G++ Internals

Dodji Seketeli <dodji@redhat.com>

Eigen Summit - February 2010 - Paris, France

Plan of the talk

Source code

Compiler pipeline

Internal representations

Class analysis process

Class template representation

Template instantiation process

Questions

Outline

Top Directory

<root>/

- gcc/ ← Code of compilers are here
- libada/
- libcpp/
- libgcc/
- libgfortran/
- libgomp/
- libiberty/
- libjava/
- libobjc/
- libstdc++-v3/ ← Code of C++ standard library

Directory gcc/

<root/gcc>/

- ada/
- cp/ ← Code of G++ is here
- doc/
- fortran/
- java/
- objc/
- objcp/

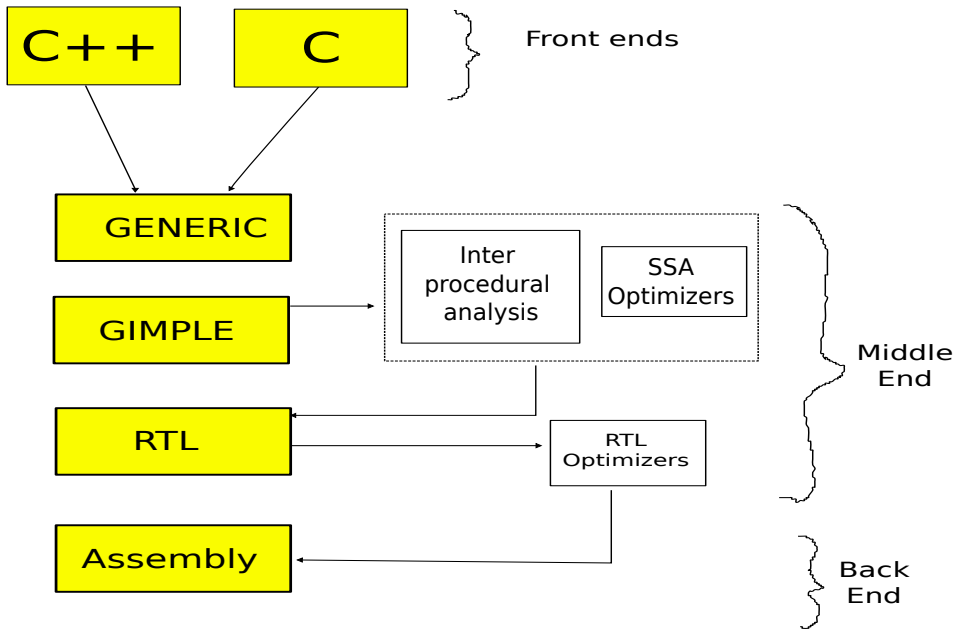
Some Numbers

- GCC is more than 2,3 millions of LOC
- G++ specific lines of code is around 84 KLOC
- GCC full testsuite is around 720 KLOC
- G++ testsuite is around 90 KLOC

