Workshop on Essential Abstractions in GCC

GCC Control Flow and Plugins

GCC Resource Center

(www.cse.iitb.ac.in/grc)

Department of Computer Science and Engineering,

Indian Institute of Technology, Bombay

July 2010

Outline

- Motivation
- Plugins in GCC
- GCC Control Flow
- Conclusions

Part 1

Motivation

Walking the Maze of a Large Code Base

- Use cscope

  ```
  cd $SOURCE
  cscope -R
  ```

- Use ctags

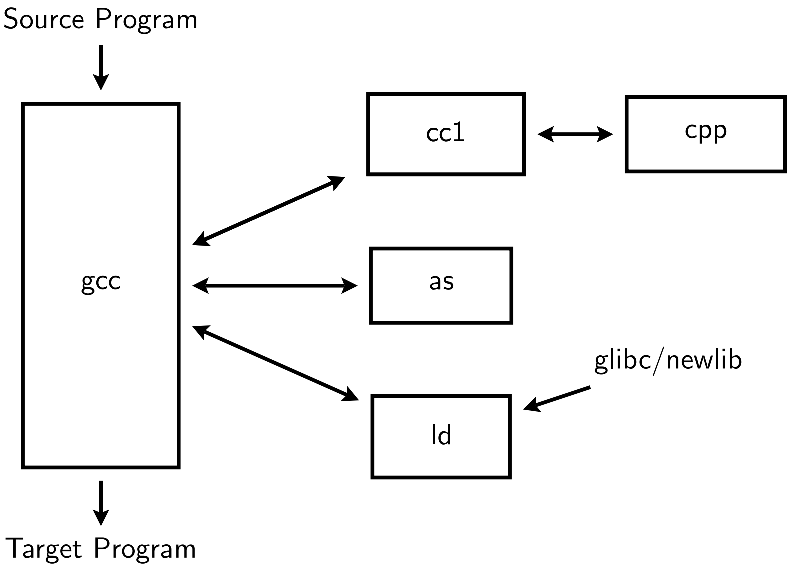
  ```
  cd $SOURCE
  ctags -R
  ```

  Make sure you use `exeburant-ctags`

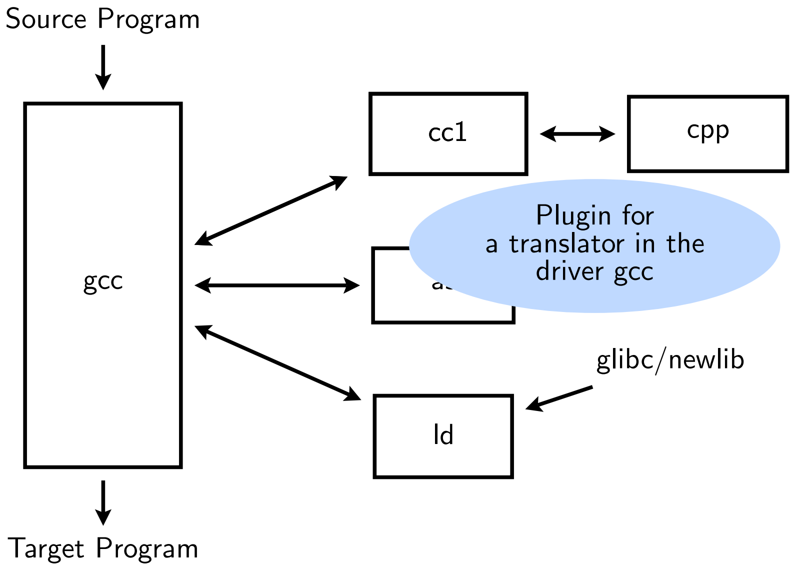
The Role of Plugins in Large Software

- A plugin is neither a stub nor a driver
- A plugin allows plugging in new modules without making changes at many places
- Most often a plugin in a C based software is a data structure containing function pointers and other related information
- The terms "plugin" and "hook" are used interchangeably

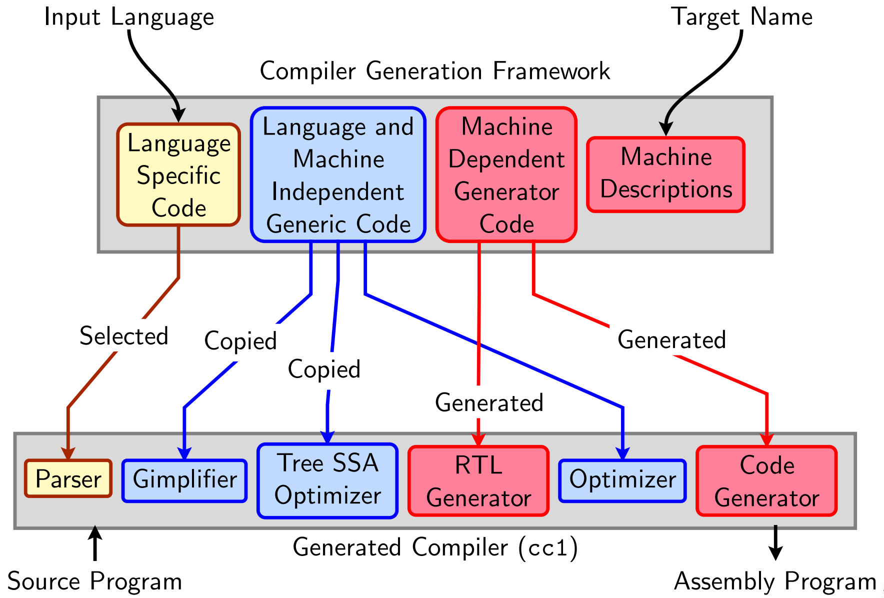
Plugins in the GCC Driver



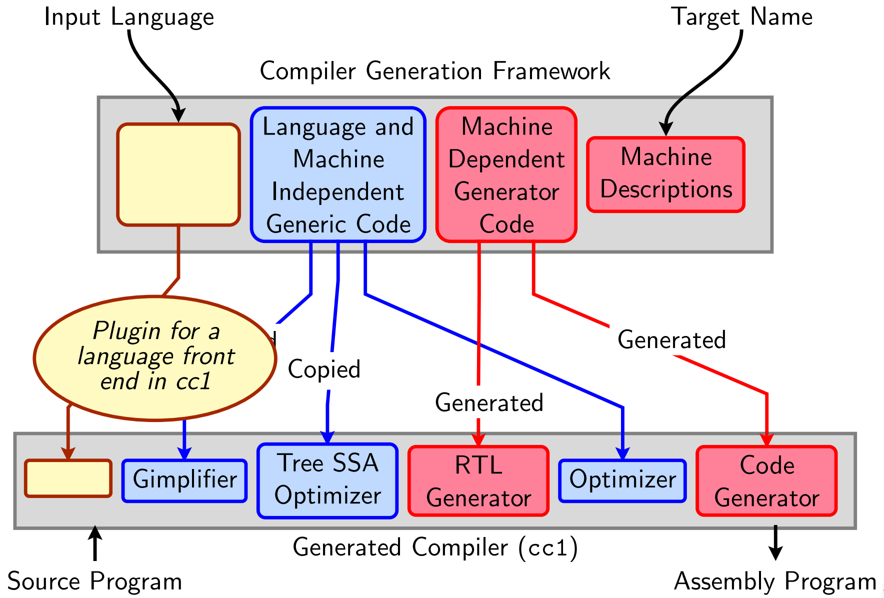
Plugins in the GCC Driver



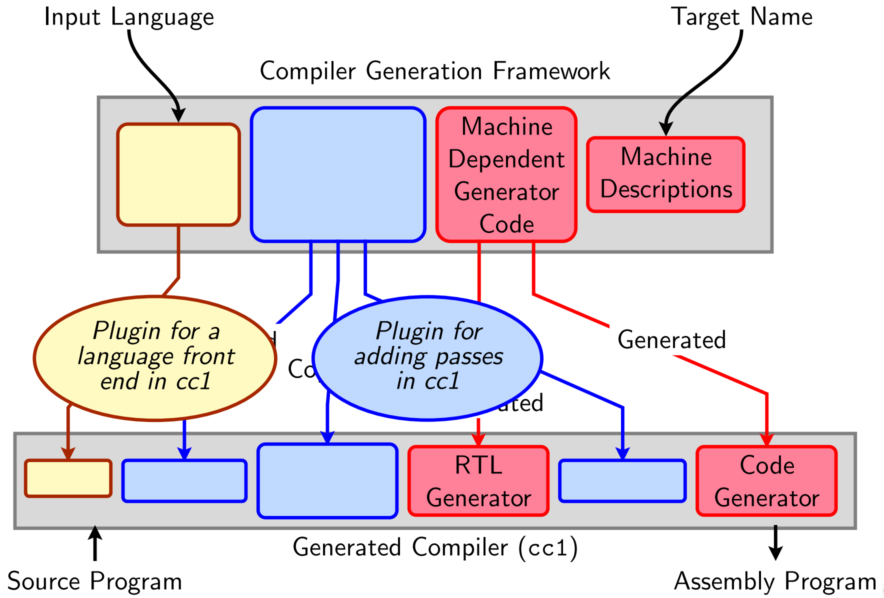
Plugins in the Generated Compiler



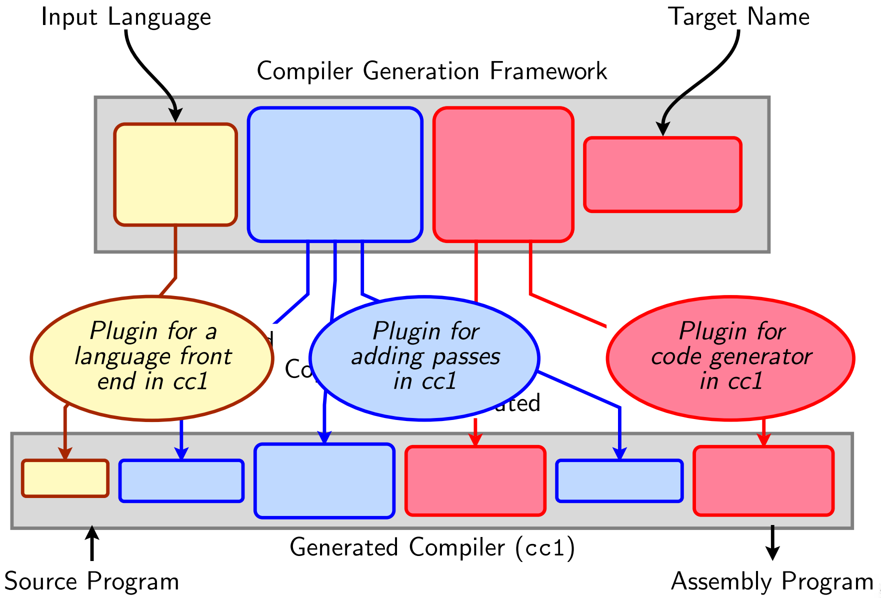
Plugins in the Generated Compiler



Plugins in the Generated Compiler



Plugins in the Generated Compiler


Input Language
Target Name
Compiler Generation Framework
Plugin for a language front end in cc1
Plugin for adding passes in cc1
Plugin for code generator in cc1
Generated Compiler (cc1)
Source Program
Assembly Program

