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

30 June 2013

Outline

- Motivation
- Plugins in GCC
- GCC Control Flow
- Link time optimization in GCC
- Conclusions

Part 1

Motivation

Module Binding Mechanisms

- The need for adding, removing, and maintaining modules relatively independently
- The mechanism for supporting this is called by many names:
  - Plugin, hook, callback, ...
  - Sometimes it remains unnamed (eg. compilers in gcc driver)
- It may involve
  - Minor changes in the main source
    Requires static linking
  - No changes in the main source
    Requires dynamic linking

Module Binding Mechanisms

- The need for adding, removing, and maintaining modules relatively independently
- The mechanism for supporting this is called by many names:
  - Plugin, hook, callback, ...
  - Sometimes it remains unnamed (eg. compilers in gcc driver)
- It may involve
  - Minor changes in the main source
    Requires static linking
    We call this a static plugin
  - No changes in the main source
    Requires dynamic linking
    We call this a dynamic plugin

Plugin as a Module Binding Mechanisms

- We view plugin at a more general level than the conventional view
  Adjectives "static" and "dynamic" create a good contrast

- Most often a plugin in a C based software is a data structure containing function pointers and other related information

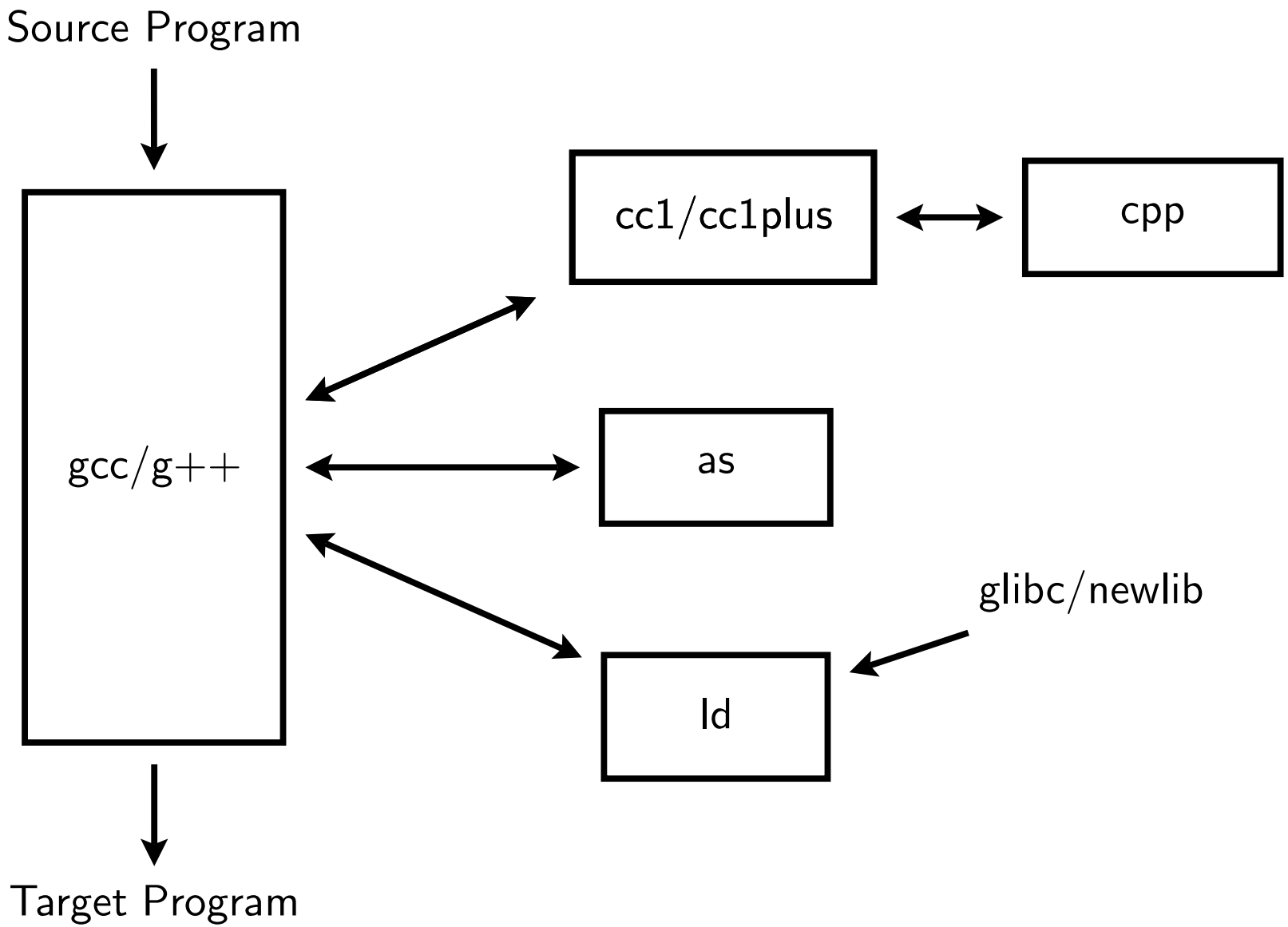
Static Vs. Dynamic Plugins

- Static plugin requires static linking
  - Changes required in `gcc/Makefile.in`, some header and source files
  - At least `cc1` may have to be rebuild
    All files that include the changed headers will have to be recompiled
- Dynamic plugin uses dynamic linking
  - Supported on platforms that support `-ldl -rdynamic`
  - Loaded using `dlopen` and invoked at pre-determined locations in the compilation process
  - Command line option

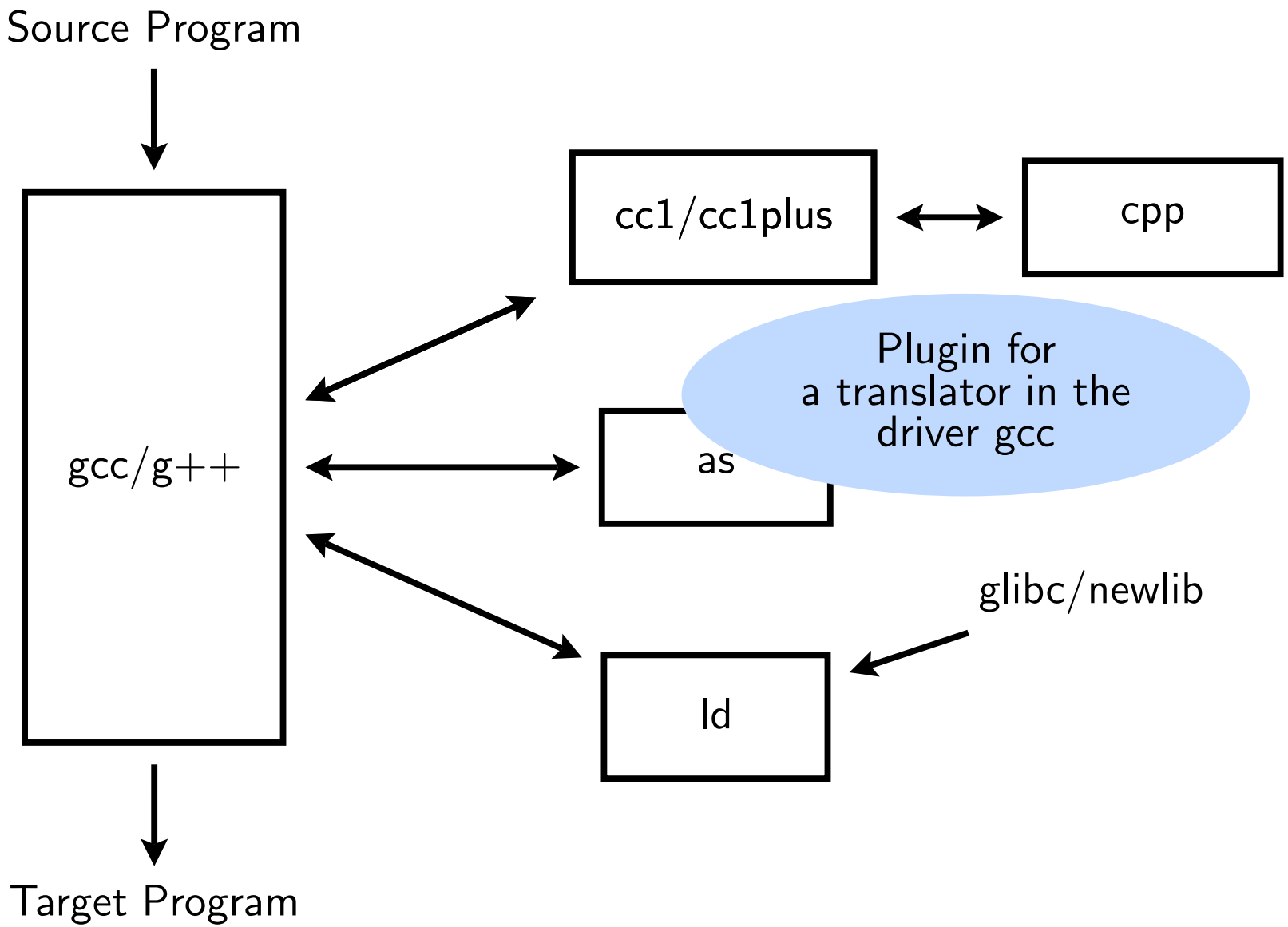
    `-fplugin=/path/to/name.so`

    Arguments required can be supplied as name-value pairs

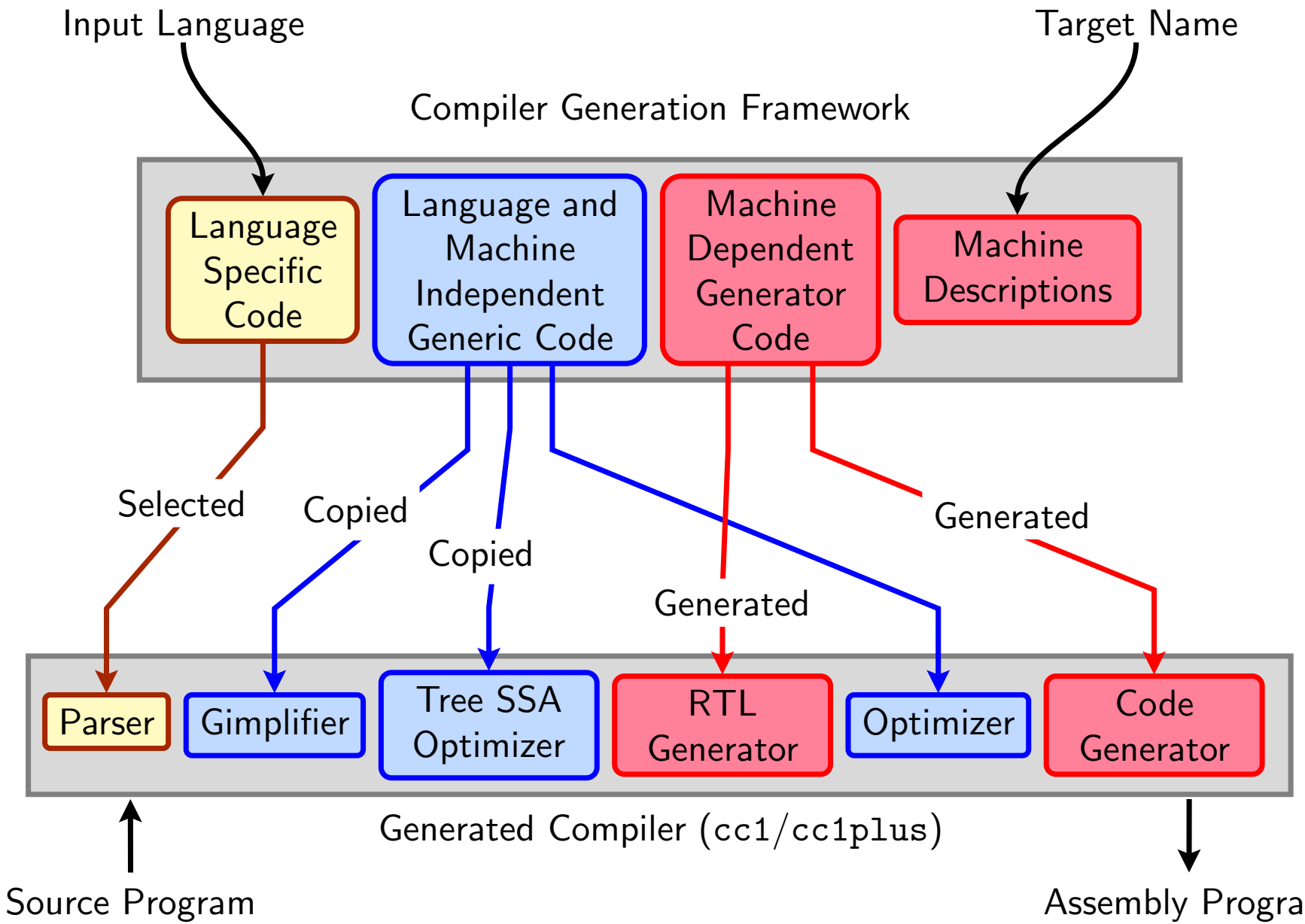
Static Plugins in the GCC Driver



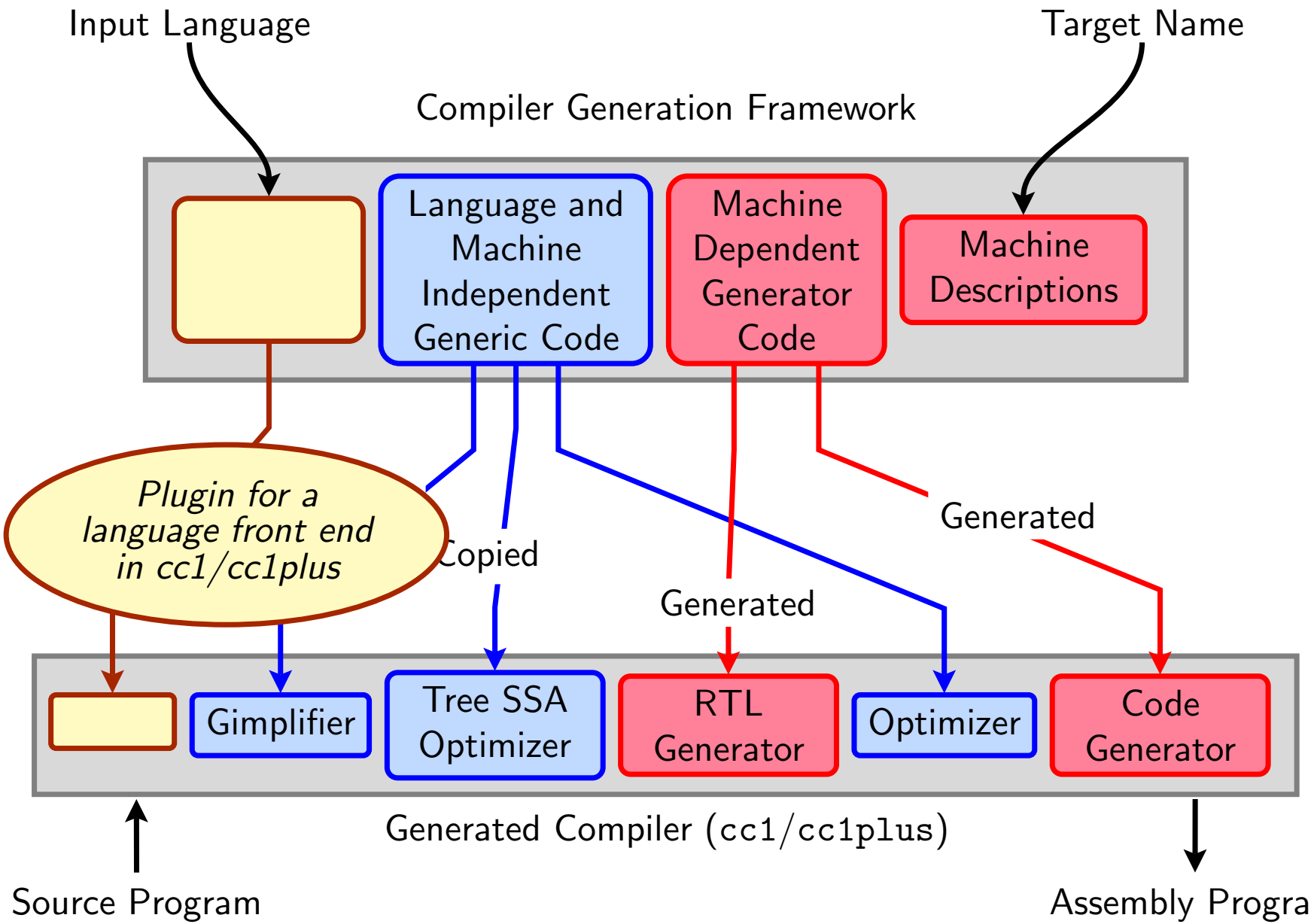
Static Plugins in the GCC Driver



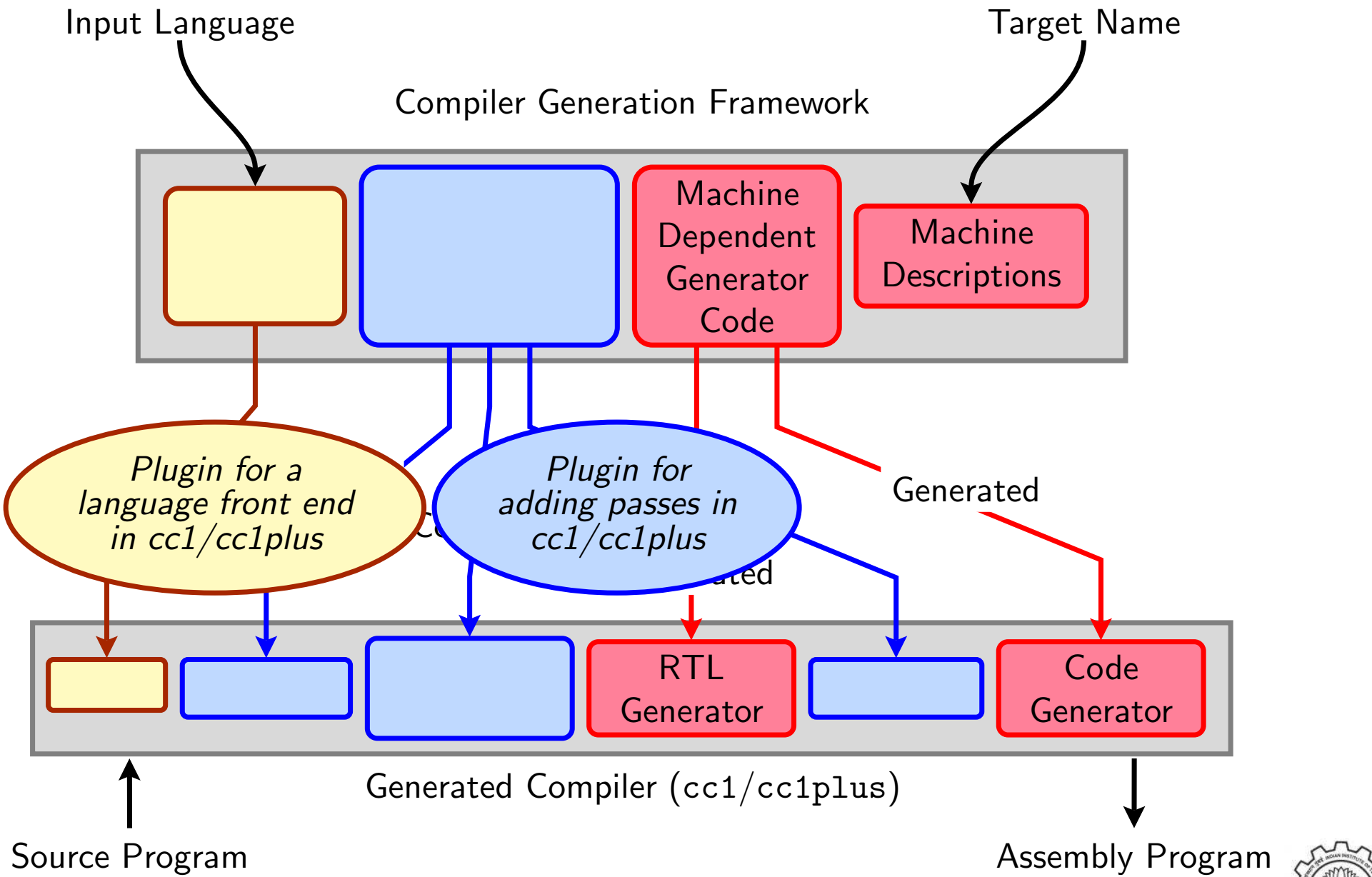
Static Plugins in the Generated Compiler



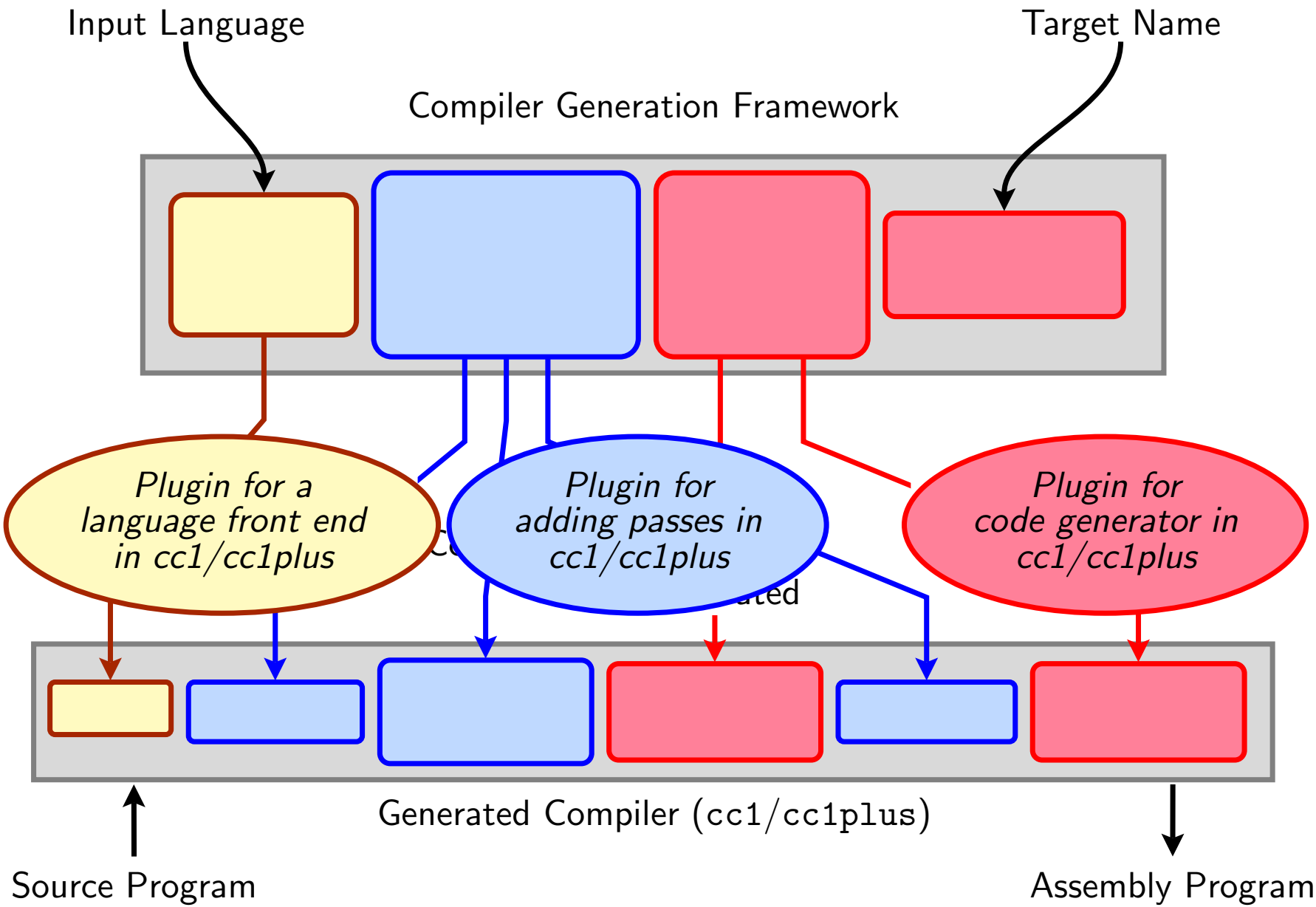
Static Plugins in the Generated Compiler



Static Plugins in the Generated Compiler



Static Plugins in the Generated Compiler



Part 2

Static Plugins in GCC

GCC's Solution

Plugin	Implementation	
	Data Structure	Initialization
Translator in `gcc/g++`	Array of C structures	Development time
Front end in `cc1/cc1plus`	C structure	Build time
Passes in `cc1/cc1plus`	Linked list of C structures	Development time
Back end in `cc1/cc1plus`	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.  */
};
```

Default Specs in the Plugin Data Structure in gcc.c

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}
}
```

Default Specs in the Plugin Data Structure in gcc.c

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}
}
```

- @: Aliased entry

Default Specs in the Plugin Data Structure in gcc.c

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}
}
```

- @: Aliased entry
- #: Default specs not available

Complete Entry for C in gcc.c

```
  {"@c",
   /* cc1 has an integrated ISO C preprocessor.  We should invoke the
      external preprocessor if -save-temps is given.  */
     "%{E|M|MM:%(trad_capable_cpp) %(cpp_options) %(cpp_debug_options)}\
      %{!E:%{!M:%{!MM:\
          %{traditional|ftraditional:\
%eGNU C no longer supports -traditional without -E}\
       %{!combine:\
  %{save-temps|traditional-cpp|no-integrated-cpp:%(trad_capable_cpp) \
%(cpp_options) -o %{save-temps:%b.i} %{!save-temps:%g.i} \n\
    cc1 -fpreprocessed %{save-temps:%b.i} %{!save-temps:%g.i} \
%(cc1_options)}\
  %{!save-temps:%{!traditional-cpp:%{!no-integrated-cpp:\
cc1 %(cpp_unique_options) %(cc1_options)}}}\
          %{!fsyntax-only:%(invoke_as)}} \
      %{combine:\
  %{save-temps|traditional-cpp|no-integrated-cpp:%(trad_capable_cpp) \
%(cpp_options) -o %{save-temps:%b.i} %{!save-temps:%g.i}}\
  %{!save-temps:%{!traditional-cpp:%{!no-integrated-cpp:\
cc1 %(cpp_unique_options) %(cc1_options)}}\
                %{!fsyntax-only:%(invoke_as)}}}}}}", 0, 1, 1},
```

Populated Plugin Data Structure for C++:

gcc/cp/lang-specs.h

```
{".cc",  "@c++", 0, 0, 0},
{".cp",  "@c++", 0, 0, 0},
{".cxx", "@c++", 0, 0, 0},
{".cpp", "@c++", 0, 0, 0},
{".c++", "@c++", 0, 0, 0},
{".C",   "@c++", 0, 0, 0},
{".CPP", "@c++", 0, 0, 0},
{".H",   "@c++-header", 0, 0, 0},
{".hpp", "@c++-header", 0, 0, 0},
{".hp",  "@c++-header", 0, 0, 0},
{".hxx", "@c++-header", 0, 0, 0},
{".h++", "@c++-header", 0, 0, 0},
{".HPP", "@c++-header", 0, 0, 0},
{".tcc", "@c++-header", 0, 0, 0},
{".hh",  "@c++-header", 0, 0, 0},
```

Populated Plugin Data Structure for C++:
gcc/cp/lang-specs.h

```
  {"@c++-header",
    "%{E|M|MM:cc1plus -E %(cpp_options) %2 %(cpp_debug_options)}\
     %{!E:%{!M:%{!MM:\
       %{save-temps|no-integrated-cpp:cc1plus -E\
%(cpp_options) %2 -o %{save-temps:%b.ii} %{!save-temps:%g.ii} \n}\
      cc1plus %{save-temps|no-integrated-cpp:-fpreprocessed %{save-temps:%
      %{!save-temps:%{!no-integrated-cpp:%(cpp_unique_options)}}\
%(cc1_options) %2\
%{!fsyntax-only:%{!fdump-ada-spec*:-o %g.s %{!o*:--output-pch=%i.gch}\
        %W{o*:--output-pch=%*}}%V}}}}",
     CPLUSPLUS_CPP_SPEC, 0, 0},
```


Populated Plugin Data Structure for C++:
gcc/cp/lang-specs.h

```
  {"@c++",
    "%{E|M|MM:cc1plus -E %(cpp_options) %2 %(cpp_debug_options)}\
     %{!E:%{!M:%{!MM:\
       %{save-temps|no-integrated-cpp:cc1plus -E\
%(cpp_options) %2 -o %{save-temps:%b.ii} %{!save-temps:%g.ii} \n}\
      cc1plus %{save-temps|no-integrated-cpp:-fpreprocessed %{save-temps:%
      %{!save-temps:%{!no-integrated-cpp:%(cpp_unique_options)}}\
%(cc1_options) %2\
       %{!fsyntax-only:%(invoke_as)}}}}",
     CPLUSPLUS_CPP_SPEC, 0, 0},
  {".ii", "@c++-cpp-output", 0, 0, 0},

  {"@c++-cpp-output",
   "%{!M:%{!MM:%{!E:\
    cc1plus -fpreprocessed %i %(cc1_options) %2\
    %{!fsyntax-only:%(invoke_as)}}}}", 0, 0, 0},
```

Populated Plugin Data Structure for LTO:

gcc/lto/lang-specs.h

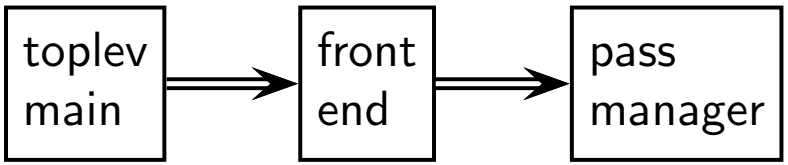
```
/* LTO contributions to the "compilers" array in gcc.c.  */

  {"@lto", "lto1 %(cc1_options) %i %{!fsyntax-only:%(invoke_as)}",
   /*cpp_spec=*/NULL, /*combinable=*/1, /*needs_preprocessing=*/0},
```

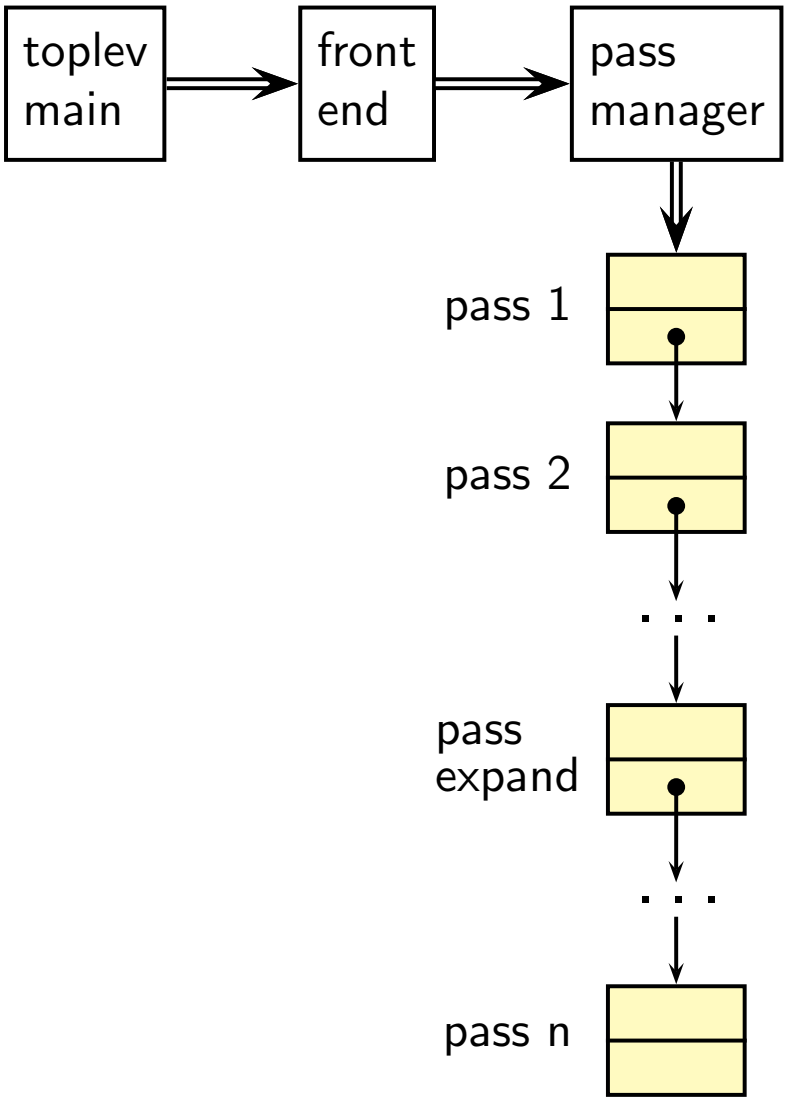

What about the Files to be Procecced by the Linker?

- 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`



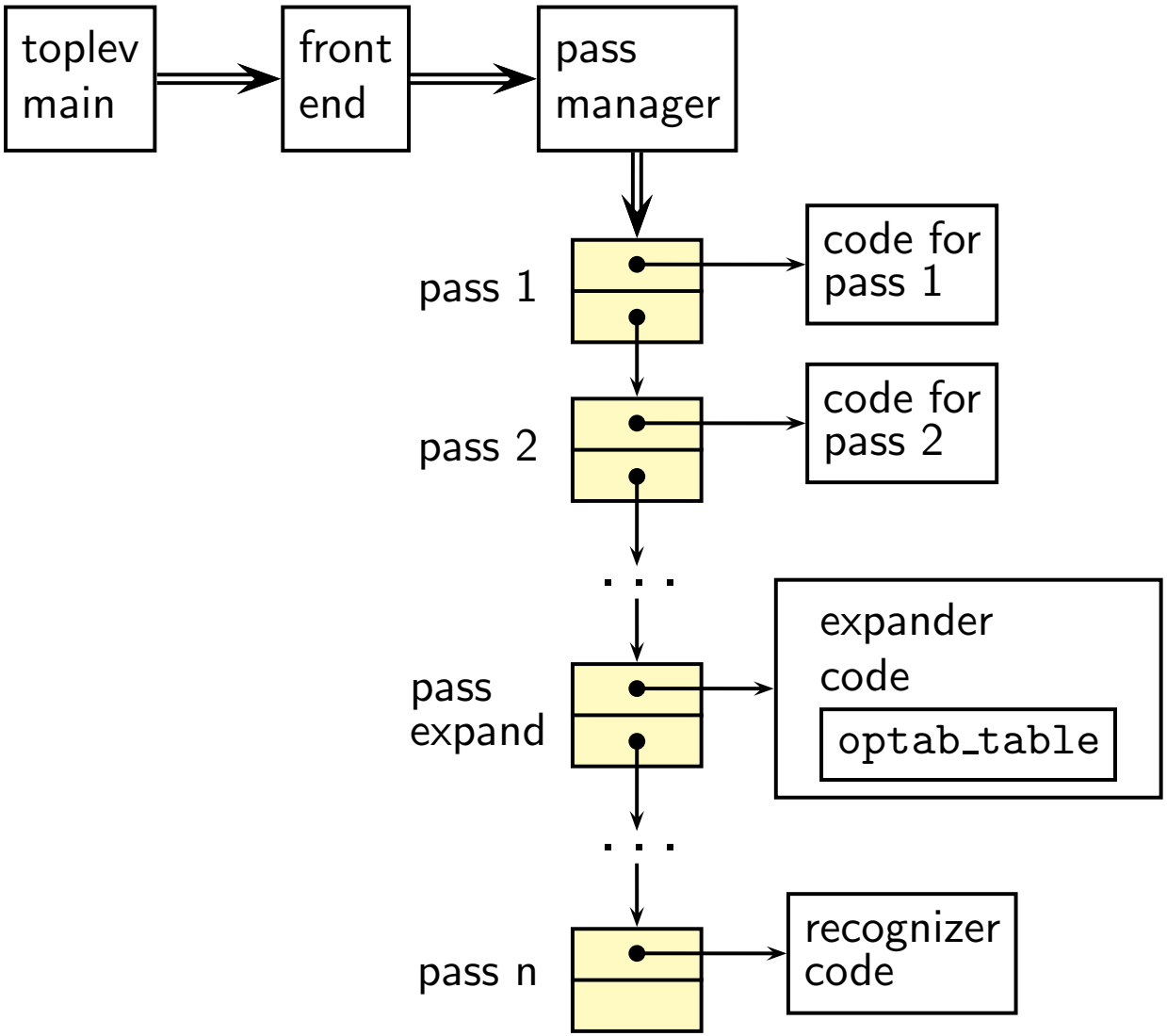
Plugin Structure in `cc1`



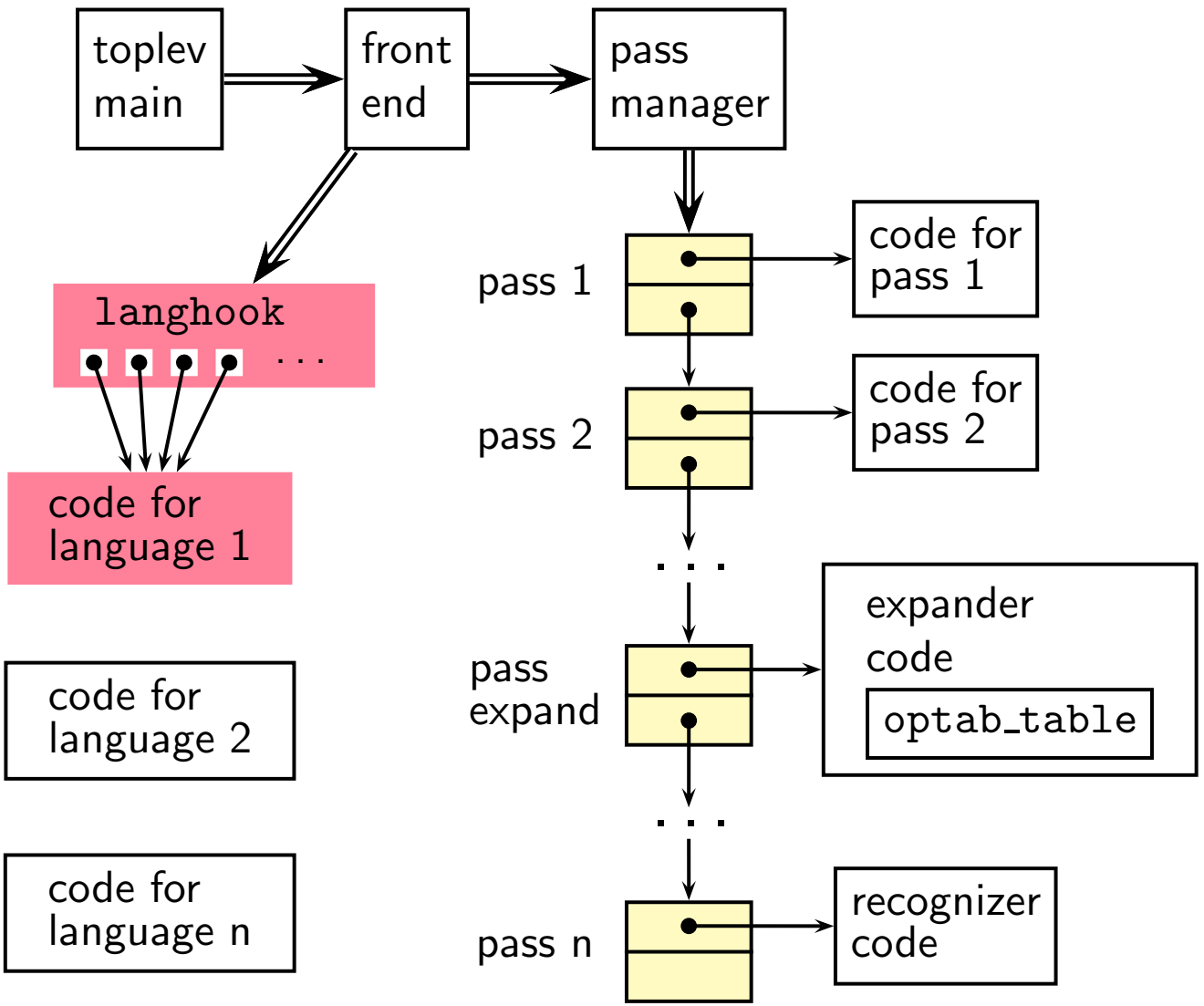
Double arrow represents control flow whereas single arrow represents pointer or index

For simplicity, we have included all passes in a single list. Actually passes are organized into five lists and are invoked as five different sequences