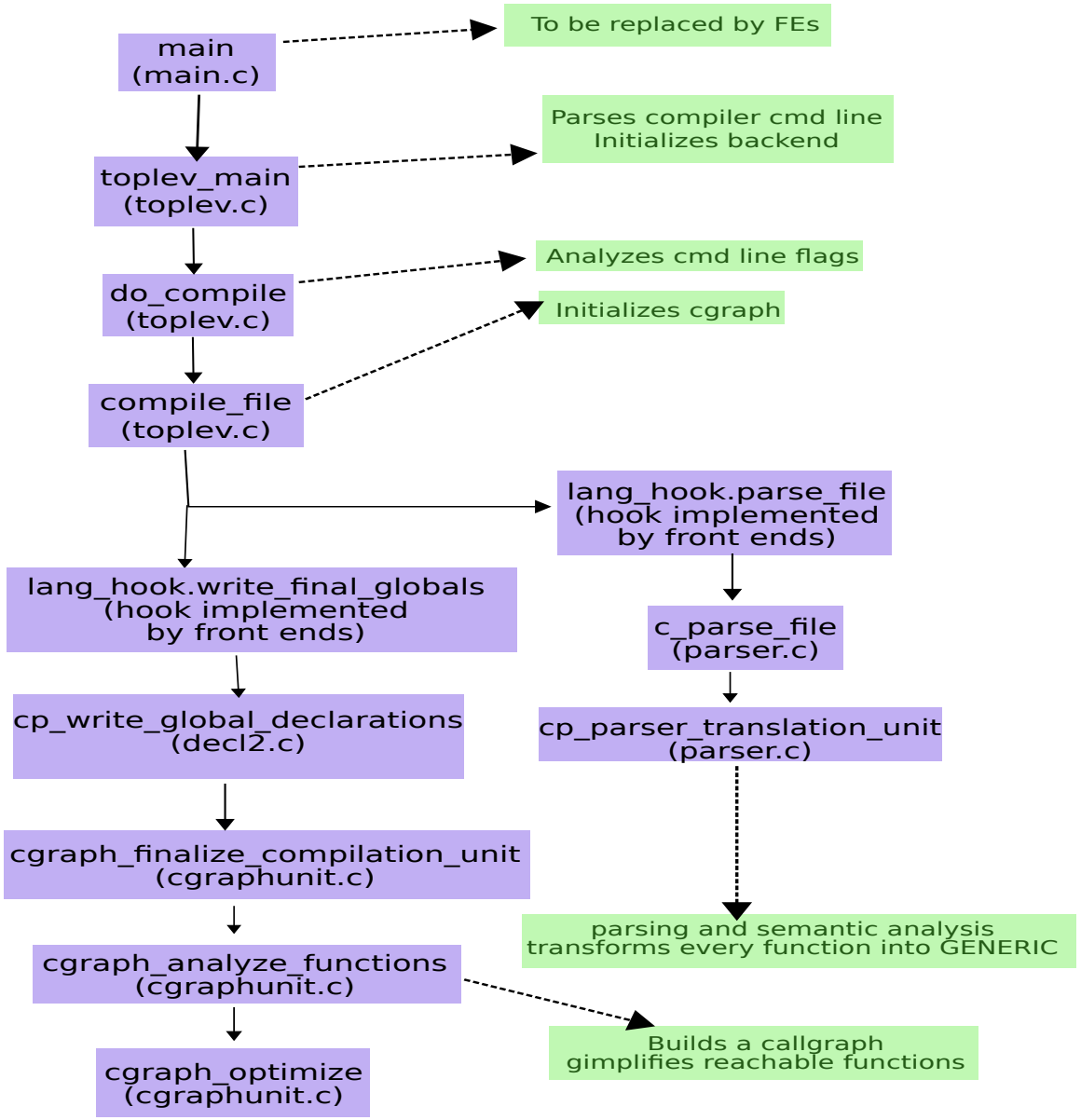
Outline

Pipeline overview

- Front-Ends
  - Have own representation
  - Emit GENERIC
- Middle-End works on GIMPLE
- Back-End works on RTL