Part 2

GCC Plugins

GCC's Solution

Plugin	Implementation	
	Data Structure	Initialization
Translator in gcc	Array of C structures	Development time
Front end in cc1	C structure	Build time
Passes in cc1	Linked list of C structures	Development time
Back end in cc1	Arrays of structures	Build time

Plugin Data Structure in the GCC Driver

```
struct compiler
{
  const char *suffix;       /* Use this compiler for input files
                               whose names end in this suffix.  */

  const char *spec;         /* To use this compiler, run this spec. *

  const char *cpp_spec;     /* If non-NULL, substitute this spec
                               for ‘%C’, rather than the usual
                               cpp_spec.  */
  const int combinable;     /* If nonzero, compiler can deal with
                               multiple source files at once (IMA).
  const int needs_preprocessing;
                            /* If nonzero, source files need to
                               be run through a preprocessor.  */
};
```

Populated Plugin Data Structure in the GCC Driver

All entries of Objective C/C++ and some entries of Fortran removed.

```
static const struct compiler default_compilers[] =
{
  {".cc", "#C++", 0, 0, 0},             {".cxx", "#C++", 0, 0, 0},
  {".cpp", "#C++", 0, 0, 0},            {".cp", "#C++", 0, 0, 0},
  {".c++", "#C++", 0, 0, 0},            {".C", "#C++", 0, 0, 0},
  {".CPP", "#C++", 0, 0, 0},            {".ii", "#C++", 0, 0, 0},
  {".ads", "#Ada", 0, 0, 0},            {".adb", "#Ada", 0, 0, 0},
  {".f", "#Fortran", 0, 0, 0},          {".F", "#Fortran", 0, 0, 0},
  {".for", "#Fortran", 0, 0, 0},        {".FOR", "#Fortran", 0, 0, 0},
  {".f90", "#Fortran", 0, 0, 0},        {".F90", "#Fortran", 0, 0, 0},
  {".p", "#Pascal", 0, 0, 0},           {".pas", "#Pascal", 0, 0, 0},
  {".java", "#Java", 0, 0, 0},          {".class", "#Java", 0, 0, 0},
  {".c", "@c", 0, 1, 1},
  {".h", "@c-header", 0, 0, 0},
  {".i", "@cpp-output", 0, 1, 0},
  {".s", "@assembler", 0, 1, 0}
}
```

Populated Plugin Data Structure in the GCC Driver

All entries of Objective C/C++ and some entries of Fortran removed.

```
static const struct compiler default_compilers[] =
{
  {".cc", "#C++", 0, 0, 0},        {".cxx", "#C++", 0, 0, 0},
  {".cpp", "#C++", 0, 0, 0},       {".cp", "#C++", 0, 0, 0},
  {".c++", "#C++", 0,                            , 0},
  {".CPP", "#C++", 0,                            0, 0},
  {".ads", "#Ada", 0,                             0, 0},
  {".f", "#Fortran", 0                           0, 0, 0},
  {".for", "#Fortran",                           , 0, 0, 0},
  {".f90", "#Fortran",                           , 0, 0, 0},
  {".p", "#Pascal", 0,                            0, 0, 0},
  {".java", "#Java", 0                            0, 0, 0},
  {".c", "@c", 0, 1, 1
  {".h", "@c-header",
  {".i", "@cpp-output", 0, 1, 0},
  {".s", "@assembler", 0, 1, 0}
}
```

What about linker files?

Populated Plugin Data Structure in the GCC Driver

All entries of Objective C/C++ and some entries of Fortran removed.

```
static const struct compiler default_compilers[] =
{
  {".cc", "#C++", 0, 0, 0},        {".cxx", "#C++", 0, 0, 0},
  {".cpp", "#C++", 0, 0, 0},       {".cp", "#C++", 0, 0, 0},
  {".c++", "#C++", 0,                                   , 0},
  {".CPP", "#C++", 0,                                  0, 0},
  {".ads", "#Ada", 0,                                   0, 0},
  {".f", "#Fortran", 0                                 0, 0, 0},
  {".for", "#Fortran",                                  , 0, 0, 0},
  {".f90", "#Fortran",                                  , 0, 0, 0},
  {".p", "#Pascal", 0,                                  0, 0, 0},
  {".java", "#Java", 0                                  0, 0, 0},
  {".c", "@c", 0, 1, 1
  {".h", "@c-header",
  {".i", "@cpp-output", 0, 1, 0},
  {".s", "@assembler", 0, 1, 0}
}
```

What about linker files?

- Linking is the last step

Populated Plugin Data Structure in the GCC Driver

All entries of Objective C/C++ and some entries of Fortran removed.

```
static const struct compiler default_compilers[] =
{
  {".cc", "#C++", 0, 0, 0},        {".cxx", "#C++", 0, 0, 0},
  {".cpp", "#C++", 0, 0, 0},       {".cp", "#C++", 0, 0, 0},
  {".c++", "#C++", 0,                         , 0},
  {".CPP", "#C++", 0,                         0, 0},
  {".ads", "#Ada", 0,                          0, 0},
  {".f", "#Fortran", 0                        0, 0, 0},
  {".for", "#Fortran",                        , 0, 0, 0},
  {".f90", "#Fortran",                        , 0, 0, 0},
  {".p", "#Pascal", 0,                         0, 0, 0},
  {".java", "#Java", 0                         0, 0, 0},
  {".c", "@c", 0, 1, 1
  {".h", "@c-header",
  {".i", "@cpp-output", 0, 1, 0},
  {".s", "@assembler", 0, 1, 0}
}
```

What about linker files?

- Linking is the last step
- Every file is passed on to linker unless it is suppressed

Populated Plugin Data Structure in the GCC Driver

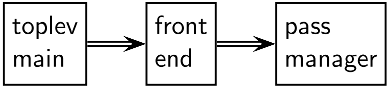
All entries of Objective C/C++ and some entries of Fortran removed.

```
static const struct compiler default_compilers[] =
{
  {".cc", "#C++", 0, 0, 0},          {".cxx", "#C++", 0, 0, 0},
  {".cpp", "#C++", 0, 0, 0},         {".cp", "#C++", 0, 0, 0},
  {".c++", "#C++", 0,                              , 0},
  {".CPP", "#C++", 0,                             0, 0},
  {".ads", "#Ada", 0,                              0, 0},
  {".f", "#Fortran", 0                          0, 0, 0},
  {".for", "#Fortran",                           , 0, 0, 0},
  {".f90", "#Fortran",                           , 0, 0, 0},
  {".p", "#Pascal", 0,                            0, 0, 0},
  {".java", "#Java", 0                            0, 0, 0},
  {".c", "@c", 0, 1, 1
  {".h", "@c-header",
  {".i", "@cpp-output", 0, 1, 0},
  {".s", "@assembler", 0, 1, 0}
}
```

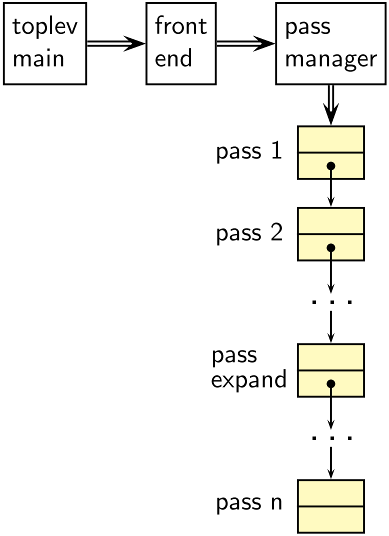
What about linker files?

- Linking is the last step
- Every file is passed on to linker unless it is suppressed
- If a translator is not found, input file is assumed to be a file for linker

Plugin Structure in `cc1`

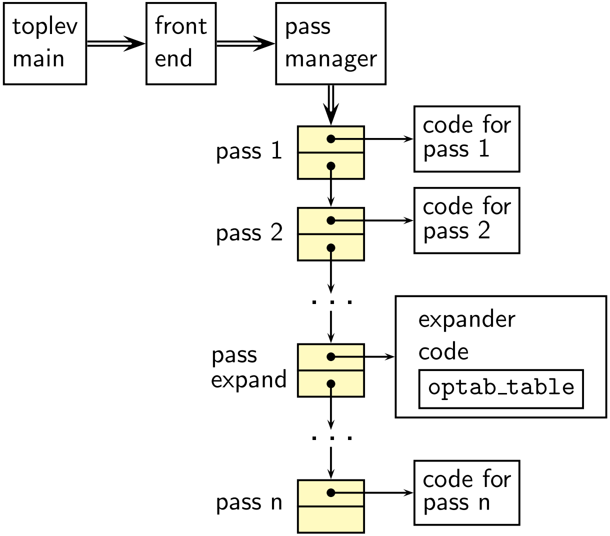


Plugin Structure in `cc1`

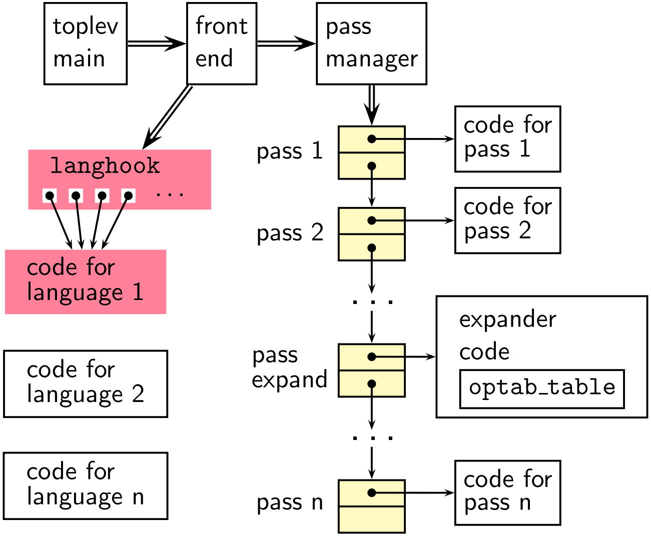


double arrow represents control flow whereas single arrow represents pointer or index