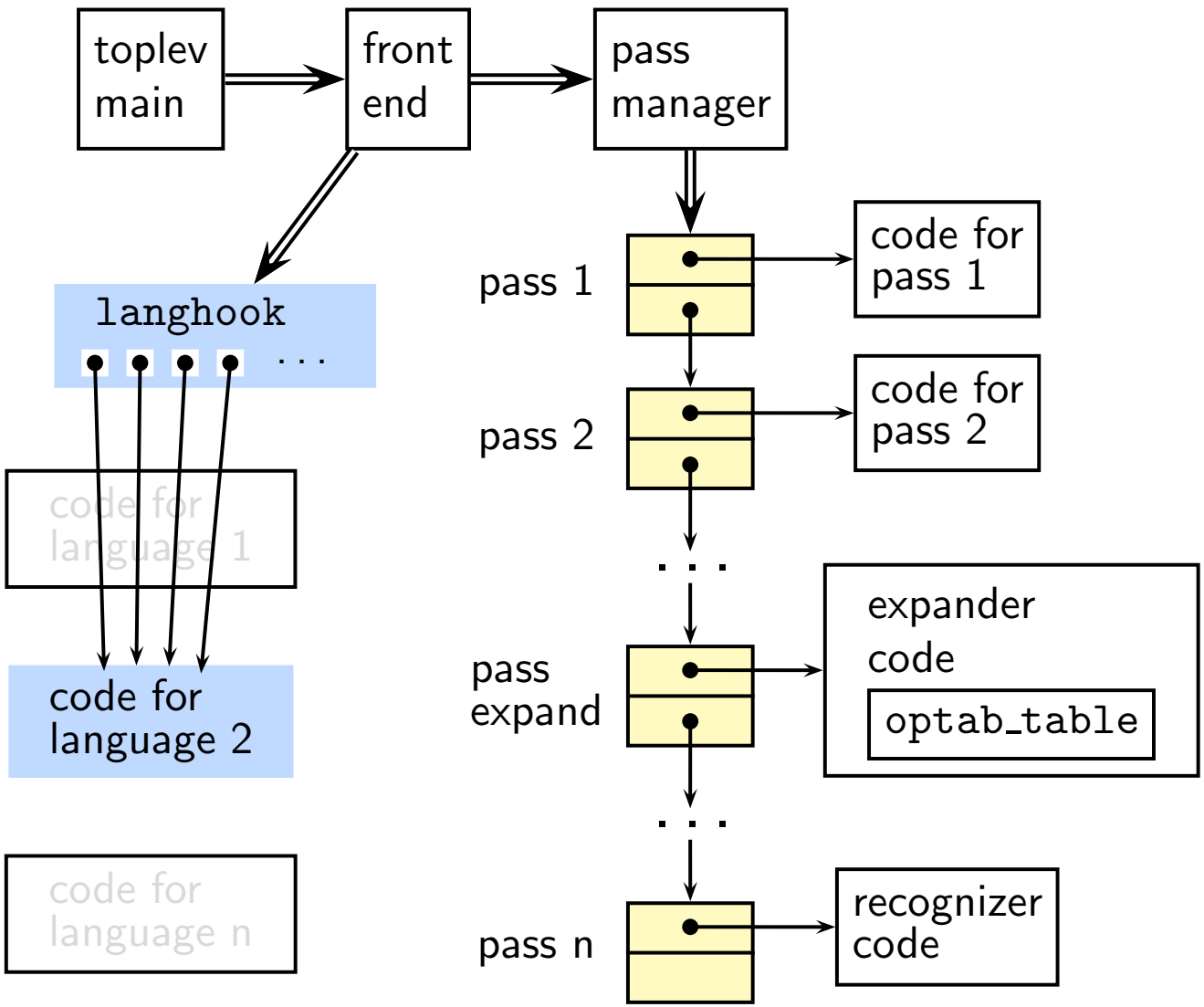
Plugin Structure in `cc1`



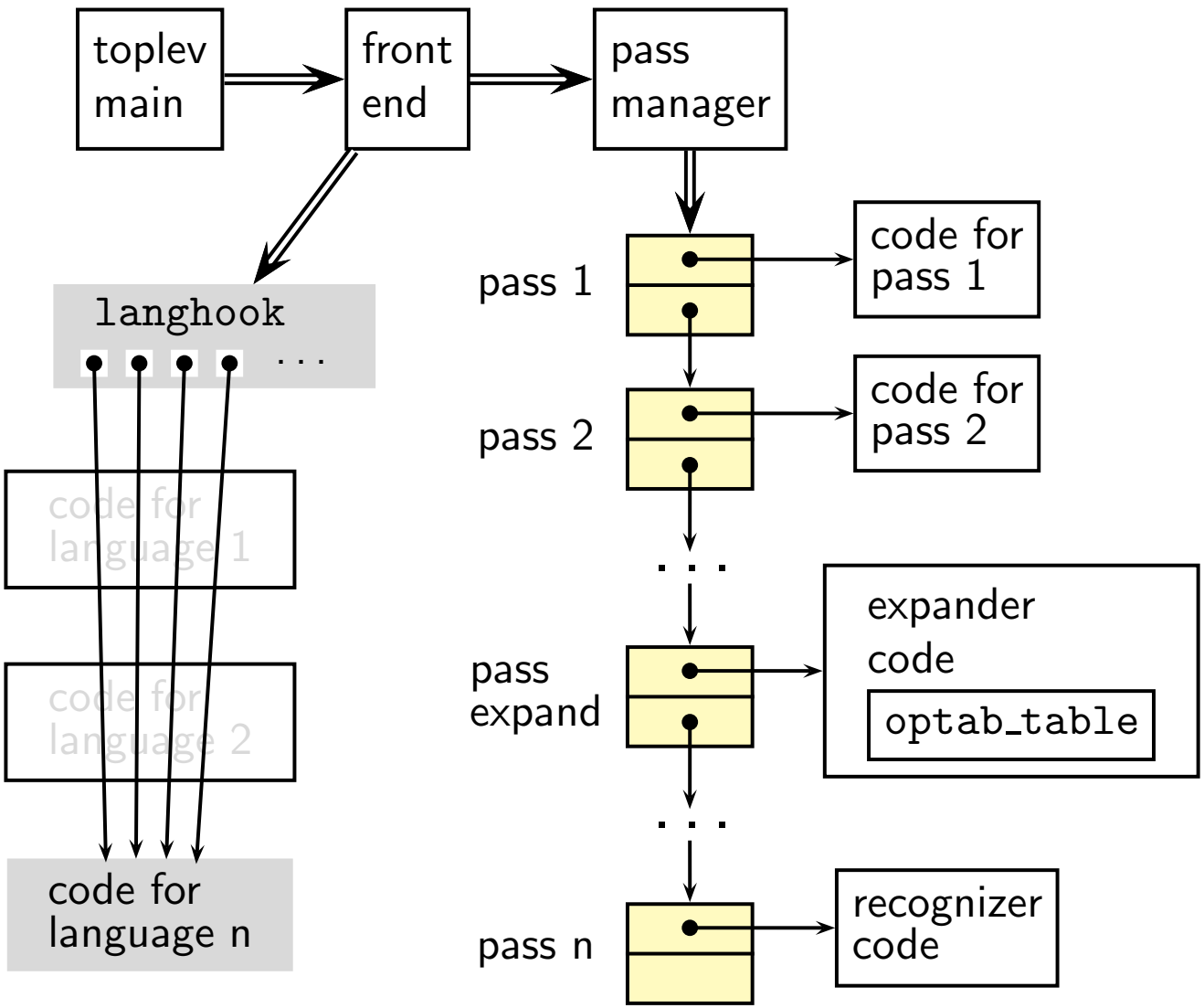
Plugin Structure in `cc1`



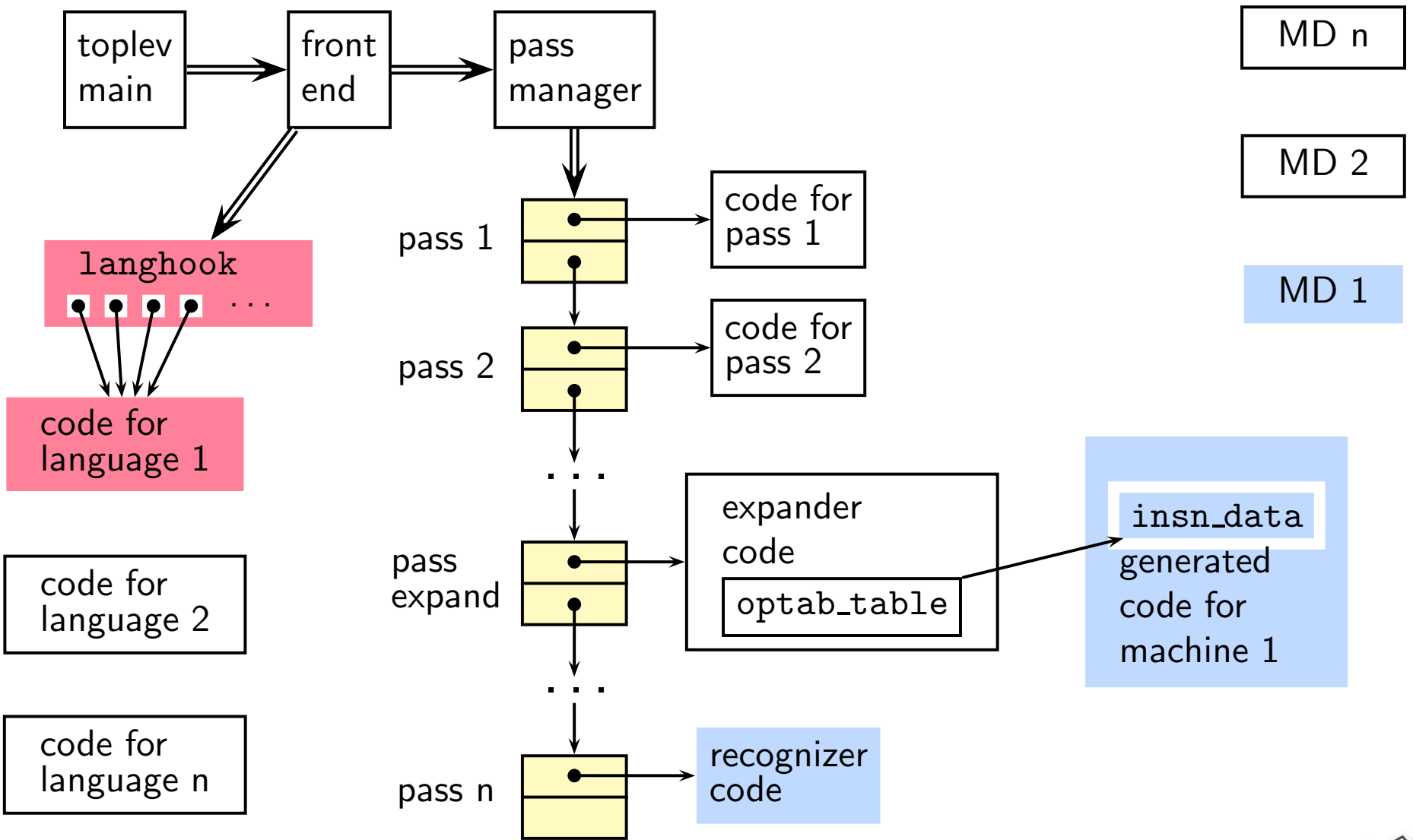
Plugin Structure in `cc1`



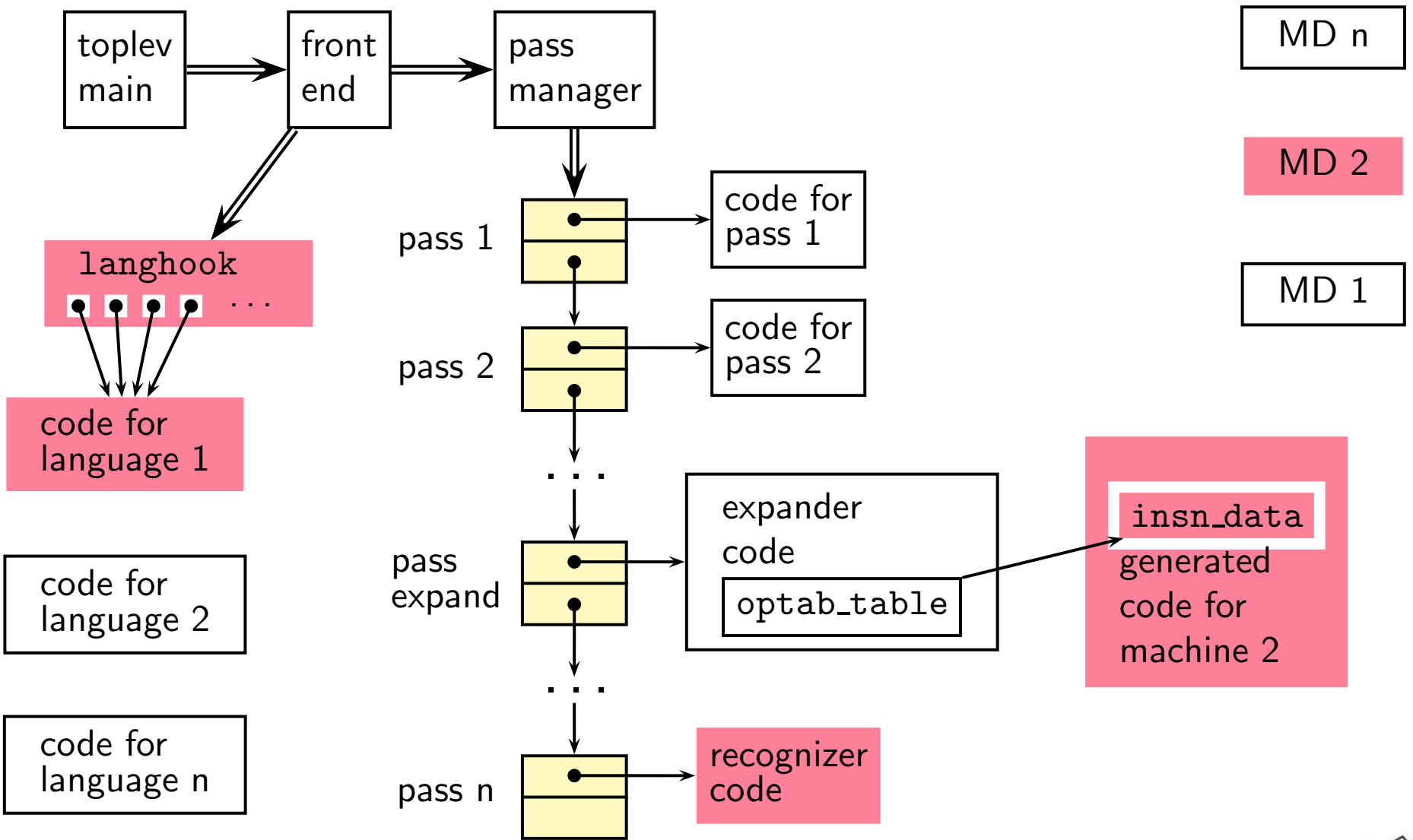
Plugin Structure in `cc1`



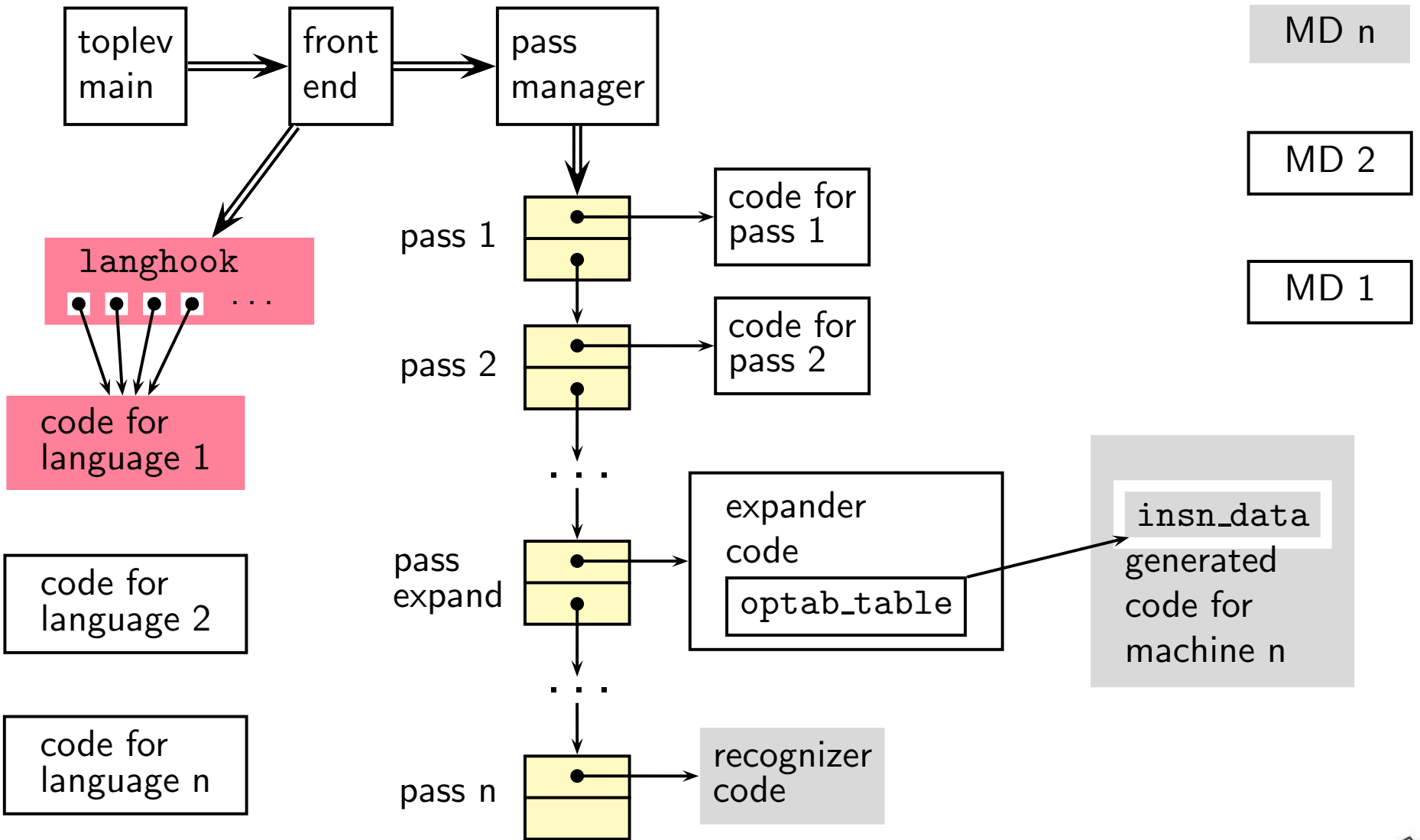
Plugin Structure in `cc1`



Plugin Structure in `cc1`



Plugin Structure in `cc1`

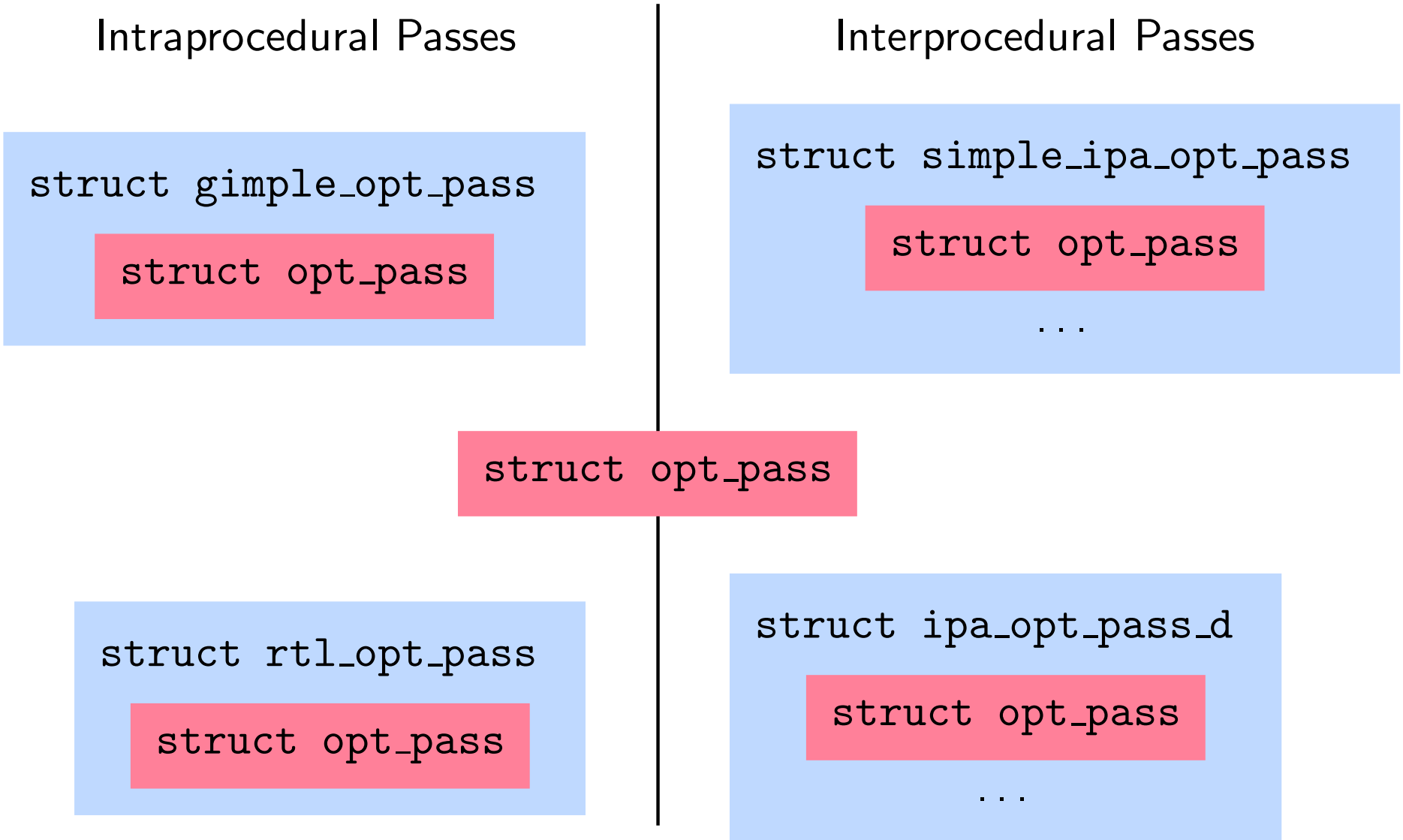


Front End Plugin

Important fields of `struct lang_hooks` instantiated for C

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

Plugins for Pass Specifications



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 on a Translation Unit

Pass variable: `all_simple_ipa_passes`

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

Plugins for Interprocedural Passes across a Translation Unit

Pass variable: `all_regular_ipa_passes`

```
struct ipa_opt_pass_d
{
  struct opt_pass pass;
  void (*generate_summary) (void);
  void (*read_summary) (void);
  void (*write_summary) (struct cgraph_node_set_def *,
                         struct varpool_node_set_def *);
  void (*write_optimization_summary)(struct cgraph_node_set_def *,
                                     struct varpool_node_set_def *);
  void (*read_optimization_summary) (void);
  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 *);
};
```

Predefined Pass Lists

Pass List	Purpose
`all_lowering_passes`	AST to CFG translation
`all_small_ipa_passes`	Interprocedural passes restricted to a single translation unit
`all_regular_ipa_passes`	Interprocedural passes on a translation unit as well as across translation units (during WPA/IPA of LTO)
`all_late_ipa_passes`	Interprocedural passes on partitions created by LTO (after WPA/IPA)
`all_lto_gen_passes`	Passes to encode program for LTO
`all_passes`	Intraprocedural passes on GIMPLE and RTL

Registering a Pass as a Static Plugin

1. Write the driver function in your file
2. Declare your pass in file `tree-pass.h`:
   `extern struct gimple_opt_pass your_pass_name;`
3. Add your pass to the appropriate pass list in `init_optimization_passes()` using the macro `NEXT_PASS`
4. Add your file details to `$SOURCE/gcc/Makefile.in`
5. Configure and build gcc
   (For simplicity, you can make `cc1` only)
6. Debug `cc1` using ddd/gdb if need arises
   (For debuging `cc1` from within `gcc`, see:
   `http://gcc.gnu.org/ml/gcc/2004-03/msg01195.html`)

Part 3

Dynamic Plugins in GCC

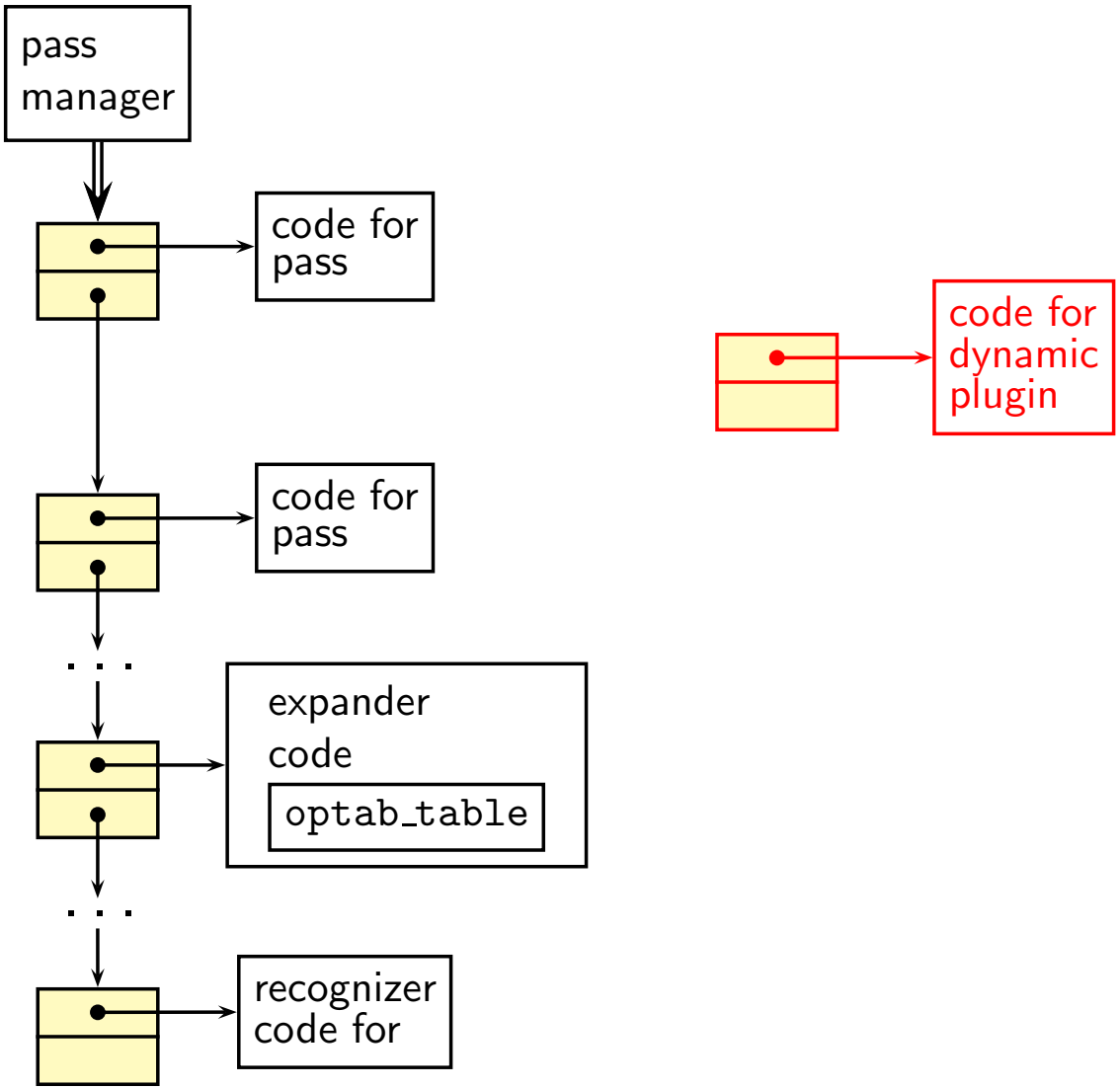
Dynamic Plugins

- Supported on platforms that support `-ldl -rdynamic`
- Loaded using `dlopen` and invoked at pre-determined locations in the compilation process
- Command line option

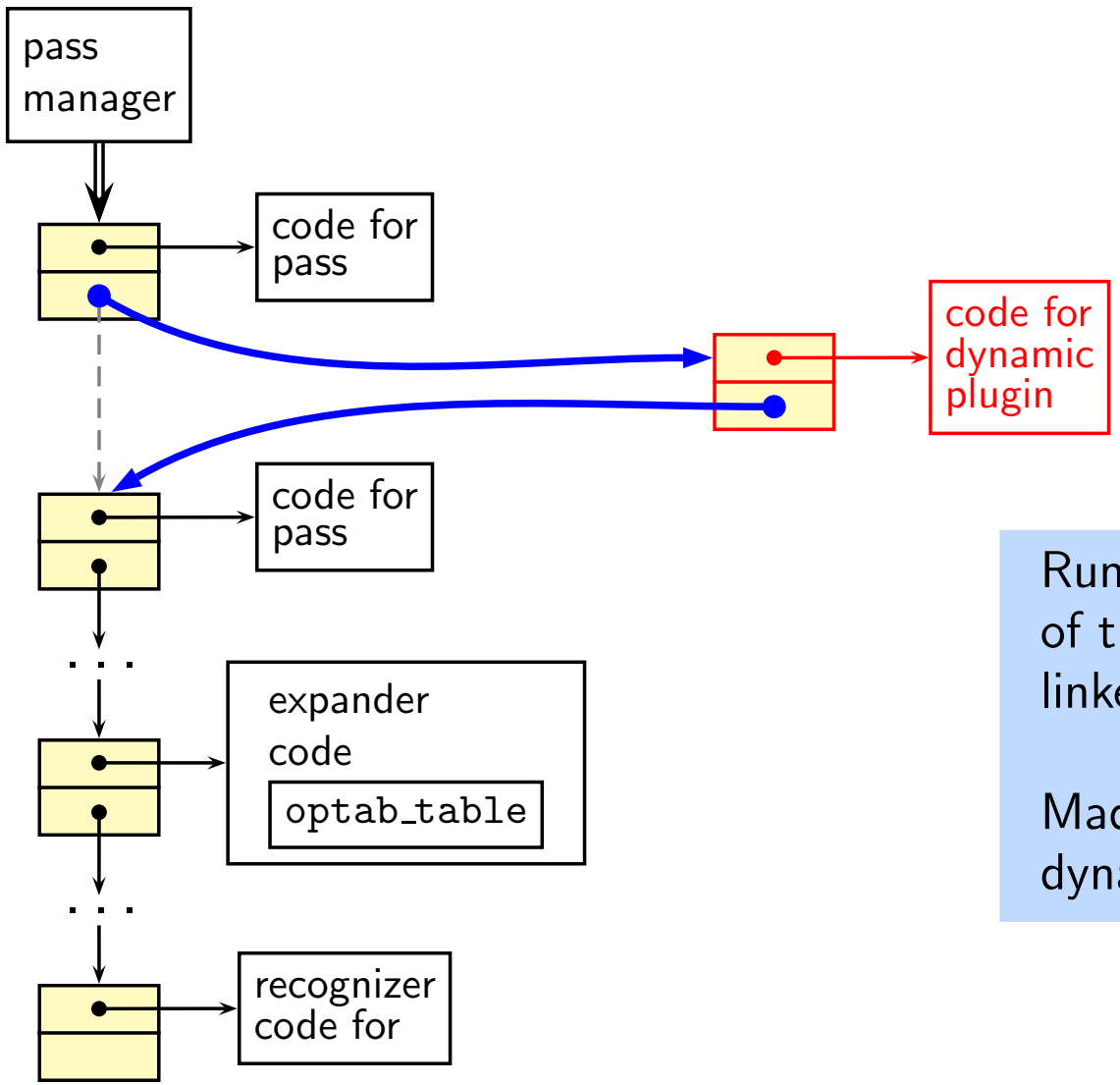
  `-fplugin=/path/to/name.so`

  Arguments required can be supplied as name-value pairs

The Mechanism of Dynamic Plugin



The Mechanism of Dynamic Plugin



Runtime initialization of the appropriate linked list of passes

Made possible by dynamic linking

Specifying an Example Pass

```
struct simple_ipa_opt_pass pass_plugin = {
  {
    SIMPLE_IPA_PASS,
    "dynamic_plug",                 /*  name */
    0,                              /*  gate */
    execute_pass_plugin,            /*  execute */
    NULL,                           /*  sub */
    NULL,                           /*  next */
    0,                              /*  static pass number */
    TV_INTEGRATION,                 /*  tv_id */
    0,                              /*  properties required */
    0,                              /*  properties provided */
    0,                              /*  properties destroyed */
    0,                              /*  todo_flags start */
    0                               /*  todo_flags end */
  }
};
```

Registering Our Pass as a Dynamic Plugin

```
struct register_pass_info pass_info = {
  &(pass_plugin.pass),      /* Address of new pass, here, the
                               struct opt_pass field of
                               simple_ipa_opt_pass defined above */
  "pta",                    /* Name of the reference pass (string
                               in the structure specification) for
                               hooking up the new pass.  */
  0,                        /* Insert the pass at the specified
                               instance number of the reference
                               pass. Do it for every instance if
                               it is 0. */
  PASS_POS_INSERT_AFTER     /* how to insert the new pass:
                               before, after, or replace. Here we
                               are inserting our pass the pass
                               named pta */
};
```

Registering Callback for Our Pass for a Dynamic Plugins

```
int plugin_init(struct plugin_name_args *plugin_info,
    struct plugin_gcc_version *version)
{ /*  Plugins are activiated using this callback  */

  register_callback (
      plugin_info->base_name,     /* char *name: Plugin name,
                                     could be any name.
                                     plugin_info->base_name
                                     gives this filename */
      PLUGIN_PASS_MANAGER_SETUP,  /* int event: The event code.
                                     Here, setting up a new
                                     pass */
      NULL,                       /* The function that handles
                                     the event */
      &pass_info);                /* plugin specific data */

  return 0;
}
```


Makefile for Creating and Using a Dynamic Plugin

```
CC = $(INSTALL_D)/bin/g++
PLUGIN_SOURCES = new-pass.c
PLUGIN_OBJECTS = $(patsubst %.c,%.o,$(PLUGIN_SOURCES ))
GCCPLUGINS_DIR = $(shell $(CC) -print-file-name=plugin)
CFLAGS+= -fPIC -O2
INCLUDE = -Iplugin/include

%.o : %.c
$(CC) $(CFLAGS) $(INCLUDE) -c $<

new-pass.so: $(PLUGIN_OBJECTS)
        $(CC) $(CFLAGS) $(INCLUDE) -shared $^ -o $@

test_plugin: test.c
        $(CC) -fplugin=./new-pass.so $^ -o $@ -fdump-tree-all
```

Part 4

Flow of Control in the Generated Compiler

Walking the Maze of a Large Code Base

- If you use conventional editors such as `vi` or `emacs`
  - Use cscope

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

  - Use ctags

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

    Make sure you use `exeburant-ctags`
- Or use IDE such as `eclipse`

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 main.c */
|   toplev_main    /* In file toplev.c */
|   |   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 main.c */
  toplev_main           /* In file toplev.c */
    decode_options
    do_compile
      compile_file
        lang_hooks                                  le
        lang_hooks
                                                    eclarations
          targetm.asm
    finalize
```

Observations

- The entire compilation is driven by functions specified in language hooks
- Not a good 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_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 */
```

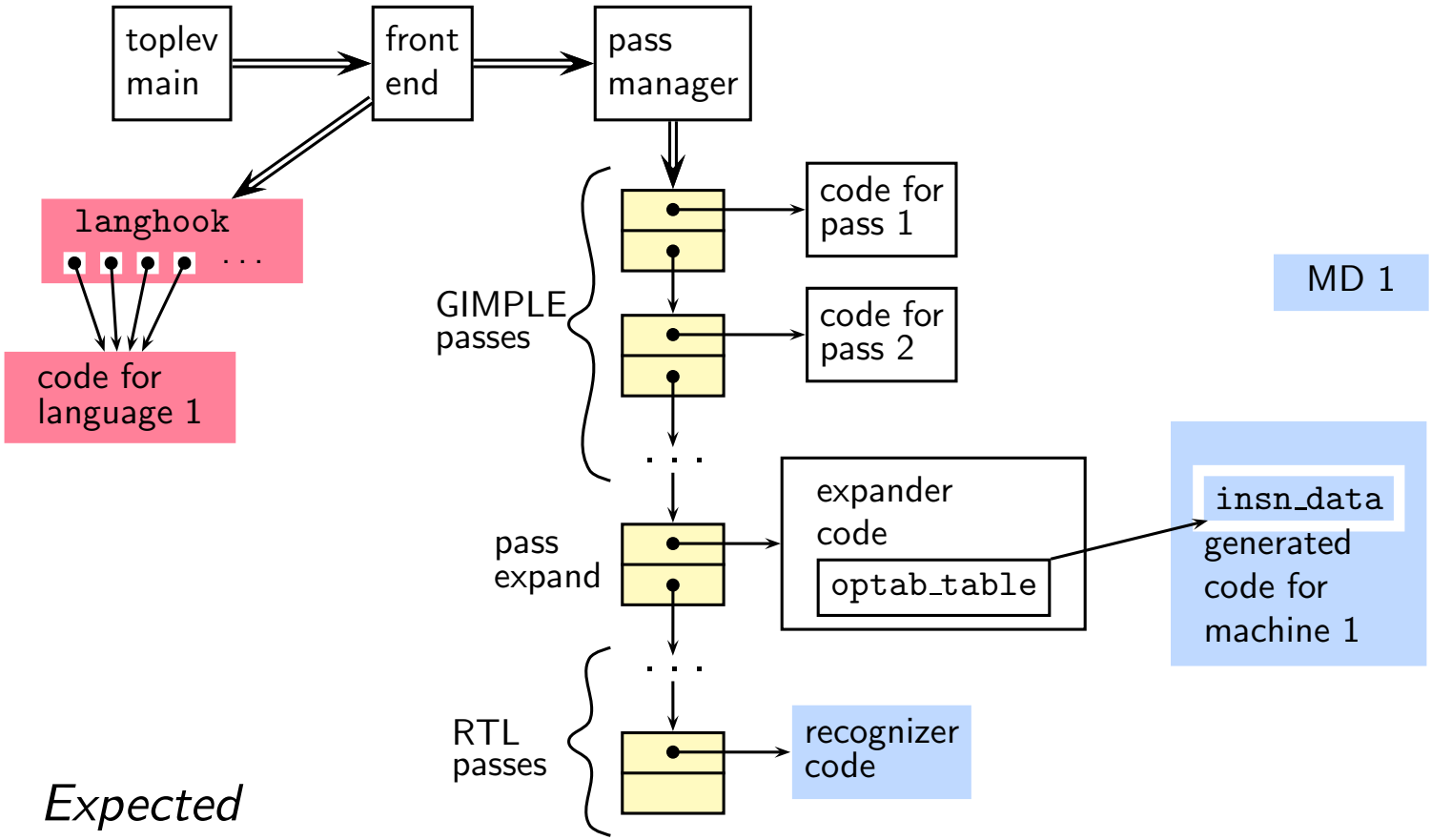

cc1 Control Flow: Parsing for C

```
lang_hooks.parse_file => c_common_parse_file
   c_parse_file
      c_parser_translation_unit
         c_parser_external_declaration
            c_parser_declaration_or_fndef
               c_pars                                    ns */
               c_pars
               finish                                    /
                  c_g
                  cgr
                  /*
```