Detailed compilation process



Outline

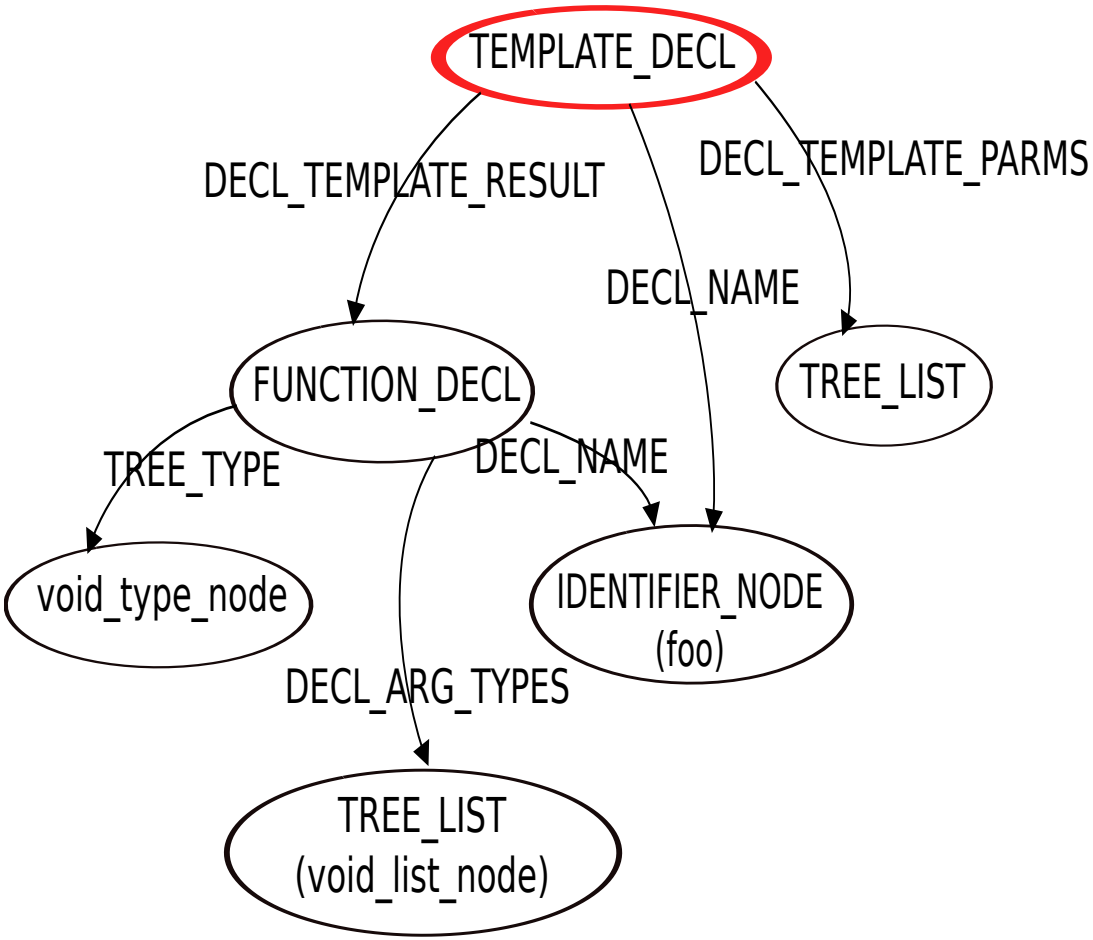
- Can represent a full translation unit
- Based on trees^{TM}
- A tree is a pointer type
- Can point to different kind of nodes
- Many macros available to manipulate trees

```
tree func = parse_function ();
if (TREE_CODE (func) == FUNCTION_DECL)
     printf ("It worked!\n");
else
     printf ("Grrr, it failed\n");
```

- GENERIC trees defined and documented in gcc/tree.def
- Accessor macros and utility functions in gcc/tree.h

- Based on trees
- Superset of GENERIC
- G++ trees defined and documented in gcc/cp/cp-tree.def
- Accessors macros and utility functions in gcc/cp/cp-tree.h and gcc/cp/tree.c

```
template<class T>
void foo(void)
{
}
```



- Used by Middle-End
- 3 address representation

```
t1 = a + b;
```

- Restricted grammar to simplify job of optimizers
- No hidden or implicit side effect
- Local variables of scalar types treated as “registers” (real operands)
- Globals and non-scalar types treated as “memory” (virtual operand).

```
t1 = a[5];
t2 = *p1;
```

- Can be incrementally lowered (2 levels currently)
  - High GIMPLE: lexical scopes

```
if (t1 == 0)
    t2 = 5;
return 0;
```

  - Low GIMPLE: no lexical scopes

```
if (t1 == 0) <L1,L2>
L1:
t2 = 5;
goto L3;
L2:
L3:
return 0;
```

- Simplified control flow
  - Loops represented as gotos
  - No lexical scope (low GIMPLE)

GENERIC

```
if (foo (a + b,c))
     c = b++ / a
endif
return c
```

High GIMPLE

```
t1 = a + b
t2 = foo (t1, c)
if (t2 != 0)
     t3 = b
     b = b + 1
     c = t3 / a
endif
return c
```