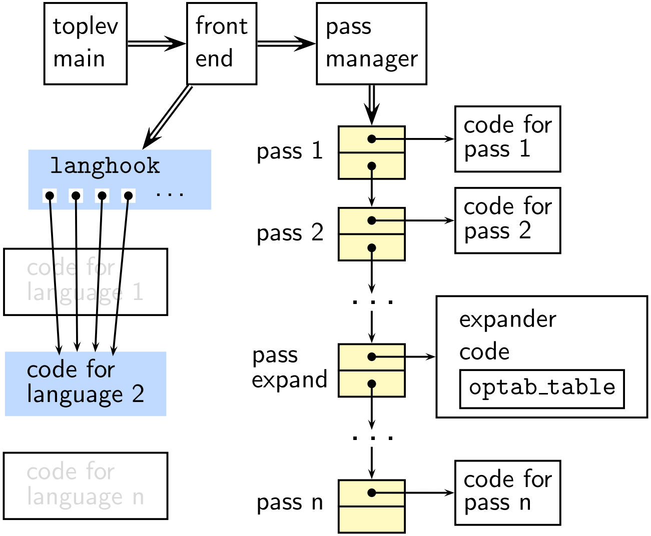
Plugin Structure in `cc1`



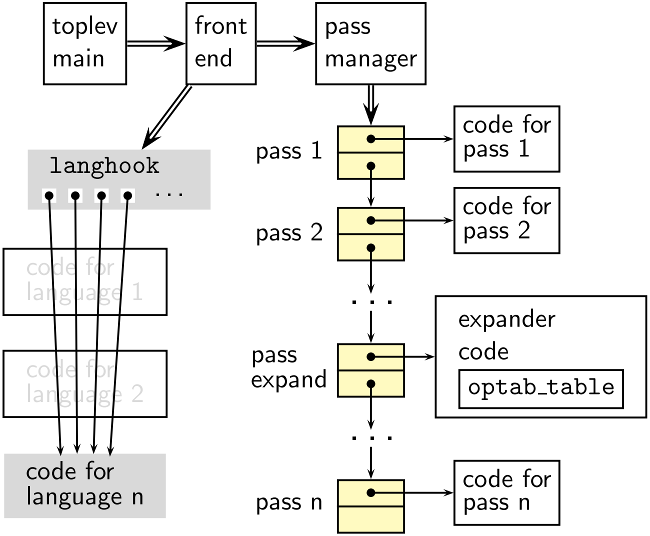
Plugin Structure in `cc1`



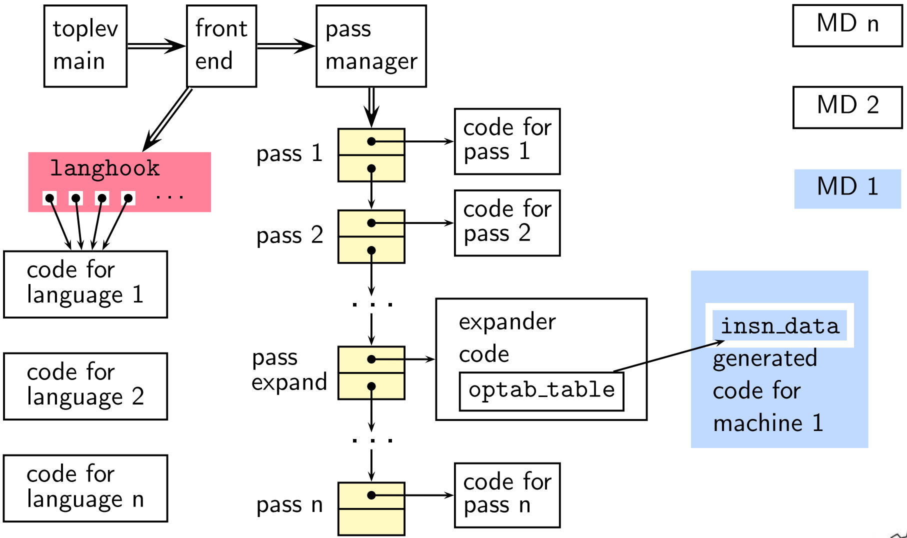
Plugin Structure in `cc1`



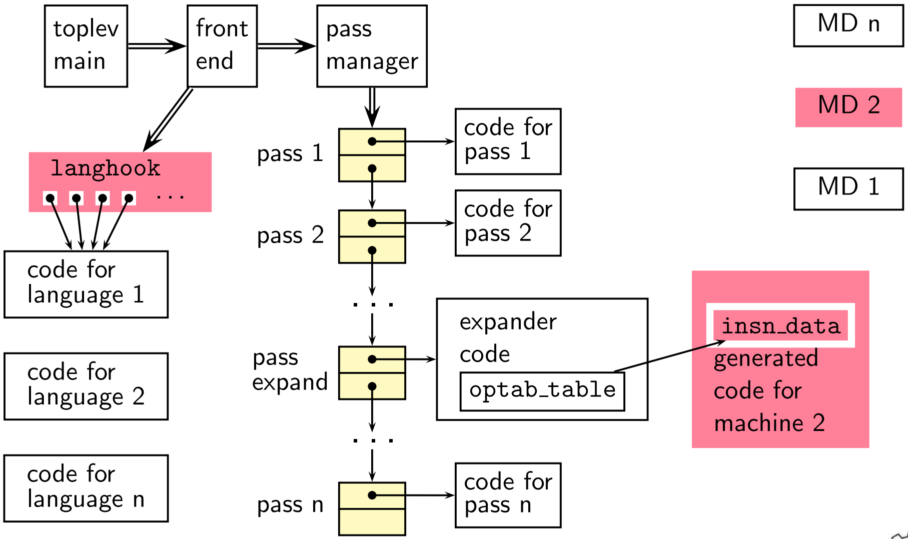
Plugin Structure in `cc1`



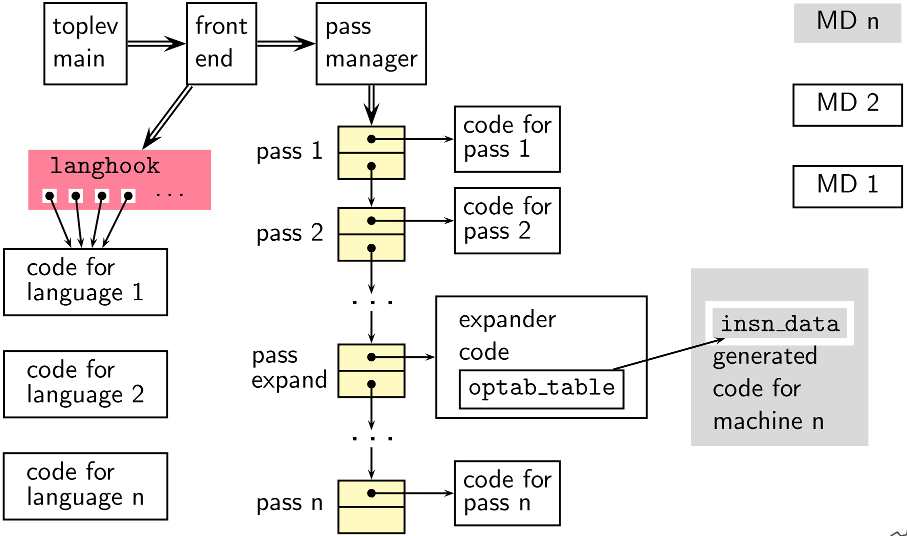
Plugin Structure in `cc1`



Plugin Structure in `cc1`



Plugin Structure in `cc1`



Front End Plugin

Important fields of `struct lang_hooks` instantiated for C

```
#define LANG_HOOKS_FINISH c_common_finish
#define LANG_HOOKS_EXPAND_EXPR c_expand_expr
#define LANG_HOOKS_PARSE_FILE c_common_parse_file
#define LANG_HOOKS_WRITE_GLOBALS c_write_global_declarations
```

Plugins for Intraprocedural Passes

```
struct opt_pass
{
  enum opt_pass_type type;
  const char *name;
  bool (*gate) (void);
  unsigned int (*execute) (void);
  struct opt_pass *sub;
  struct opt_pass *next;
  int static_pass_number;
  timevar_id_t tv_id;
  unsigned int properties_required;
  unsigned int properties_provided;
  unsigned int properties_destroyed;
  unsigned int todo_flags_start;
  unsigned int todo_flags_finish;
};
```

```
struct gimple_opt_pass
{
  struct opt_pass pass;
};

struct rtl_opt_pass
{
  struct opt_pass pass;
};
```

Plugins for Interprocedural Passes

```
struct ipa_opt_pass_d
{
  struct opt_pass pass;
  void (*generate_summary) (void);
  void (*write_summary) (struct cgraph_node_set_def *);
  void (*read_summary) (void);
  void (*function_read_summary) (struct cgraph_node *);
  void (*stmt_fixup) (struct cgraph_node *, gimple *);
  unsigned int function_transform_todo_flags_start;
  unsigned int (*function_transform) (struct cgraph_node *);
  void (*variable_transform) (struct varpool_node *);
};

struct simple_ipa_opt_pass
{
  struct opt_pass pass;
};
```

Part 3

GCC Control Flow

gcc Driver Control Flow

```
main   /* In file gcc.c */
   validate_all_switches
   lookup_compiler
   do_spec
      do_spec_2
         do_spec_1  /* Get the name of the compiler */
      execute
         pex_init
         pex_run
            pex_run_in_environment
               obj->funcs->exec_child
```

gcc Driver Control Flow

```
main   /* In file gcc.c */
   validate_all_switches
   lookup_compiler
   do_spec
      do_spec_2
         do_spec_1  /*
      execute
         pex_init
         pex_run
            pex_run_in
               obj->fu
```

Observations

- All compilers are invoked by this driver
- Assembler is also invoked by this driver
- Linker is invoked in the end by default

cc1 Top Level Control Flow

```
main   /* In file toplev.c */
   toplev_main
     decode_options
     do_compile
        compile_file
          lang_hooks.parse_file => c_common_parse_file
          lang_hooks.decls.final_write_globals =>
                                   c_write_global_declarations
          targetm.asm_out.file_end
     finalize
```

cc1 Top Level Control Flow

```
main   /* In file toplev.c */
   toplev_main
     decode_options
     do_compile
        compile_file
          lang_hooks.p                                    e
          lang_hooks.d
                                                        larations

          targetm.asm_
     finalize
```