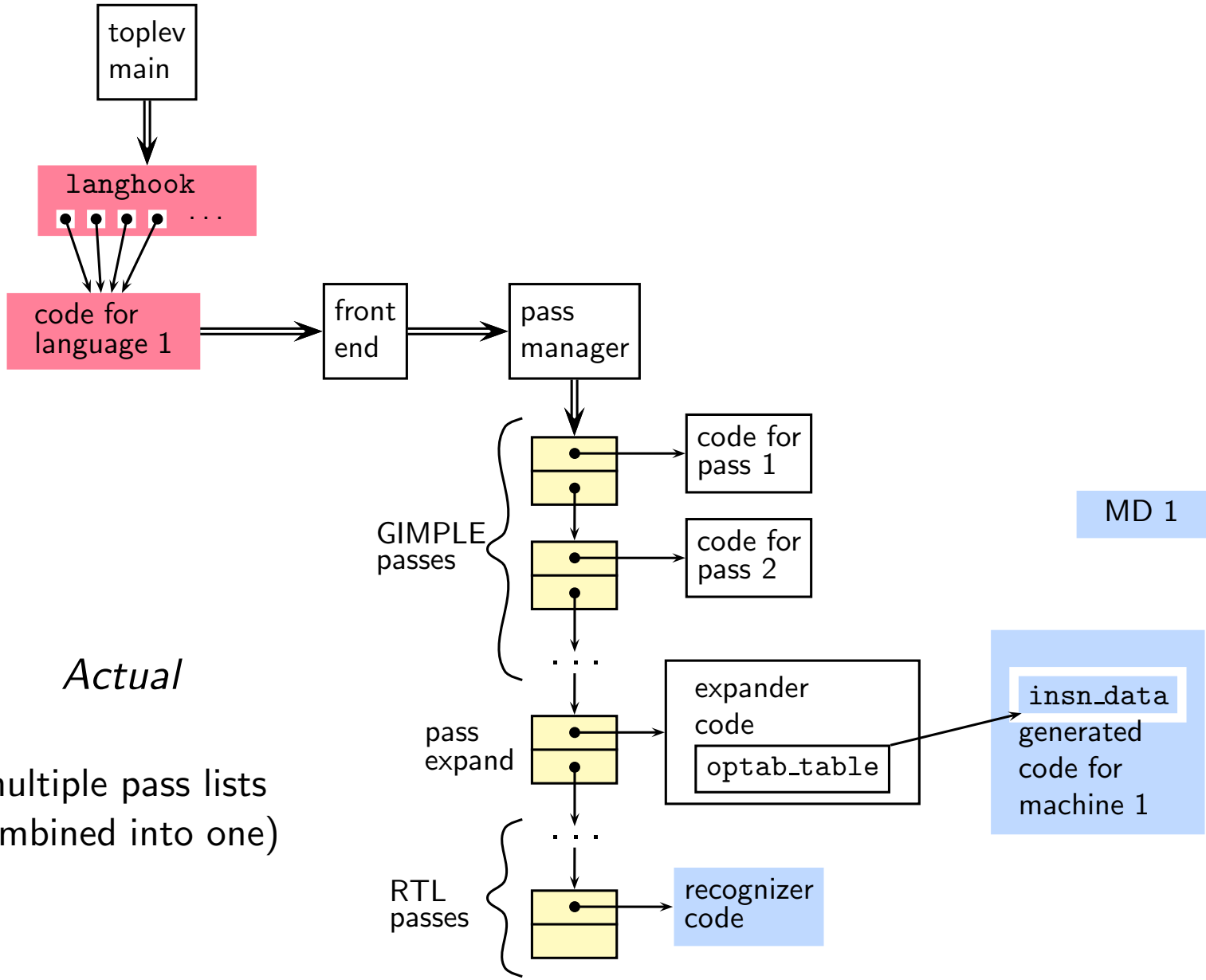
Observations

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

Expected Vs. Actual Schematic of the Front End



Expected Vs. Actual Schematic of the Front End



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_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_analyze_functions     /* Create GIMPLE */
          cgraph_analyze_function
             gimplify
                gimpl
                   gi

             cgraph_l                                  ural */
                tree_
                   ex                                  asses)
```

Observations

- Lowering passes are language independent
- Yet they are being called from a function in language hooks
- Not a good design!

Organization of Passes

Order	Task	IR	Level	Pass data structure
1	Lowering	GIMPLE	Intra	`gimple_opt_pass`
2	Optimizations	GIMPLE	Inter	`simple_ipa_opt_pass`
3	Optimizations	GIMPLE	Inter	`ipa_opt_pass_d`
4	Optimizations	GIMPLE	Intra	`gimple_opt_pass`
5	RTL Generation	GIMPLE	Intra	`rtl_opt_pass`
6	Optimization	RTL	Intra	`rtl_opt_pass`

cc1 Control Flow: Optimization and Code Generation Passes

```
cgraph_analyze_function          /* Create GIMPLE */
...                              /* previous slide */
cgraph_optimize
   ipa_passes
      execute_ipa_pass_list(all_small_ipa_passes)  /*!in_lto_p*/
      execute_ipa_summary_passes(all_regular_ipa_passes)
      execute_ipa_summary_passes(all_lto_gen_passes)
      ipa_write_summaries
   execute_ipa_pass_list(all_late_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

```
cgraph_analyze_function          /* Create GIMPLE */
...                              /* previous slide */
cgraph_optimize
  ipa_passes
    execute_ipa_pass_list(all_small_ipa_passes)  /*!in_lto_p*/
    execute_ipa_summary_passes(all_regular_ipa_passes)
    execute_ipa_summary_passes(all_lto_gen_passes)
    ipa_write_summaries
  execute_ipa_pass_list(all_late_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