Low GIMPLE

```
t1 = a + b
t2 = foo (t1, c)
if (t2 != 0) <L1,L2>
L1:
t3 = b
b = b + 1
c = t3 / a
goto L3
L2:
L3:
return c
```

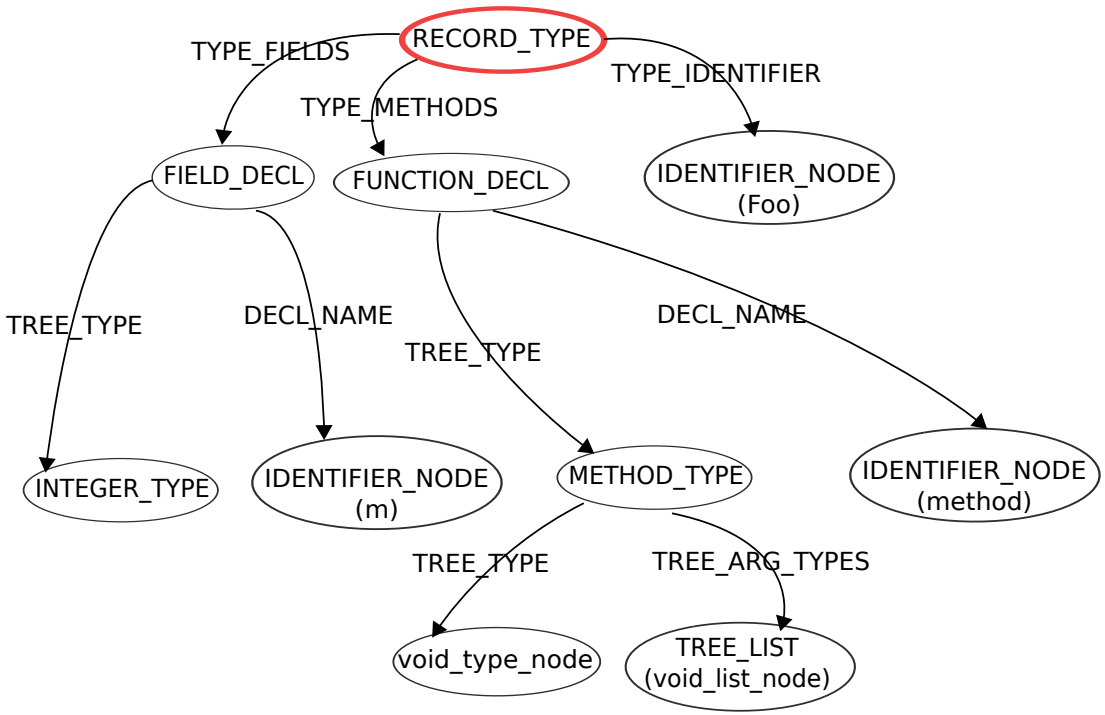
