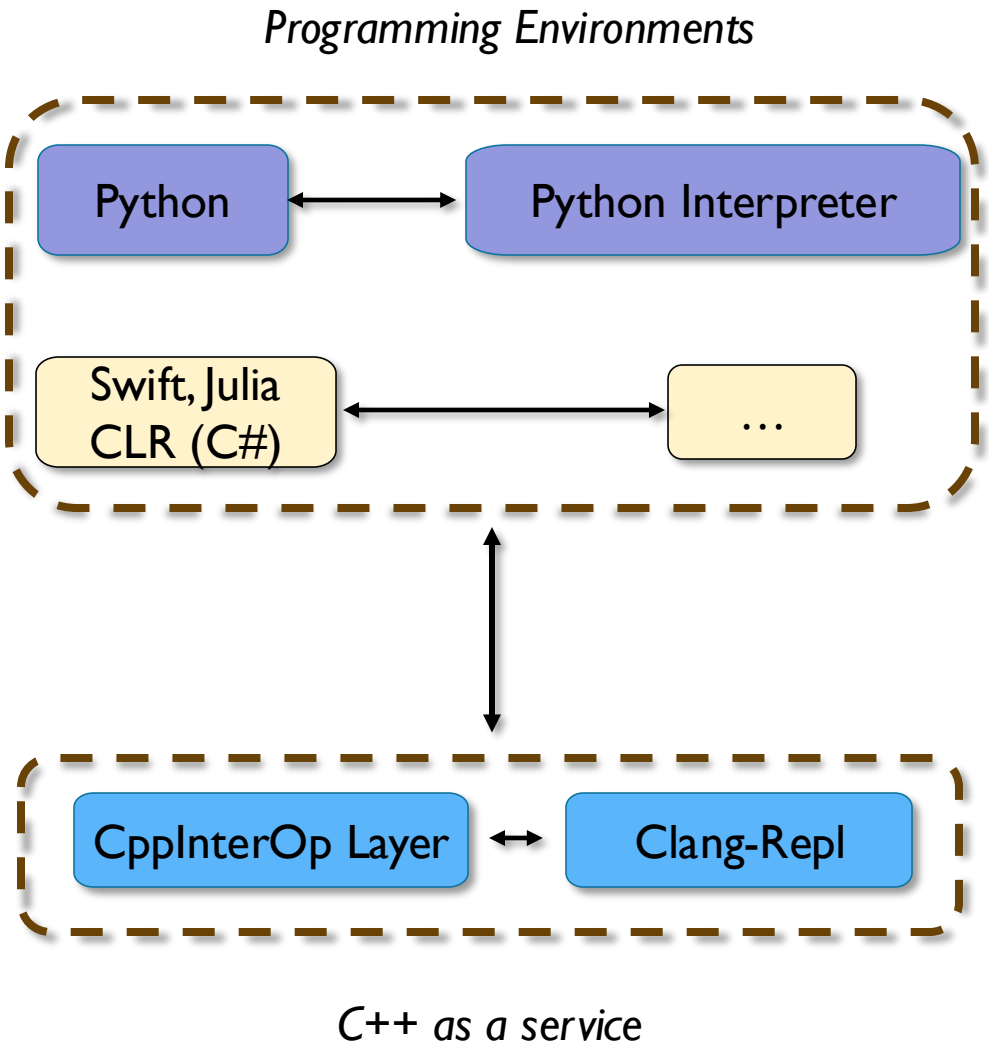
- CppInterOp allows Cppyy to use LLVM's Clang-REPL as a runtime compiler
- This avoids the string parsing logic used with the current Cling based cppyy-backend
- Opens up more C++ features that can be used by Cppyy users
- Lower dependencies leads to performance improvement
- CppInterOp unit tests verify the API that is used for proxy creation, lookups, function reflection, etc



# CPPINTEROP

- CppInterOp enables dynamic C++ interactions with multiple languages and diverse computing environments like Jupyter