```
cgraph_analyze_function          /* Create GIMPLE */
...                              /* previous slide */
cgraph_optimize
   ipa_passes
      execute_ipa_pas                                  !in_lto_p*/
      execute_ipa_sum                                  ses)
      execute_ipa_sum
      ipa_write_summa
   execute_ipa_pass_l
   cgraph_expand_all_
       cgraph_expand_
       /* Intraproced
       /* expansion p
            tree_rest
               execut
```

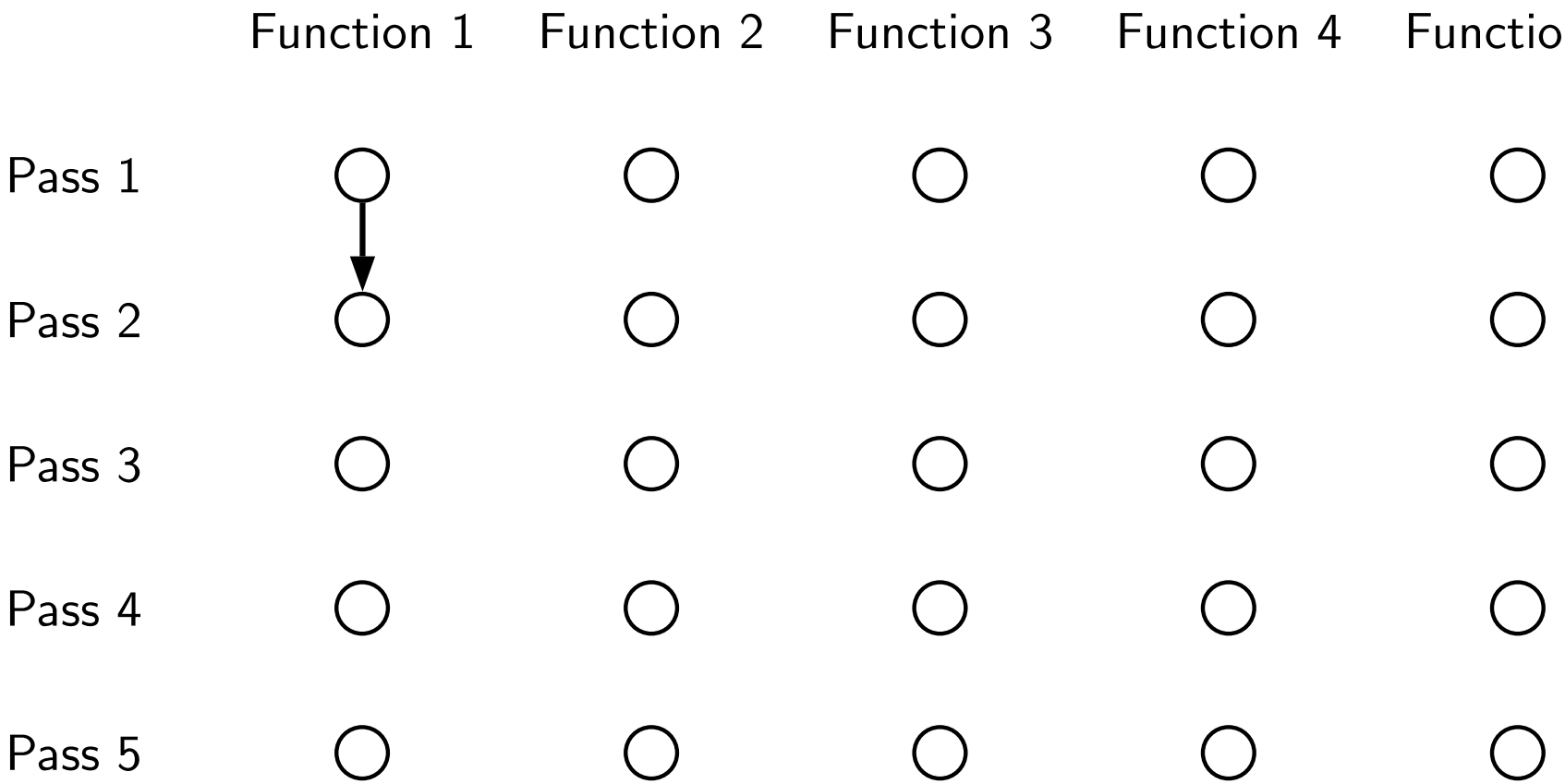
Observations

- Optimization and code generation passes are language independent
- Yet they are being called from a function in language hooks
- Not a good design!

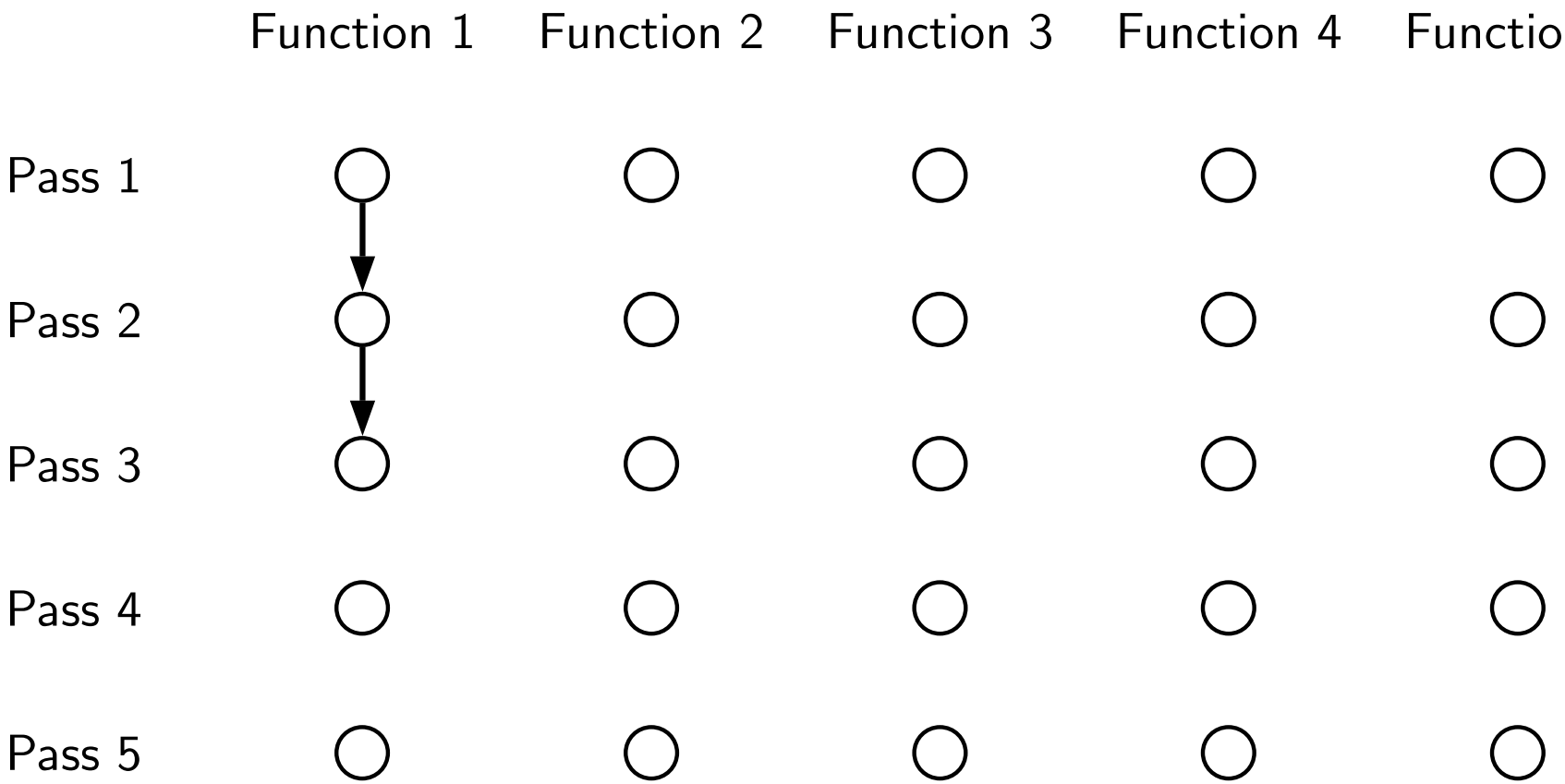
Execution Order in Intraprocedural Passes

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

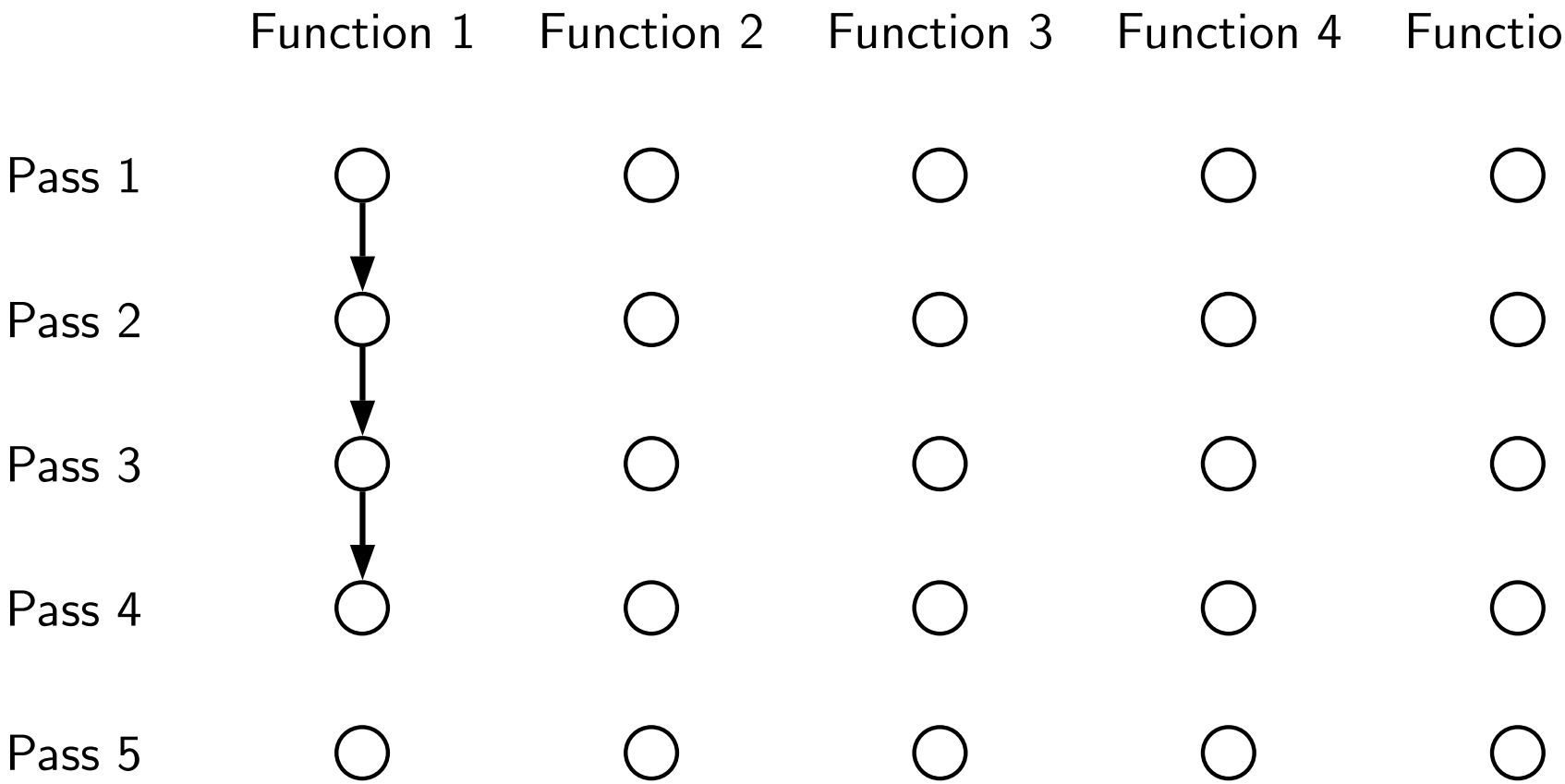
Execution Order in Intraprocedural Passes



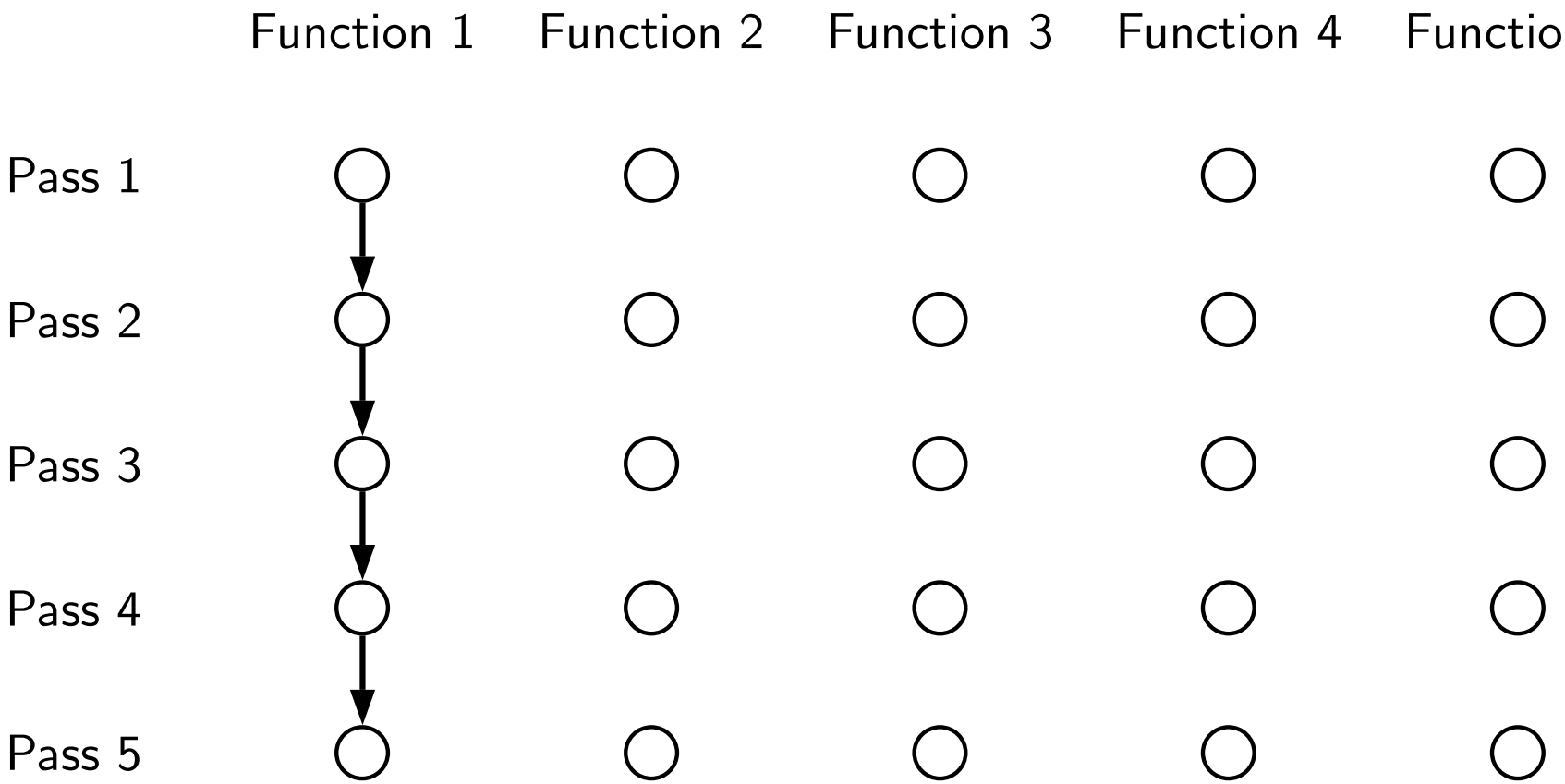
Execution Order in Intraprocedural Passes



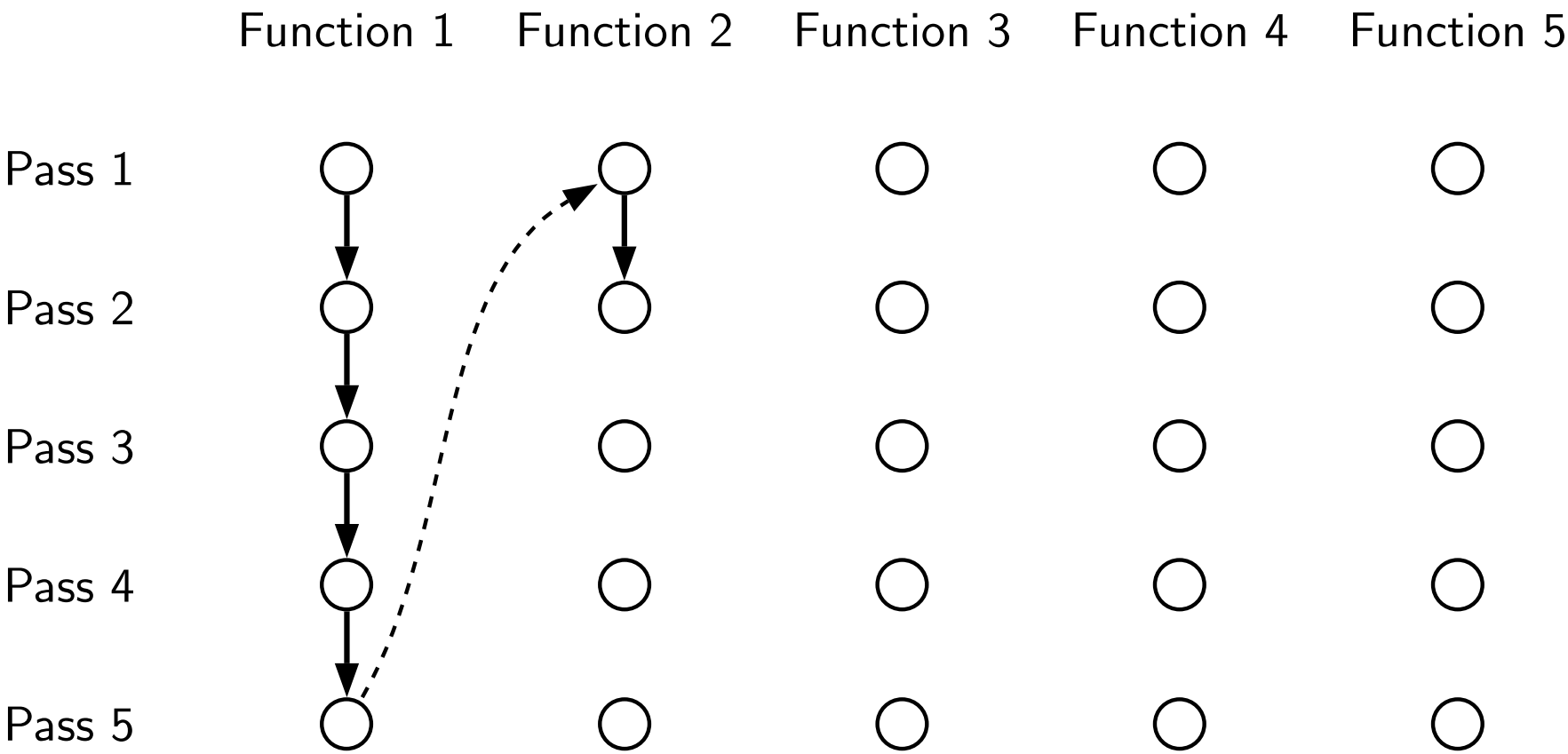
Execution Order in Intraprocedural Passes



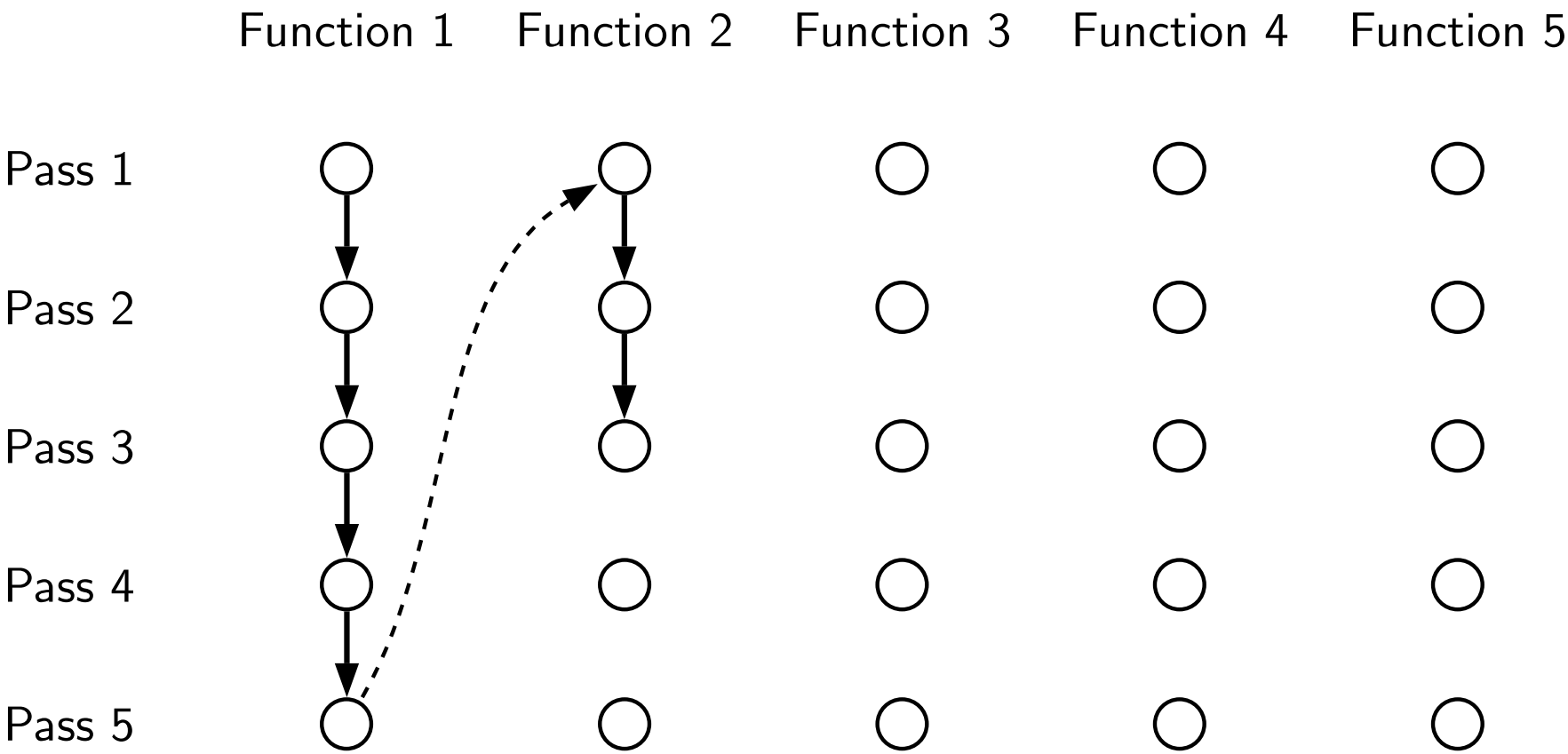
Execution Order in Intraprocedural Passes



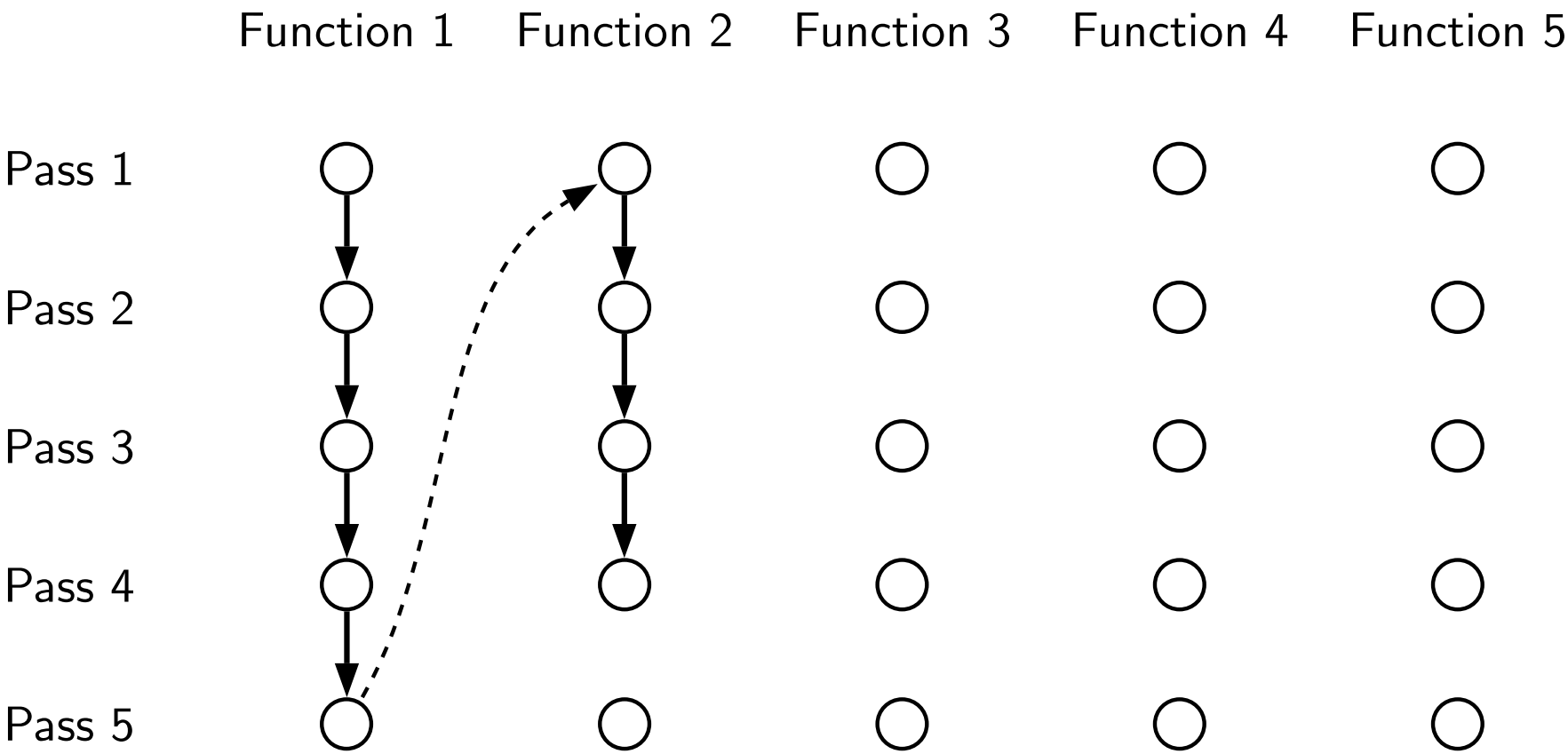
Execution Order in Intraprocedural Passes



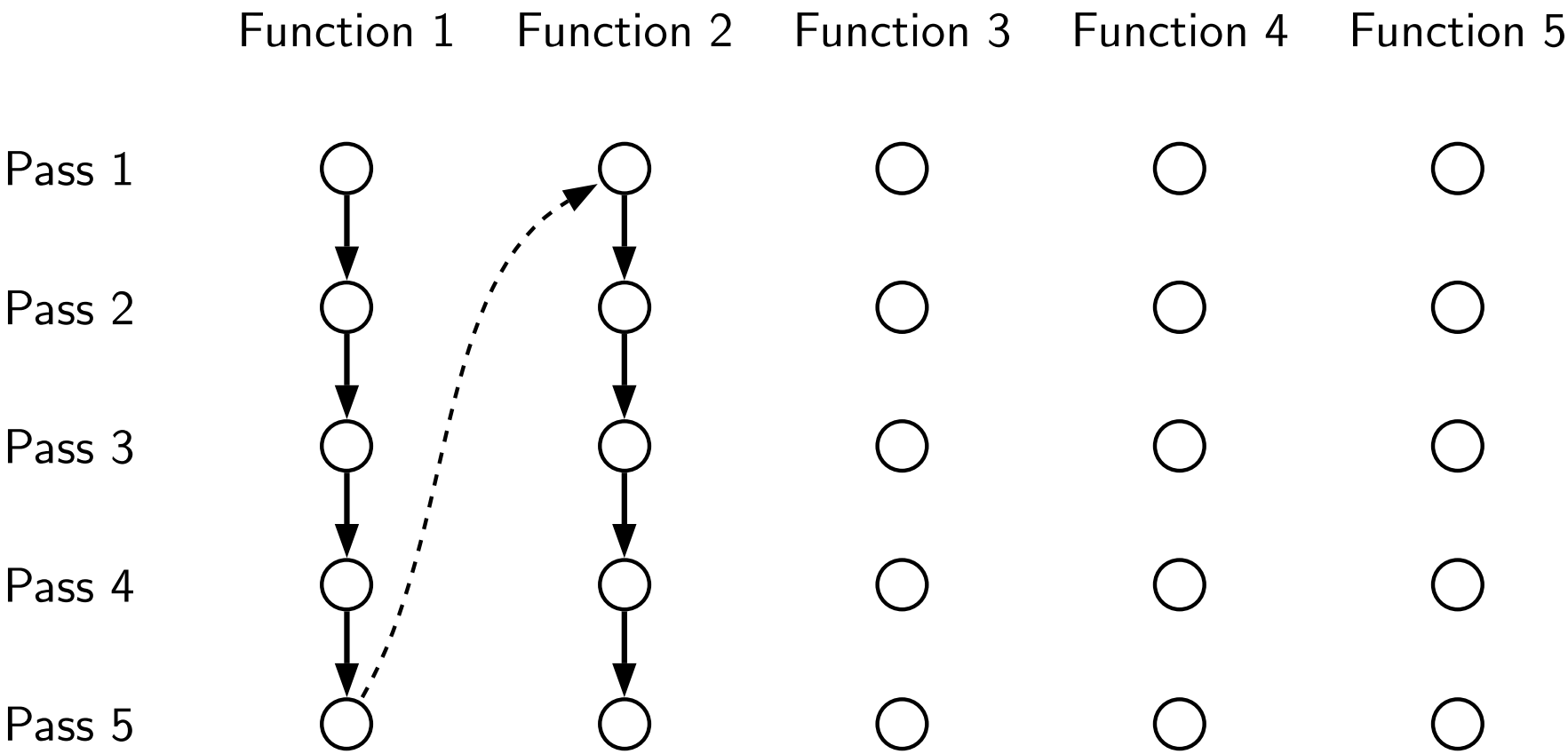
Execution Order in Intraprocedural Passes



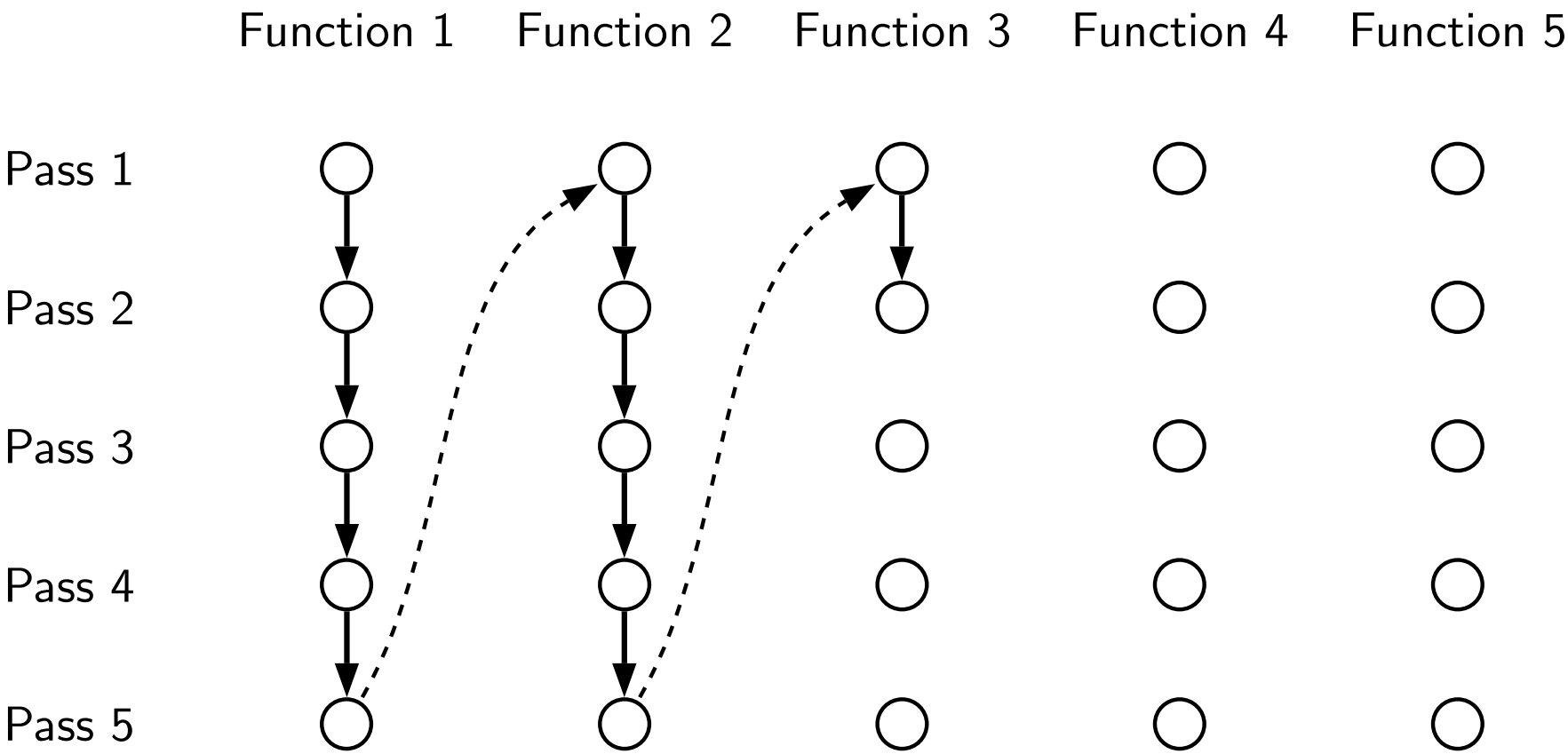
Execution Order in Intraprocedural Passes



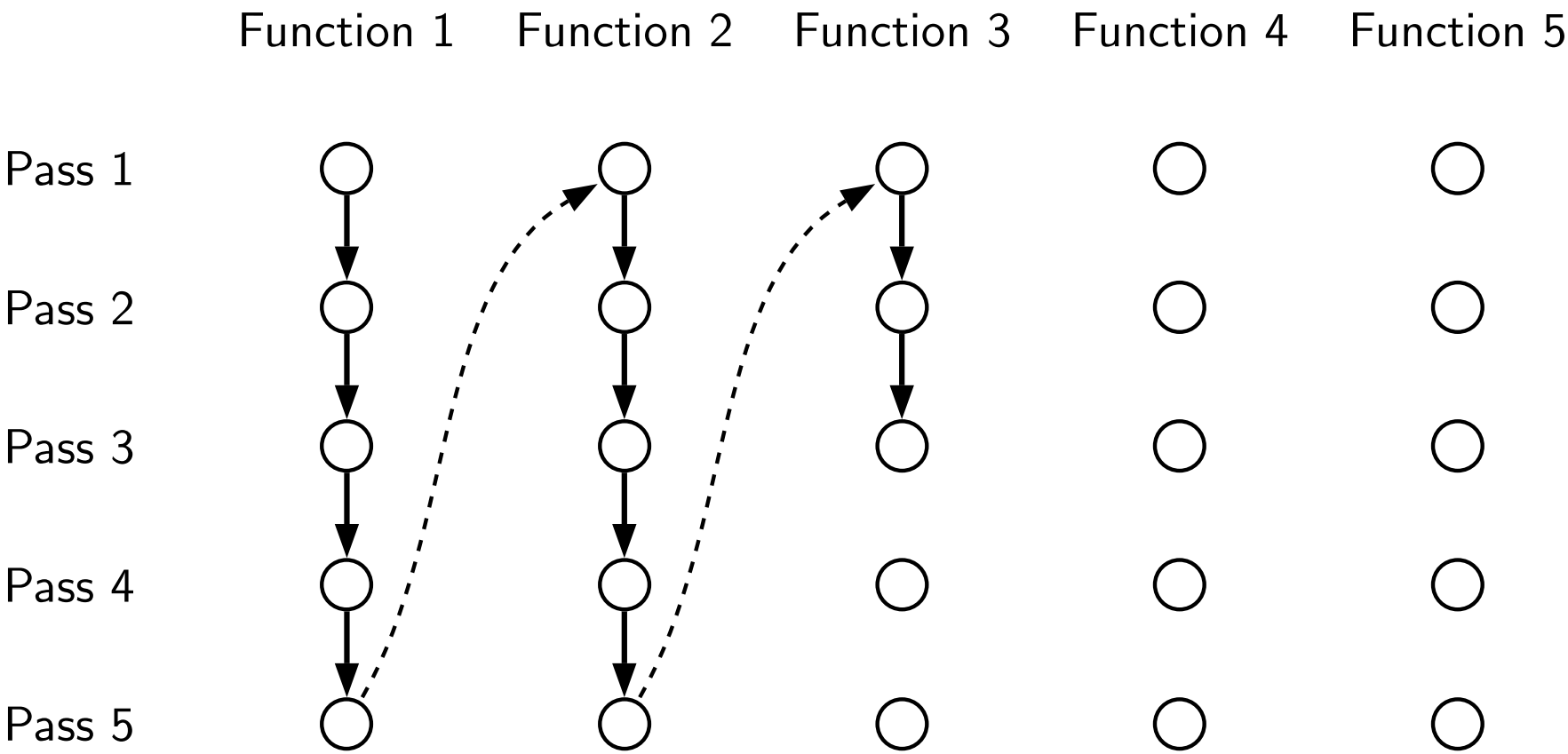
Execution Order in Intraprocedural Passes



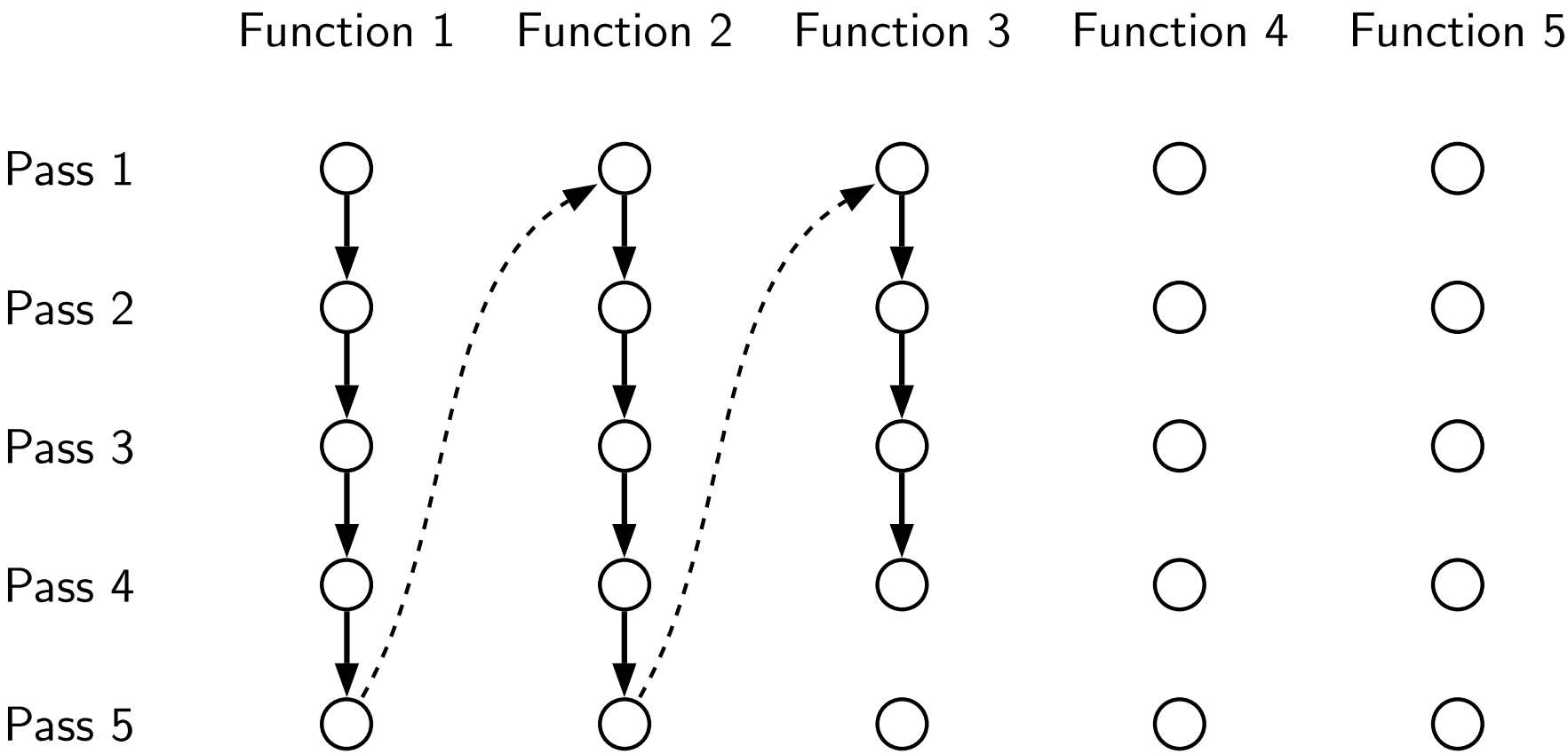
Execution Order in Intraprocedural Passes



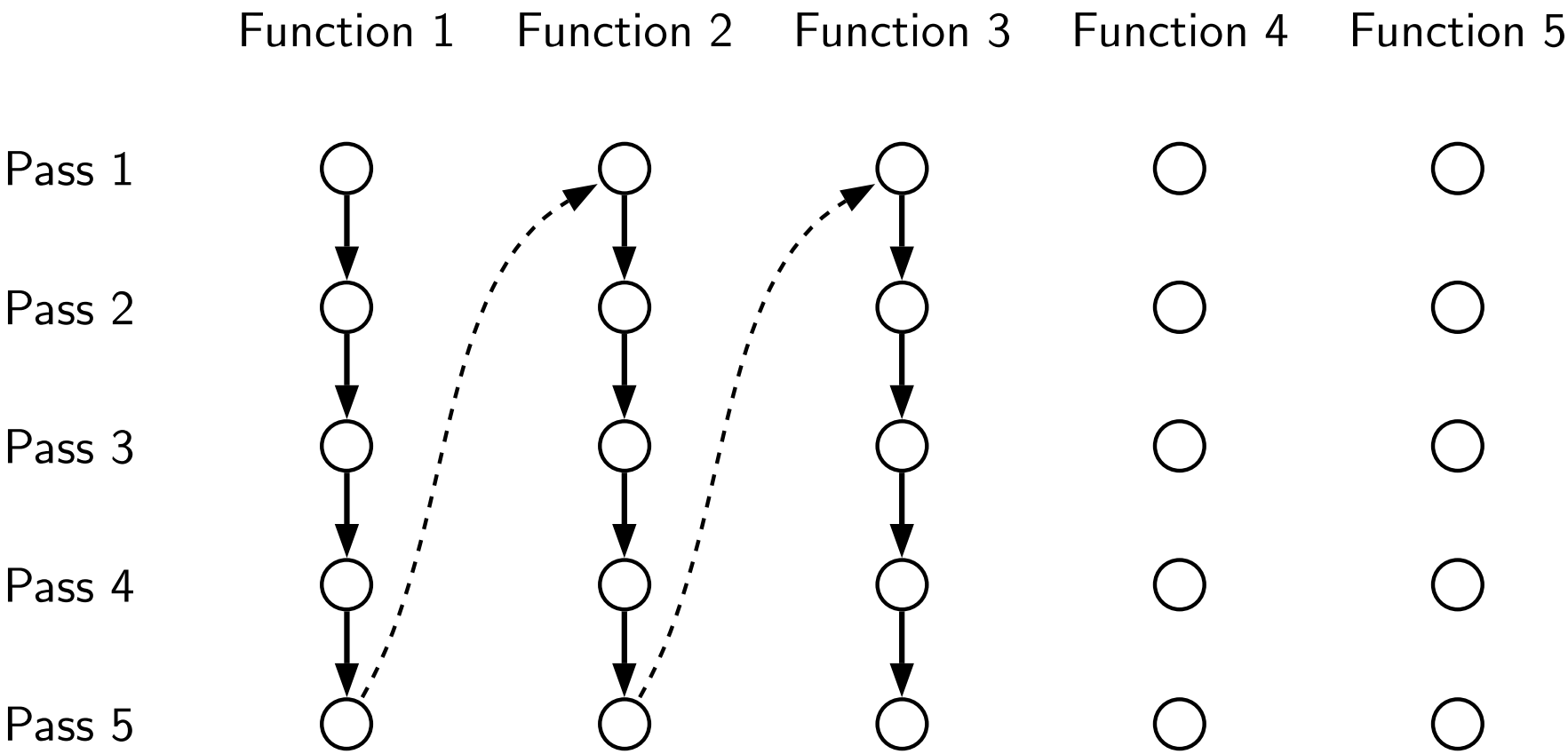
Execution Order in Intraprocedural Passes



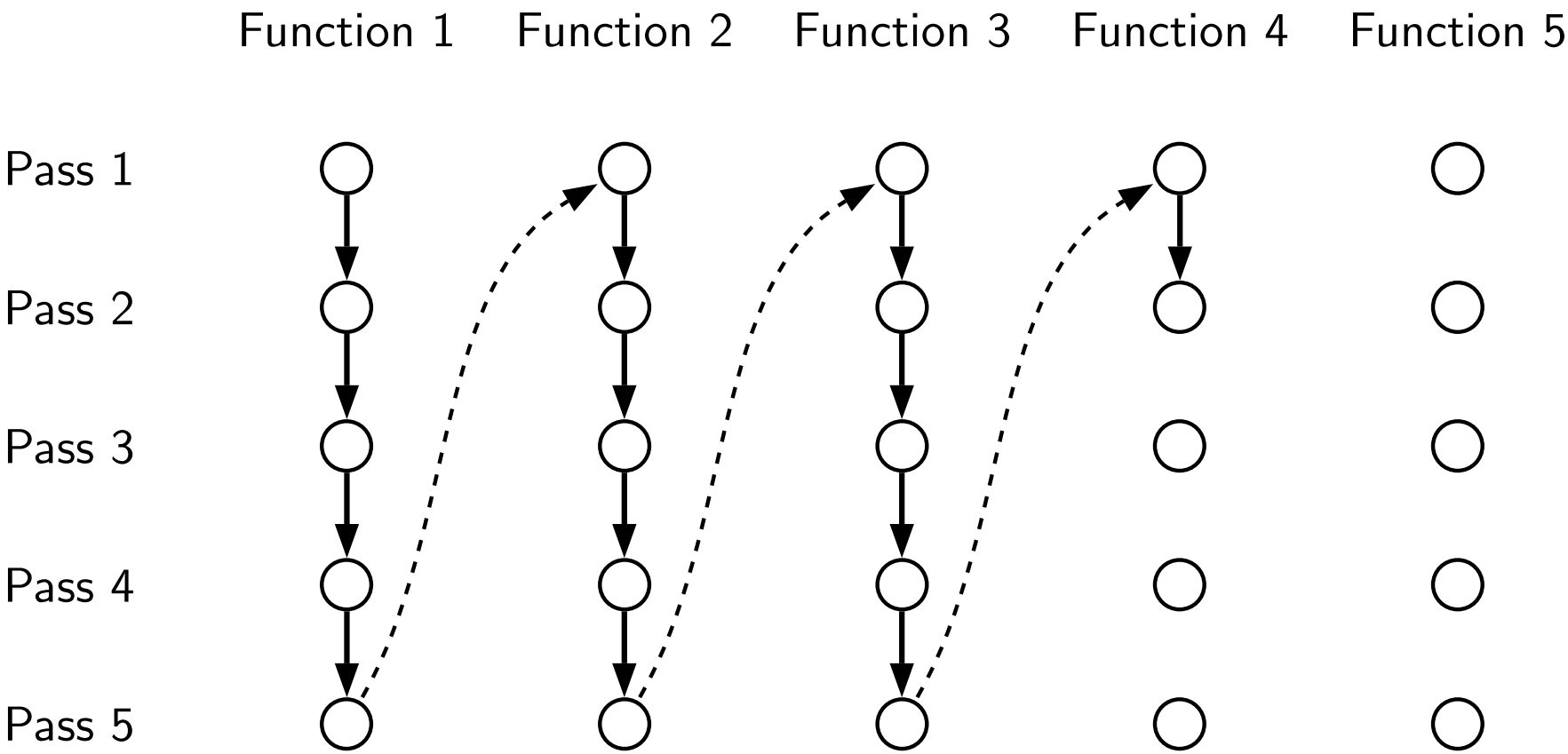
Execution Order in Intraprocedural Passes



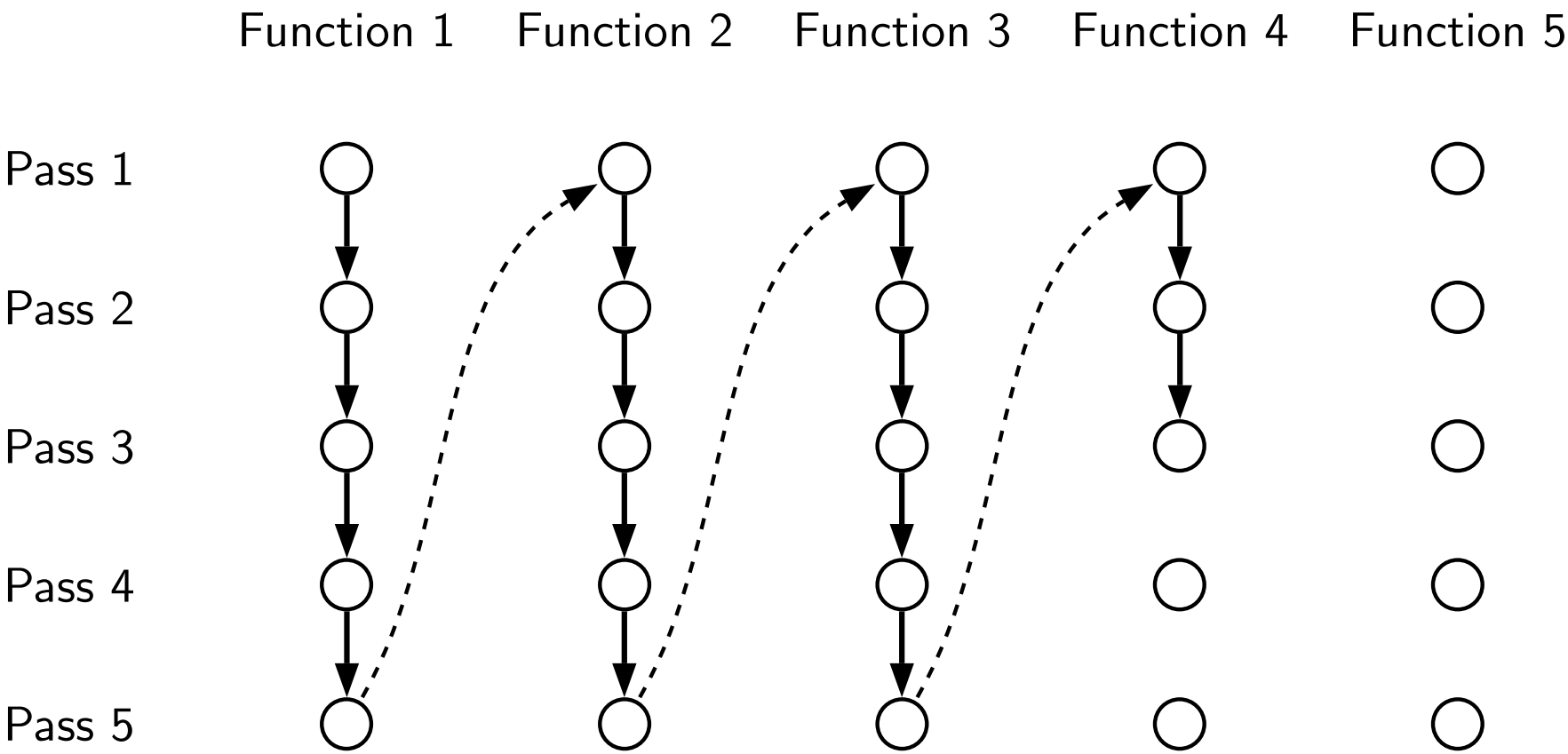
Execution Order in Intraprocedural Passes



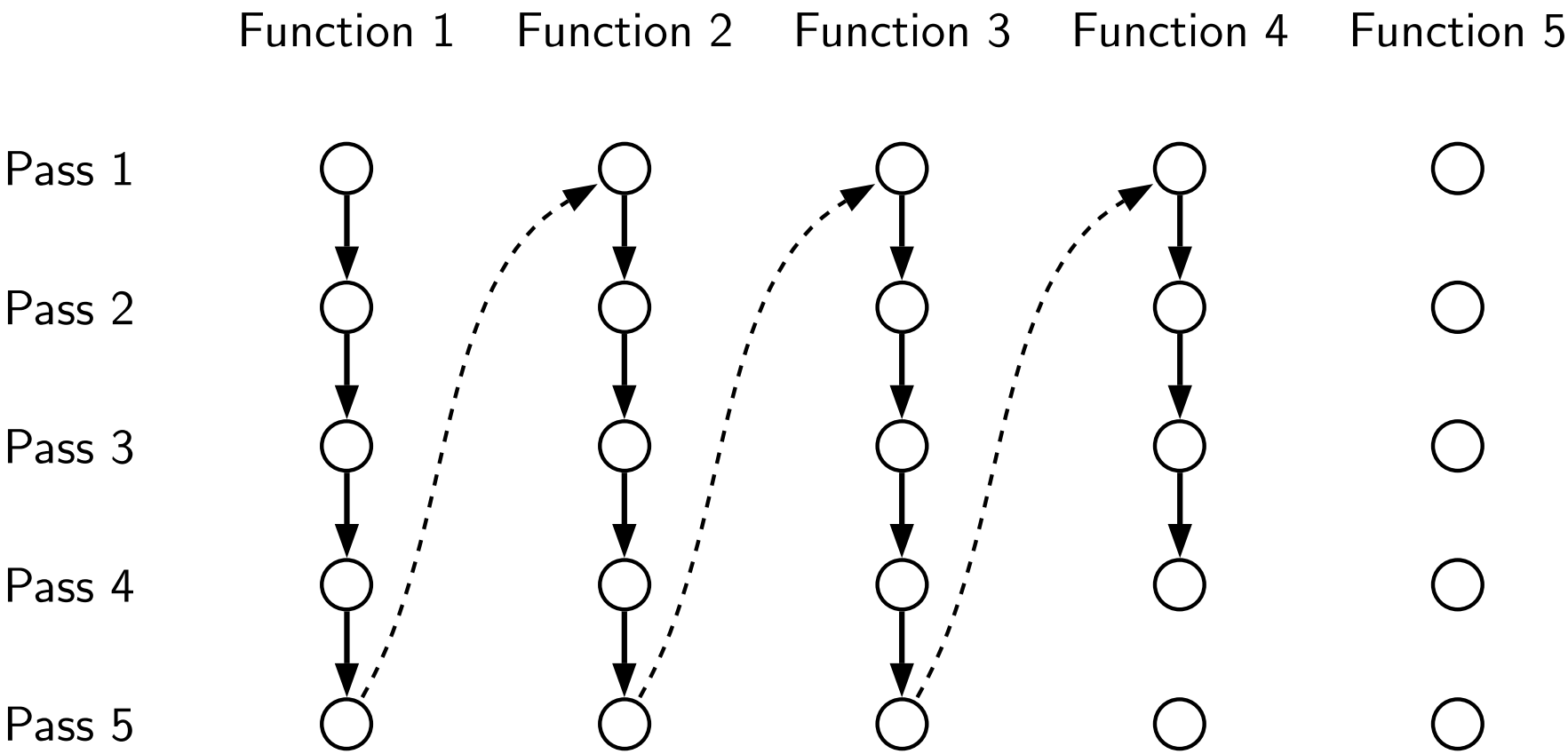
Execution Order in Intraprocedural Passes



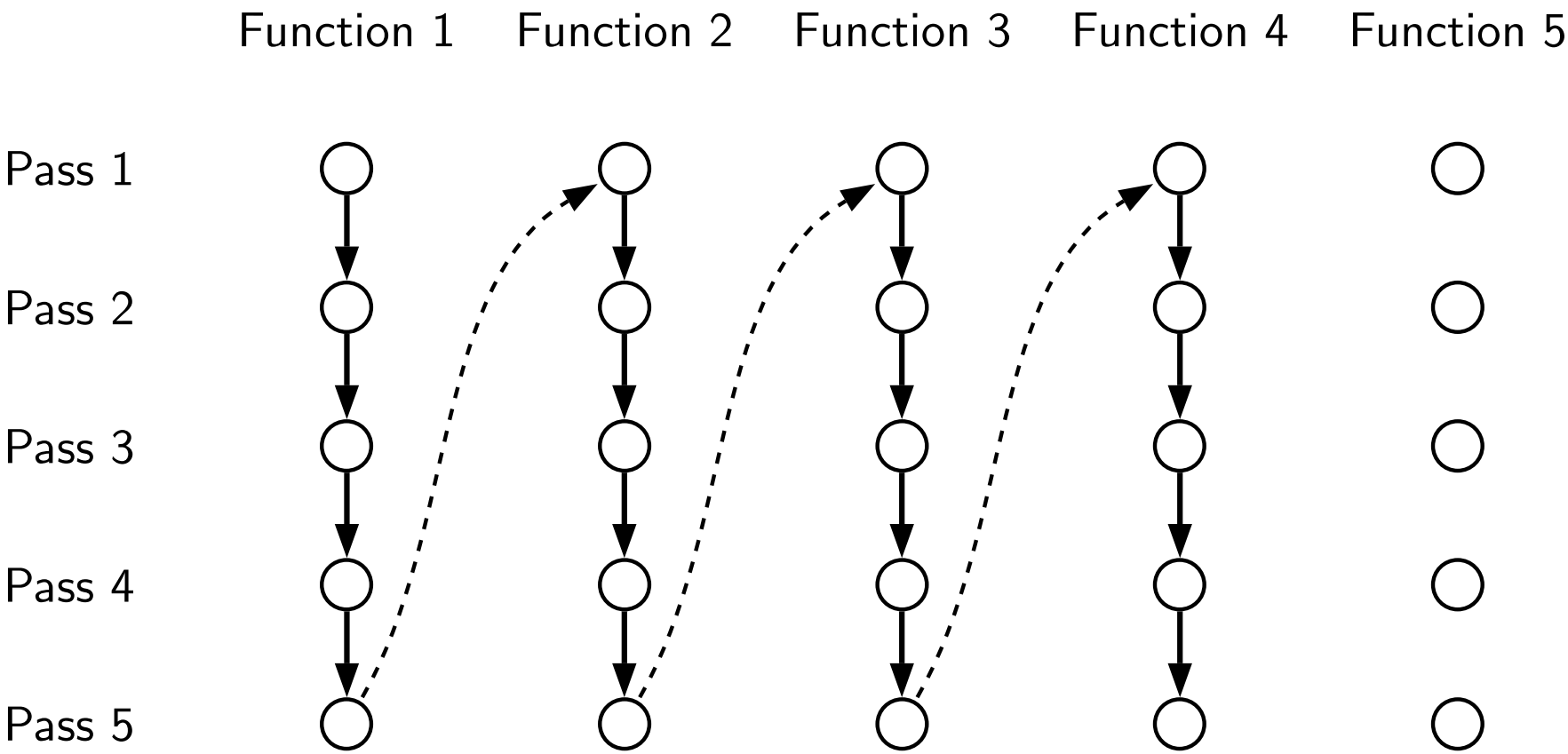
Execution Order in Intraprocedural Passes



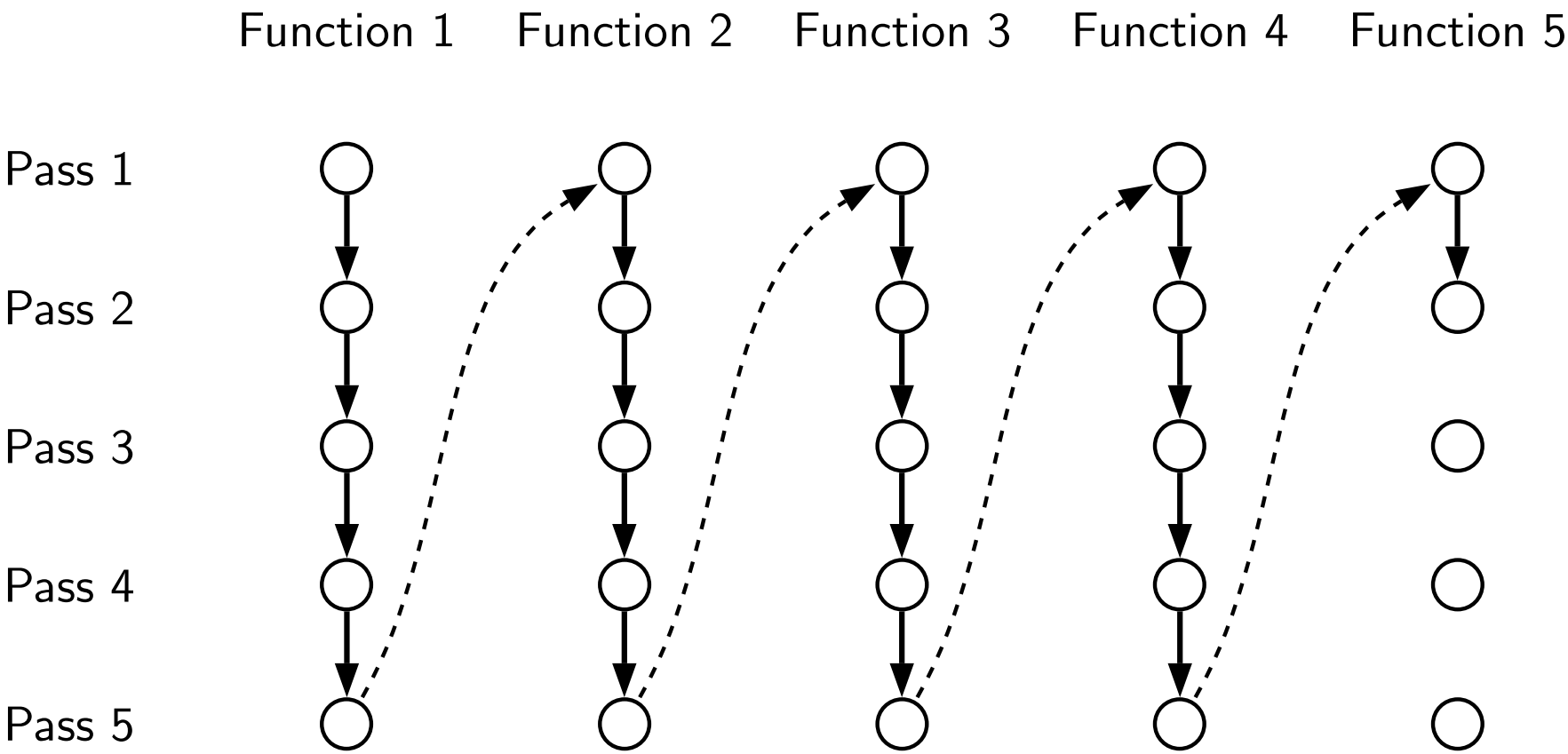
Execution Order in Intraprocedural Passes



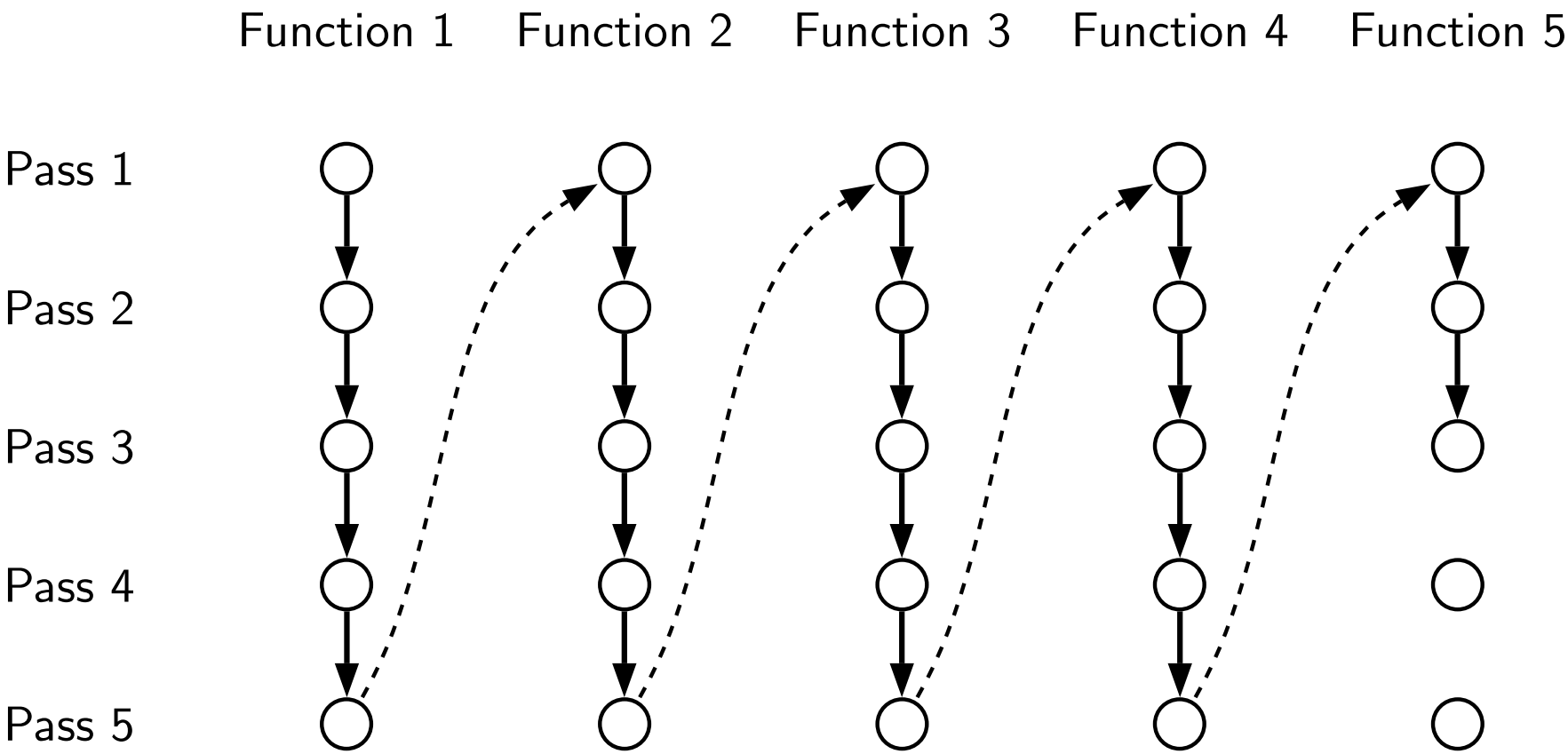