Observations

- The entire compilation is driven by functions specified in language hooks
- Bad design!

cc1 Control Flow: Parsing for C

```
lang_hooks.parse_file => c_common_parse_file
   c_parse_file
         c_parser_translation_unit
            c_parser_external_declaration
               c_parser_declaration_or_fndef
                  c_parser_declspecs /* parse declarations */
                  c_parser_compound_statement
                  finish_function    /* finish parsing */
                     c_genericize
                     cgraph_finalize_function
                     /* finalize AST of a function */
```

cc1 Control Flow: Parsing for C

```
lang_hooks.parse_file => c_common_parse_file
   c_parse_file
         c_parser_translation_unit
            c_parser_e
               c_parse
                  c_pa                                         ions */
                  c_pa
                  fini                                       ; */
                     c
                     c
                     /
```

Observations

- GCC has moved to a recursive descent parser from version 4.1.0
- Earlier parser was generated using Bison specification

cc1 Control Flow: Lowering Passes for C

```
lang_hooks.decls.final_write_globals =>
                                  c_write_global_declarations
       cgraph_finalize_compilation_unit
             cgraph_analyze_functions     /* Create GIMPLE */
                cgraph_analyze_function
                     gimplify_function_tree
                        gimplify_body
                           gimplify_stmt
                              gimplify_expr
                cgraph_lower_function    /* Intraprocedural */
                   tree_lowering_passes
                      execute_pass_list (all_lowering_passes)
```

cc1 Control Flow: Lowering Passes for C

```
lang_hooks.decls.final_write_globals =>
                                      c_write_global_declarations
       cgraph_finalize_compilation_unit
             cgraph_an                               MPLE */
                cgraph
                     g




                cgraph                                dural */
                   tre
                                                     passes)
```

Observations

- Lowering passes are language independent
- Yet they are being called from a function in language hooks
- Bad design!

cc1 Control Flow: Optimization and Code Generation Passes

```
lang_hooks.decls.final_write_globals =>
                                   c_write_global_declarations
    cgraph_finalize_compilation_unit
        cgraph_analyze_function      /* Create GIMPLE */
        cgraph_optimize
           ipa_passes
           cgraph_expand_all_functions
                cgraph_expand_function
                /* Intraprocedural passes on GIMPLE,  */
                /* expansion pass, and passes on RTL. */
                     tree_rest_of_compilation
                         execute_pass_list (&all_passes)
```

cc1 Control Flow: Optimization and Code Generation Passes

```
lang_hooks.decls.final_write_globals =>
                                   c_write_global_declarations
    cgraph_finalize_co
        cgraph_analyze                                         */
        cgraph_optimiz
           ipa_passes
           cgraph_expa
                cgraph
                /* Int                                        */
                /* exp                                        */
                     t
                                                              s)
```

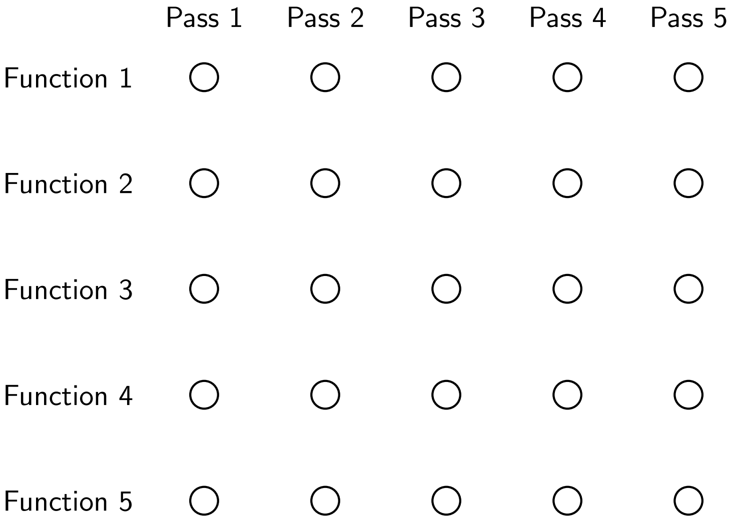
Observations

- Optimization and code generation passes are language independent
- Yet they are being called from a function in language hooks
- Bad design!

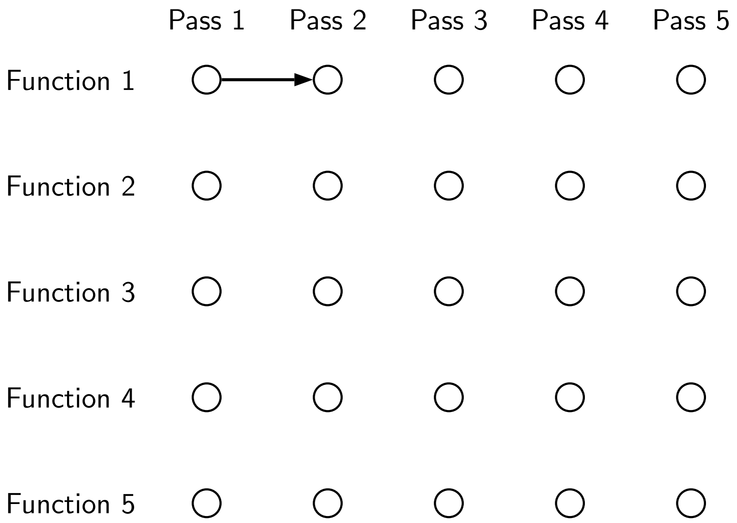
Organization of Passes

Order	Task	IR	Level	Pass data structure
1	Lowering	GIMPLE	Intraproc.	`struct gimple_opt_pass`
2	Optimizations	GIMPLE	Interproc.	`struct ipa_opt_pass`
3	Optimizations	GIMPLE	Intraproc.	`struct gimple_opt_pass`
4	RTL Generation	GIMPLE	Intraproc.	`struct rtl_opt_pass`
5	Optimization	RTL	Intraproc.	`struct rtl_opt_pass`

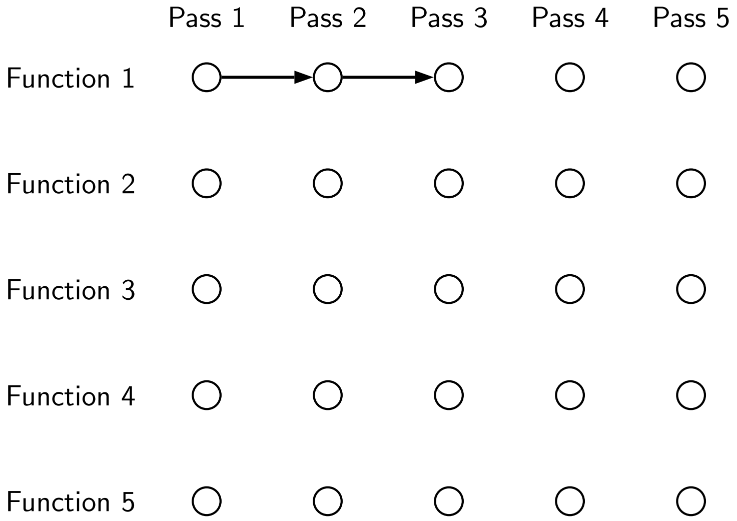
Execution Order in Intraprocedural Passes



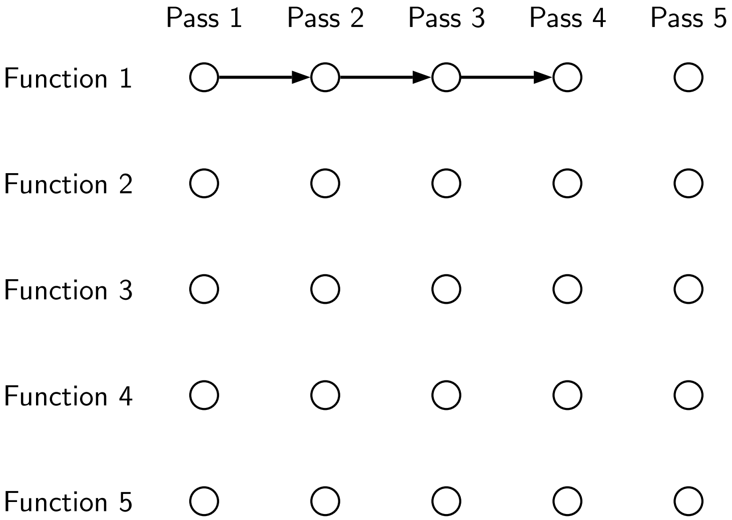
Execution Order in Intraprocedural Passes



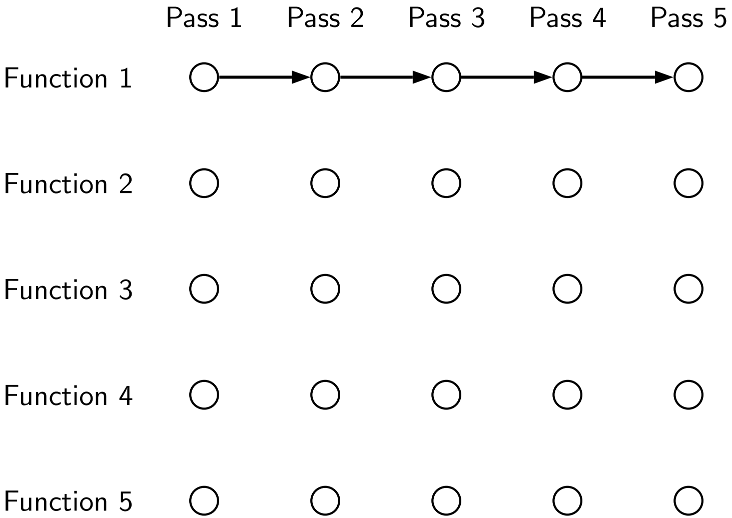
Execution Order in Intraprocedural Passes