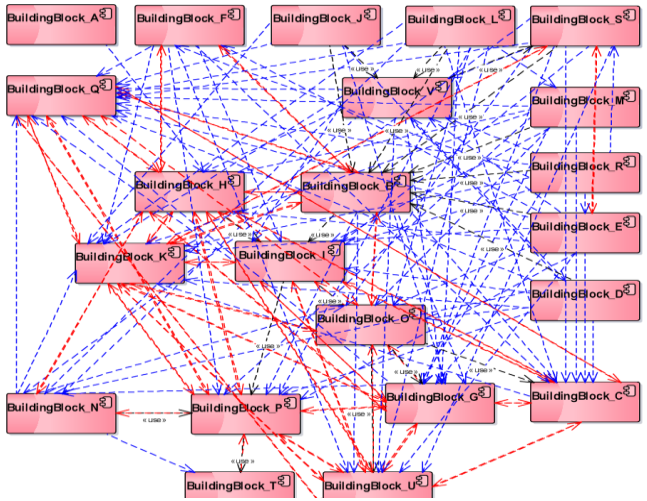
- Provides other languages/environments with:
  - A performant JIT, to incrementally compile C++ code
  - A reflection API to drive bindings generation.

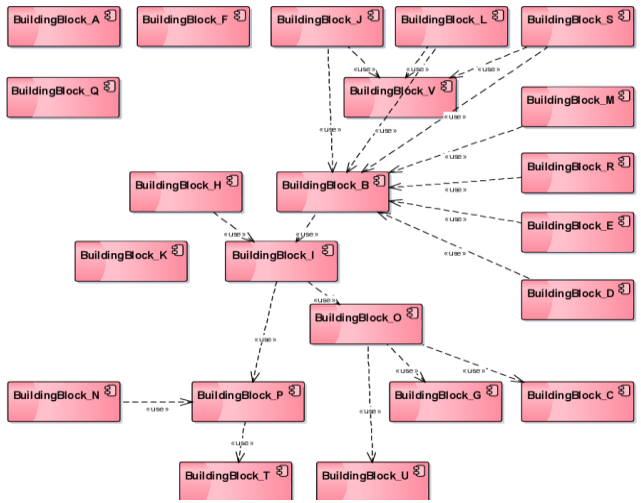


# ADVANTAGES OF TRANSITIONING TO CLANG-REPL

**Simplified codebase:**

- Compiler features are abstracted to the InterOp layer interfaces
- Easier to extend: Modular development of tests/features
- Reduced string manipulation with parsed code



# ADVANTAGES OF TRANSITIONING TO CLANG-REPL

## Simplified codebase

Example: Cppyy::GetMethodTemplate

### Cppyy Upstream - clingwrapper

```
TFunction * func = nullptr;
ClassInfo_t * cl = nullptr;
if (scope == (cppyy_scope_t) GLOBAL_HANDLE) {
  func = gROOT -> GetGlobalFunctionWithPrototype(name.c_str(), proto.c_str());
  if (func && name.back() == '>') {
if (!template_compare(name, func -> GetName()))
      func = nullptr; // happens if implicit conversion matches the overload
  }
} else {
  TClassRef & cr = type_from_handle(scope);
  if (cr.GetClass()) {
    func = cr -> GetMethodWithPrototype(name.c_str(), proto.c_str());
    if (!func) {
      cl = cr -> GetClassInfo();
TCppIndex_t nbases = GetNumBases(scope);
      for (TCppIndex_t i = 0; i < nbases; ++i) {
        TClassRef & base = type_from_handle(GetScope(GetBaseName(scope, i)));
        if (base.GetClass()) {
          func = base -> GetMethodWithPrototype(name.c_str(), proto.c_str());
          if (func) break;
        }
      }
    }
  }
```

### With CppInterOp

```
Cppyy::TCppMethod_t Cppyy::GetMethodTemplate(
TCppScope_t scope, const std::string& name, const std::string& proto)
{
        Cpp::GetMethodTemplate(scope, name);
}

TCppFunction_t GetTemplatedMethod(const std::string& name,
TCppScope_t parent, const std::string& filter)
{
DeclContext *Within = 0;
DeclContext::decl_iterator decl;
if (parent) {
auto *D = (Decl *)parent;
Within = llvm::dyn_cast<DeclContext>(D);
}

auto *ND = Cpp_utils::Lookup::Named(&getSema(), name, Within); if
((intptr_t) ND == (intptr_t) 0)
return nullptr;

if ((intptr_t) ND != (intptr_t) -1)
return (TCppFunction_t)(ND->getCanonicalDecl());
}
```