Execution Order in Intraprocedural Passes



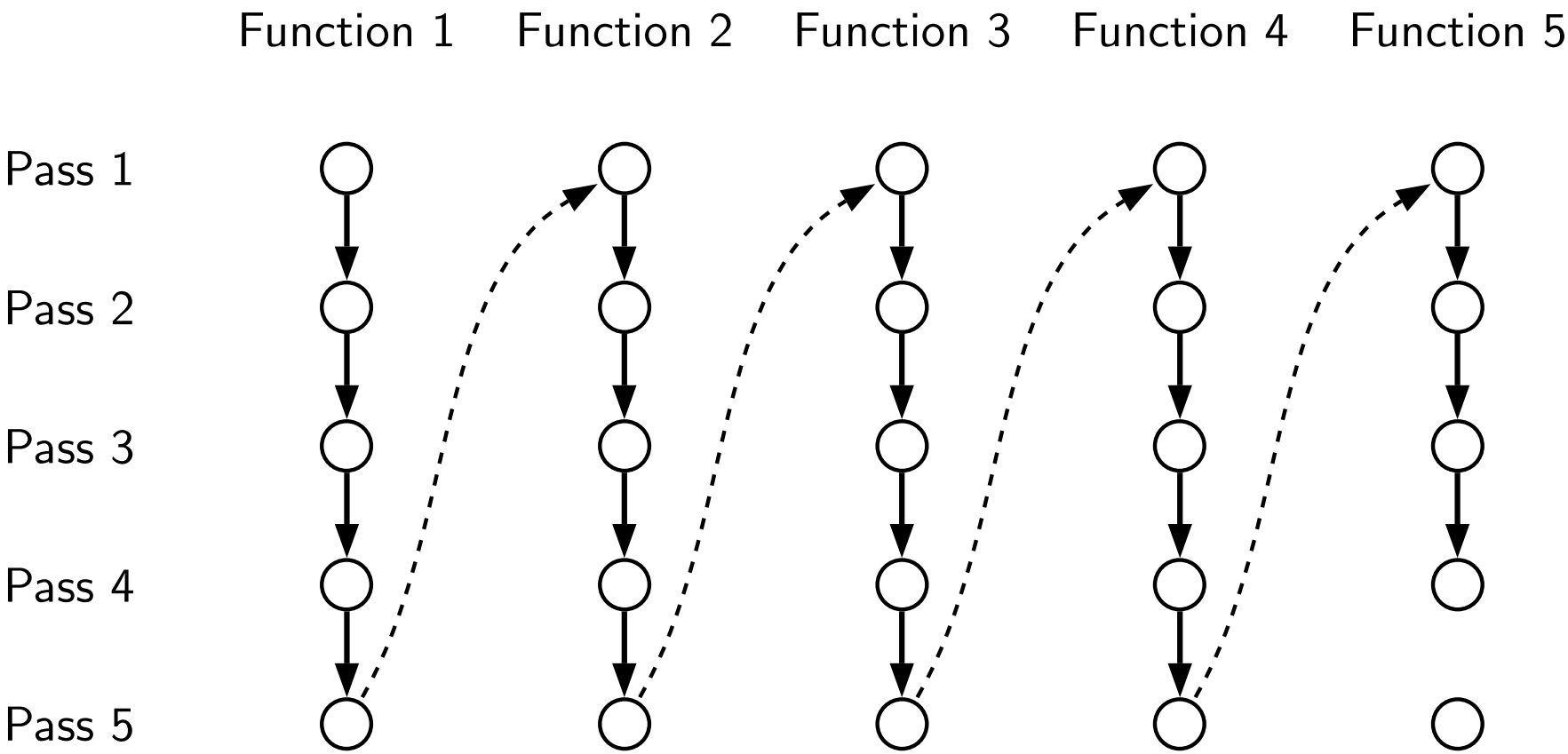
Execution Order in Intraprocedural Passes



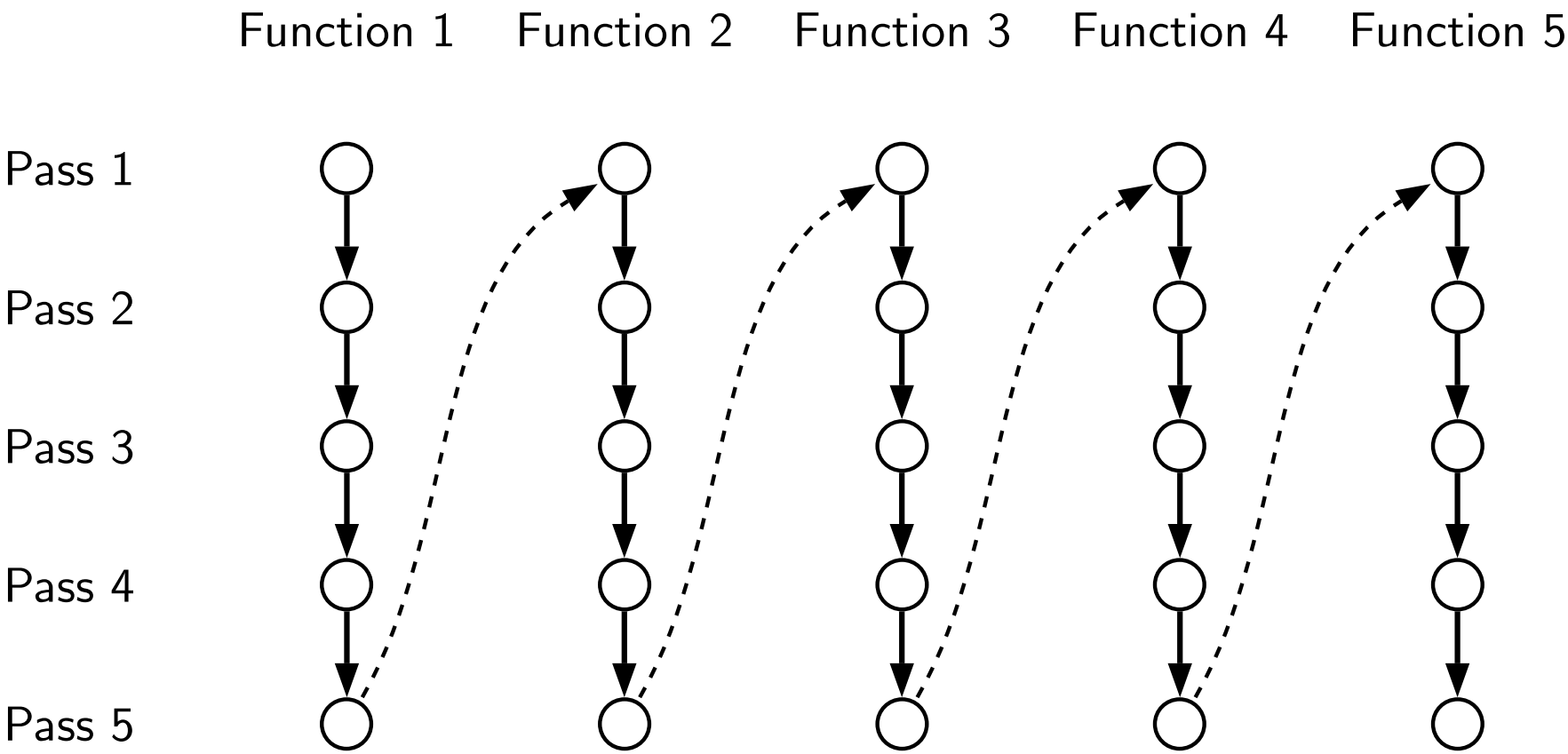
Execution Order in Intraprocedural Passes



Execution Order in Intraprocedural Passes



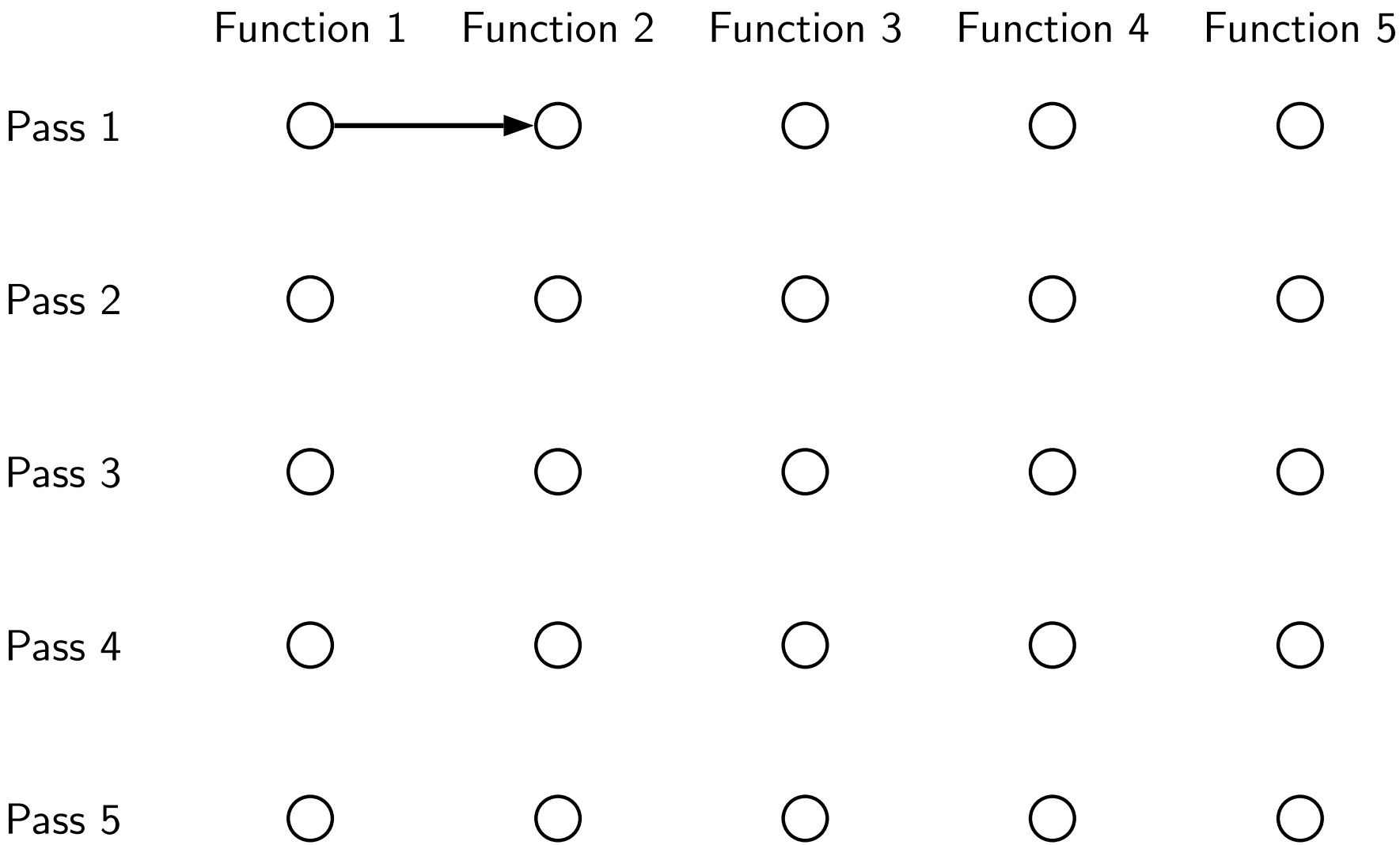
Execution Order in Intraprocedural Passes



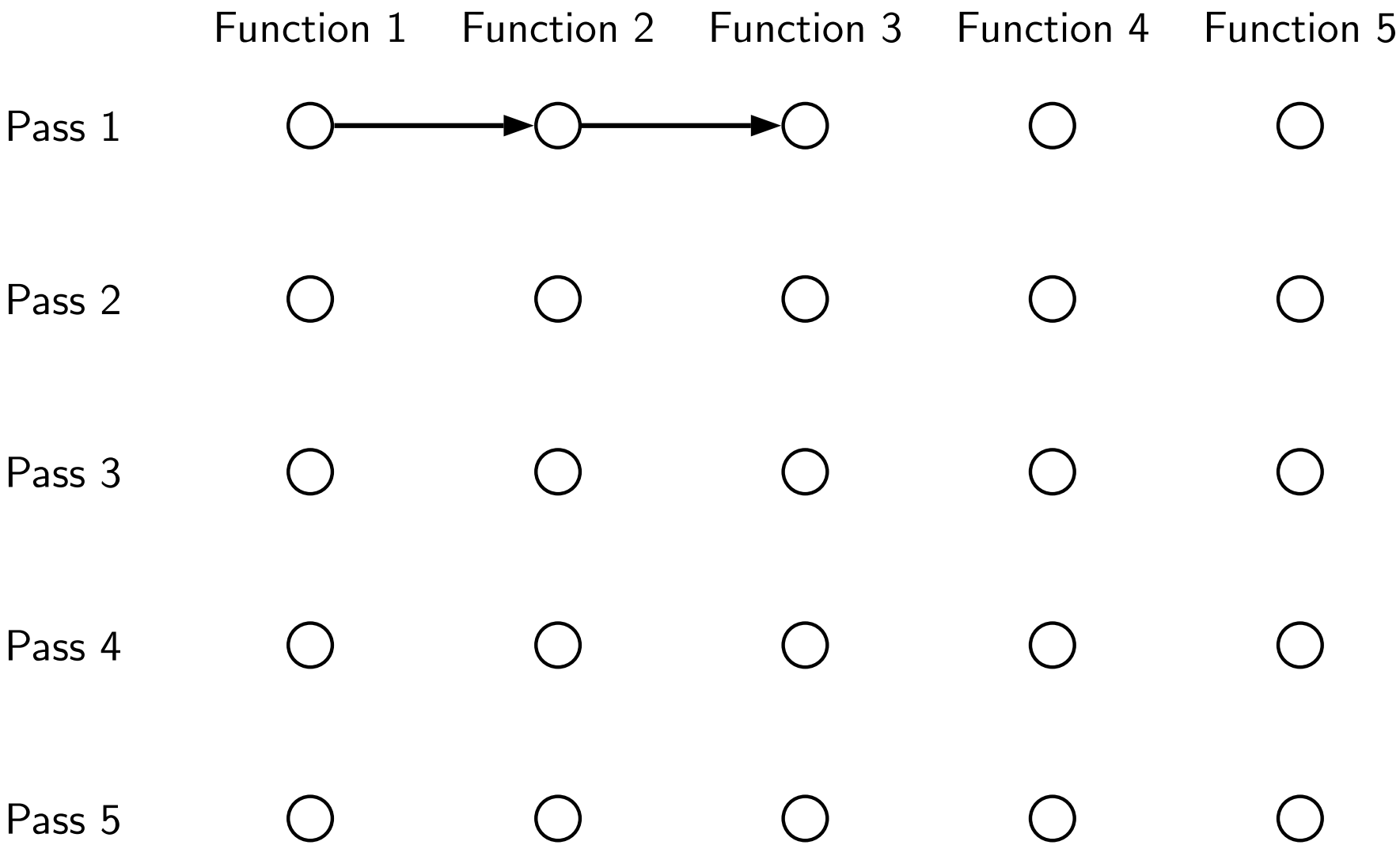
Execution Order in Interprocedural Passes

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

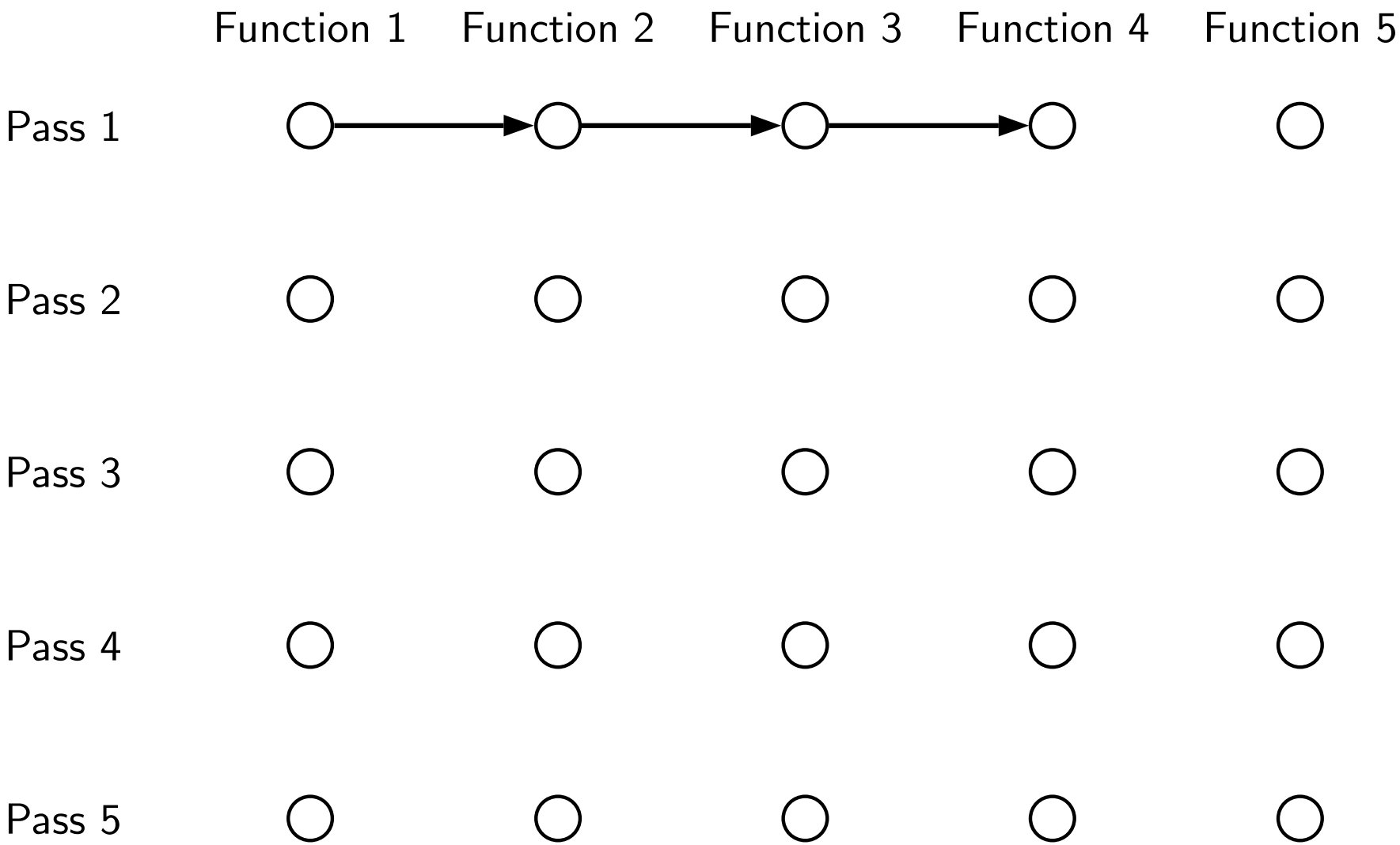
Execution Order in Interprocedural Passes



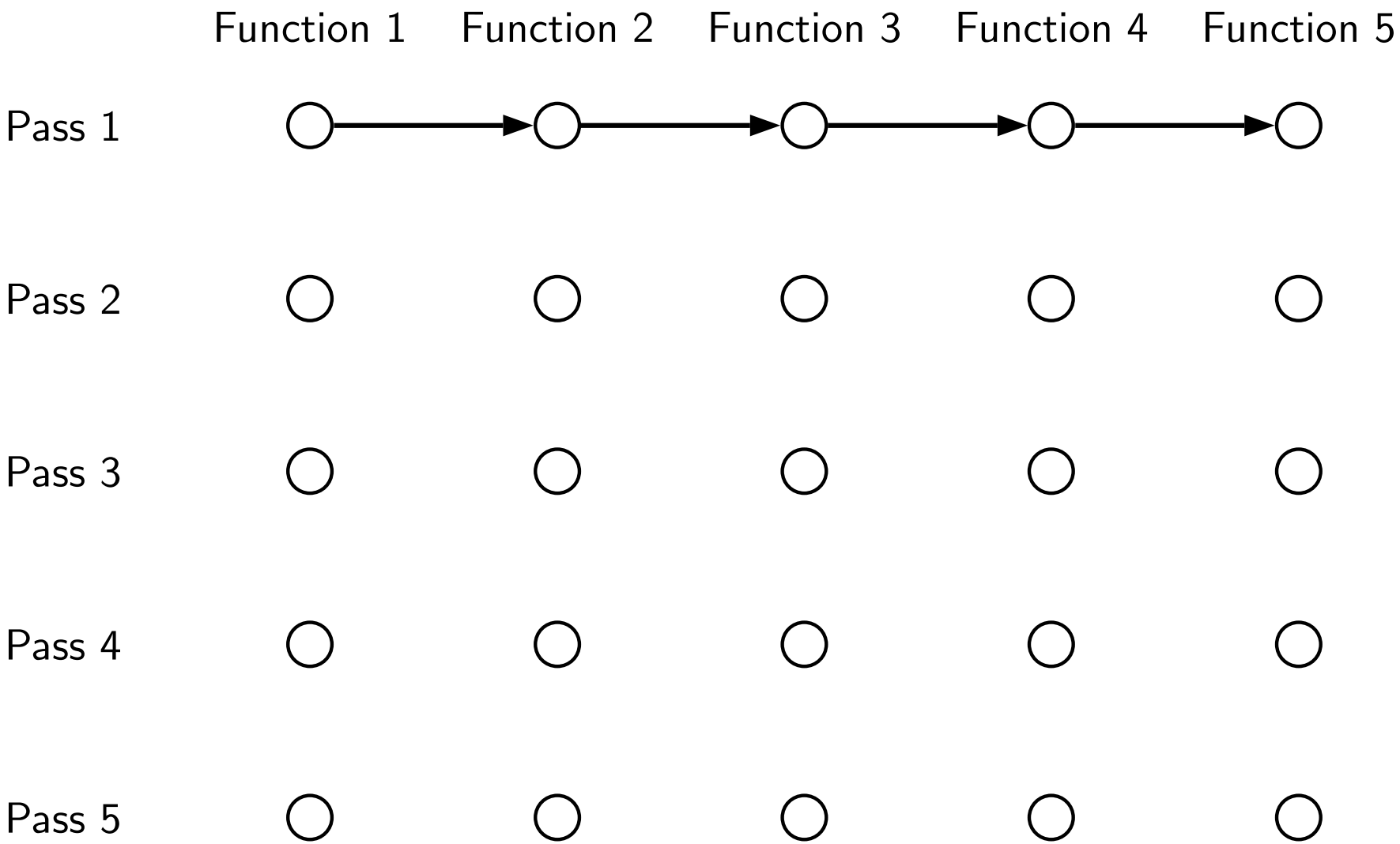
Execution Order in Interprocedural Passes



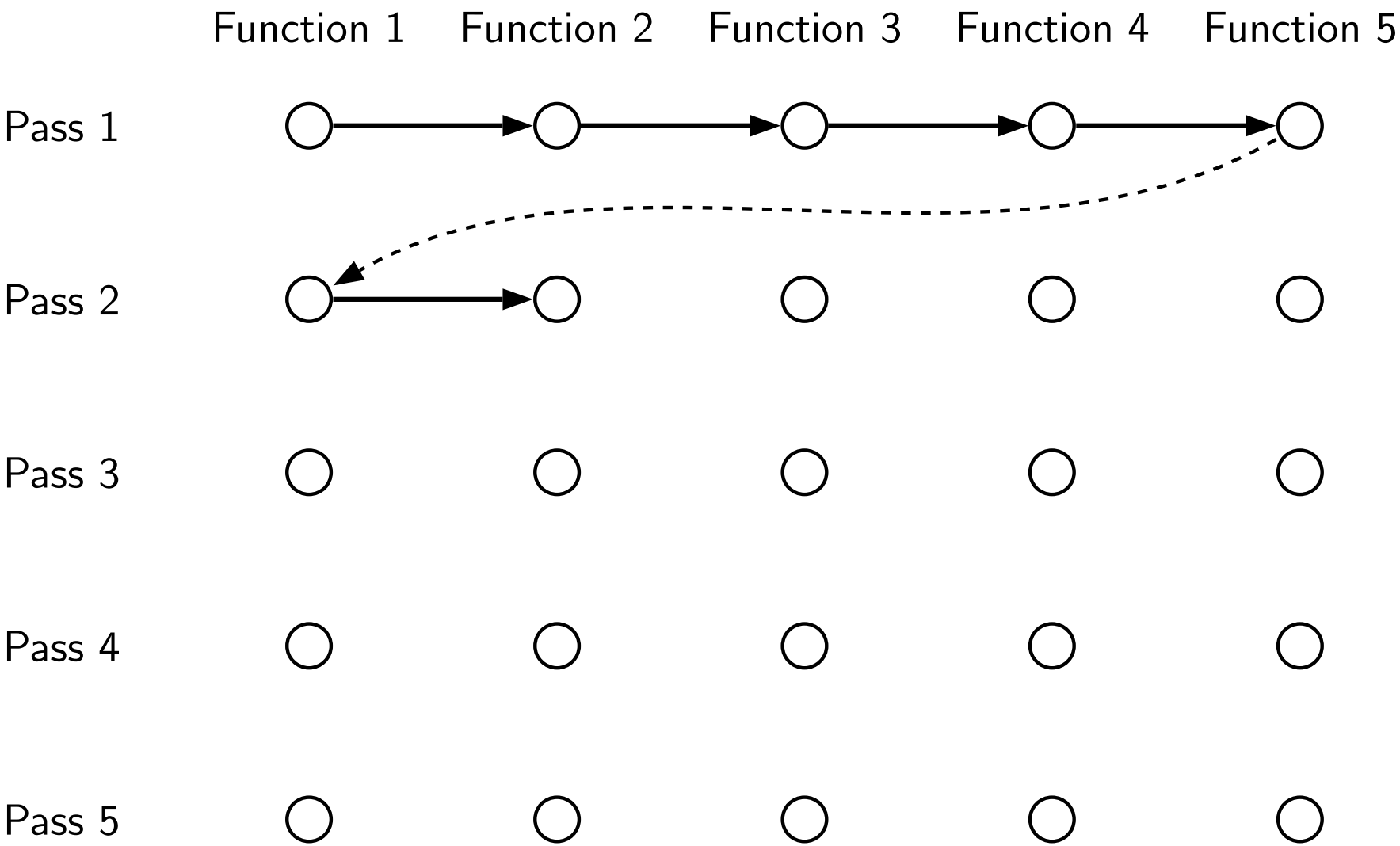
Execution Order in Interprocedural Passes



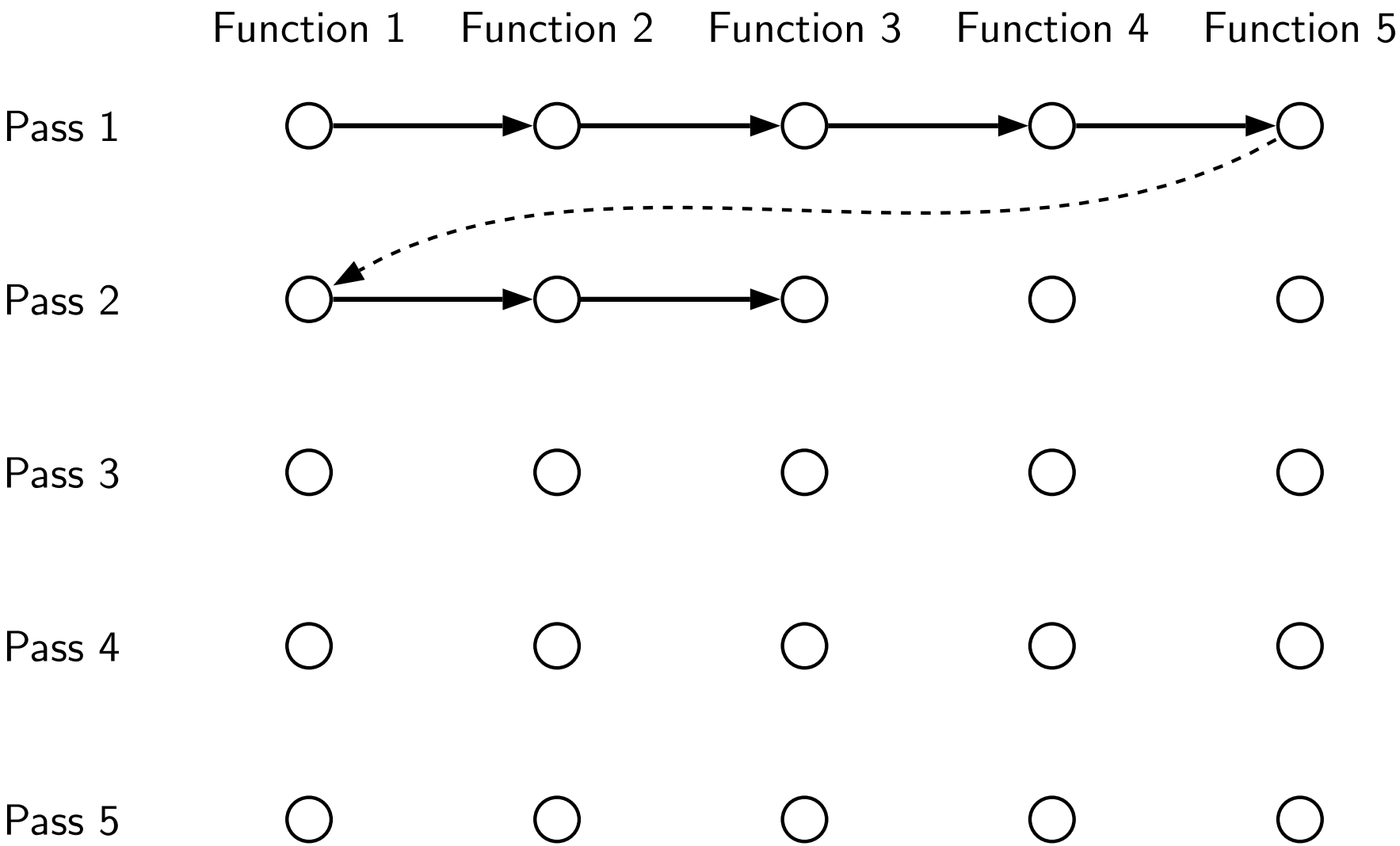
Execution Order in Interprocedural Passes



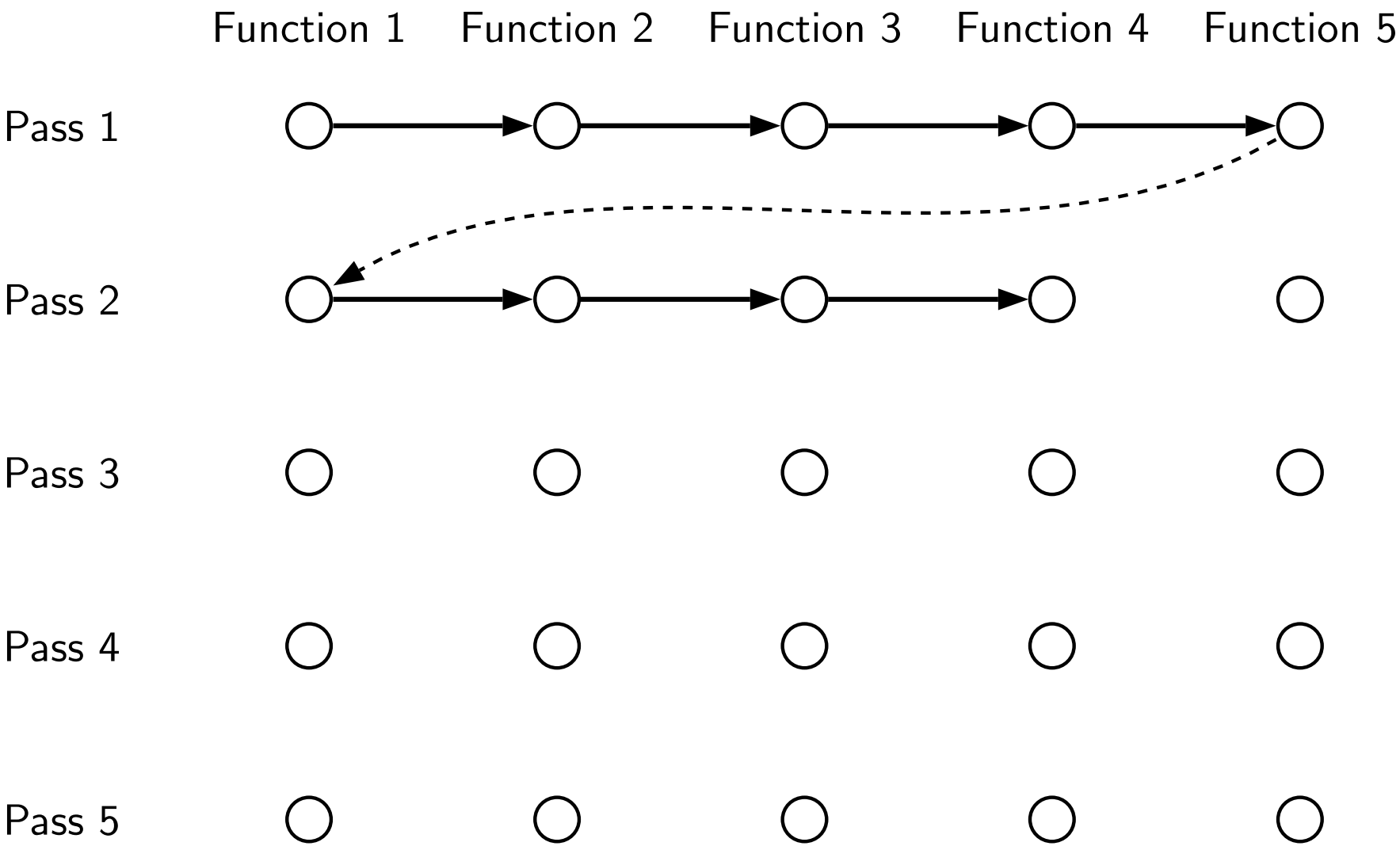
Execution Order in Interprocedural Passes



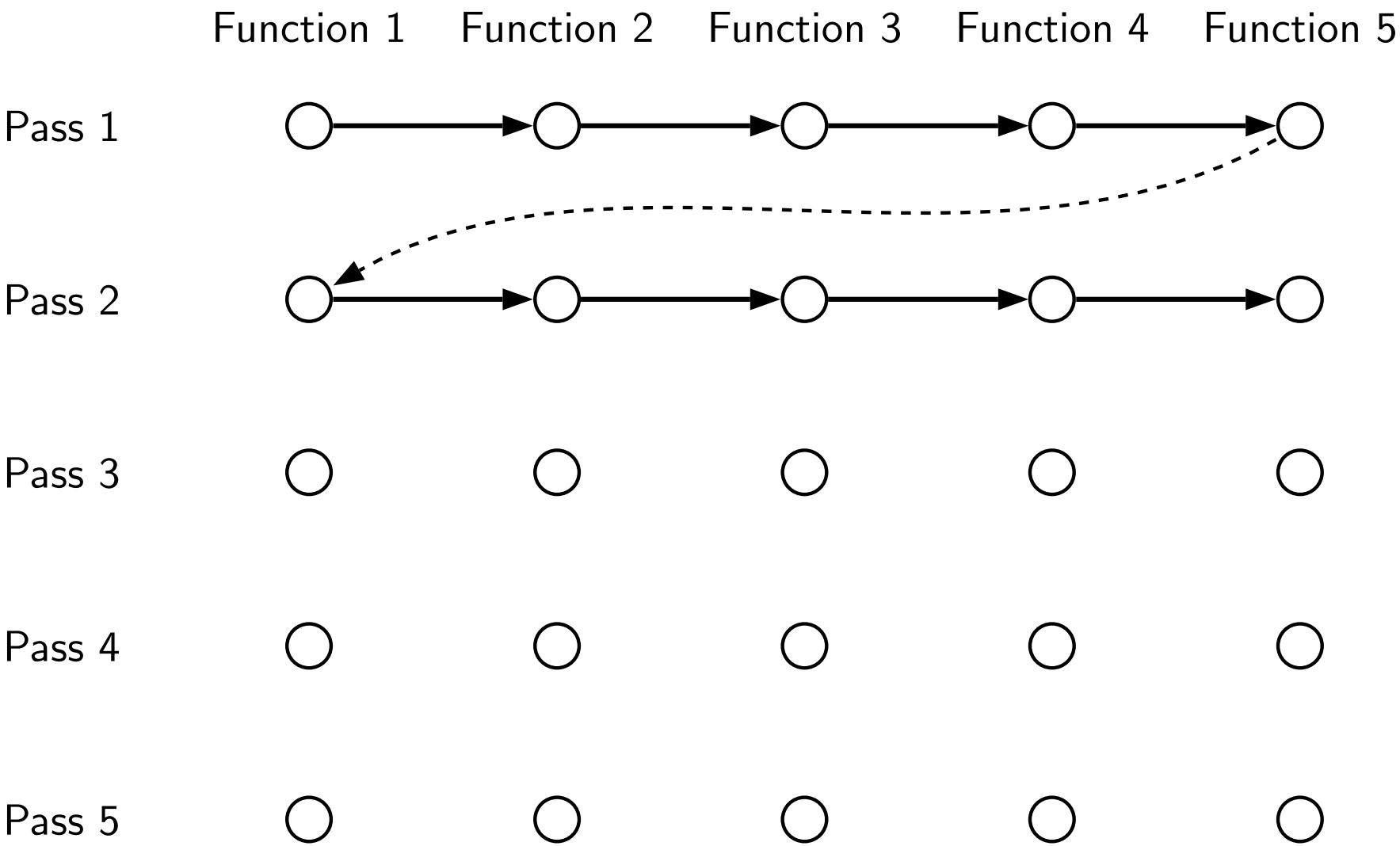
Execution Order in Interprocedural Passes



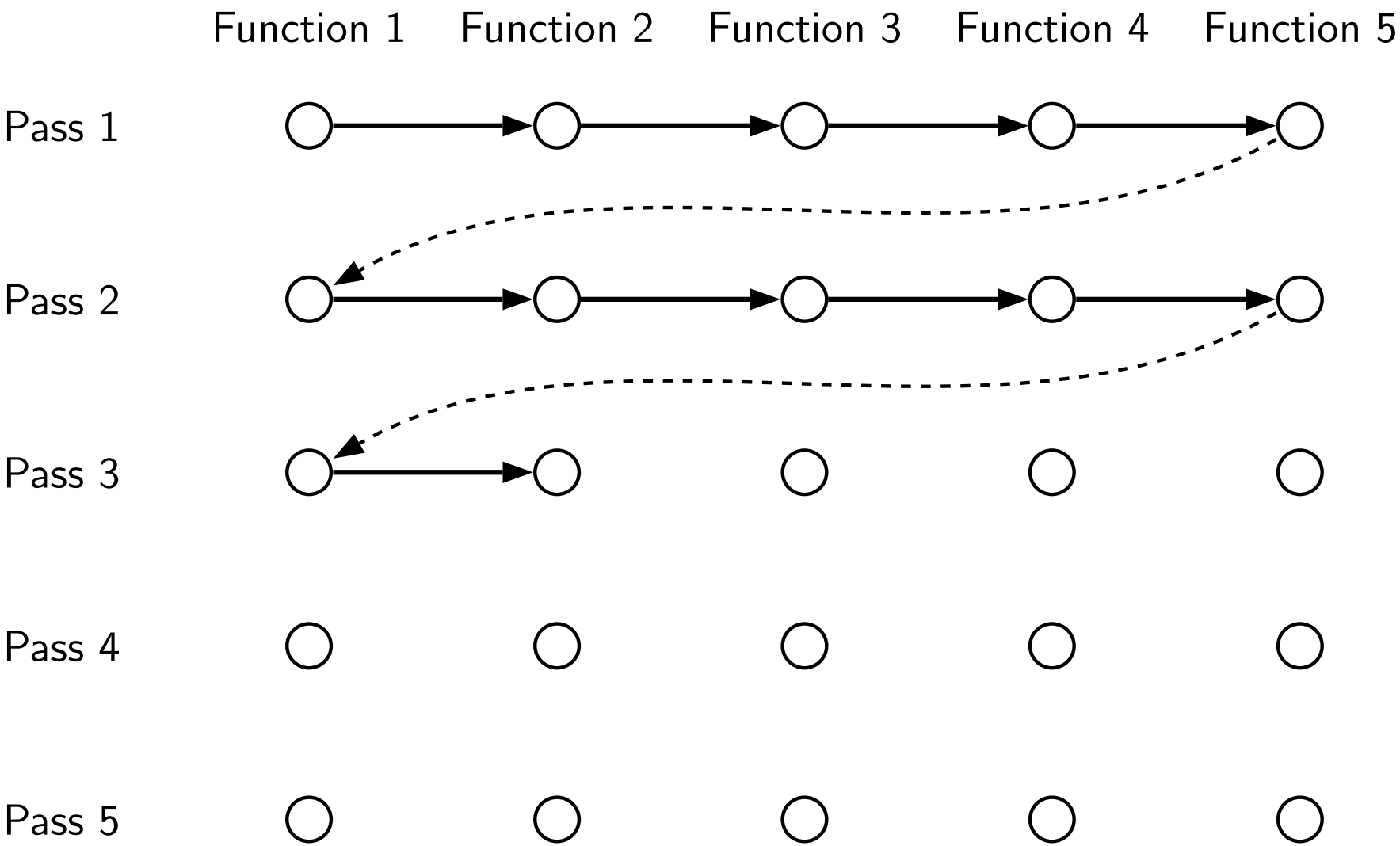
Execution Order in Interprocedural Passes



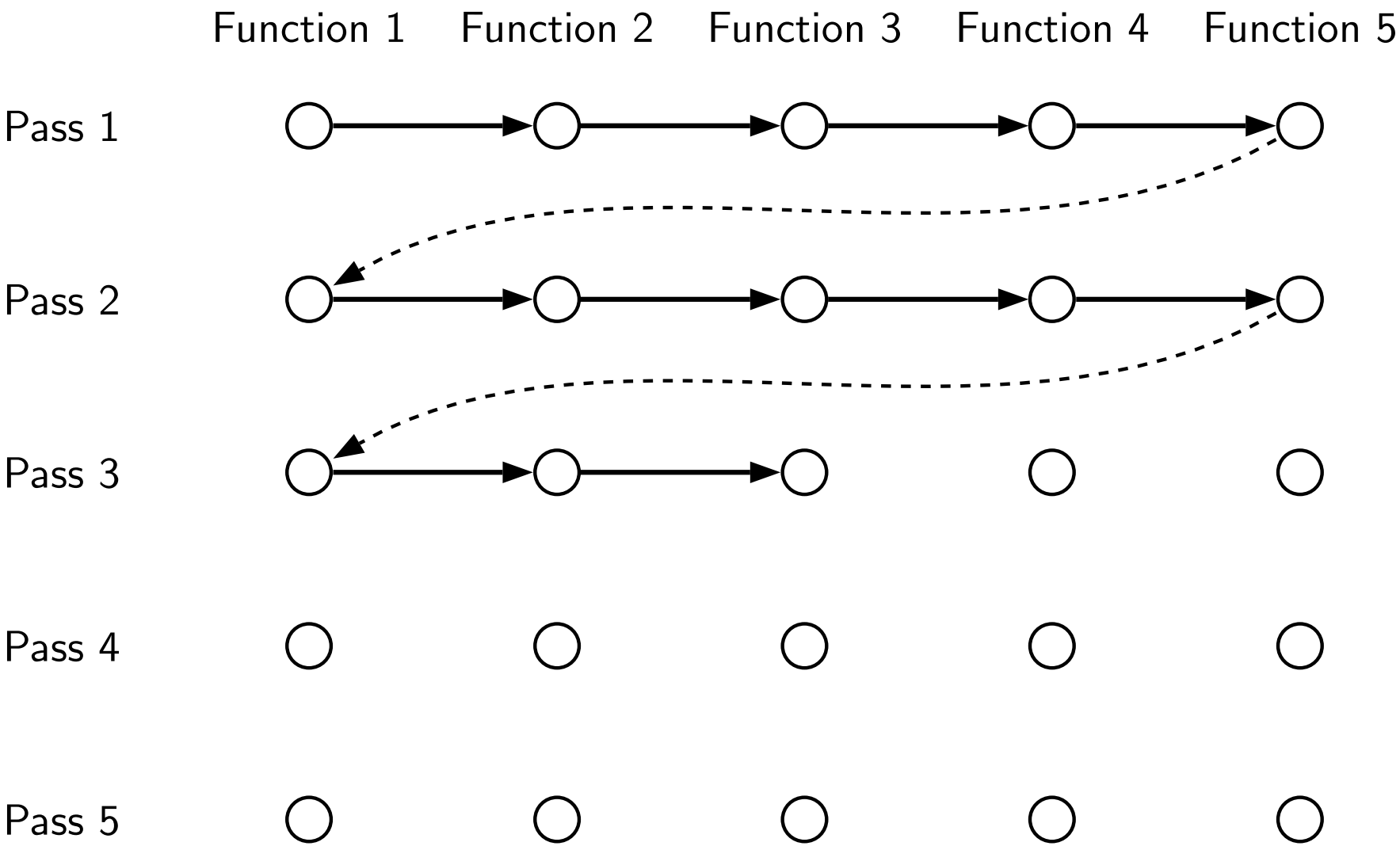
Execution Order in Interprocedural Passes



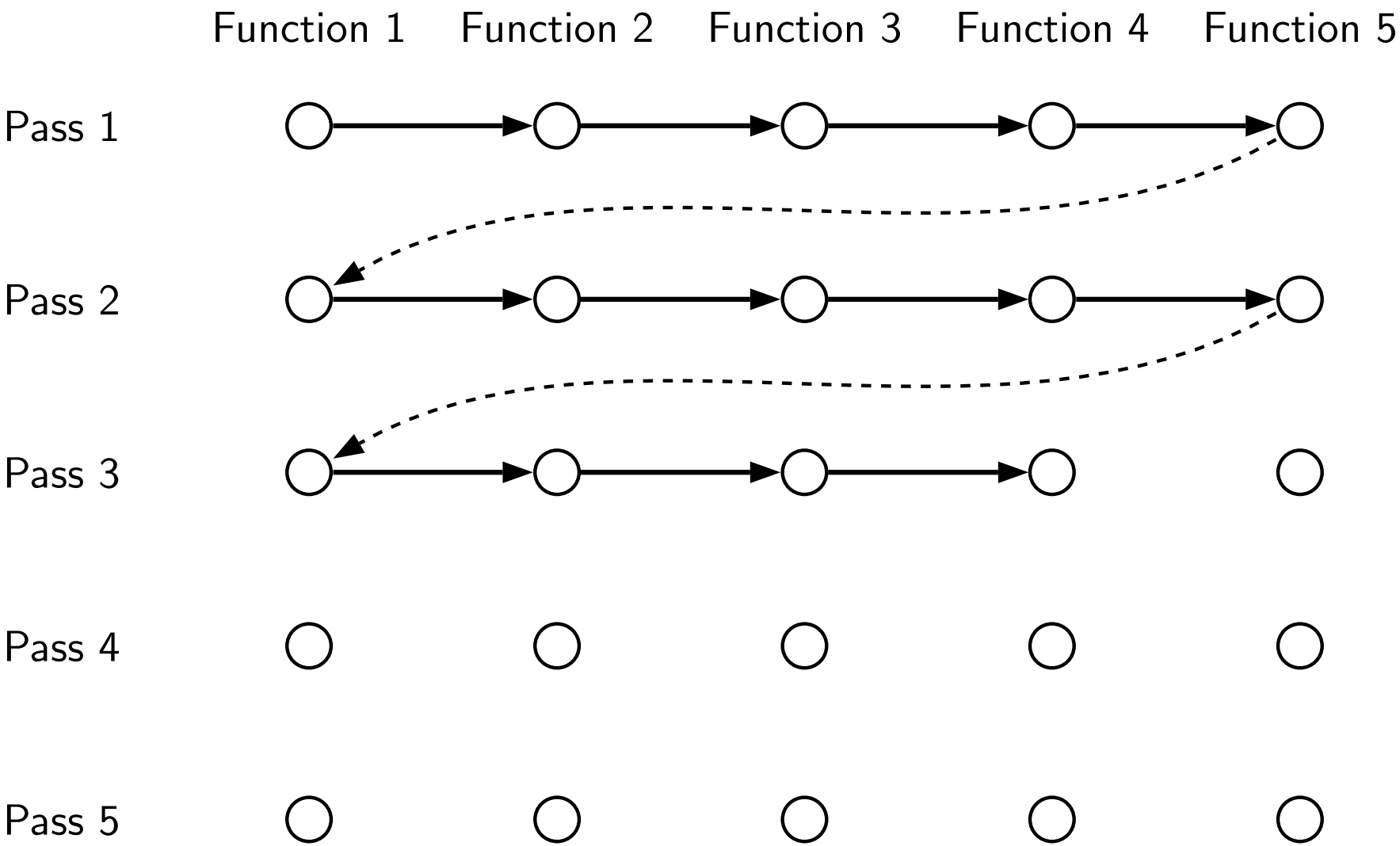
Execution Order in Interprocedural Passes



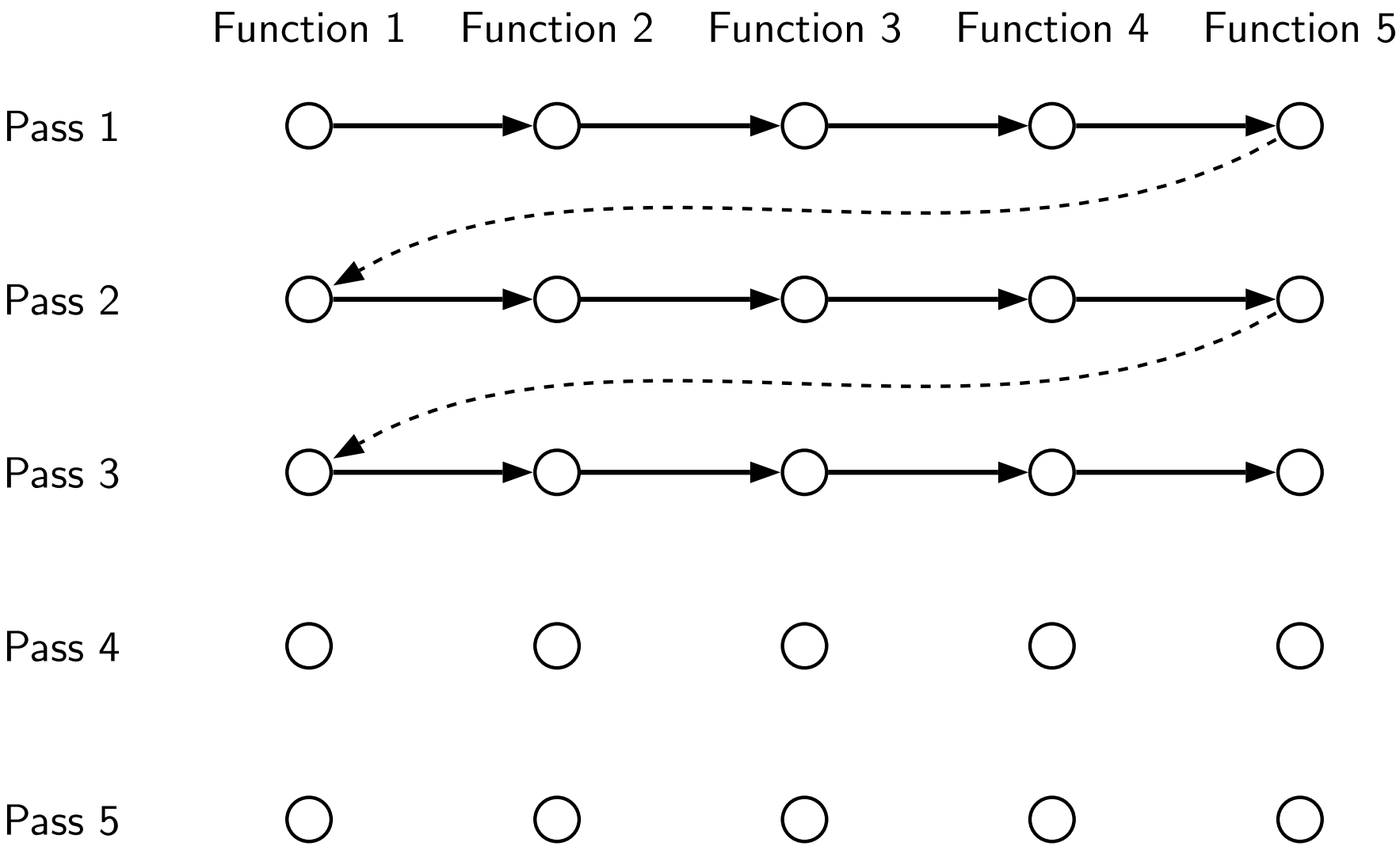
Execution Order in Interprocedural Passes



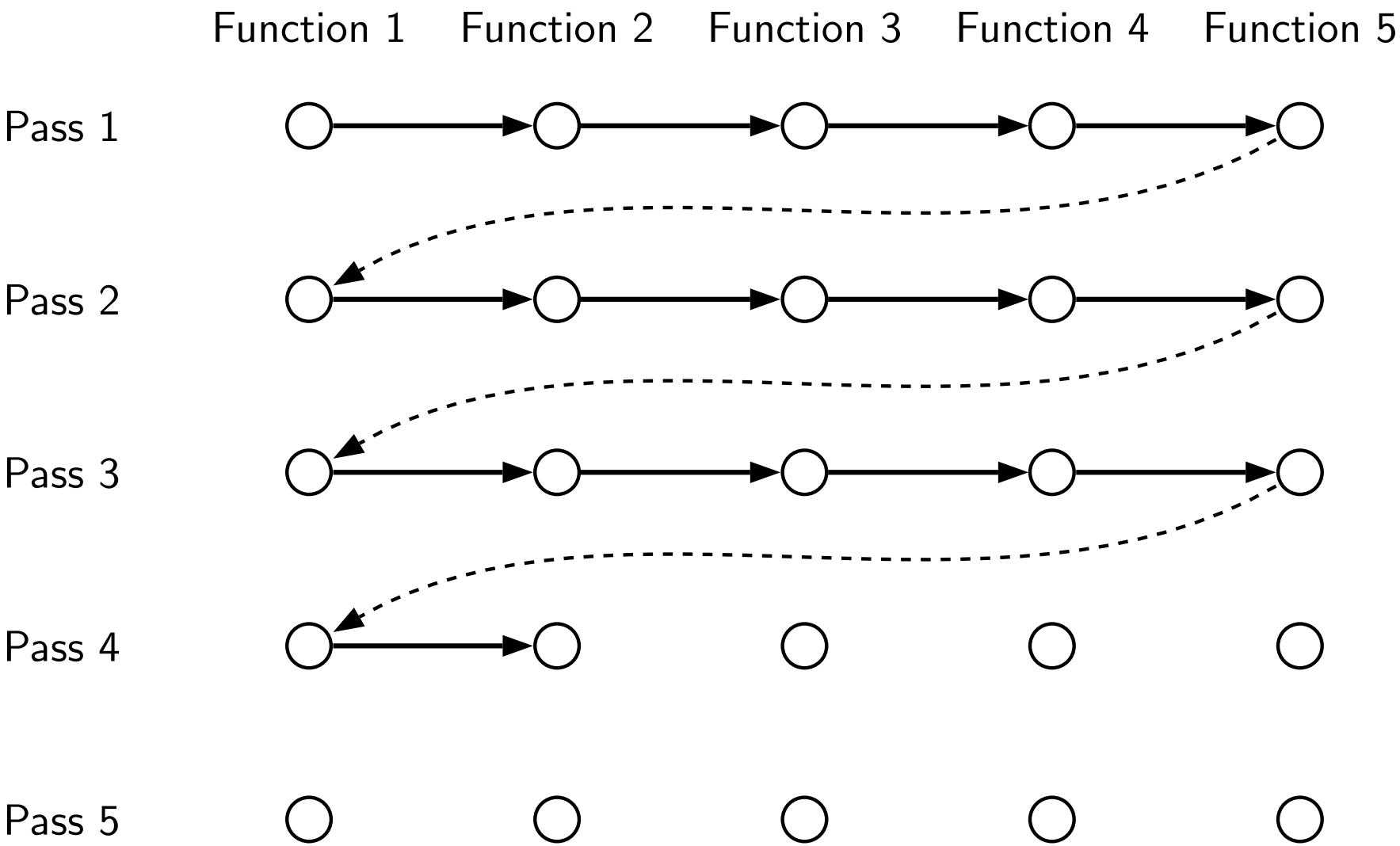
Execution Order in Interprocedural Passes



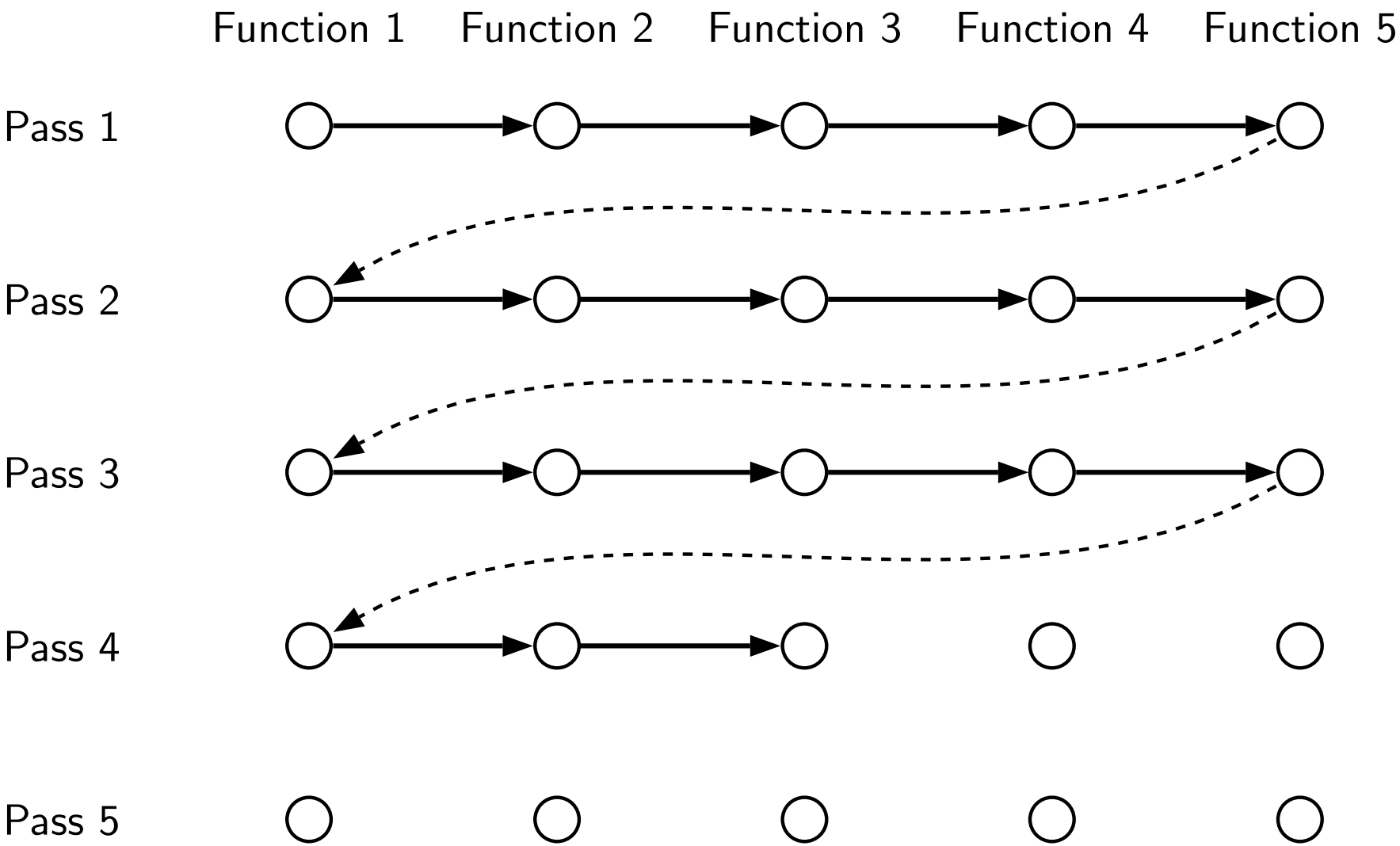
Execution Order in Interprocedural Passes



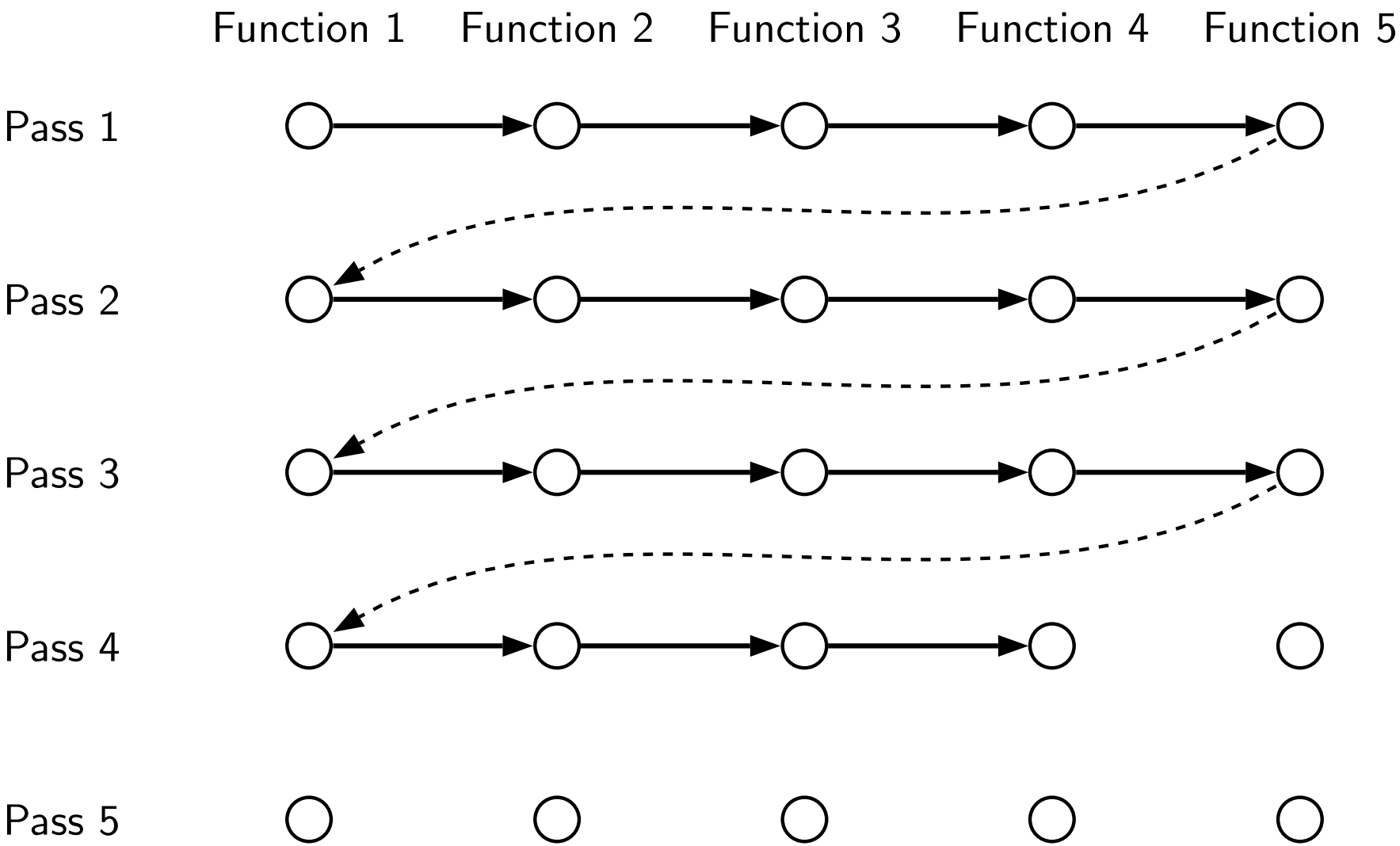
Execution Order in Interprocedural Passes



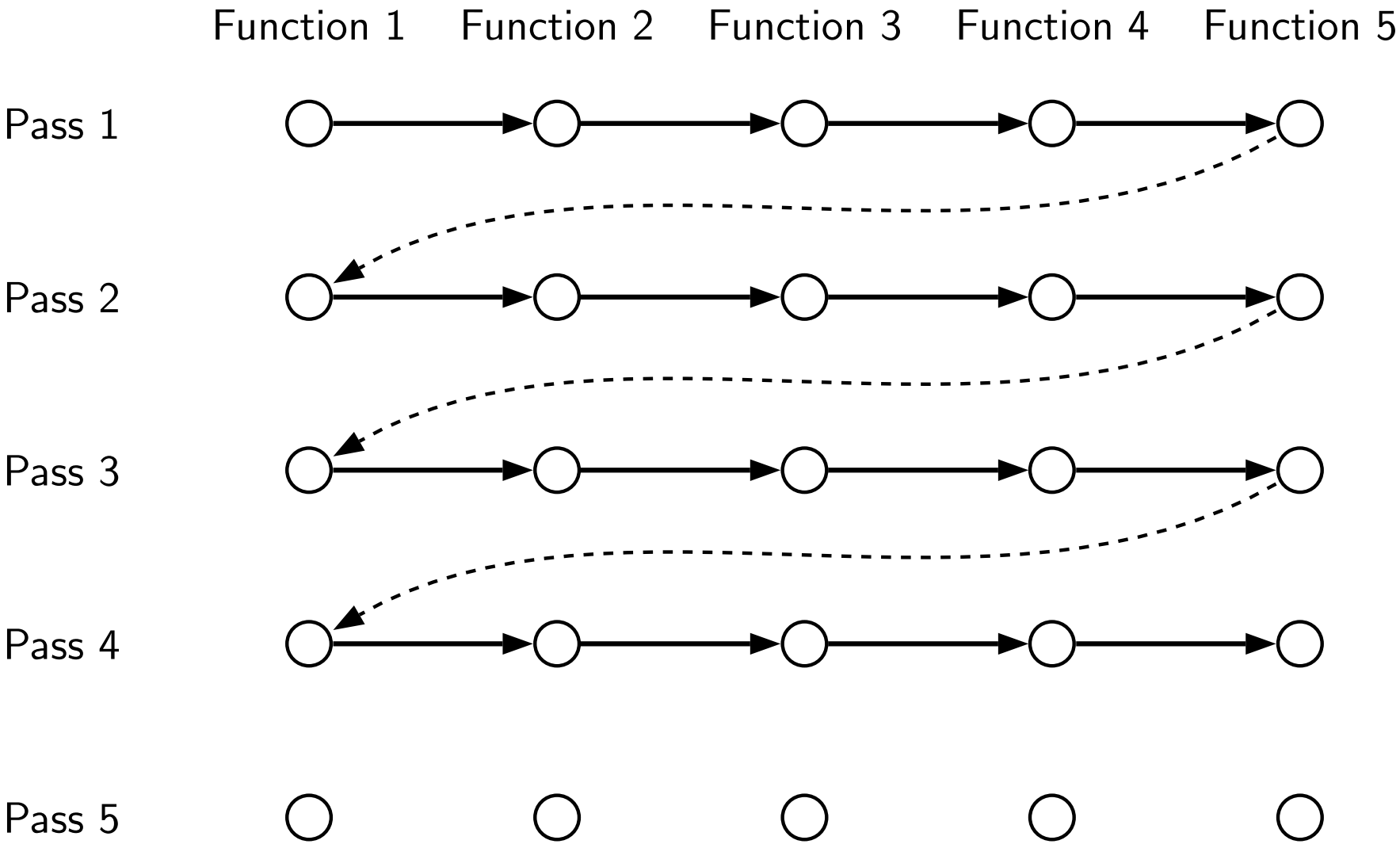
Execution Order in Interprocedural Passes



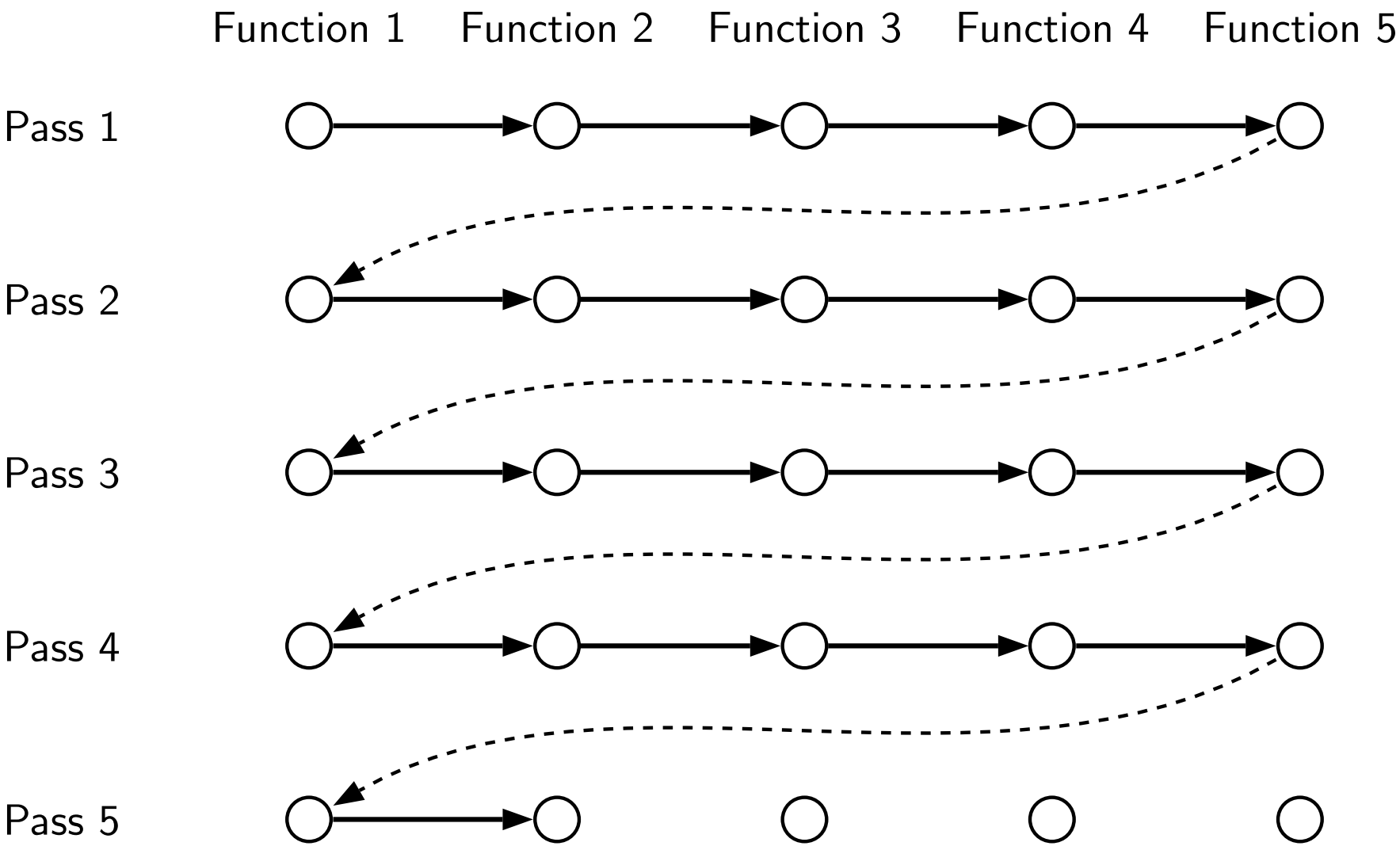
Execution Order in Interprocedural Passes



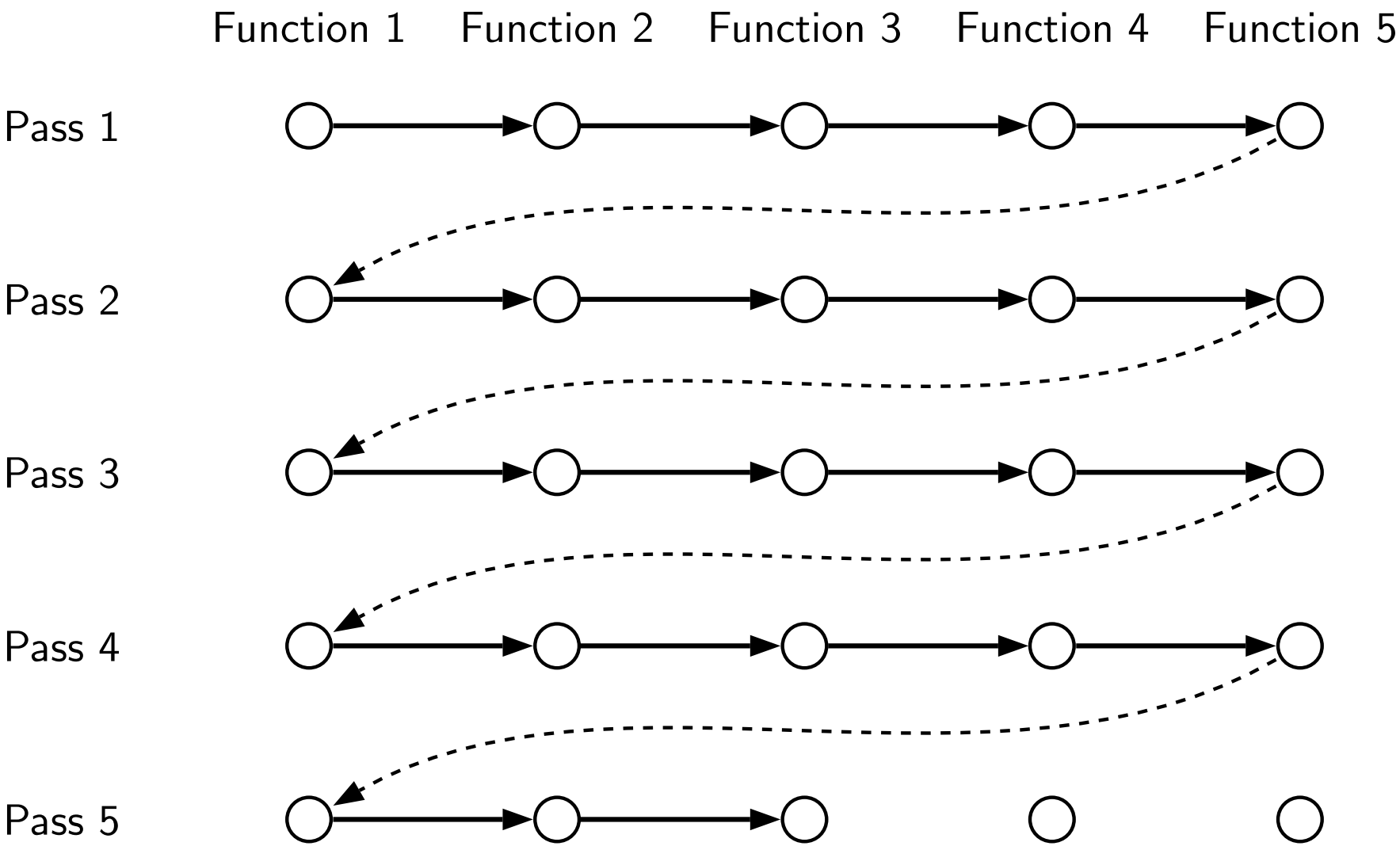
Execution Order in Interprocedural Passes



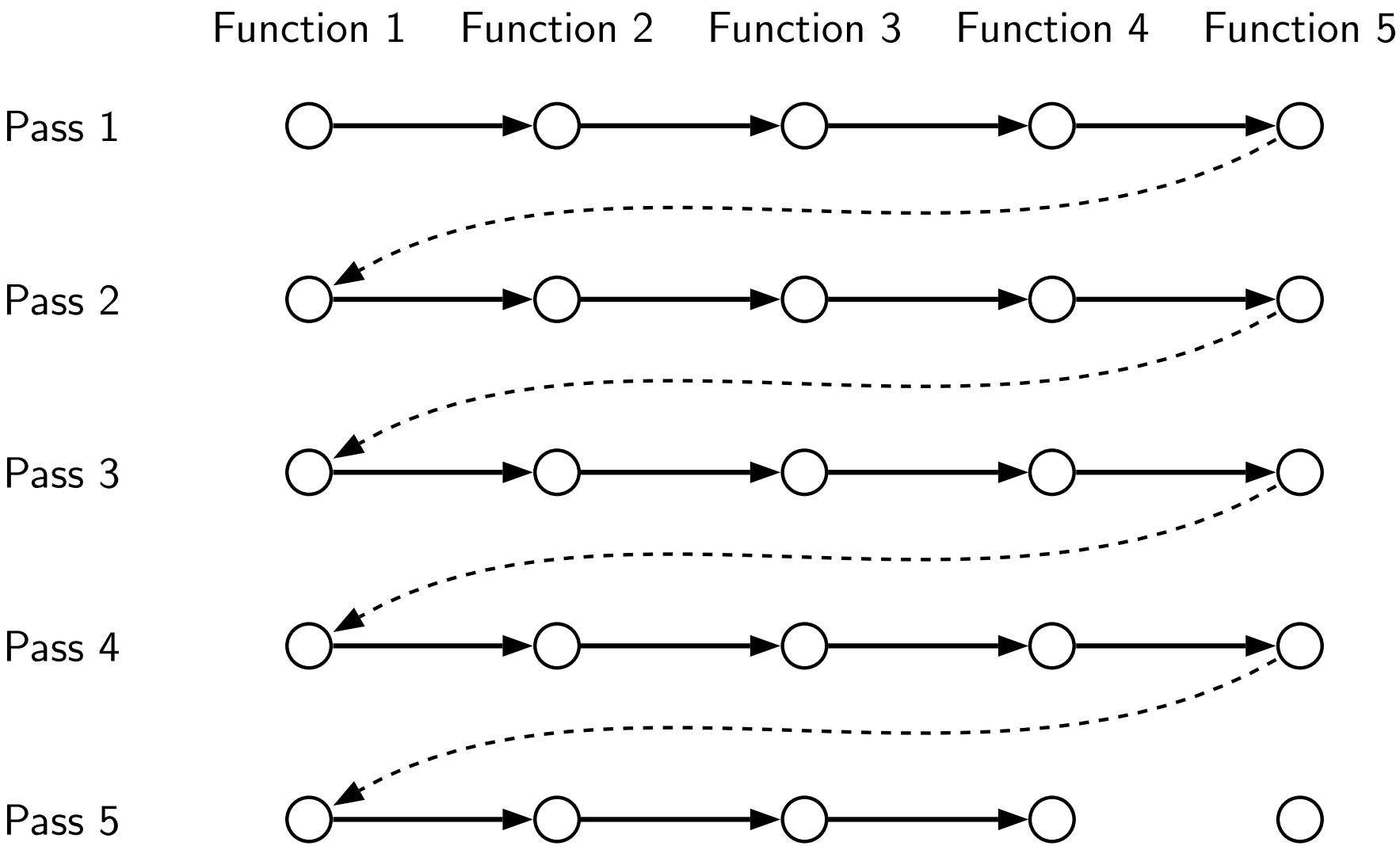
Execution Order in Interprocedural Passes



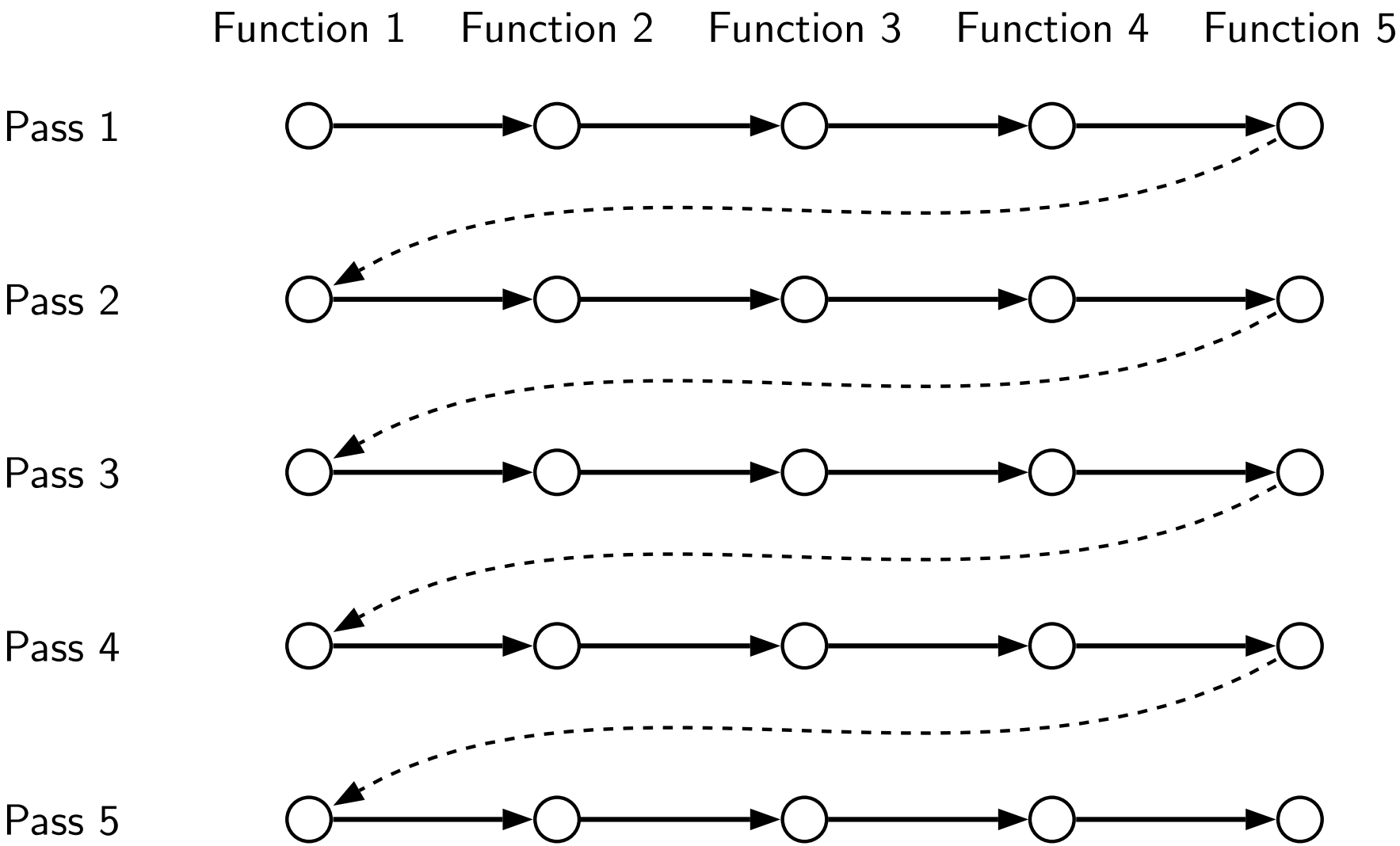
Execution Order in Interprocedural Passes



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);
                    GEN_FCN
                    emit_insn
```