Outline

Code

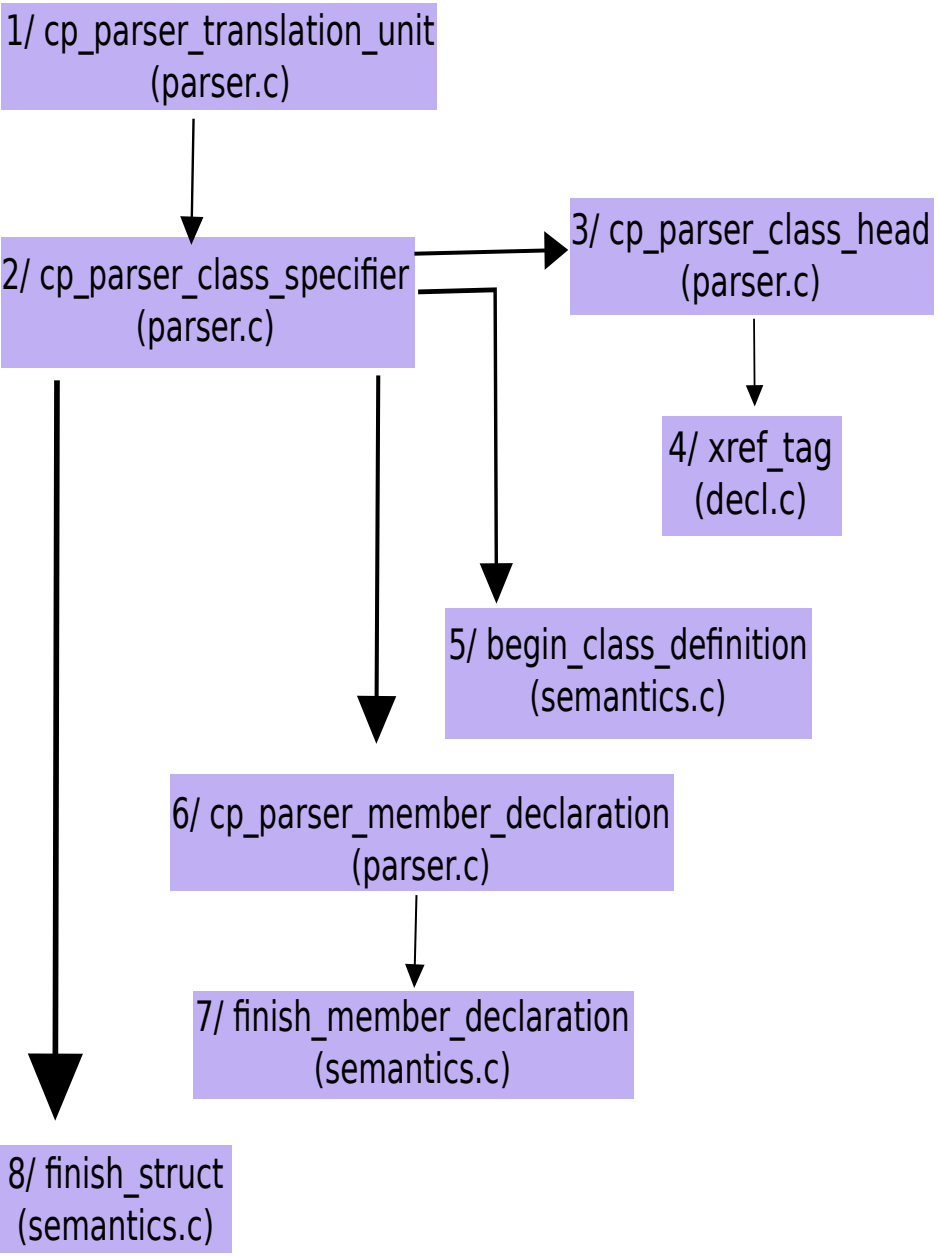
```
struct Foo
{
    int m;

    void
    method()
    {
        char j;
    }
};
```