# ADVANTAGES OF TRANSITIONING TO CLANG-REPL

InterOp Unit Tests

compiler features tested on InterOp layer

```
TEST(CUDATest, CUDAH) {
if (!HasCudaSDK())
return;

Cpp::CreateInterpreter({}, {"--cuda"});
bool success = !Cpp::Declare("#include <cuda.h>");
EXPECT_TRUE(success);
}

TEST(CUDATest, CUDARuntime) {
if (!HasCudaSDK())
return;

EXPECT_TRUE(HasCudaRuntime());
}

EXPECT_EQ(Cpp::GetTypeAsString(Cpp::GetType("struct")),"NULL TYPE");
EXPECT_EQ(Cpp::GetTypeAsString(Cpp::GetType("char")), " char");
```

```
TEST(TypeReflectionTest, GetSizeOfType) {
std::vector<Decl *> Decls;
std::string code = R"(
struct S {
int a;
double b;
};

char ch;
int n;
S s;
)";

GetAllTopLevelDecls(code, Decls);

EXPECT_EQ(Cpp::GetSizeOfType(Cpp::GetVariableType(Decls[1])), 1);
EXPECT_EQ(Cpp::GetSizeOfType(Cpp::GetVariableType(Decls[2])), 4);
EXPECT_EQ(Cpp::GetSizeOfType(Cpp::GetVariableType(Decls[3])), 16);
```

# TEMPLATE SUPPORT WITH CLANG-REPL

C++ source #1

```cpp
class MyTemplatedMethodClass {
  public:
      template<class A> long get_size(A&);
      template<class A> long get_size();
      template<class A, class B> long get_size(A a, B
  };
  template<class A>
  long MyTemplatedMethodClass::get_size(A&) {
      return sizeof(A);
  }
  template<class A>
  long MyTemplatedMethodClass::get_size() {
      return sizeof(A) + 1;
  }
  template<class A, class B>
  long MyTemplatedMethodClass::get_size(A a, B b) {
      return sizeof(A) + sizeof(B);
  }
```

```
| | |-CopyConstructor simple trivial has_const_param ne
| | |-MoveConstructor exists simple trivial needs_impli
| | |-CopyAssignment simple trivial has_const_param nee
| | |-MoveAssignment exists simple trivial needs_implic
| | `-Destructor simple irrelevant trivial needs_implic
| |-CXXRecordDecl <col:5, col:11> col:11 implicit class
| |-AccessSpecDecl <line:2:7, col:13> col:7 public
| |-FunctionTemplateDecl <line:3:11, col:45> col:34 get
| | |-TemplateTypeParmDecl <col:20, col:26> col:26 refe
| | `-CXXMethodDecl <col:29, col:45> col:34 get_size '
| |   `-ParmVarDecl <col:43, col:44> col:45 'A &'
| |-FunctionTemplateDecl <line:4:11, col:43> col:34 get
| | |-TemplateTypeParmDecl <col:20, col:26> col:26 clas
| | `-CXXMethodDecl <col:29, col:43> col:34 get_size '
| `-FunctionTemplateDecl <line:5:11, col:60> col:43 get
```

An overload is assumed to be “correct” when the type translation for a set of template parameters (types in the angular brackets instantiating the type) and function parameters (standard function arguments) can find a Clang declaration on its syntax tree generated by the compiler.

A visualisation of the Abstract Syntax Tree (AST) generated by Clang

# TEMPLATE SUPPORT WITH CLANG-REPL

Design for a higher level interface (in cppyy-backend/clingwrapper) that is called by the aforementioned overload selection logic in CPyCppyy.

Once the type translation and name is obtained, we call GetMethodTemplate:

```
Cppyy::TCppMethod_t Cppyy::GetMethodTemplate(
    TCppScope_t scope, const std::string& name, const std::string& proto)
{
    std::string pureName;
    std::string explicit_params;

    if (name.find('<') != std::string::npos) {
        pureName = name.substr(0, name.find('<'));
        size_t start = name.find('<');
        size_t end = name.find('>');
        explicit_params = name.substr(start + 1, end - start - 1);
    }

    else pureName = name;

    std::vector<Cppyy::TCppMethod_t> unresolved_candidate_methods;
    Cpp::GetClassTemplatedMethods(pureName, scope,
                                  unresolved_candidate_methods);

    // CPyCppyy assumes that we attempt instantiation here
    std::vector<Cpp::TemplateArgInfo> arg_types;
    std::vector<Cpp::TemplateArgInfo> templ_params;
    Cppyy::AppendTypesSlow(proto, arg_types);
    Cppyy::AppendTypesSlow(explicit_params, templ_params);

    Cppyy::TCppMethod_t cppmeth = Cpp::BestTemplateFunctionMatch(unresolved_candidate_methods, temp

    if(!cppmeth){
        return nullptr;
    }

    return cppmeth;
```