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);
                          GEN_FCN
                          emit_insn
```


Part 5

Link Time Optimization

Motivation for Link Time Optimization

- Default `cgraph` creation is restricted to a translation unit (i.e. a single file)
  $\Rightarrow$ Interprocedural analysis and optimization is restricted to a single file
- All files (or their equivalents) are available only at link time
  (assuming static linking)
- LTO enables interprocedural optimizations across different files

Link Time Optimization

- LTO framework supported from GCC-4.6.0
- Use `-flto` option during compilation
- Generates conventional `.o` files with GIMPLE level information inserted
  Complete translation is performed in this phase
- During linking all object modules are put together and `lto1` is invoked
- `lto1` re-executes optimization passes from the function `cgraph_optimize`

Basic Idea: Provide a larger call graph to regular ipa passes

Understanding LTO Framework

```
main ()
{
     printf ("hello, world\n");
}
```

Assembly Output without LTO Information (1)

```
        .file "t0.c"
        .section .rodata
.LC0:
        .string "hello, world"
        .text
        .globl main
        .type main, @function
main:
.LFB0:
        .cfi_startproc
        pushl %ebp
        .cfi_def_cfa_offset 8
        .cfi_offset 5, -8
        movl %esp, %ebp
        .cfi_def_cfa_register 5
        andl $-16, %esp
        subl $16, %esp
        movl $.LC0, (%esp)
        call puts
        leave
        .cfi_restore 5
        .cfi_def_cfa 4, 4
        ret
        .cfi_endproc
.LFE0:
        .size main, .-main
        .ident "GCC: (GNU) 4.7.2"
        .section .note.GNU-stack,"",@pro
```


Assembly Output with LTO Information (2)

```
.ascii "\b"
.text
.section  .gnu.lto_.refs.57f4e8b14959f6c4,"",@progbits
.string "x\234cb‘d‘f‘‘‘b\200\001"
.string ""
.string "\204"
.ascii "\t"
.text
.section .gnu.lto_.statics.57f4e8b14959f6c4,"",@progbits
.string "x\234cb‘d‘b\300\016@\342\214\020&"
.string ""
.string "\375"
.ascii "\t"
.text
.section  .gnu.lto_.decls.57f4e8b14959f6c4,"",@progbits
.string "x\234\215R=O\002A\020\2359Ne\303IB!\201\n\032M\224h\374\0
.string "\3218\311\313\275\333\233\2317\363n5@\020q@p(\2565\200E\3
.string "\2004\370!\336mB\003~\2068\017\022tB\230‘\020\232\2046\24
.string "\022Z\023\372\b\345\247\261^\t\270\341v\357\355\210\307>\
```

Assembly Output with LTO Information (3)

```
.string "\3474\030\205KN\321;\346\034\367L\324\031\304\301"
.string "\3040\023\202\202\031\f\324\002&\336aT\261\"
.string "\024\313k\260\004\017\\\306 O\245\323 \375\347iWu\001\232
.string "\"\343\245\226\225\032\242\322\306\004\024]\261\244'\246"
.string "\273%\262\367P\3440\360\245A\b.8\257q~\302\263\257\341"
.string "\377\r\037\020\236h\020A\257qK-\"\277\300hO\006g\262"
.string "\347/vE^Ovc\036\032r\343\032\232\230a\324%.N\317G\006"
.string "\366\3442L\222\270\242\334Q\201\216\307\334o\207\276\342"
.string "\270%&\2661\3446E\377\037\374Q\320\364\013\"P\027\003\333
.string "\007\257\212^\335\254\252\353bD2\345\305\300\030\231\362"
.string "\273\326#\372[\0321\230\031j\204$\334Jg9\r\237\236\363\35
.string "\377\335\273%d\363\346V>\271\221J\301Teu\245"
.ascii "o\026\005\213."
.text
.section .gnu.lto_.symtab.57f4e8b14959f6c4,"",@progbits
.string "main"
.string ""
.string ""
.string ""
.string ""
```


Assembly Output with LTO Information (4)

```
        .string ""
        .string ""
        .string ""
        .string ""
        .string ""
        .string ""
        .string ""
        .string "\240"
        .string ""
        .string ""
        .text
        .section .gnu.lto_.opts,"",@progbits
        .string "'-fexceptions''-mtune=generic''-march=pentiumpro''-flto'"
        .text
        .section .rodata
.LC0:
        .string "hello, world"
```

Assembly Output with LTO Information (5)

```
        .text
        .globl main
        .type main, @function
main:
.LFB0:
        .cfi_startproc
        pushl %ebp
        .cfi_def_cfa_offset 8
        .cfi_offset 5, -8
        movl %esp, %ebp
        .cfi_def_cfa_register 5
        andl $-16, %esp
        subl $16, %esp
        movl $.LC0, (%esp)
        call puts
```


Assembly Output with LTO Information (6)