Representation



Class analysis process



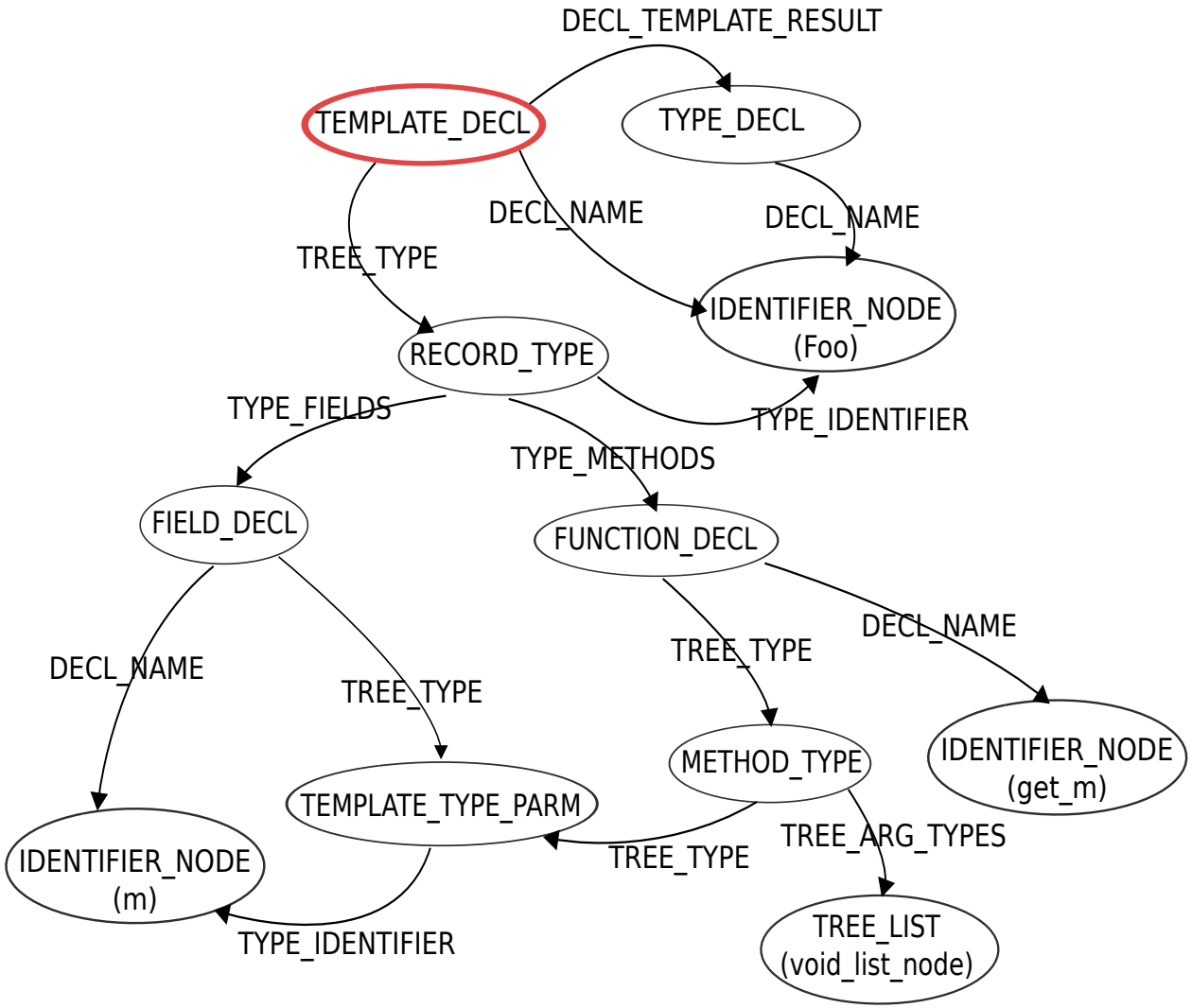
1. cp_parser_translation_unit
   - Parser entry point (recursive descent)
2. cp_parser_class_specifier
   - parses the entire class
3. cp_parser_class_head
   - Parses E.g: class foo
4. xref_tag : Declare the new type
   - Create (non-complete) type
     Inserts it into symbols table
5. begin_class_definition
   - Setup new type definition
     E.g: Open class scope
6. cp_parser_member_declaration
   - called to parse each class member
7. finish_member_declaration
   - Add parsed member to current class type
8. finish_struct
   - Layout the type
   - Emit type debug info

