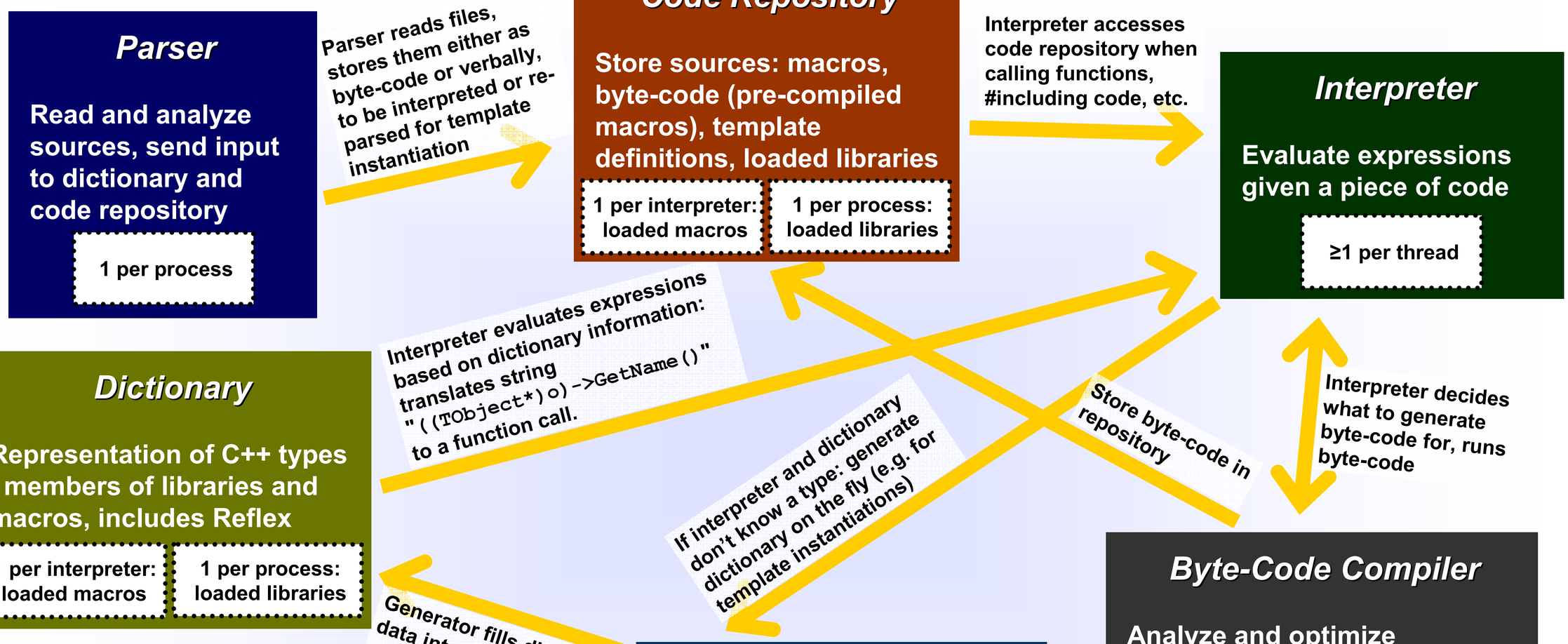
CINT & Reflex – The Future

CINT's Future Layout



Restructure CINT to be object-oriented, with focus on thread-safety, ease of maintenance, speed; consider requirements of dependent projects (ROOT I/O, GUI, PyROOT, CINT users)

Reflex API

Simple interface: everything is types, scopes, and/or member
Consistent navigation: by name, iteration, direct access
Very small API objects: just a pointer (scope: +int), no virtual funcs
Hide complex back-end implementation that will allow unloading, persistency, dynamic updates, forward declarations, etc.

Reflex::Type

Generic type description: enclosing scope; can be built as sequence of pointer, const, reference,…; can have bases if it's a class, can convert to a Reflex::Scope if it's a class, struct, or union; functions also have a type, e.g. parameters, return type, constness

Reflex::Scope

Generic scope description: enclosing scope; can convert to a Reflex::Type if it's a class, struct, or union; allows access to enclosed scopes, types, members

Reflex::Member

Describe data and function members; iterate through them, get their types, for functions: access list of parameters, get the return type

Get Reflex::Type representing a `char`, build `const char*`. `ToType()` returns the underlying type, `TypeType()` the kind of type – `char` is a fundamental type.

```
Type tC("char");
Type tCC = ConstBuilder(tC);
Type tPCC = PointerBuilder(tCC);
tCC == tPCC.ToType(); // true
tC.TypeType(); // FUNDAMENTAL
```

A class is a type, too: look it up by name, get its enclosing scope `"ROOT::Math"`, test its properties, get its name including scopes.

```
Type tI("ROOT::Math::Integrator");
Scope sRM = tI.DeclaringScope();
tI.IsClass(); tI.IsPointer(); tI.IsConst(); // true,false,false
tI.Name(SCOPED); // "ROOT::Math::Integrator"
```

Find a scope by name, as it's a class we can convert it to a type. Access types defined within a scope by iterator or name. Compare two representations of the same type, access a type from the global scope.

```
Scope sRMI("ROOT::Math::Integrator");
Type tRMI = sRMI;
Type_Iterator iSubType = sRMI.DeclaringScope().SubType_Begin();
Type tSubType = sRMI.SubTypeByName("Integrator");
tSubType == tRMI; // true
Scope sR = Scope::ByName("ROOT");
Scope::GlobalScope().ByName("ROOT") == sR // true
```

Retrieve a scope's data member by name and iterator, get a member's type.

```
Scope sNamed("TNamed");
Member mName = sNamed.DataMemberByName("fName");
Member_Iterator iMember = sNamed.Member_Begin();
Type tName = sNamed.GetType();
tName == Type::ByName("TString"); // true
```

Get member function `SetTitle(const char* t="")` by name, get a function member by direct access. Member functions have a type, too. Query parameters and their default values.

```
Member mSetTitle = sNamed.FunctionMemberByName("SetTitle");
Member iMember = sNamed.FunctionMemberAt(0);
Type t = mSetTitle.GetType(); // (void TNamed::*)(const char*)
int numparams = mSetTitle.FunctionParameterSize(); // 1
mSetTitle.FunctionParameterDefaultAt(0); // ""
```

Work In Progress:

- **Use Reflex to store dictionary data**
  - *Smaller memory footprint*
  - *First step to modularized code*
- **Remove function wrappers in CINT dictionaries for inherited virtual functions**
  - *Saves 30% of e.g. G__Cont.o*
  - *Can be implemented on all platforms*
  - *Call through base class's wrapper*
- **Replace function wrappers by direct library calls on selected systems**
  - *Saves 55% of e.g. G__Cont.o*
  - *Highly platform dependent, will only work for selected architectures (GCC, ICC, MSVC)*
- **Shuffle CINT's functionality into an object-oriented layout**
  - *See above*
  - *Prerequisite for thread-safety*
- **Make CINT thread-safe**
- **On-the-fly dictionary generation**
  - *Useful for e.g. templated classes*
  - *Powerful once direct library calls work*
- **New implementation of the byte-code compiler**
  - *Major source of ISO C++ incompatibilities*