```
        leave
        .cfi_restore 5
        .cfi_def_cfa 4, 4
        ret
        .cfi_endproc
.LFE0:
        .size main, .-main
        .comm __gnu_lto_v1,1,1
        .ident "GCC: (GNU) 4.7.2"
        .section .note.GNU-stack,"",@progbits
```

Interprocedural Optimizations Using LTO

Whole program optimization needs to see the entire program

- Does it need the entire program *together* in the memory?

  Load only the call graph without function bodies

  - Independent computation of summary information of functions
  - "Adjusting" summary information through whole program analysis over the call graph
  - Perform transformation independently on functions

- Process the entire program together

Why Avoid Loading Function Bodies?

- Practical programs could be rather large and compilation could become very inefficient
- Many optimizations decisions can be taken by looking at the call graph alone
  - Procedure Inlining: just looking at the call graph is sufficient
    Perhaps some summary size information can be used
  - Procedure Cloning: some additional summary information about actual parameters of a call is sufficient

Partitioned and Non-Partitioned LTO

Load complete call graph

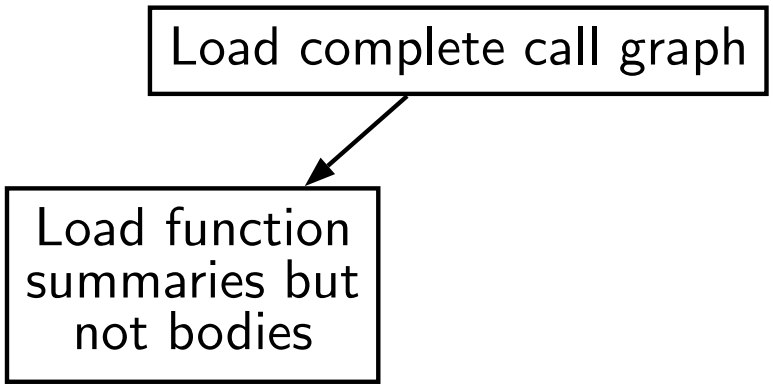
Analysis

Sequential Analysis

Transformation

Partitioned and Non-Partitioned LTO

Analysis

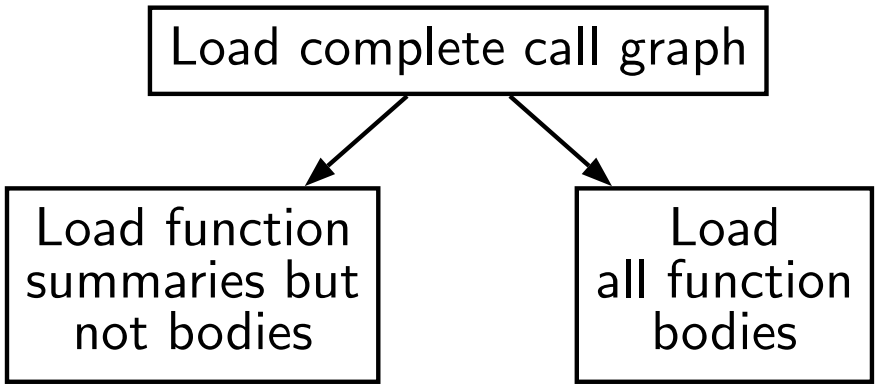


Sequential Analysis

Transformation

Partitioned and Non-Partitioned LTO

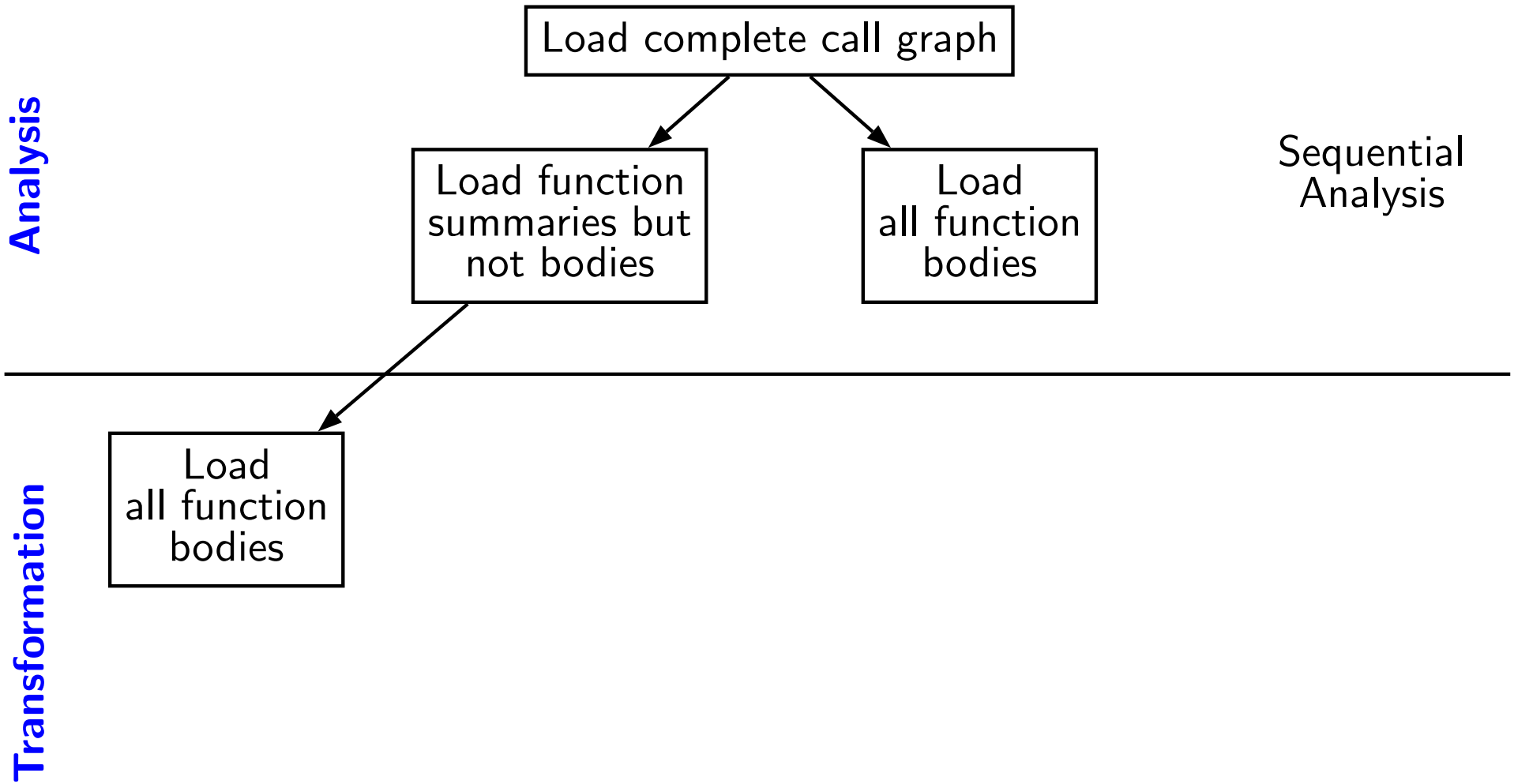
Analysis



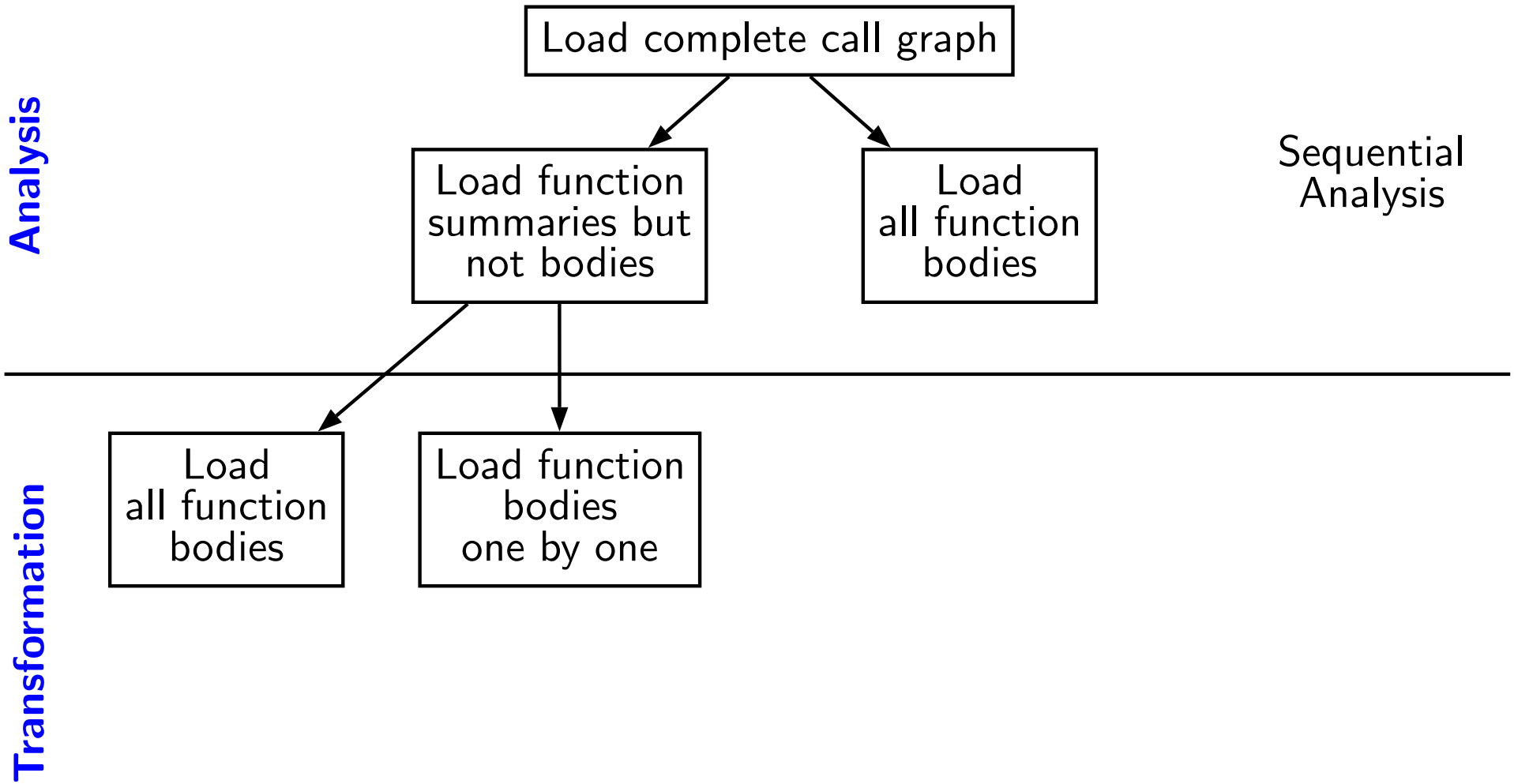
Sequential Analysis

Transformation

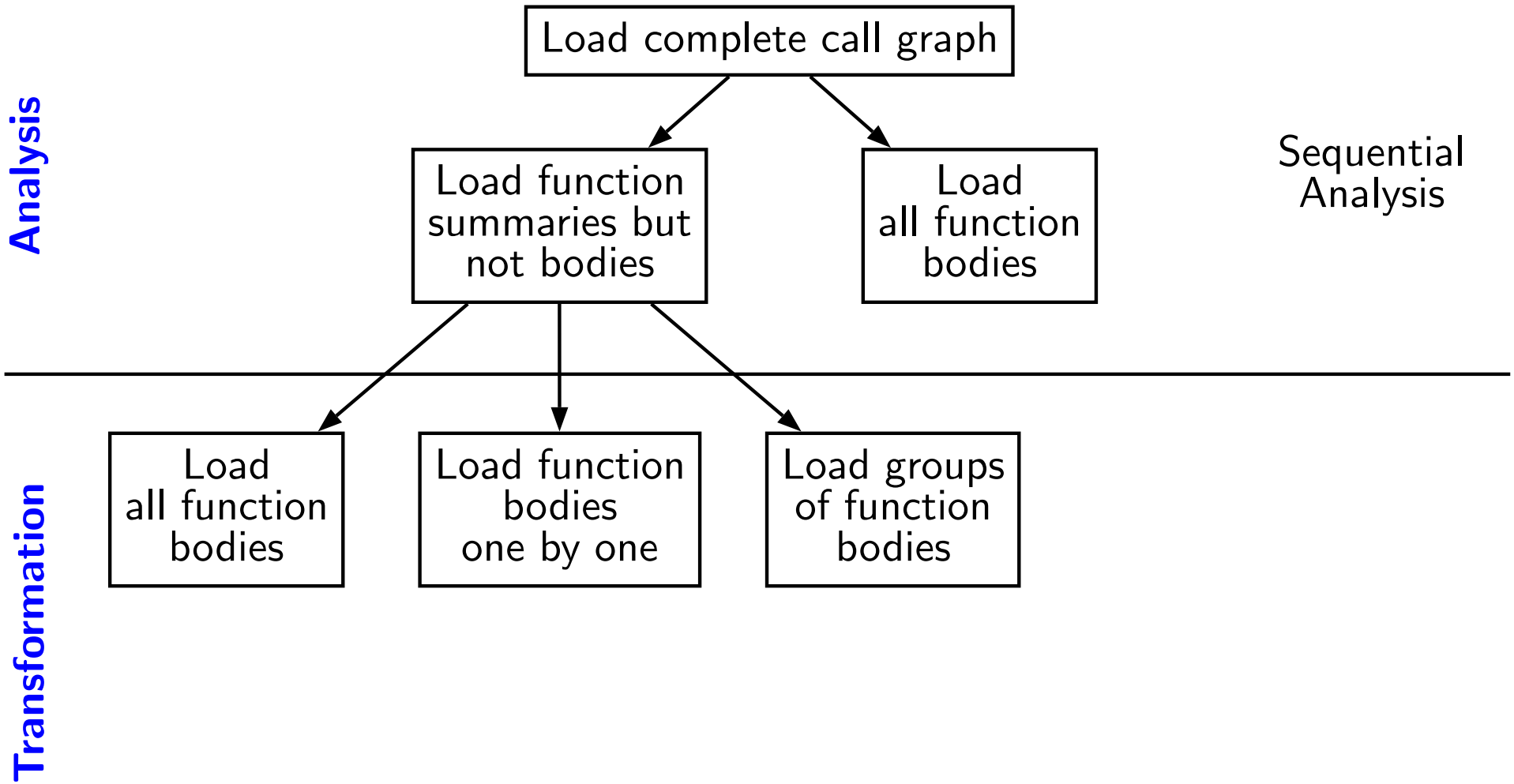
Partitioned and Non-Partitioned LTO



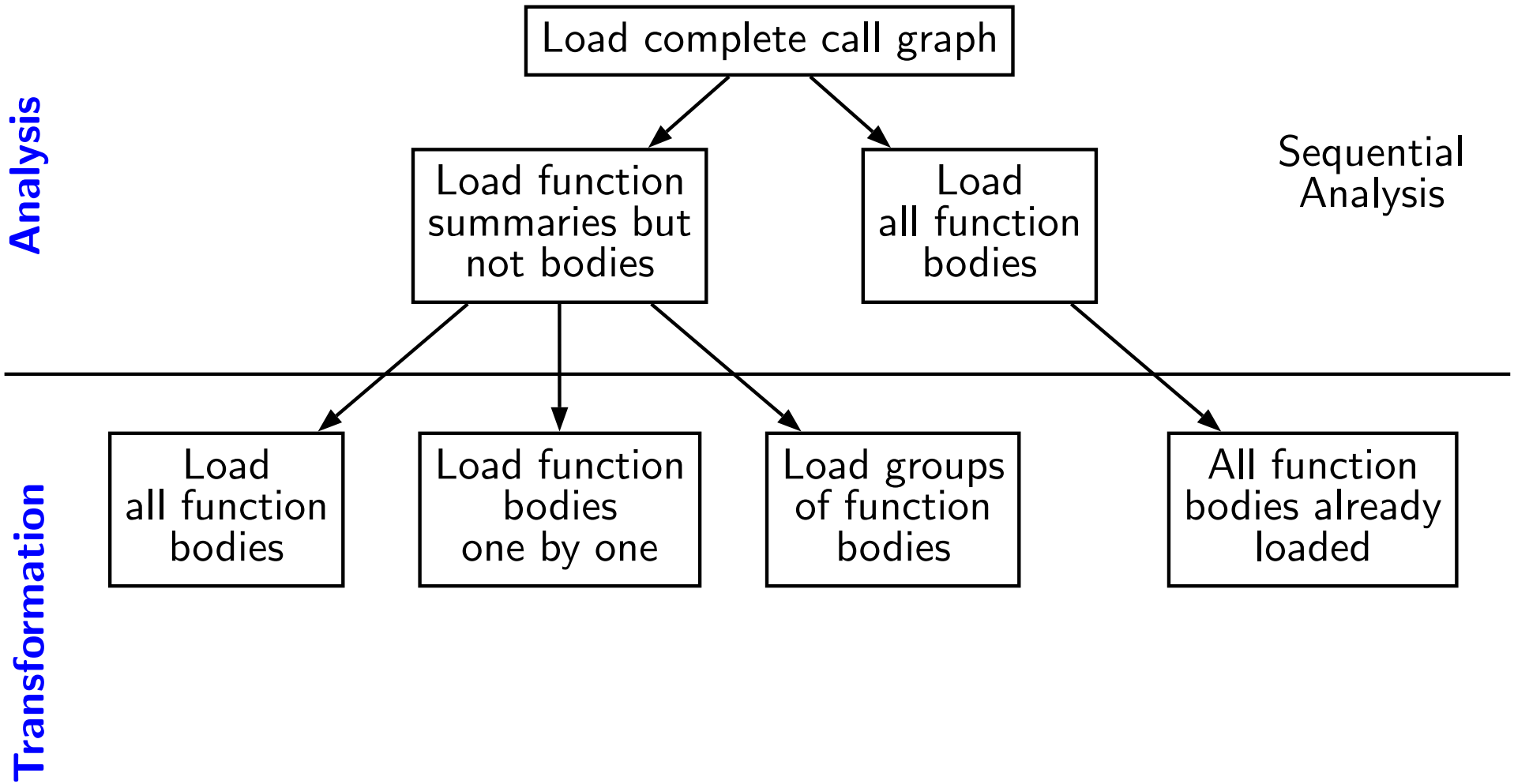
Partitioned and Non-Partitioned LTO



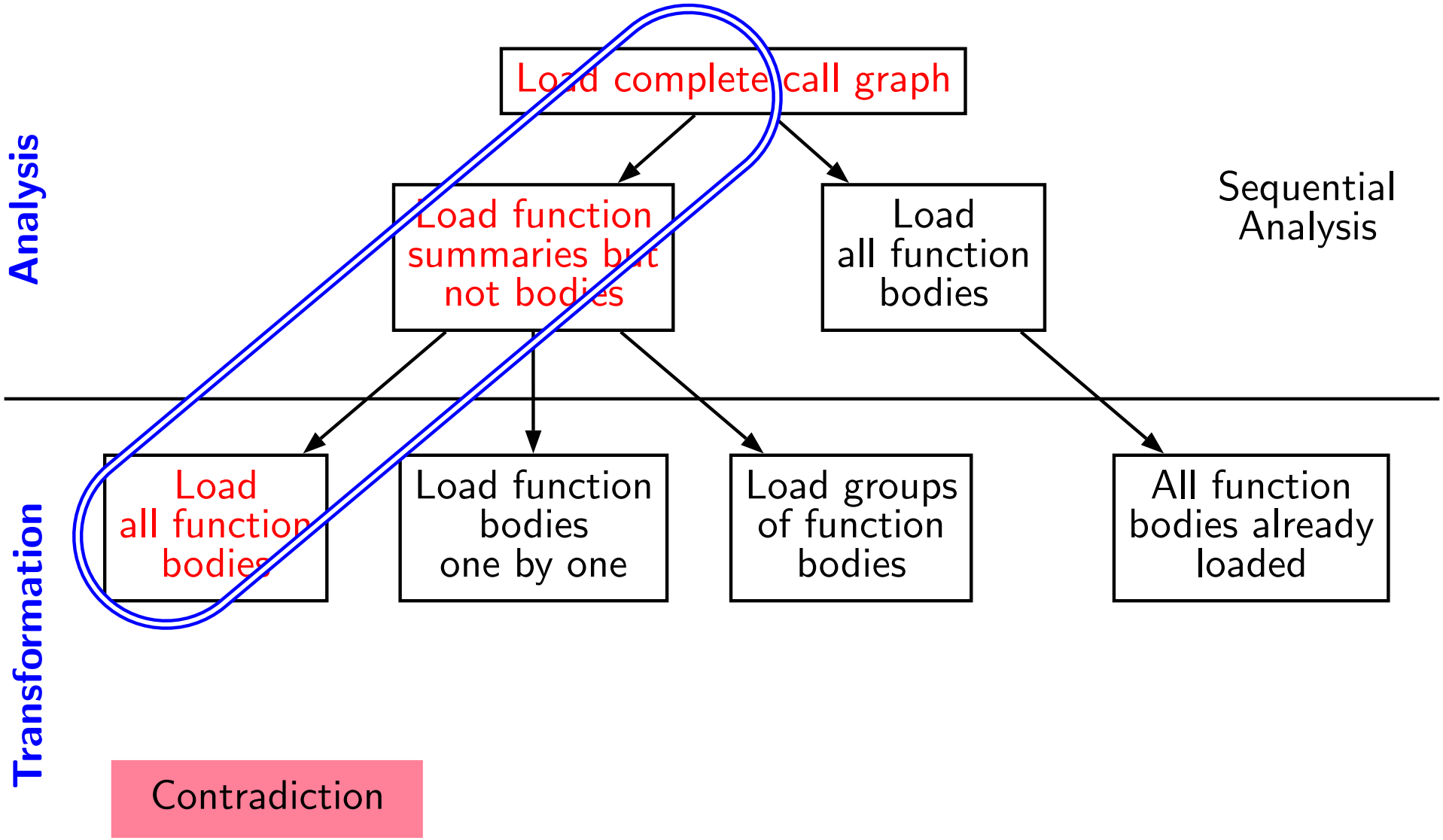
Partitioned and Non-Partitioned LTO



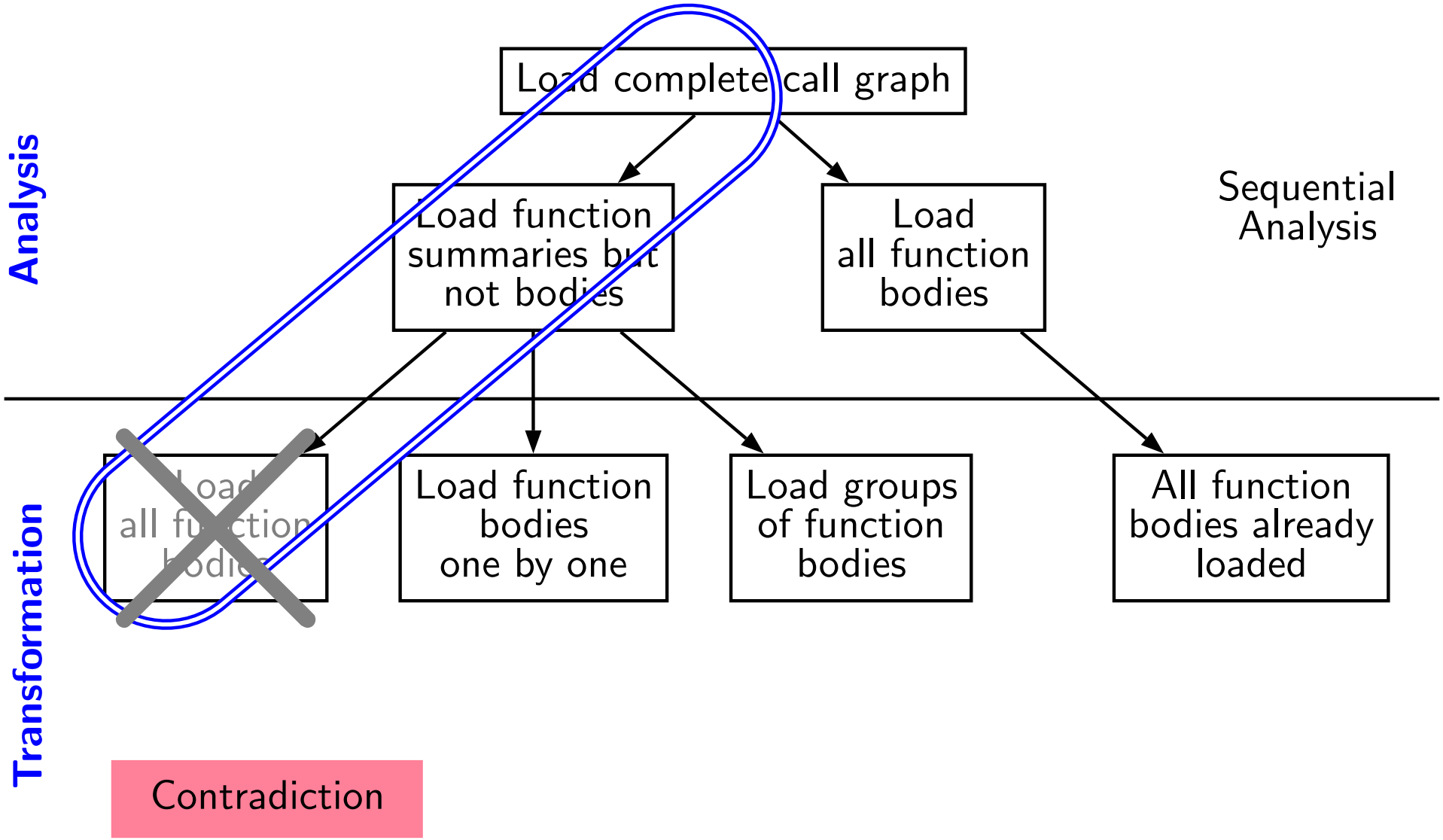
Partitioned and Non-Partitioned LTO



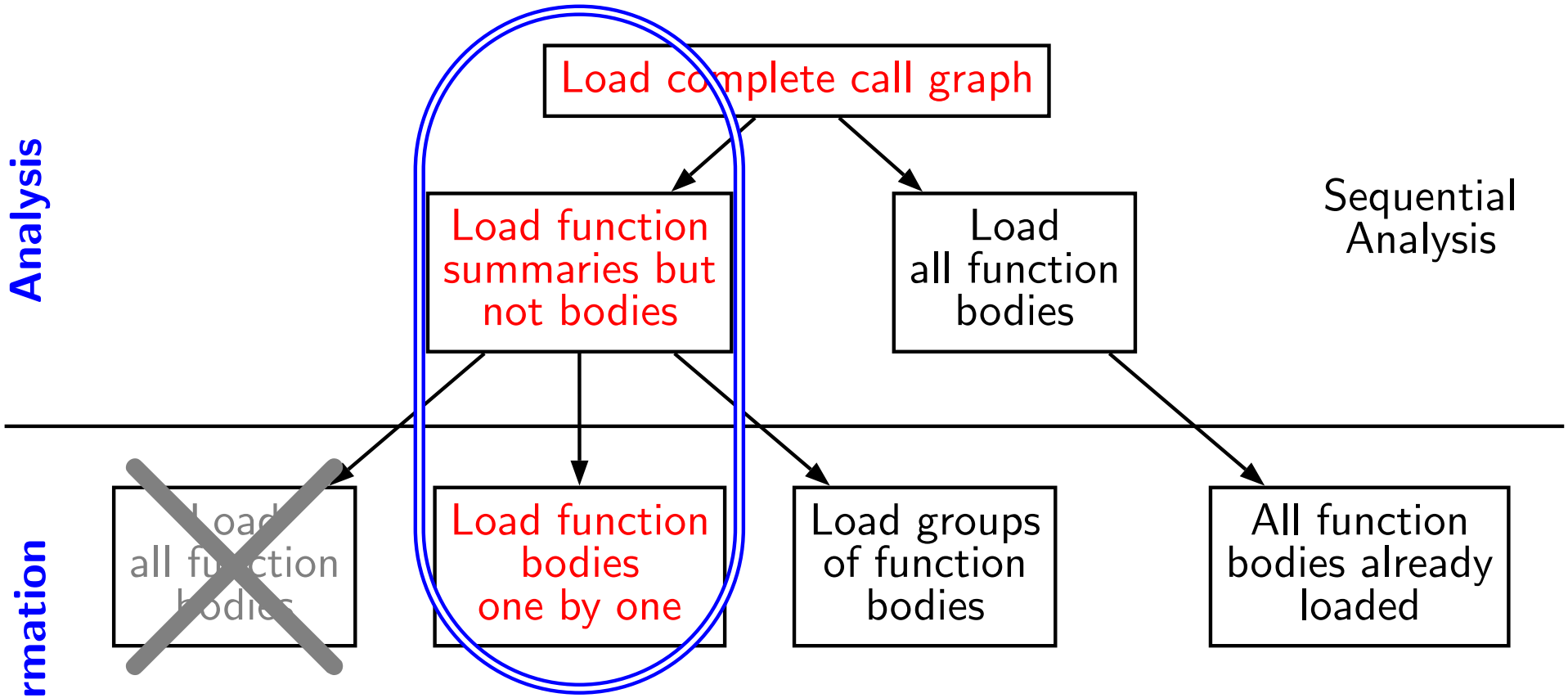
Partitioned and Non-Partitioned LTO



Partitioned and Non-Partitioned LTO

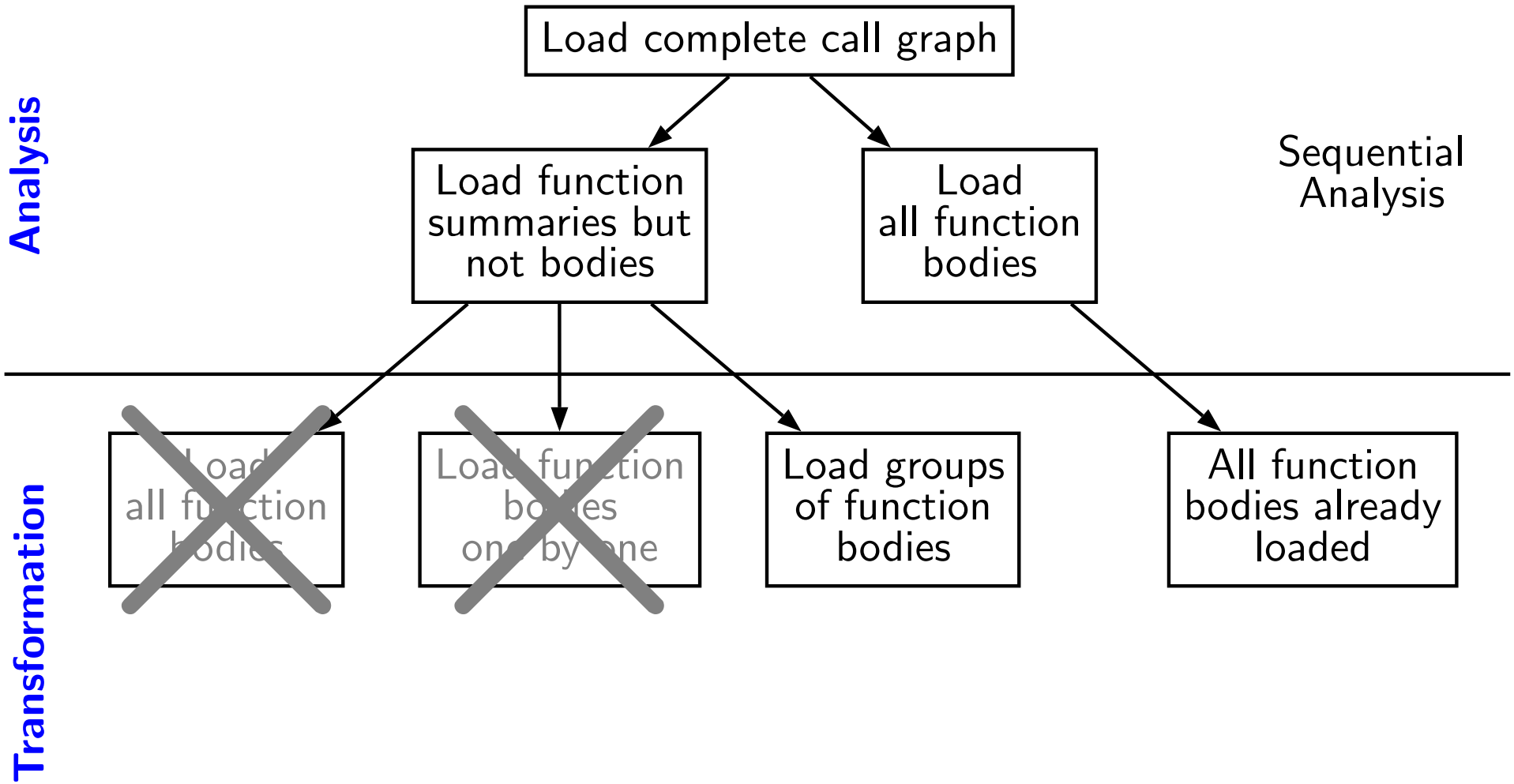


Partitioned and Non-Partitioned LTO

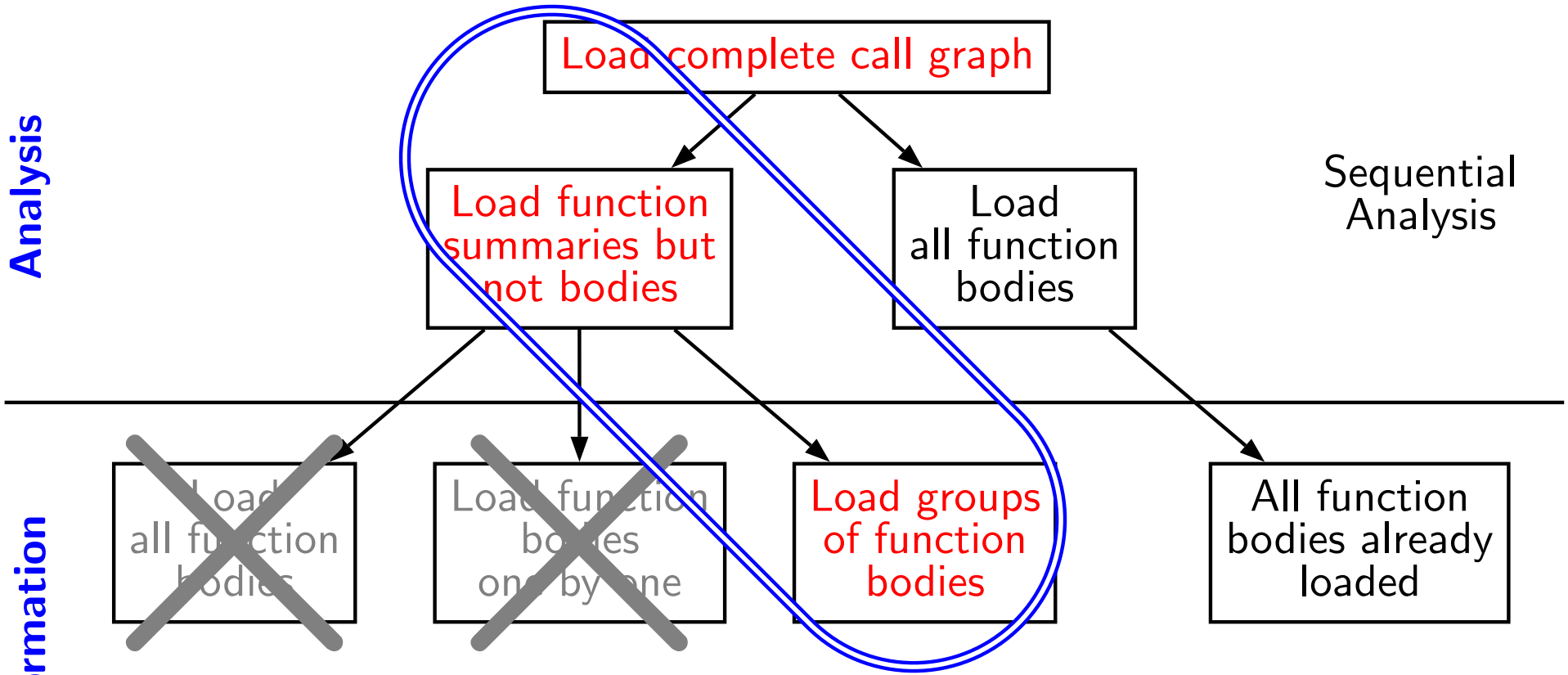


IPA not possible (only one function body at a time)
Strictly sequential transformations

Partitioned and Non-Partitioned LTO

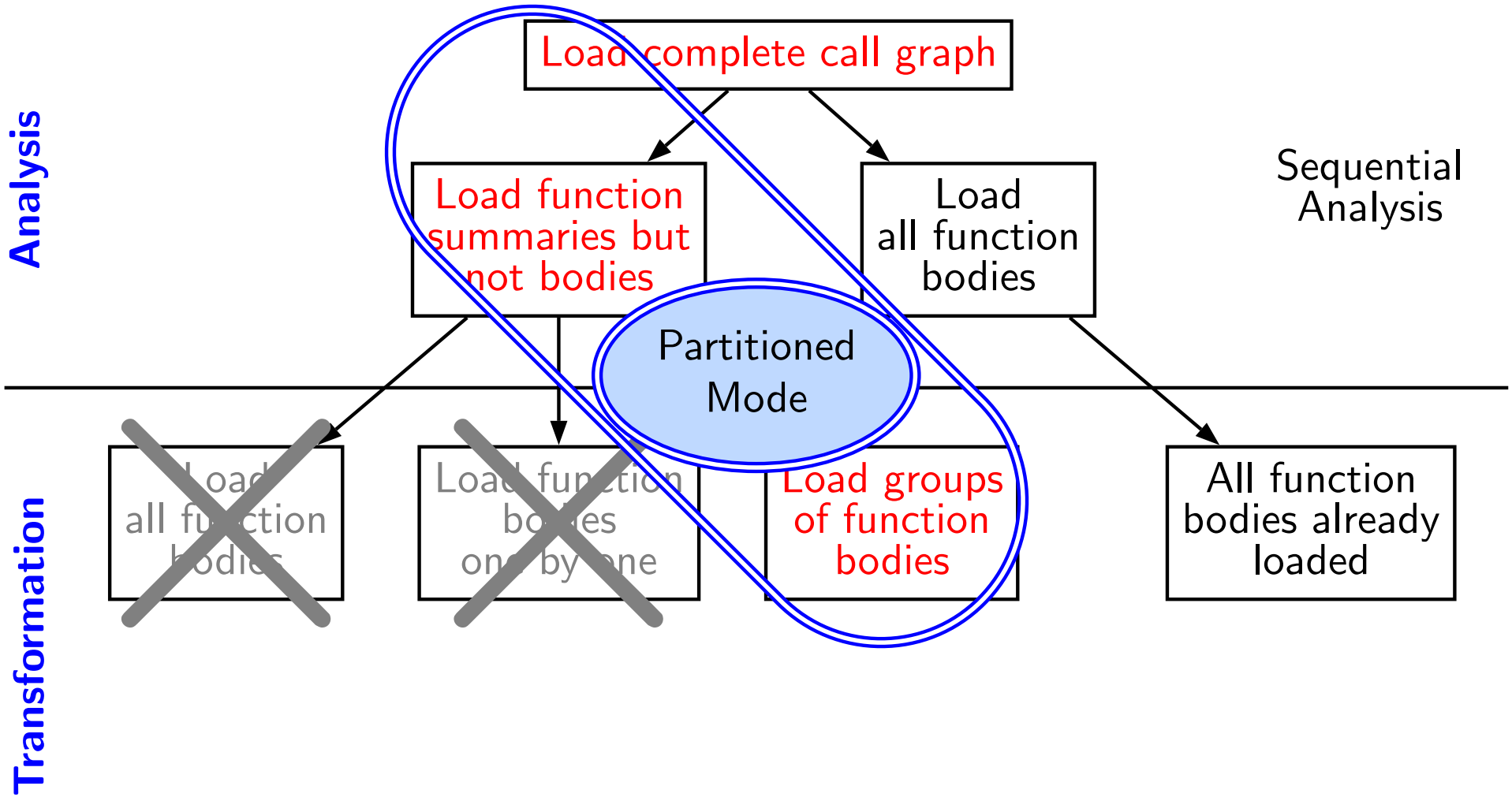


Partitioned and Non-Partitioned LTO

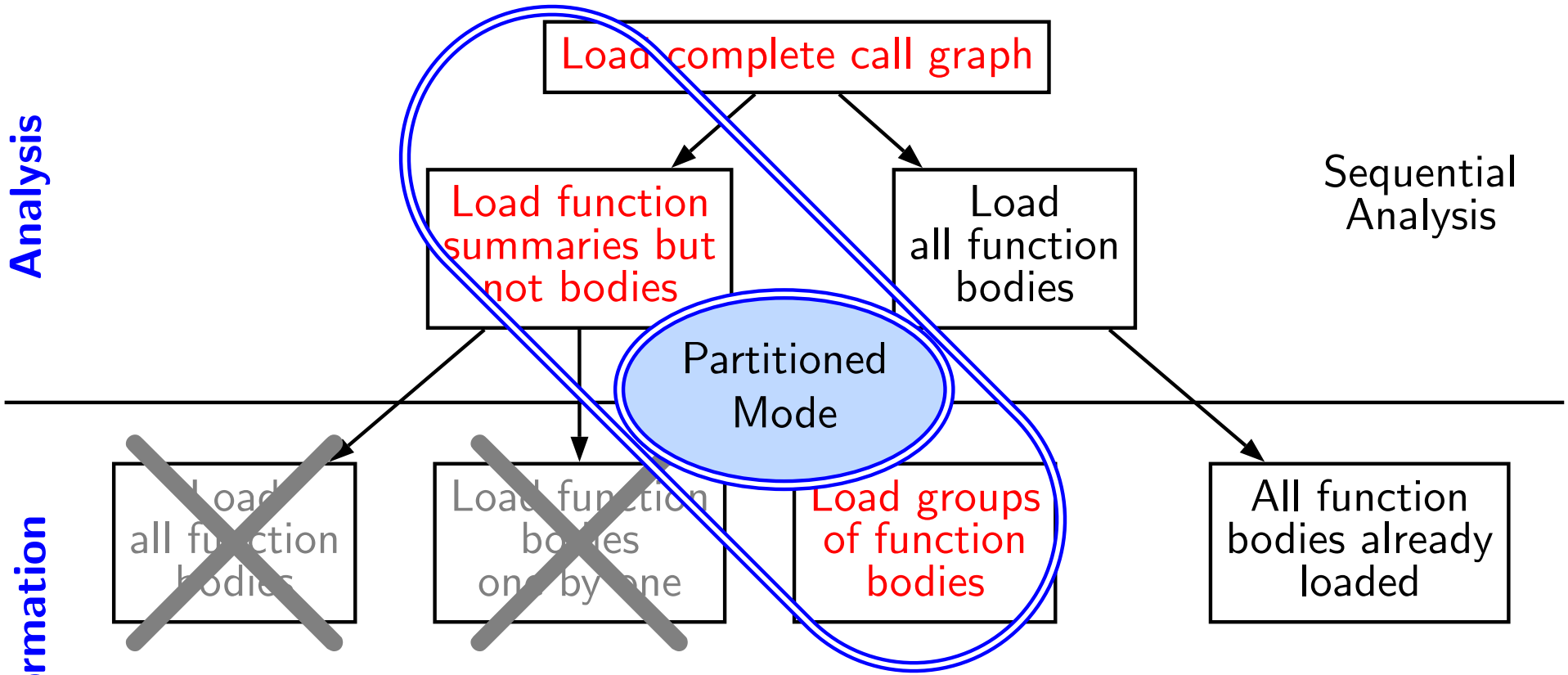


No need to load the entire program in memory
IPA possible (multiple function bodies)
Parallel transformations possible
Analysis and transformations in independent processes

Partitioned and Non-Partitioned LTO

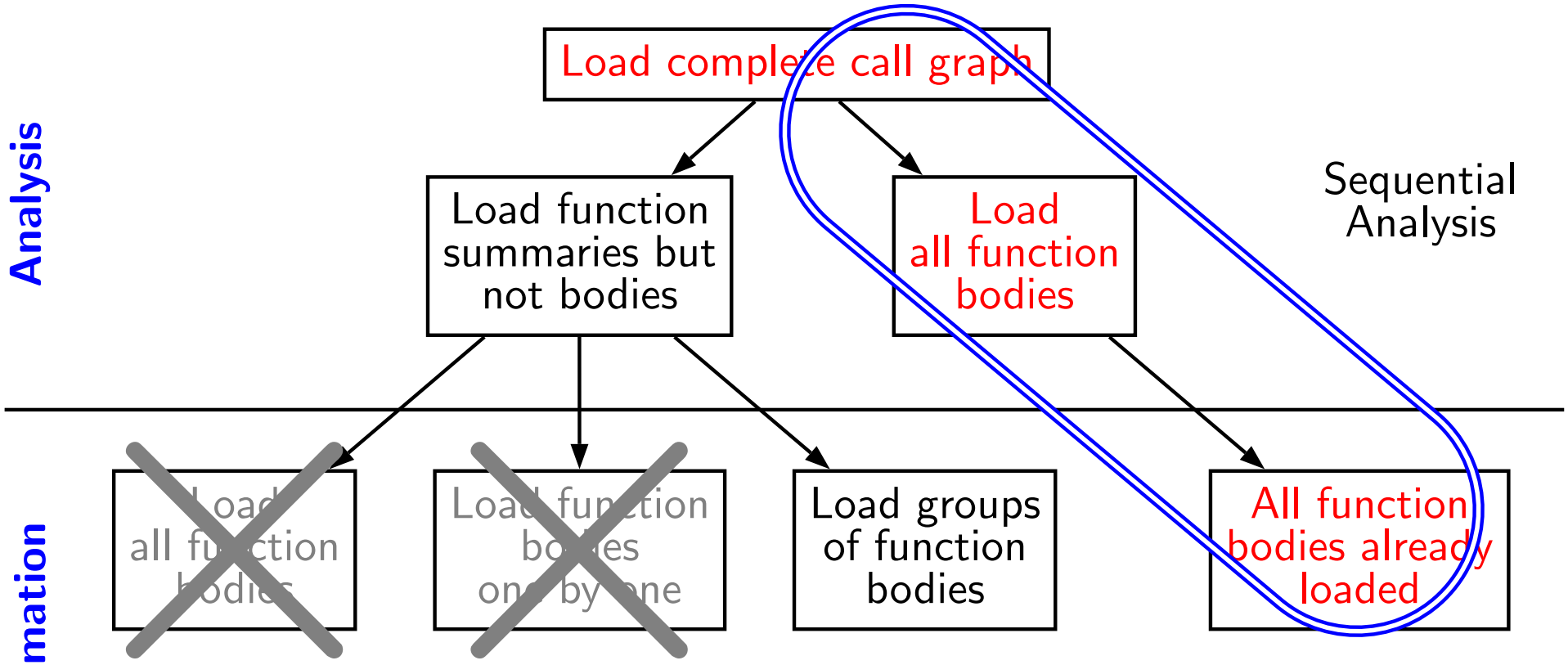


Partitioned and Non-Partitioned LTO



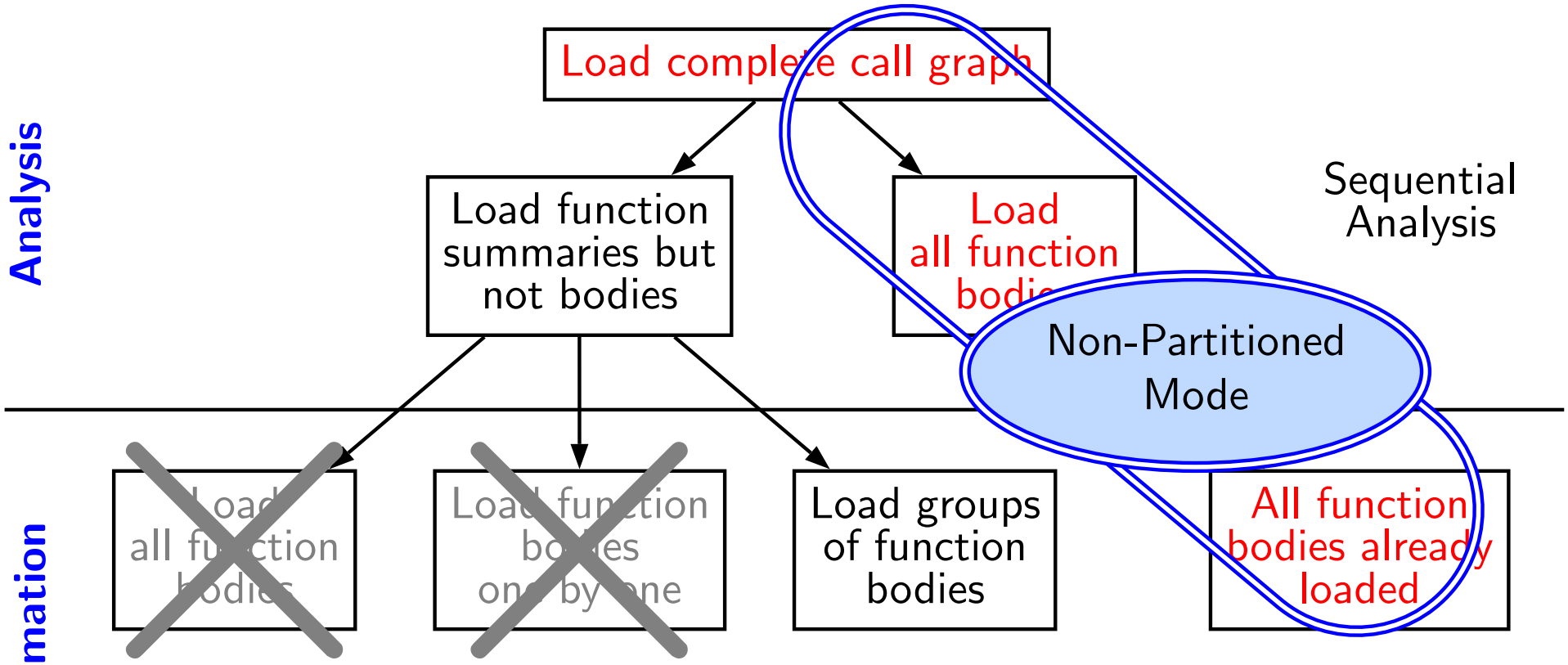
Balanced partitions `-flto -flto-partitions=balanced`
One Partition per file `-flto -flto-partitions=1to1`
Partitions by number `-flto --params lto-partitions=n`
Partitions by size `-flto --params lto-min-partition=s`

Partitioned and Non-Partitioned LTO



Entire program needs to be loaded in memory
No partitions `-flto -flto-partitions=none`
Strictly sequential transformations
Analysis and transformations in the same processes

Partitioned and Non-Partitioned LTO



Entire program needs to be loaded in memory
No partitions `-flto -flto-partitions=none`
Strictly sequential transformations
Analysis and transformations in the same processes

Partitioned LTO (aka WHOPR Mode of LTO)

- Three steps
  - LGEN:
    - generation of summary information
    - generation of translation unit information
  - WPA: Whole Program Analysis
    - Reads the call graph and not function bodies
    - Summary information for each function
  - LTRANS: Local Transformations

Partitioned LTO (aka WHOPR Mode of LTO)

- Three steps
  - LGEN: Potentially Parallel
    - generation of summary information
    - generation of translation unit information
  - WPA: Whole Program Analysis
    - Reads the call graph and not function bodies
    - Summary information for each function
  - LTRANS: Local Transformations

Partitioned LTO (aka WHOPR Mode of LTO)

- Three steps
  - LGEN: Potentially Parallel
    - generation of summary information
    - generation of translation unit information
  - WPA: Whole Program Analysis Sequential
    - Reads the call graph and not function bodies
    - Summary information for each function
  - LTRANS: Local Transformations

Partitioned LTO (aka WHOPR Mode of LTO)

- Three steps
  - LGEN: Potentially Parallel
    - generation of summary information
    - generation of translation unit information
  - WPA: Whole Program Analysis Sequential
    - Reads the call graph and not function bodies
    - Summary information for each function
  - LTRANS: Local Transformations Potentially Parallel

Partitioned LTO (aka WHOPR Mode of LTO)

- Three steps
  - LGEN: Potentially Parallel
    - generation of summary information
    - generation of translation unit information
  - WPA: Whole Program Analysis Sequential
    - Reads the call graph and not function bodies
    - Summary information for each function
  - LTRANS: Local Transformations Potentially Parallel
- Processing sequence

Partitioned LTO (aka WHOPR Mode of LTO)

- Three steps
  - LGEN: Potentially Parallel
    - generation of summary information
    - generation of translation unit information
  - WPA: Whole Program Analysis Sequential
    - Reads the call graph and not function bodies
    - Summary information for each function
  - LTRANS: Local Transformations Potentially Parallel
- Processing sequence
  - `gcc` executes LGEN
  - Subsequent process of `lto1` executes WPA
  - Subsequent independent processes of `lto1` execute LTRANS

Non-Partitioned LTO

- Two steps
  - LGEN:
    - generation of translation unit information
    - no summary
  - IPA: Inter-Procedural Analysis
    - Reads the call graph and function bodies
    - Performs analysis and transformation

  IPA is a whole program analysis
  (processes the entire program together)

Non-Partitioned LTO

- Two steps
  - LGEN: Potentially Parallel
    - generation of translation unit information
    - no summary
  - IPA: Inter-Procedural Analysis
    - Reads the call graph and function bodies
    - Performs analysis and transformation

  IPA is a whole program analysis
  (processes the entire program together)

Non-Partitioned LTO

- Two steps
  - LGEN: Potentially Parallel
    - generation of translation unit information
    - no summary
  - IPA: Inter-Procedural Analysis Sequential
    - Reads the call graph and function bodies
    - Performs analysis and transformation

  IPA is a whole program analysis
  (processes the entire program together)

Non-Partitioned LTO

- Two steps
  - LGEN: Potentially Parallel
    - generation of translation unit information
    - no summary
  - IPA: Inter-Procedural Analysis Sequential
    - Reads the call graph and function bodies
    - Performs analysis and transformation

    IPA is a whole program analysis
    (processes the entire program together)

- Processing sequence

Non-Partitioned LTO

- Two steps
  - LGEN: Potentially Parallel
    - generation of translation unit information
    - no summary
  - IPA: Inter-Procedural Analysis Sequential
    - Reads the call graph and function bodies
    - Performs analysis and transformation

    IPA is a whole program analysis
    (processes the entire program together)

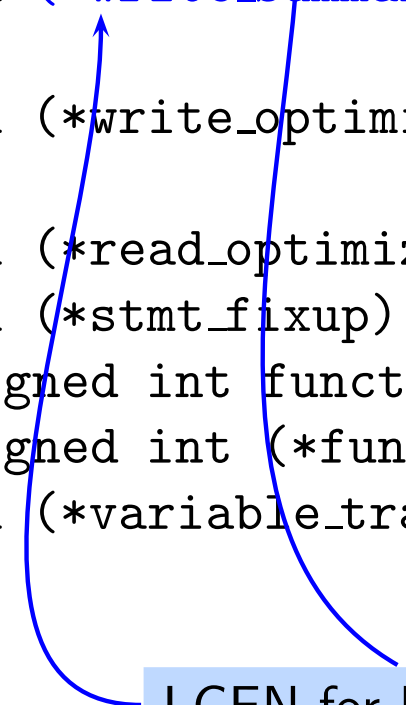
- Processing sequence
  - `gcc` executes LGEN
  - Subsequent process of `lto1` executes IPA

LTO Pass Hooks