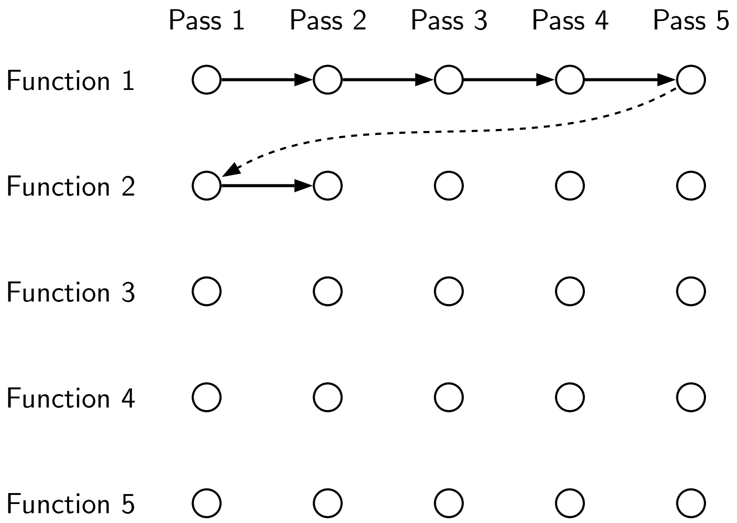
Execution Order in Intraprocedural Passes



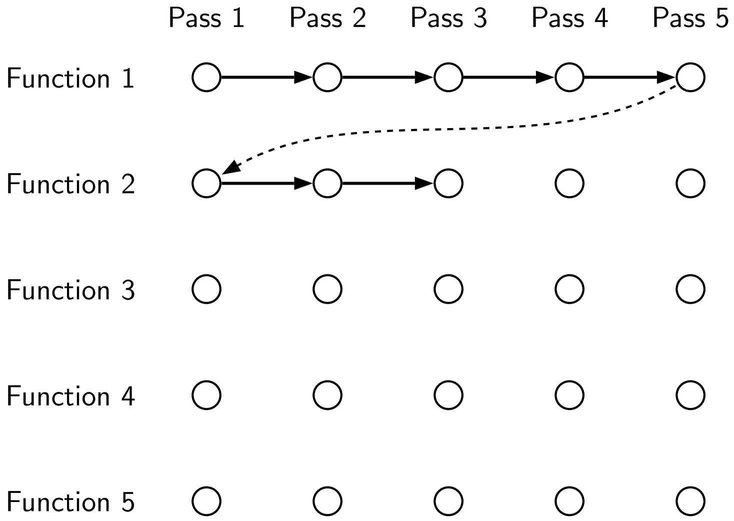
Execution Order in Intraprocedural Passes



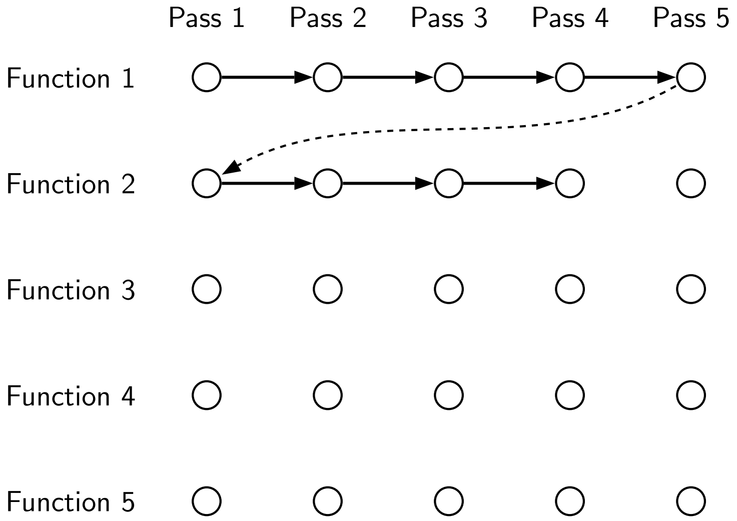
Execution Order in Intraprocedural Passes



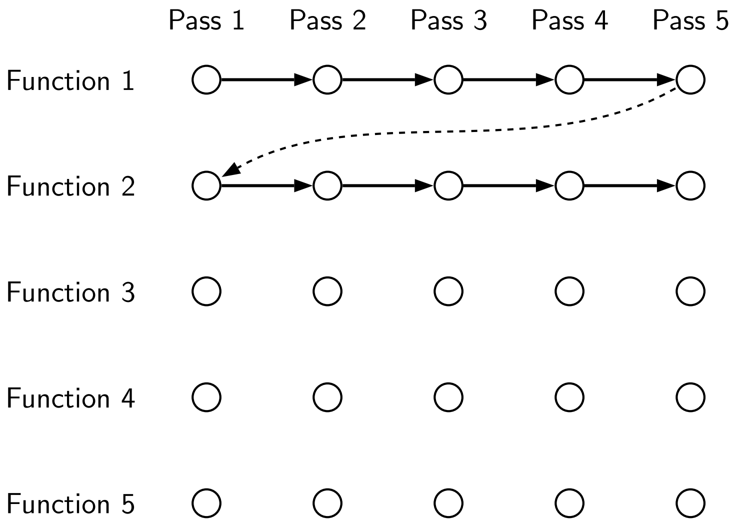
Execution Order in Intraprocedural Passes



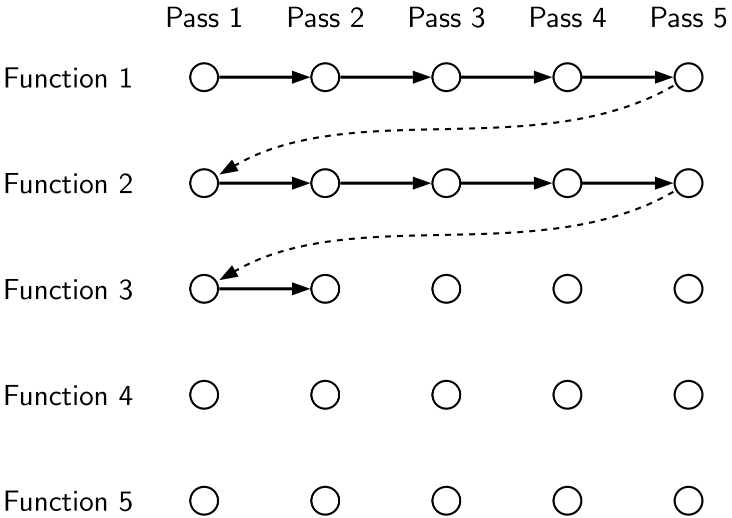
Execution Order in Intraprocedural Passes



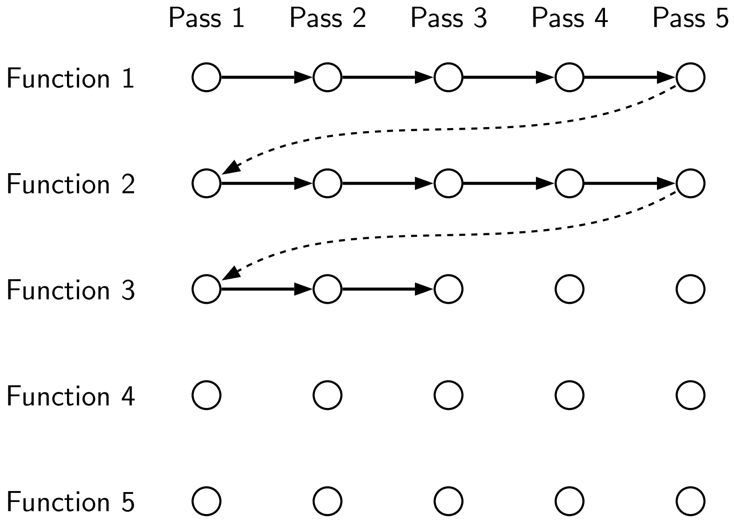
Execution Order in Intraprocedural Passes



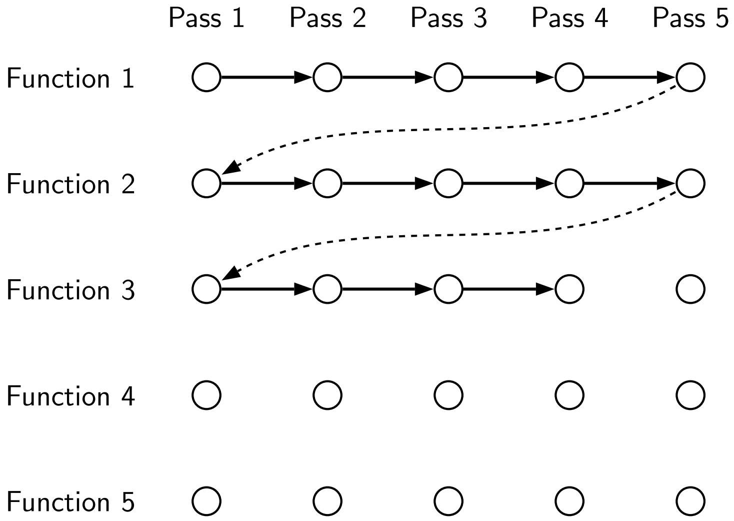
Execution Order in Intraprocedural Passes



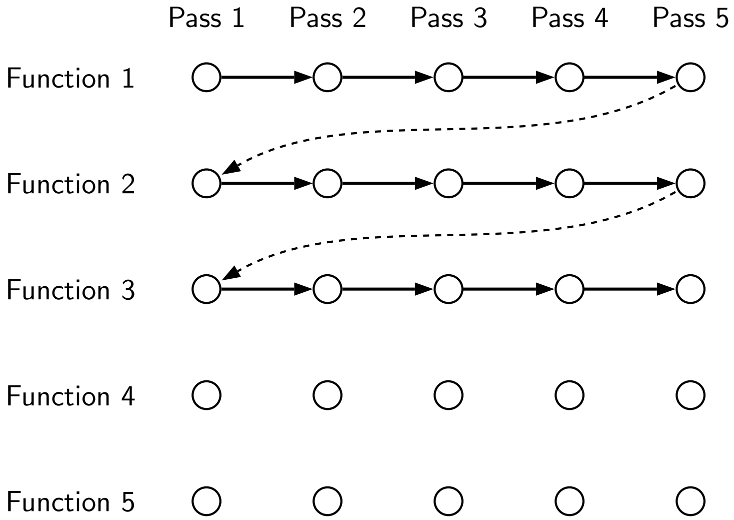
Execution Order in Intraprocedural Passes