# TEMPLATE SUPPORT WITH CLANG-REPL

- We use an InterOp level interface that exposes and works with the Clang API to return the memory address (Cppyy::TCppMethod_t is a typedef'd void pointer) of the best suited templated function.

- This selection logic algorithm is specifically designed to consider the rules of overload resolution and explicit instantiations

```
TCppFunction_t
BestTemplateFunctionMatch(const std::vector<TCppFunction_t>& candidates,
                          const std::vector<TemplateArgInfo>& explicit_type
                          const std::vector<TemplateArgInfo>& arg_types) {

  for (const auto& candidate : candidates) {
    auto* TFD = (FunctionTemplateDecl*)candidate;
    clang::TemplateParameterList* tpl = TFD->getTemplateParameters();

    // template parameter size does not match
    if (tpl->size() < explicit_types.size())
      continue;

    // right now uninstantiated functions give template typenames instead c
    // actual types. We make this match solely based on count

    const FunctionDecl* func = TFD->getTemplatedDecl();

#ifdef USE_CLING
    if (func->getNumParams() > arg_types.size())
      continue;
#else // CLANG_REPL
    if (func->getMinRequiredArguments() > arg_types.size())
      continue;
#endif

    // FIXME : first score based on the type similarity before forcing
```

# TEMPLATE SUPPORT WITH CLANG-REPL

```
static Decl* InstantiateTemplate(TemplateDecl* TemplateD,
                                 TemplateArgumentListInfo& TLI, Sema& S) {
  // This is not right but we don't have a lot of options to choose from as a
  // template instantiation requires a valid source location.
  SourceLocation fakeLoc = GetValidSLoc(S);
  if (auto* FunctionTemplate = dyn_cast<FunctionTemplateDecl>(TemplateD)) {
    FunctionDecl* Specialization = nullptr;
    clang::sema::TemplateDeductionInfo Info(fakeLoc);
    if (Sema::TemplateDeductionResult Result = S.DeduceTemplateArguments(
            FunctionTemplate, &TLI, Specialization, Info,
            /*IsAddressOfFunction*/ true)) {
      // FIXME: Diagnose what happened.
      (void)Result;
    }
    return Specialization;
  }

  if (auto* VarTemplate = dyn_cast<VarTemplateDecl>(TemplateD)) {
    DeclResult R = S.CheckVarTemplateId(VarTemplate, fakeLoc, fakeLoc, TLI);
    if (R.isInvalid()) {
      // FIXME: Diagnose
    }
    return R.get();
  }

  // This will instantiate tape<T> type and return it.
  SourceLocation noLoc;
  QualType TT = S.CheckTemplateIdType(TemplateName(TemplateD), noLoc, TLI);

  // Perhaps we can extract this into a new interface.
  S.RequireCompleteType(fakeLoc, TT, diag::err_tentative_def_incomplete_type);
  return GetScopeFromType(TT);
```

Here, the interface InstantiateTemplate is designed to provide on demand instantion of a template declaration (TemplateDecl*):

# TEMPLATE SUPPORT WITH CLANG-REPL

This also involved bumping up all the Clang API in CppInterOp to handle template functions to work with the Cppyy engine

compiler-research/CppInterOp **Update GetFunctionRequiredArgs for template functions** ✓
#220 by aaronj0 was merged on Apr 7 • Approved

compiler-research/CppInterOp **Update GetFunctionNumArgs API for template functions** ✗
#219 by aaronj0 was merged on Apr 6 • Approved

compiler-research/CppInterOp **Update GetFunctionReturnType API for template functions** ✓
#218 by aaronj0 was merged on Apr 7 • Approved

compiler-research/CppInterOp **Fix usage of InstantiateTemplate** ✓
#217 by aaronj0 was merged on Apr 5

compiler-research/CPyCppyy **Fix missed rename case for InstantiateTemplate**
#49 by aaronj0 was merged on Apr 4

compiler-research/CPyCppyy **Rename template instantiation API usage**
#48 by aaronj0 was merged on Apr 4

compiler-research/cppyy-backend **Rename InterOp interface for template instantiation** ✗
#94 by aaronj0 was merged on Apr 4