Outline

class template representation

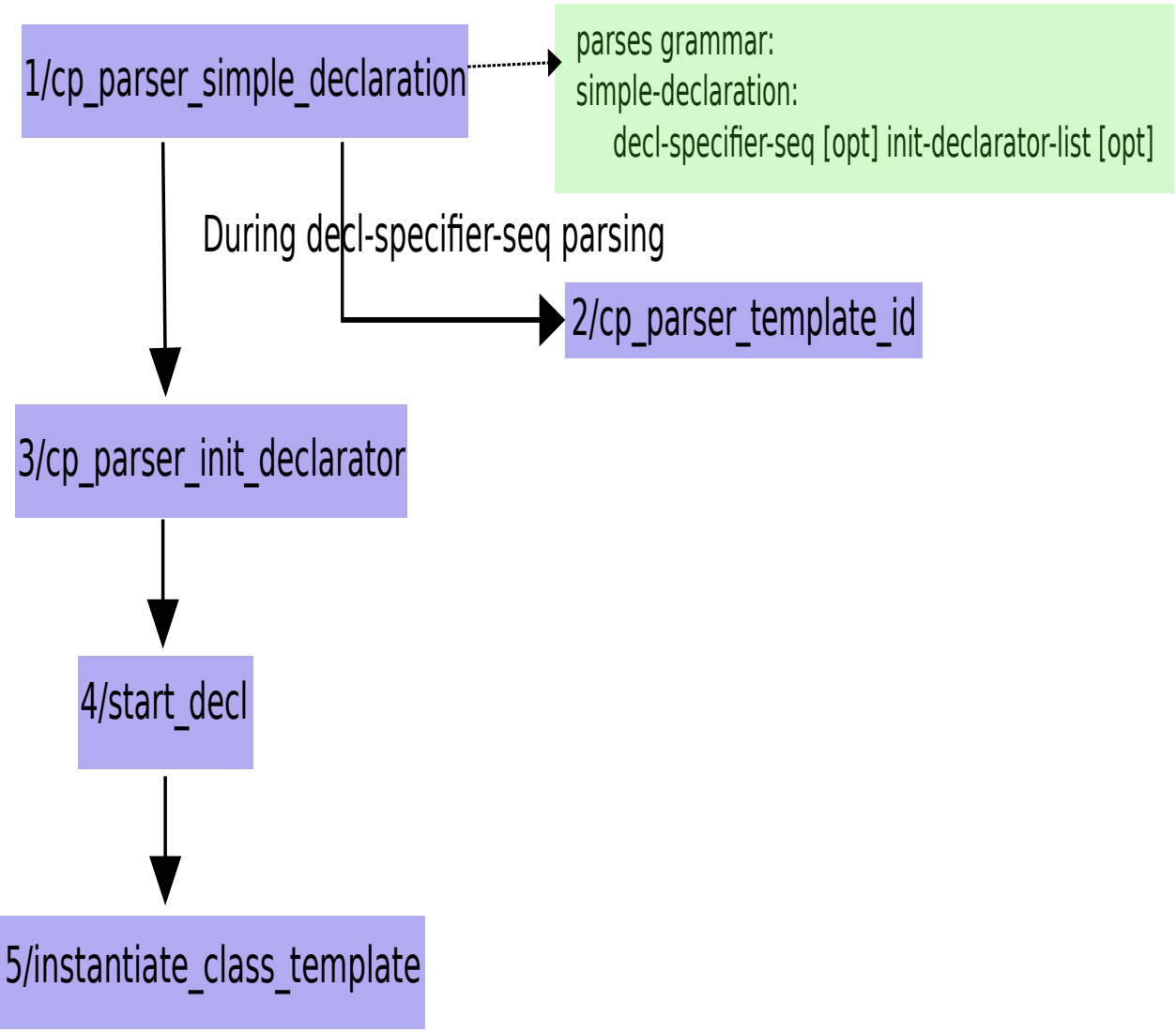
```
template<class T>
struct Foo
{
    T m;

    T
    get_m ()
    {
        return m;
    }
};
```



Outline

Template instantiation process



- So we want to parse:
  Foo<int> bar;

1. Parses the declaration
   - Sequence of decl specifiers
   - Init declarator
2. Parses `template-id`
   - Parse template name
   - Look it up
   - Parses templ arg int
   - Create non-complete type
3. Parse the `init-declarator`
   - Parse declarator bar
4. Add declarator into sym table
   - Complete type Foo<int>
   - Semantic analysis
5. Instantiate Foo<int>
   - Find most specialized template Foo
   - Substitute int into template parms
   - Semantic analysis

Outline

Contact

- irc://oftc.net#gcc
- git clone git://gcc.gnu.org/git/gcc
- http://gcc.gnu.org/onlinedocs/gccint/
- http://gcc.gnu.org

Questions?