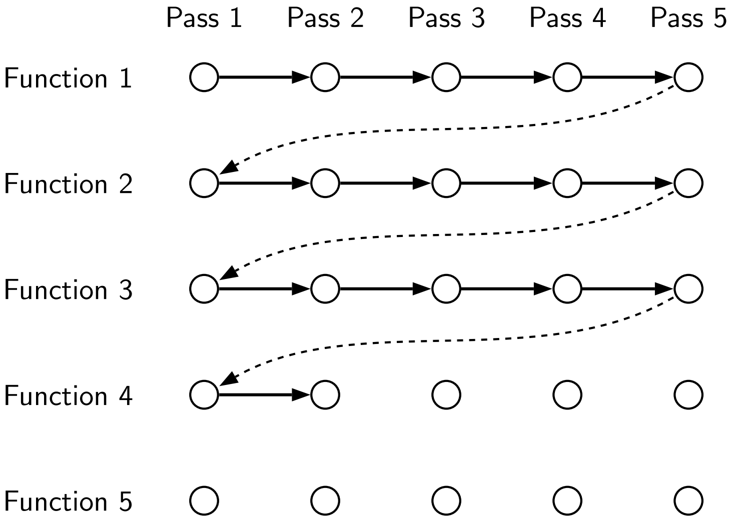
Execution Order in Intraprocedural Passes



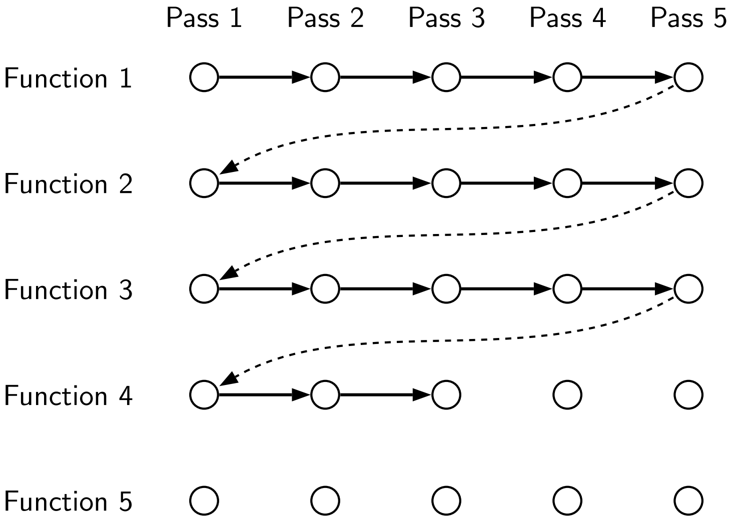
Execution Order in Intraprocedural Passes



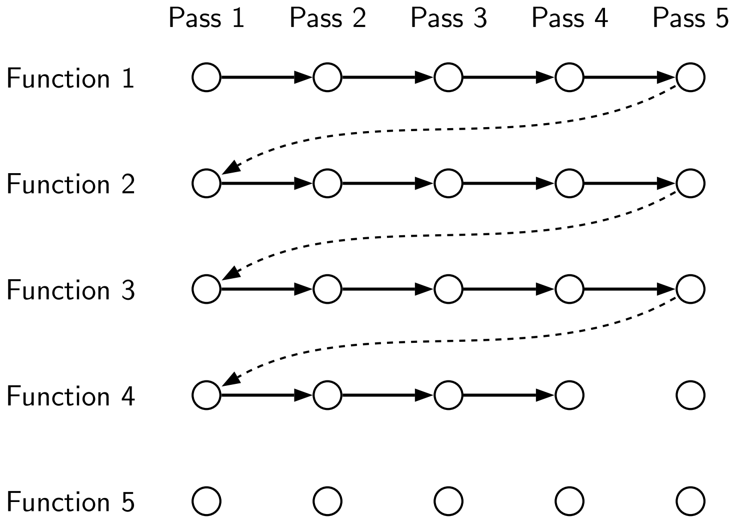
Execution Order in Intraprocedural Passes



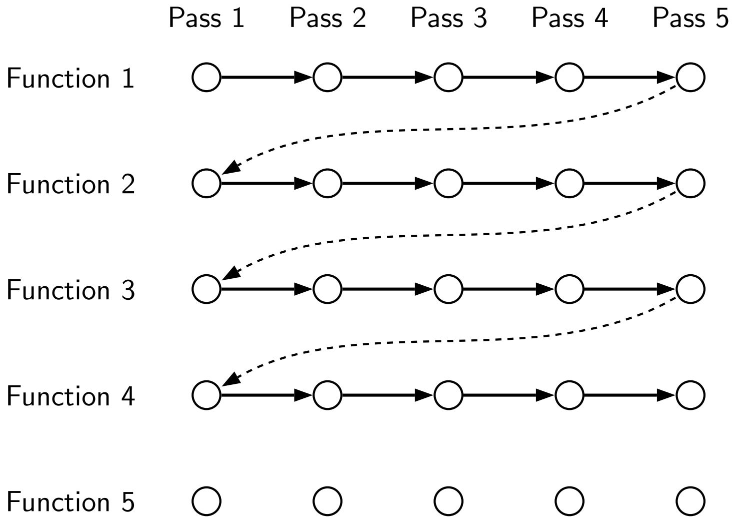
Execution Order in Intraprocedural Passes



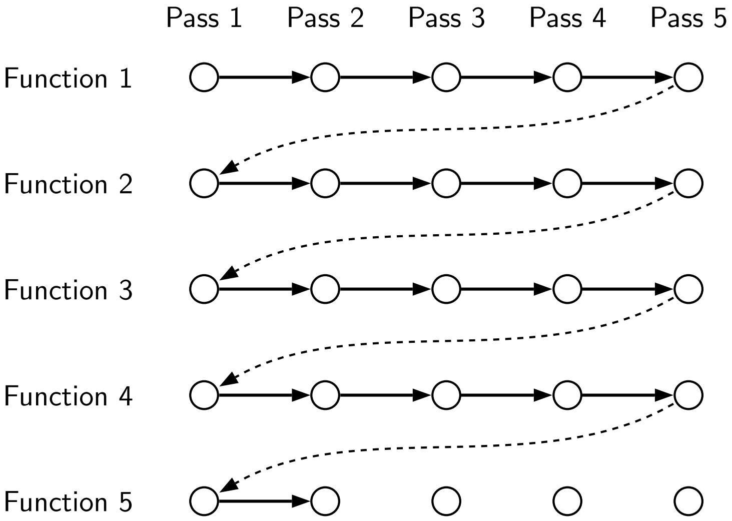
Execution Order in Intraprocedural Passes



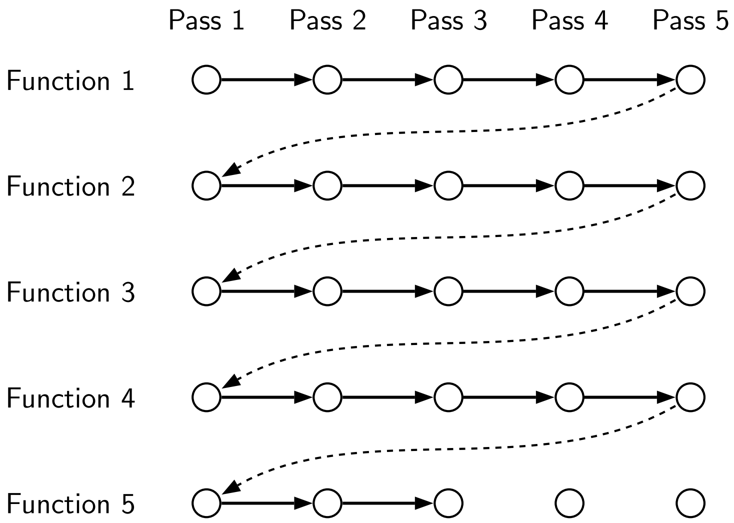
Execution Order in Intraprocedural Passes



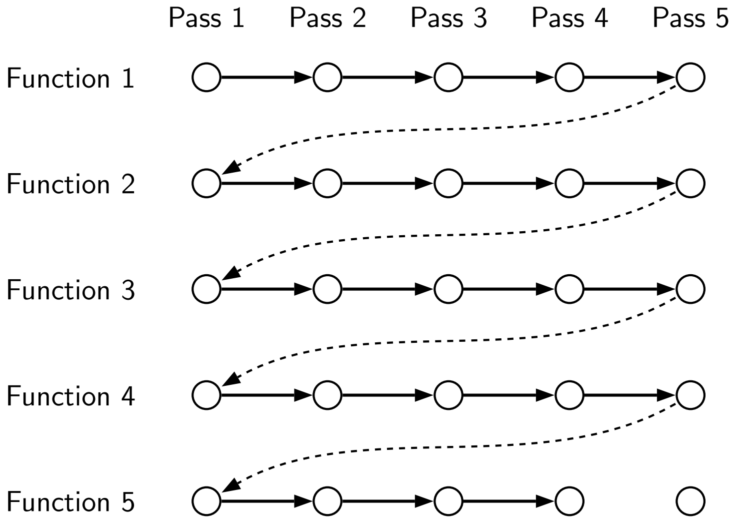
Execution Order in Intraprocedural Passes



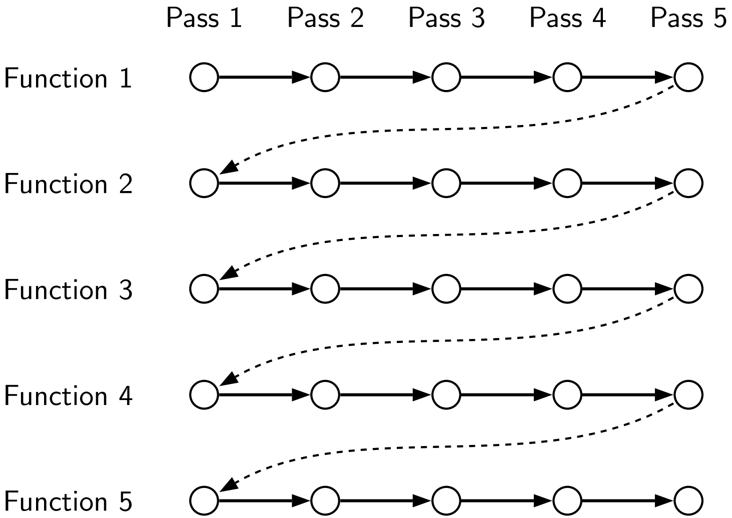
Execution Order in Intraprocedural Passes