# RESULTS AND FUTURE WORK

Added features:

- Near 100% feature support for:

- Python-C++ CrossInheritance(+10 tests)
- Overload selection tests(8/10)
- Pythonify tests (19/22)
- Concurrency(4/7)
- Conversion tests (3/4)

```python
from cppyy.gbl.CrossInheritance import TBase1, TDerived1, TBase1_I

class TPyDerived1(TBase1_I):
    def __init__(self):
        super(TBase1_I, self).__init__()

    def get_value(self):
        return 13

b1, b2 = TBase1[int](), TBase1_I()
assert b1.get_value() == 42
assert b2.get_value() == 42

d1 = TDerived1()
assert d1.get_value() == 27
```

```python
@mark.xfail(condition=IS_MAC_X86 or IS_MAC_ARM, reason="Fails on OS X")
def test03_instance_conversion(self):
    """Proxy object conversions"""

    import cppyy
    cpp = cppyy.gbl
    API = cpp.CPyCppyy

    cppyy.cppdef("""
    class APICheck2 {
    public:
      virtual ~APICheck2() {}
    };""")

    m = cpp.APICheck2()

    voidp = API.Instance_AsVoidPtr(m)
    m2 = API.Instance_FromVoidPtr(voidp, 'APICheck2')
    assert m is m2
```

# RESULTS AND FUTURE WORK

100% passing tests for:

- API

- Memory leak checks

compiler-research/CppInterOp **[Interpreter] Forward declare input to process so it's registered in JIT** ✓
#238 by maximusron was merged 1 hour ago • Approved

compiler-research/cppyy **[test] Enable xpasses**
#66 by maximusron was merged 3 days ago

compiler-research/CppInterOp **[ci] Cppyy tests - fix pytest failing** ✓
#233 by maximusron was merged last week

compiler-research/CppInterOp **[ci] Cppyy tests - tail last two lines on pytest log** ✓
#232 by maximusron was closed 2 weeks ago

compiler-research/CppInterOp **[ci] Add boost and eigen for Cppyy test suite** ✓
#231 by maximusron was merged 2 weeks ago • Approved

compiler-research/CppInterOp **Tests for static GetClassDecls** ✓
#230 by maximusron was merged 2 weeks ago • Approved

compiler-research/CppInterOp **[ci]Update clang-format job** ✓
#216 by maximusron was merged 3 weeks ago

compiler-research/CppInterOp **Enable template features** ✗
#215 by maximusron was merged 2 weeks ago • Approved

compiler-research/cppyy-backend **Template fix patch based on updated InterOp API** ✗
#93 by maximusron was merged 2 weeks ago

compiler-research/cppyy **[tests] Update pytest tags on tests based on template-fix patch**
#65 by maximusron was merged 2 weeks ago • Approved

compiler-research/CPyCppyy **Fix bugs related to reflex, python side attribute lookup and template instantiation**
#47 by maximusron was merged 2 weeks ago

# RESULTS AND FUTURE WORK

Specific features enabled:

- std::basic_string support in Converters
- Class loading
- Template bindings
- R-Value templates
- Std::pair initialization
- Vector of Pair
- Builtin casts

```python
def test02_builtin_cpp_casts(self):
    """C++ casting of builtin types"""

    from cppyy import ll

    for cast in (ll.cast, ll.static_cast):
        assert type(cast[float](1)) == float
        assert cast[float](1) == 1.

        assert type(cast[int](1.1)) == int
        assert cast[int](1.1) == 1

    assert len(ll.reinterpret_cast['int*'](0)) == 0
    raises(ReferenceError, ll.reinterpret_cast['int*'](0).__getitem__, 0)
```

```python
def test11_vector_of_pair(self):
    """Use of std::vector<std::pair>"""

    import cppyy

    cppyy.cppdef("""
    class PairVector {
    public:
        std::vector<std::pair<double, double>> vector_pair(const std::vector<std::pair<double, double>>& a) {
            return a;
        }
    };
    """)

    from cppyy.gbl import PairVector
    a = PairVector()
    ll = [[1., 2.], [2., 3.], [3., 4.], [4., 5.]]
    v = a.vector_pair(ll)
```

# RESULTS AND FUTURE WORK

Interpreter – Forward declaring input to register in JIT

Added API's for support cppyy support with Clang-REPL

Support for basic templated functions and methods

Enabling libboost and eigen functionality

compiler-research/CppInterOp **[Interpreter] Forward declare input to process so it's registered in JIT** ✓
#238 by maximusron was merged 1 hour ago • Approved

compiler-research/cppyy **[test] Enable xpasses**
#66 by maximusron was merged 3 days ago

compiler-research/CppInterOp **[ci] Cppyy tests - fix pytest failing** ✓
#233 by maximusron was merged last week

compiler-research/CppInterOp **[ci] Cppyy tests - tail last two lines on pytest log** ✓
#232 by maximusron was closed 2 weeks ago

compiler-research/CppInterOp **[ci] Add boost and eigen for Cppyy test suite** ✓
#231 by maximusron was merged 2 weeks ago • Approved

compiler-research/CppInterOp **Tests for static GetClassDecls** ✓
#230 by maximusron was merged 2 weeks ago • Approved

compiler-research/CppInterOp **[ci]Update clang-format job** ✓
#216 by maximusron was merged 3 weeks ago

compiler-research/CppInterOp **Enable template features** ✗
#215 by maximusron was merged 2 weeks ago • Approved

compiler-research/cppyy-backend **Template fix patch based on updated InterOp API** ✗
#93 by maximusron was merged 2 weeks ago

compiler-research/cppyy **[tests] Update pytest tags on tests based on template-fix patch**
#65 by maximusron was merged 2 weeks ago • Approved

compiler-research/CPyCppyy **Fix bugs related to reflex, python side attribute lookup and template instantiation**
#47 by maximusron was merged 2 weeks ago

# TOTAL STATS

| TEST | Total Count | Previously passing | New pass count | Increase |
|---|---|---|---|---|
| CROSSINHERITANCE | 38 | 11 | 28 | +17 |
| DATATYPES | 49 | 18 | 27 | +9 |
| DOCFEATURES | 61 | 29 | 40 | +11 |
| EIGEN | 5 | 0 | 1 | +1 |
| FRAGILE | 30 | 16 | 17 | +1 |

# TOTAL STATS

| TEST | Total Count | Previously passing | New pass count | Increase |
|---|---|---|---|---|
| API | 5 | 2 | 5 | +3 |
| LEAKCHECK | 6 | 1 | 6 | +5 |
| LOWLEVEL | 21 | 8 | 12 | +4 |

# TOTAL STATS

| | | |
|---|---|---|
| FAIL | 54 | 0 |
| PASS | 188 | 276 |
| XFAIL | 262 | 227 |

```
276 passed, 1 skipped, 227 xfailed, 4 warnings in 322.35s (0:05:22)
```

# THE CI


CppInterOp's CI is intended to ensure extensive testing and development across several settings. A caching phase is used to guarantee that the essential build artifacts are readily available, minimizing the time required for subsequent builds.

The CI matrix includes a variety of operating systems (Ubuntu, Windows, macOS), compilers (GCC, MSVC, Clang), and configurations (various Clang runtime versions, with and without Cling and Cppyy).

# THE CI

compiler-research/cppyy [test] Suppress Eigen regression for OS X x86 ✓
#84 by aaronj0 was merged on May 9

compiler-research/cppyy Add final set of failing tests for OS X cling and disable docfeatures ✓
#83 by aaronj0 was merged on May 8

compiler-research/cppyy [tests] Tag OS X arm cling tests ✗
#81 by aaronj0 was merged on May 7

compiler-research/CppInterOp [ci] Fixes for crashes on workflow ✗
#268 by aaronj0 was merged on May 7

compiler-research/CppInterOp No longer use true for OS X pytest ✗
#263 by aaronj0 was merged on May 5

compiler-research/cppyy [ci] Disable crashing tests on Apple Silicon ✗
#79 by aaronj0 was merged on May 4

compiler-research/cppyy Update crashing test tag for os x arm ✗
#78 by aaronj0 was merged on May 3

compiler-research/cppyy Pytest tags for OS X ✗
#76 by aaronj0 was merged on May 3

compiler-research/cppyy Mark exception failing tests on OS X ✗
#75 by aaronj0 was merged on May 3