Execution Order in Intraprocedural Passes



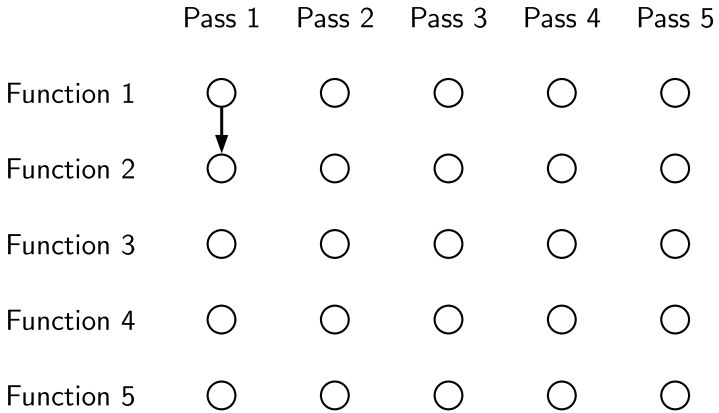
Execution Order in Intraprocedural Passes



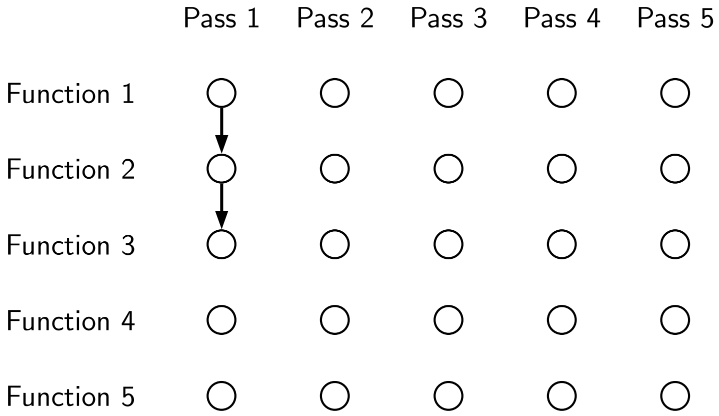
Execution Order in Interprocedural Passes

	Pass 1	Pass 2	Pass 3	Pass 4	Pass 5
Function 1	○	○	○	○	○
Function 2	○	○	○	○	○
Function 3	○	○	○	○	○
Function 4	○	○	○	○	○
Function 5	○	○	○	○	○

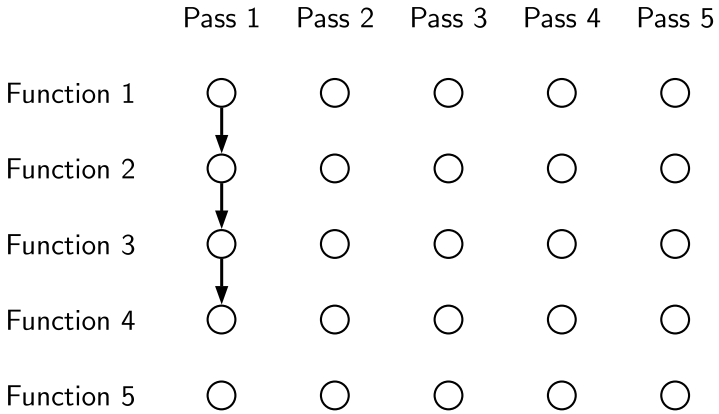
Execution Order in Interprocedural Passes



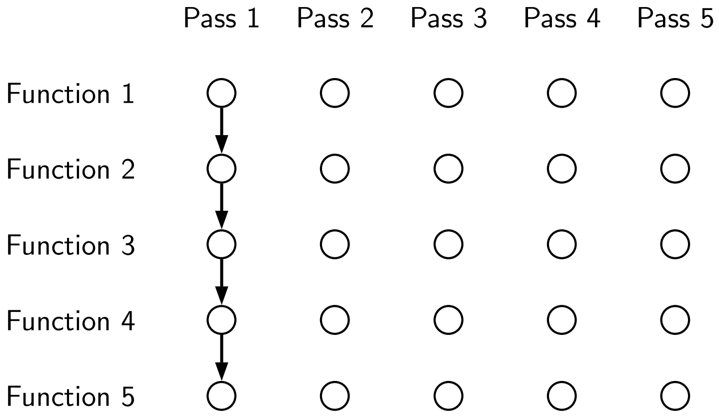
Execution Order in Interprocedural Passes



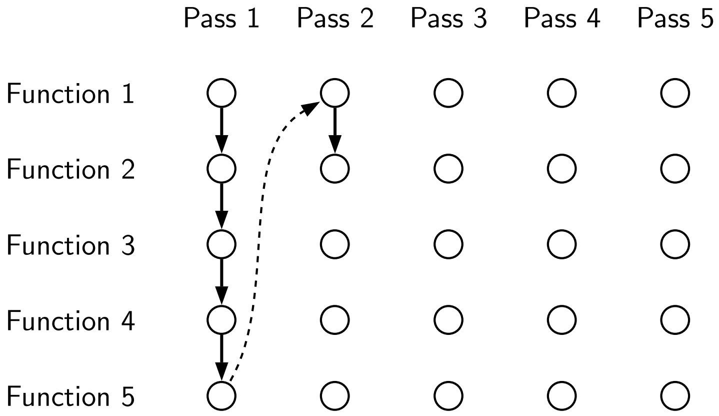
Execution Order in Interprocedural Passes



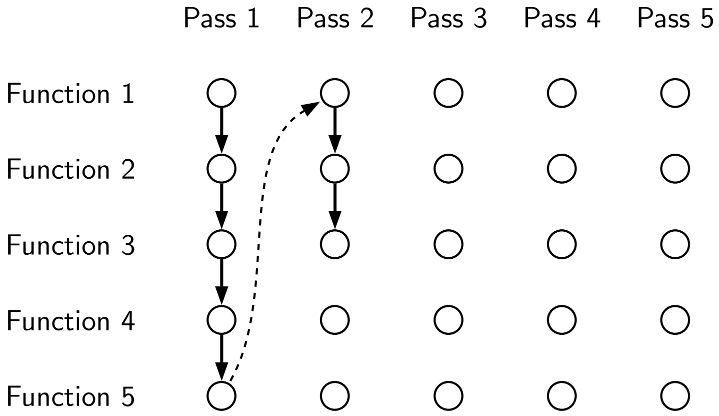
Execution Order in Interprocedural Passes



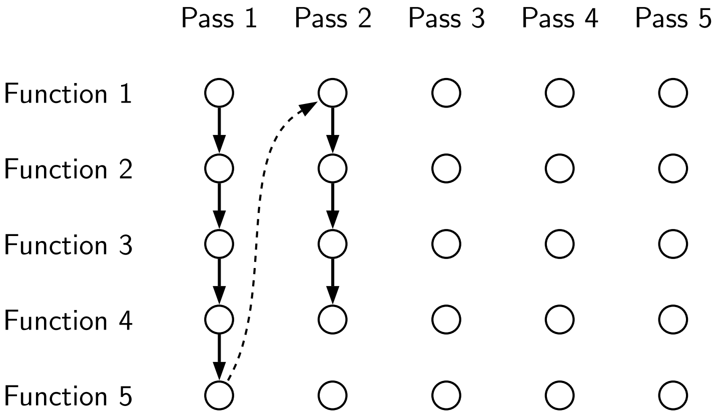
Execution Order in Interprocedural Passes



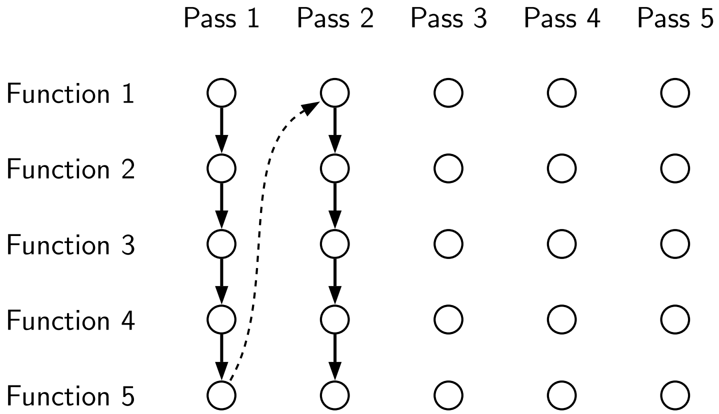
Execution Order in Interprocedural Passes



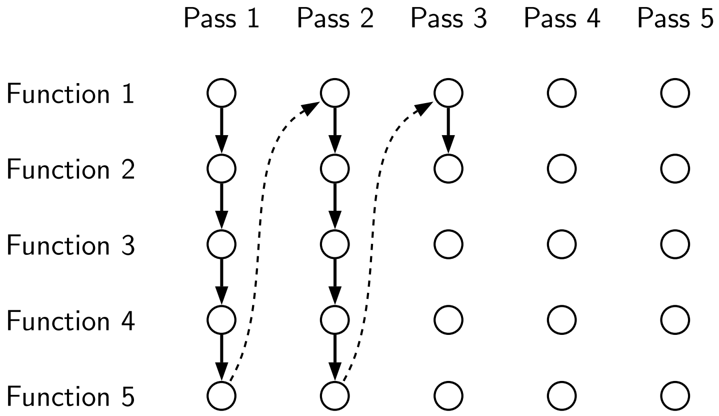
Execution Order in Interprocedural Passes



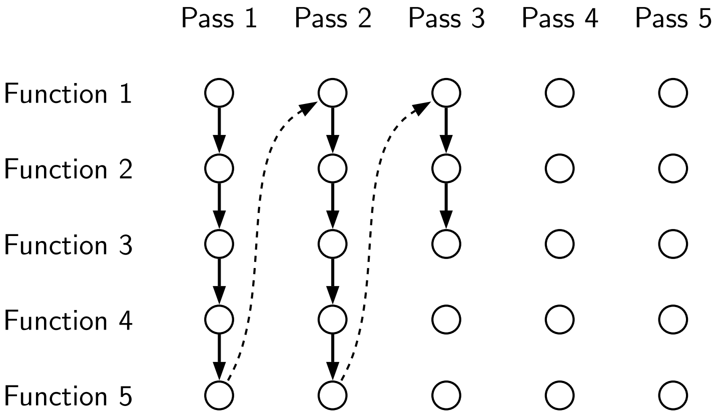
Execution Order in Interprocedural Passes