The CI infrastructure of CppInterOp was significantly improved to catch regressions, over multiple repositories (3x cppyy packages, InterOp)

# THE CI

Specific improvements in platform specific support, variations in libc++ vs libstdc++ with Mac, and architecture specific bugs with x86 vs ARM on OS X

compiler-research/CppInterOp **[ci] Update OS X and clang 16 exitcodes** ✕
#253 by aaronj0 was merged on May 3

compiler-research/cppyy-backend **[ci] Update CI for clang 16 valgrind** ✕
#98 by aaronj0 was merged on May 3

compiler-research/cppyy **Update test tag and CI** ✕
#73 by aaronj0 was merged on May 2

compiler-research/CppInterOp **[ci] Show error on valgrind** ✕
#252 by aaronj0 was merged on May 1

compiler-research/cppyy-backend **[ci] Capture exit code on full test run by setting pipefail** ✕
#97 by aaronj0 was merged on Apr 27

compiler-research/CppInterOp **[ci] Revert pytest-xdist worker restart for cppyy tests** ✕
#241 by aaronj0 was merged on Apr 26

compiler-research/cppyy **[ci] Update cppyy testing job** ✕
#68 by aaronj0 was merged on Apr 26

# OTHER OUTCOMES

Accepted abstract at CHEP 2024:

*CppInterOp: Advancing Interactive C++ for High Energy Physics*

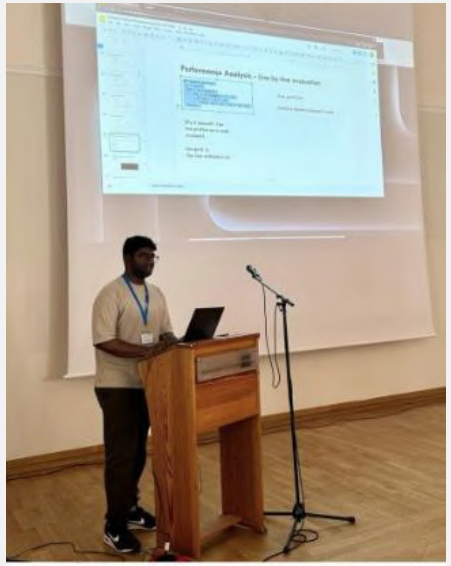
Mentoring 2 projects on Google Summer of Code 2024:

- Improved Numpy-STL integration in Cppyy
- LLM in Xeus-Cpp

Open Source community development:

- Reviewing PR's from external contributers

Talk on High performance python at the Fast and Efficient Python Computing school in Aachen, Germany



# WHAT NEXT?

- Working on PyROOT with the ROOT team.

- Reduction of technical debt with Cppyy/PyROOT divergence (8 patches away from syncing with upstream)

- Fixing bugs and issues reported by experiments/users on PyROOT

- Improvement of pythonic interfaces in ROOT

Adoption of CppInterOp in ROOT to provide an improved interpreter infrastructure

ROOT
Data Analysis Framework

root-project/root [PyROOT] Perform function-style casts when returning multi-keyword types in:PyROOT
#16197 by aaronj0 was merged 3 weeks ago • Approved 1 task done

root-project/root [v632][PyROOT] add Sequence_Check to the public API pr:backport
#16183 by aaronj0 was merged 3 weeks ago • Approved

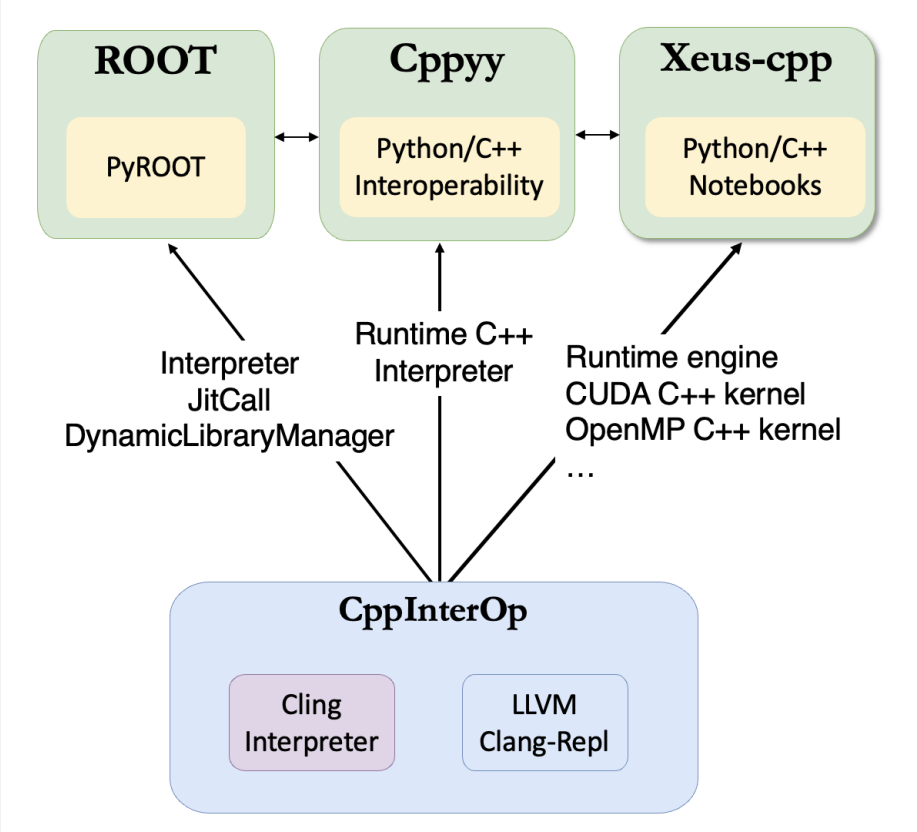
root-project/roottest [PyROOT] Loop nJIT function more to prevent sporadic failures
#1176 by aaronj0 was merged 2 days ago • Approved

root-project/root [PyROOT] Return scope in GetActualClass based on Interpreter ClassInfo bug in:PyROOT
#16277 by aaronj0 was merged last week • Approved

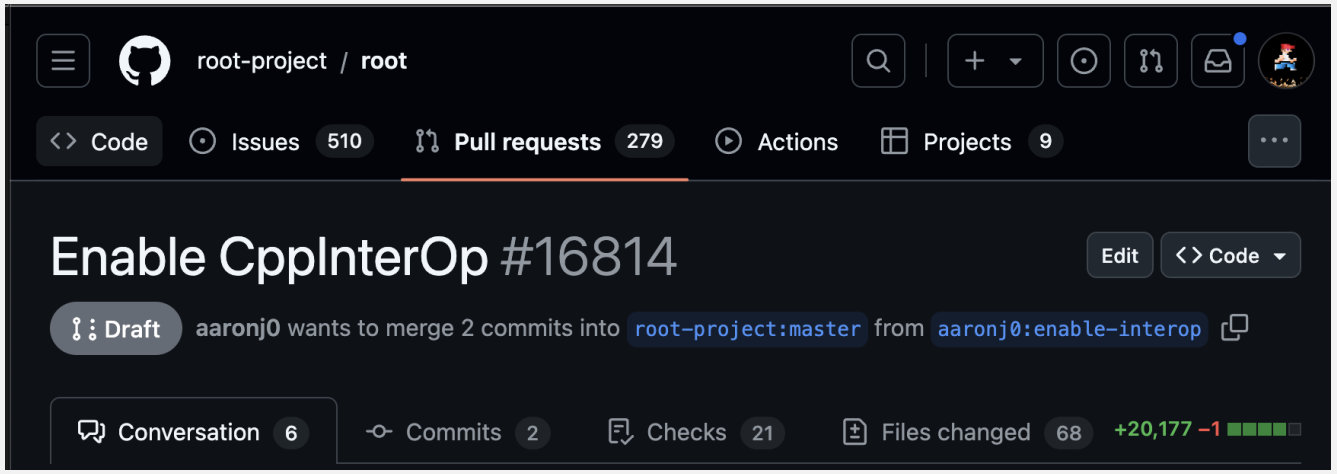
root-project/root [ci] Enable numpy 2.0
#16238 by aaronj0 was merged 2 weeks ago • Approved

# WHAT NEXT?

- Adoption of CppInterOp in ROOT to provide an improved interpreter infrastructure







# Thank You!

Aaron Jomy