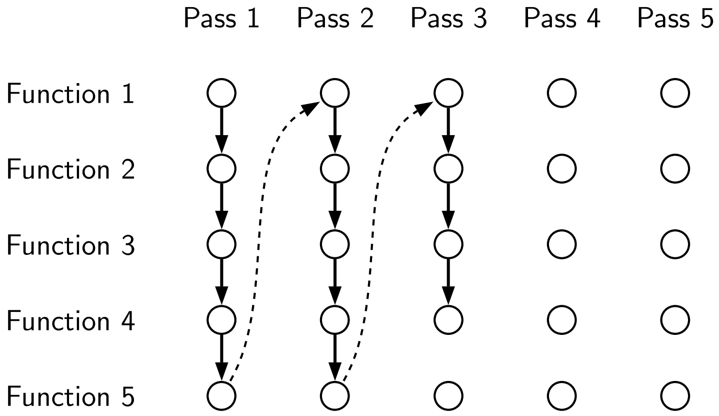
Execution Order in Interprocedural Passes



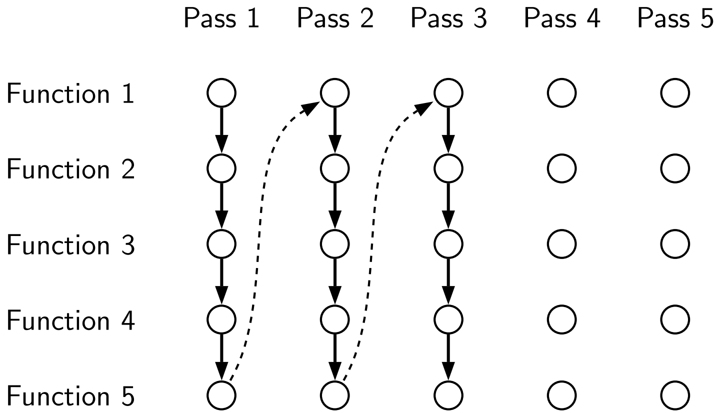
Execution Order in Interprocedural Passes



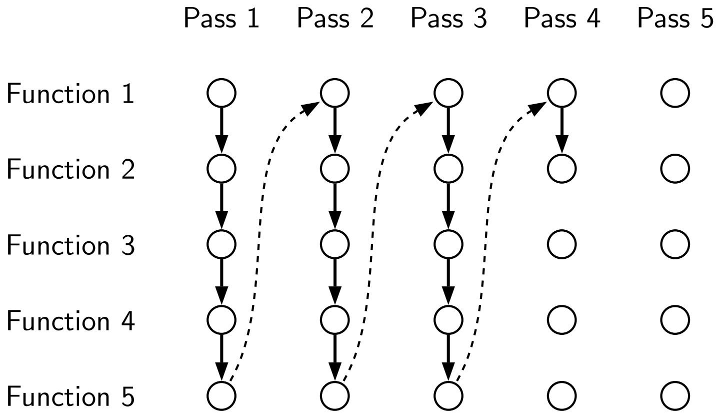
Execution Order in Interprocedural Passes



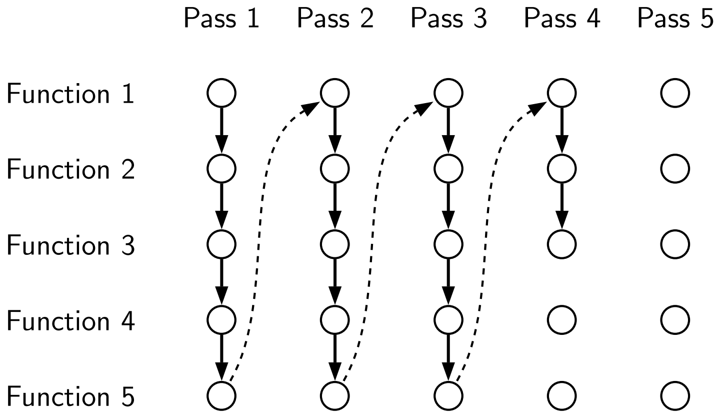
Execution Order in Interprocedural Passes



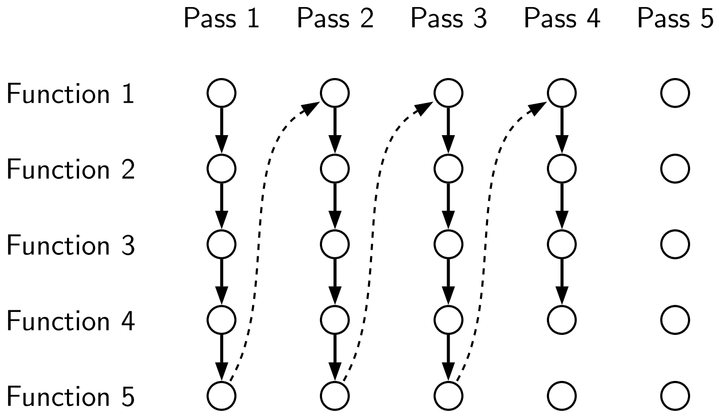
Execution Order in Interprocedural Passes



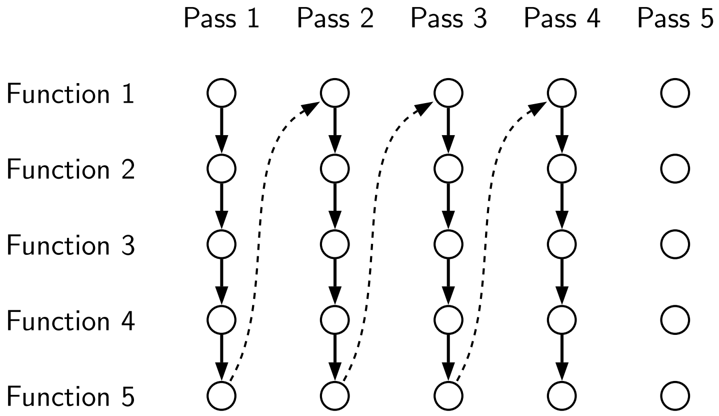
Execution Order in Interprocedural Passes



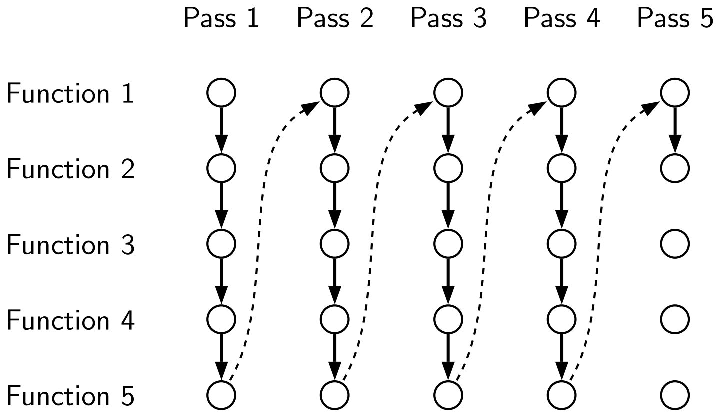
Execution Order in Interprocedural Passes



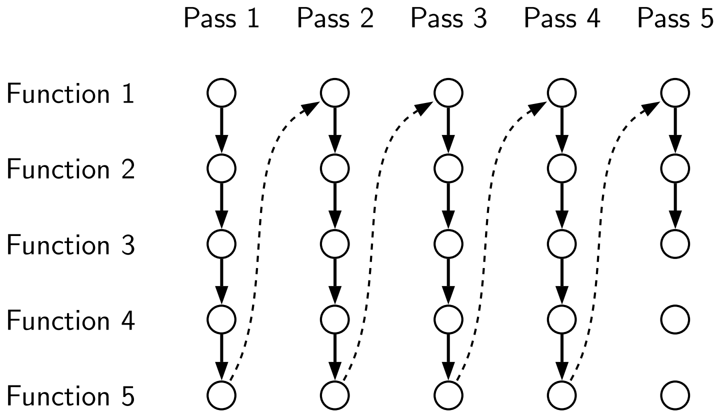
Execution Order in Interprocedural Passes



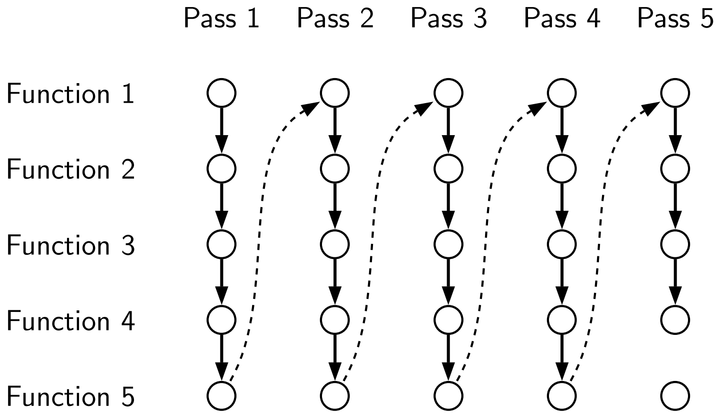
Execution Order in Interprocedural Passes



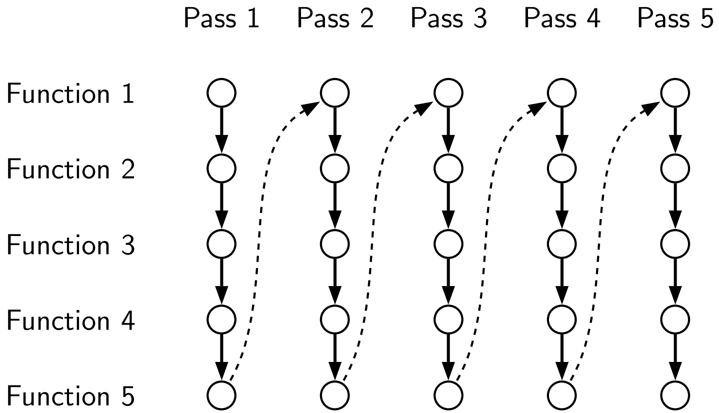
Execution Order in Interprocedural Passes



Execution Order in Interprocedural Passes



Execution Order in Interprocedural Passes



cc1 Control Flow: GIMPLE to RTL Expansion (pass_expand)

```
gimple_expand_cfg
  expand_gimple_basic_block(bb)
     expand_gimple_cond(stmt)
     expand_gimple_stmt(stmt)
         expand_gimple_stmt_1 (stmt)
             expand_expr_real_2
                expand_expr   /* Operands */
                    expand_expr_real
                optab_for_tree_code
                expand_binop /* Now we have rtx for operands */
                   expand_binop_directly
                     /* The plugin for a machine */
                     code=optab_handler(binoptab,mode)->insn_code;
                     GEN_FCN
                     emit_insn
```

Part 4

Conclusions

Conclusions

- Excellent mechanism of plugging in different
  - translators in the main driver
  - front ends, passes, and back ends in the main compiler
- However, the plugins have been used in an adhoc manner