```
struct ipa_opt_pass_d
{
  struct opt_pass pass;
  void (*generate_summary) (void);
  void (*read_summary) (void);
  void (*write_summary) (struct cgraph_node_set_def *,
                         struct varpool_node_set_def *);
  void (*write_optimization_summary)(struct cgraph_node_set_def *,
                                     struct varpool_node_set_def *);
  void (*read_optimization_summary) (void);
  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 *);
};
```

LTO Pass Hooks

```
struct ipa_opt_pass_d
{
  struct opt_pass pass;
  void (*generate_summary) (void);
  void (*read_summary) (void);
  void (*write_summary) (struct cgraph_node_set_def *,
                         struct varpool_node_set_def *);
  void (*write_optimization_summary)(struct cgraph_node_set_def *,
                                     struct varpool_node_set_def *);
  void (*read_optimization_summary) (void);
  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 *);
};
```



LGEN for Partitioned LTO

LTO Pass Hooks

```
struct ipa_opt_pass_d
{
  struct opt_pass pass;
  void (*generate_summary) (void);
  void (*read_summary) (void);
  void (*write_summary) (struct cgraph_node_set_def *,
                         struct varpool_node_set_def *);
  void (*write_optimization_summary)(struct cgraph_node_set_def *,
                                     struct varpool_node_set_def *);
  void (*read_optimization_summary) (void);
  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 *);
};
```

LGEN for Non-Partitioned LTO

LTO Pass Hooks

```
struct ipa_opt_pass_d
{
  struct opt_pass pass; (member  void (*execute) (void);)
  void (*generate_summary) (void);
  void (*read_summary) (void);
  void (*write_summary) (struct cgraph_node_set_def *,
                         struct varpool_node_set_def *);
  void (*write_optimization_summary)(struct cgraph_node_set_def *,
                                     struct varpool_node_set_def *);
  void (*read_optimization_summary) (void);
  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 *);
};
```

WPA for Partitioned LTO

LTO Pass Hooks

```
struct ipa_opt_pass_d
{
  struct opt_pass pass; (member  void (*execute) (void);)
  void (*generate_summary) (void);
  void (*read_summary) (void);
  void (*write_summary) (struct cgraph_node_set_def *,
                         struct varpool_node_set_def *);
  void (*write_optimization_summary)(struct cgraph_node_set_def *,
                                     struct varpool_node_set_def *);
  void (*read_optimization_summary) (void);
  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 *);
};
```

IPA for Non-Partitioned LTO

LTO Pass Hooks

```
struct ipa_opt_pass_d
{
  struct opt_pass pass;
  void (*generate_summary) (void);
  void (*read_summary) (void);
  void (*write_summary) (struct cgraph_node_set_def *,
                         struct varpool_node_set_def *);
  void (*write_optimization_summary)(struct cgraph_node_set_def *,
                                     struct varpool_node_set_def *);
  void (*read_optimization_summary) (void);
  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 *);
};
```

LTRANS for Partitioned LTO

lto1 Control Flow

```
lto_main
   lto_init
      lto_process_name
      lto_reader_init
   read_cgraph_and_symbols
   if (flag_wpa)
      /* WPA for partitioned LTO */
      do_whole_program_analysis
         materialize_cgraph
         execute_ipa_pass_list (all_regular_ipa_passes)
         lto_wpa_write_files
   else
      /* IPA for non-partitioned LTO */
      /* Only LTRANS for partitioned LTO */
      materialize_cgraph
      cgraph_optimize
```

cc1 Control Flow: A Recap

```
toplev_main   /* In file toplev.c */
  compile_file
    lang_hooks.parse_file=>c_common_parse_file
    lang_hooks.decls.final_write_globals=>c_write_global_declarations
      cgraph_finalize_compilation_unit
        cgraph_analyze_functions     /* Create GIMPLE */
          cgraph_analyze_function       /* Create GIMPLE */
              ...
          cgraph_optimize
            ipa_passes
              execute_ipa_pass_list(all_small_ipa_passes) /*!in lto*/
              execute_ipa_summary_passes(all_regular_ipa_passes)
              execute_ipa_summary_passes(all_lto_gen_passes)
              ipa_write_summaries
            execute_ipa_pass_list(all_late_ipa_passes)
            cgraph_expand_all_functions
              cgraph_expand_function
              /* Intra. GIMPLE, expansion, and RTL passes */
```


cc1 and Non-Partitioned lto1

```
toplev_main
...
  compile_file
  ...
    cgraph_analyze_function
```

cc1

```
cgraph_optimize
...
  ipa_passes
  ...
  cgraph_expand_all_functions
    ...
    tree_rest_of_compilation
```

cc1 and Non-Partitioned lto1

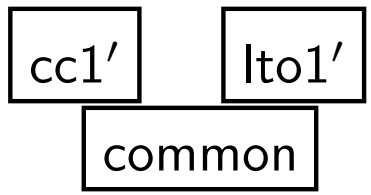
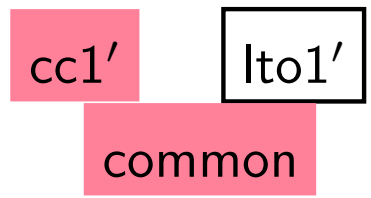
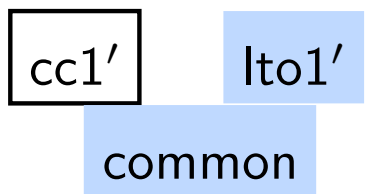
```
toplev_main
...
  compile_file
  ...
    cgraph_analyze_function
```

```
lto_main
...
  read_cgraph_and_symbols
  ...
    materialize_cgraph
```

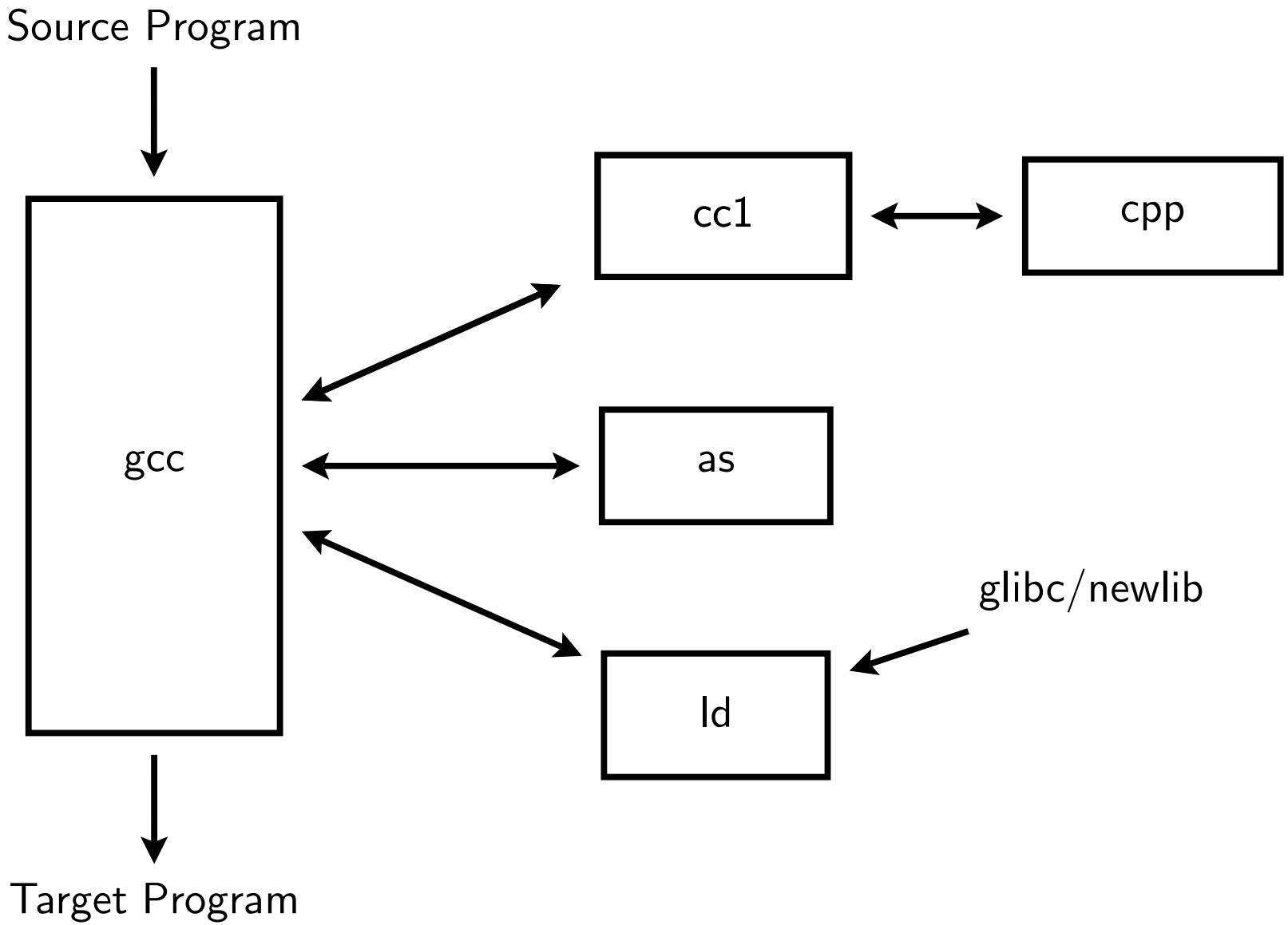
```
cgraph_optimize
...
  ipa_passes
  ...
  cgraph_expand_all_functions
    ...
    tree_rest_of_compilation
```

lto1

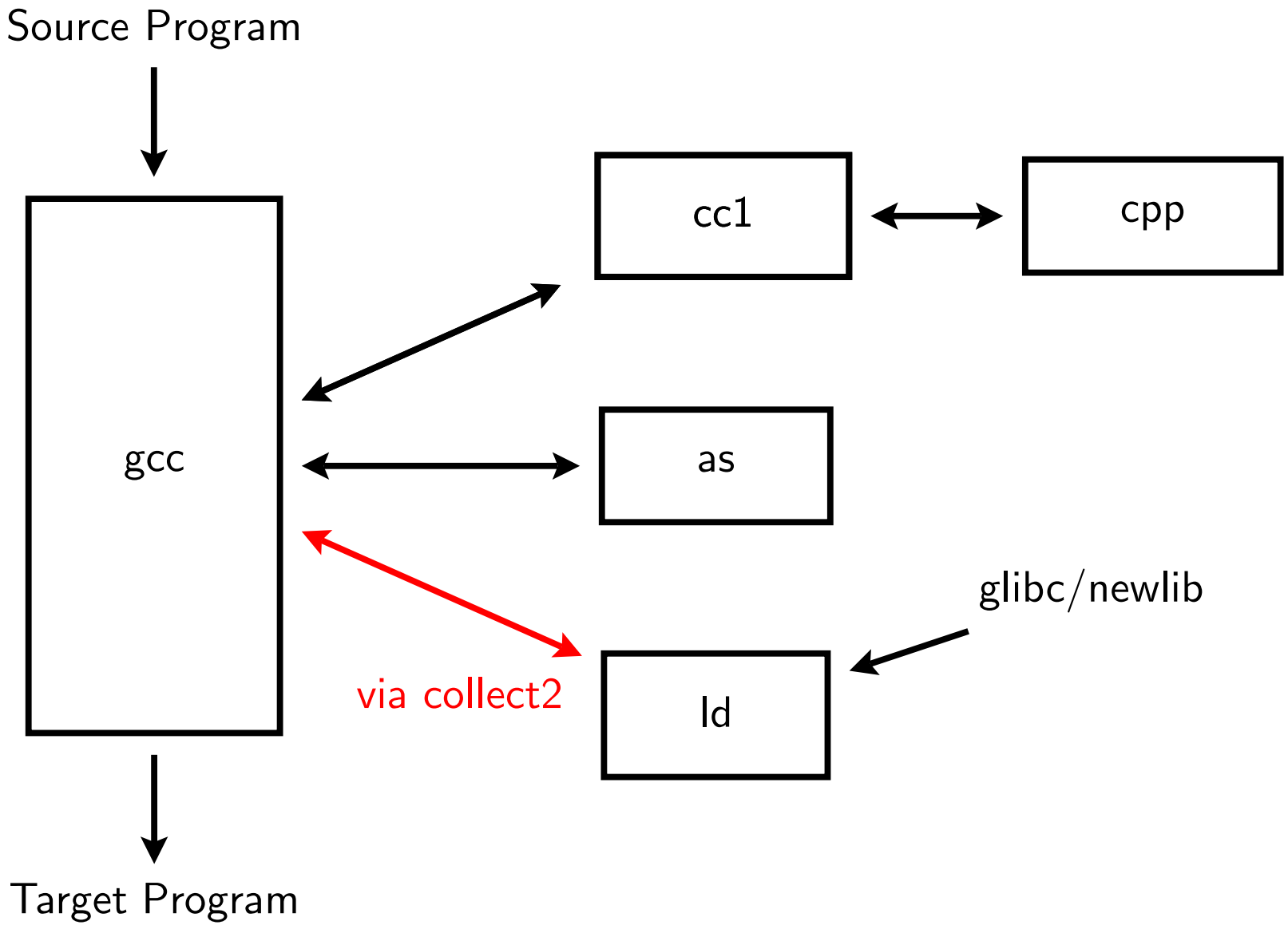
Our Pictorial Convention

Source code	
cc1 executable	
lto1 executable	

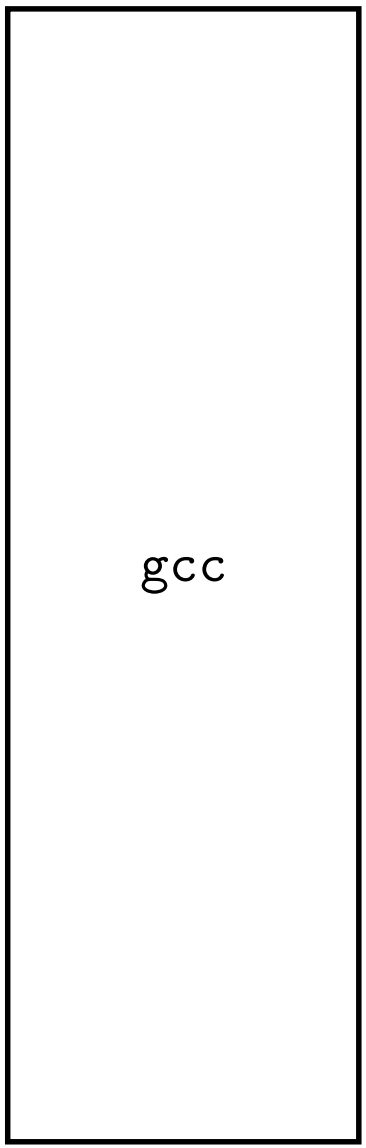
The GNU Tool Chain: Our First Picture



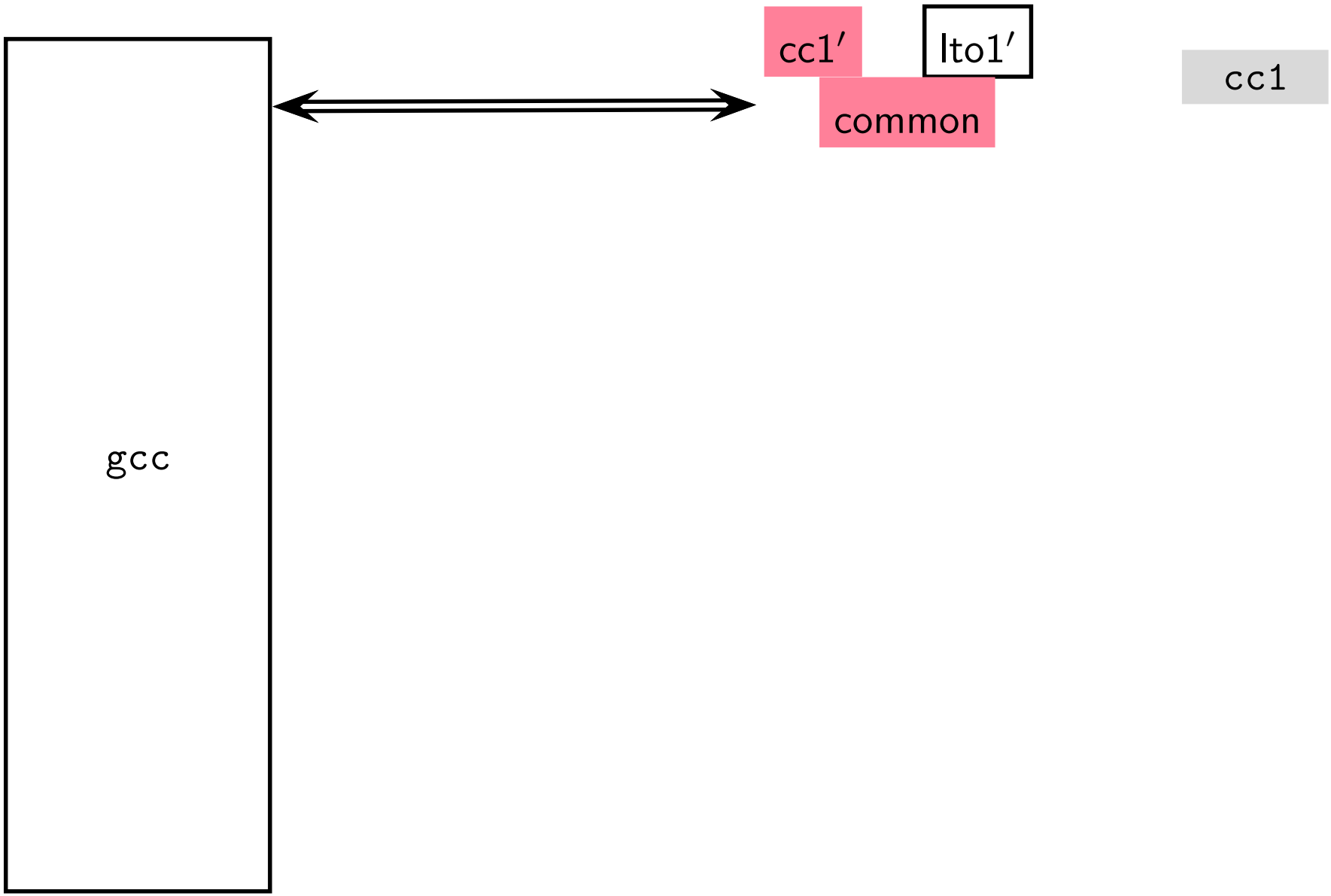
The GNU Tool Chain: Our First Picture



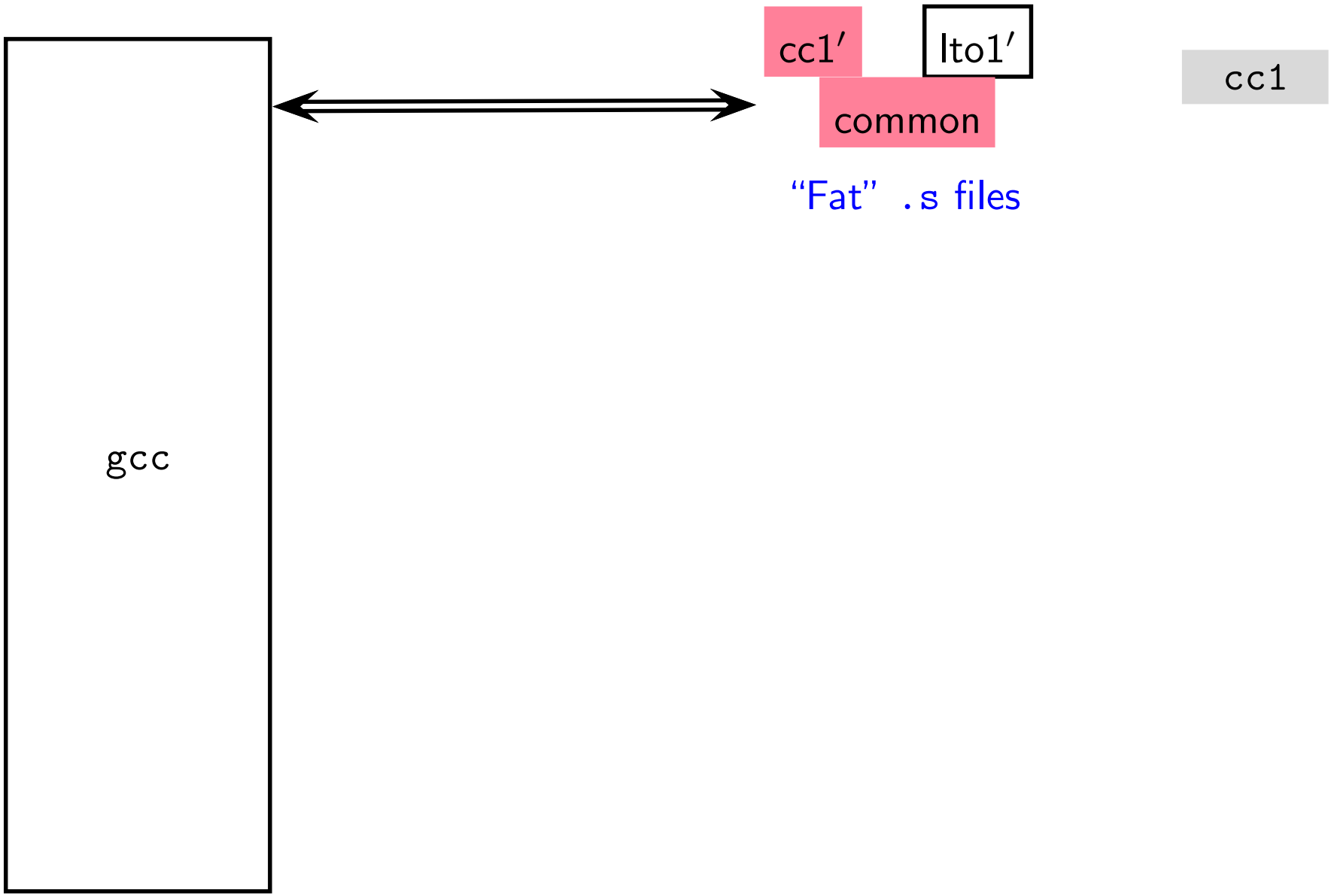
The GNU Tool Chain for Non-Partitioned LTO Support



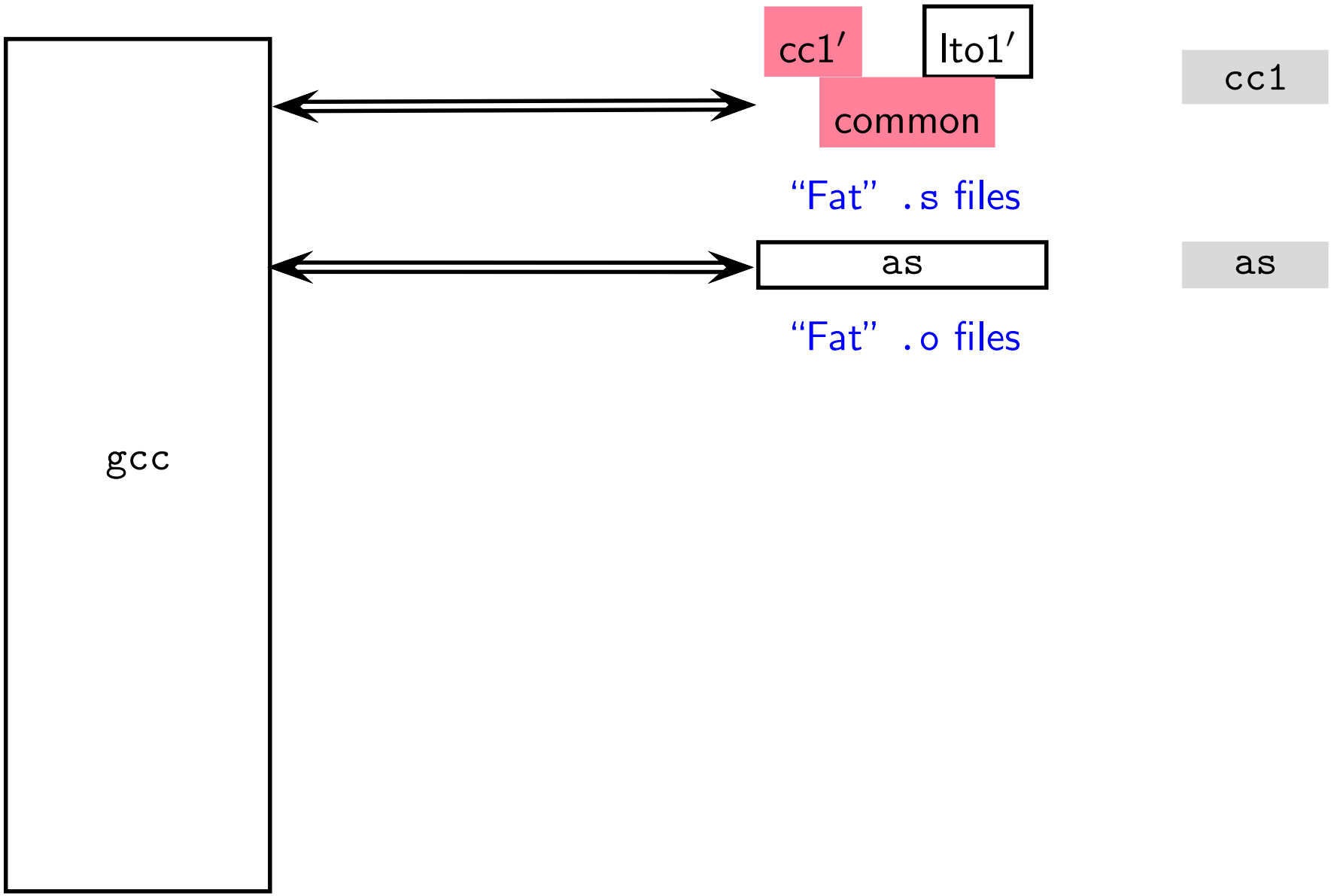
The GNU Tool Chain for Non-Partitioned LTO Support



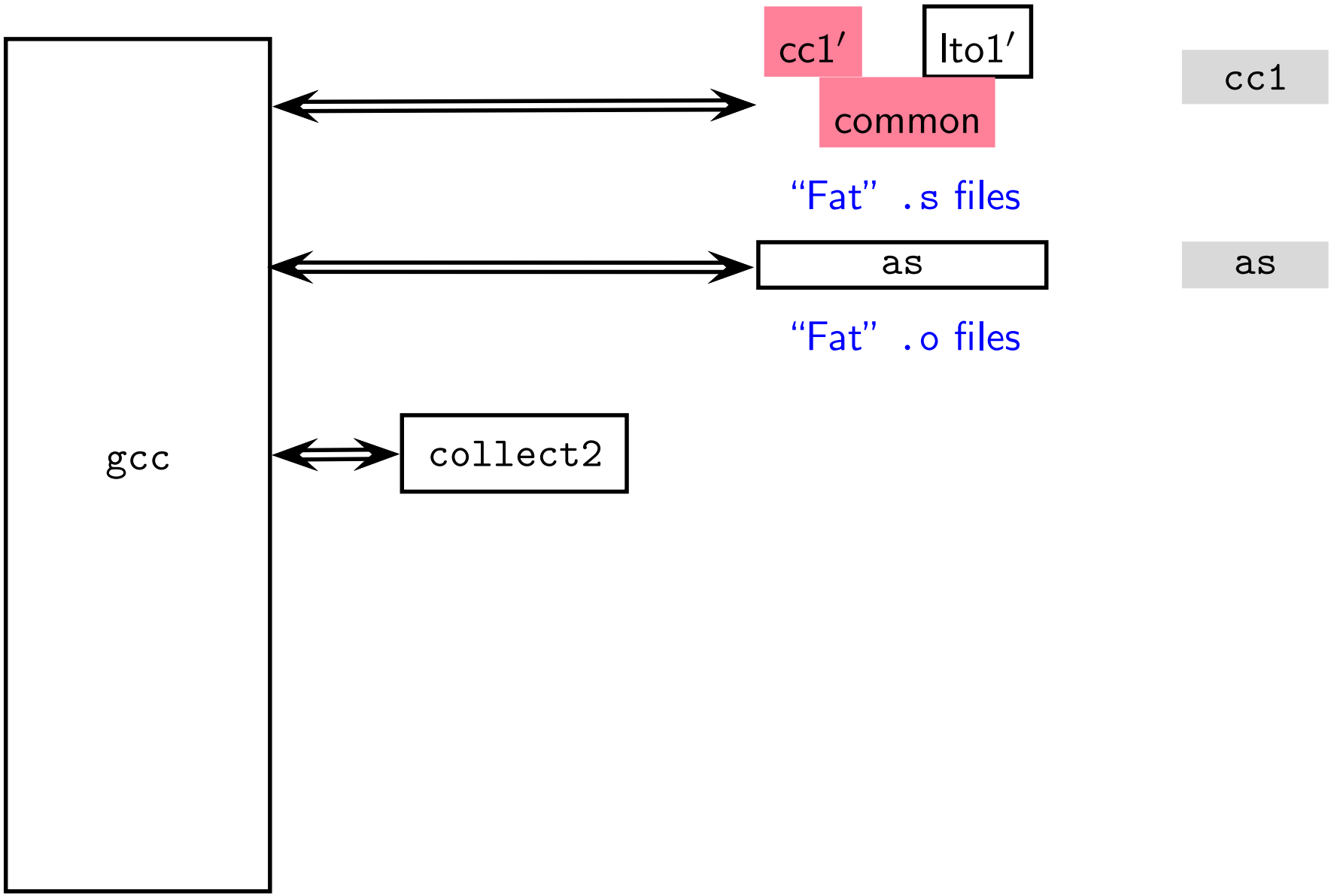
The GNU Tool Chain for Non-Partitioned LTO Support



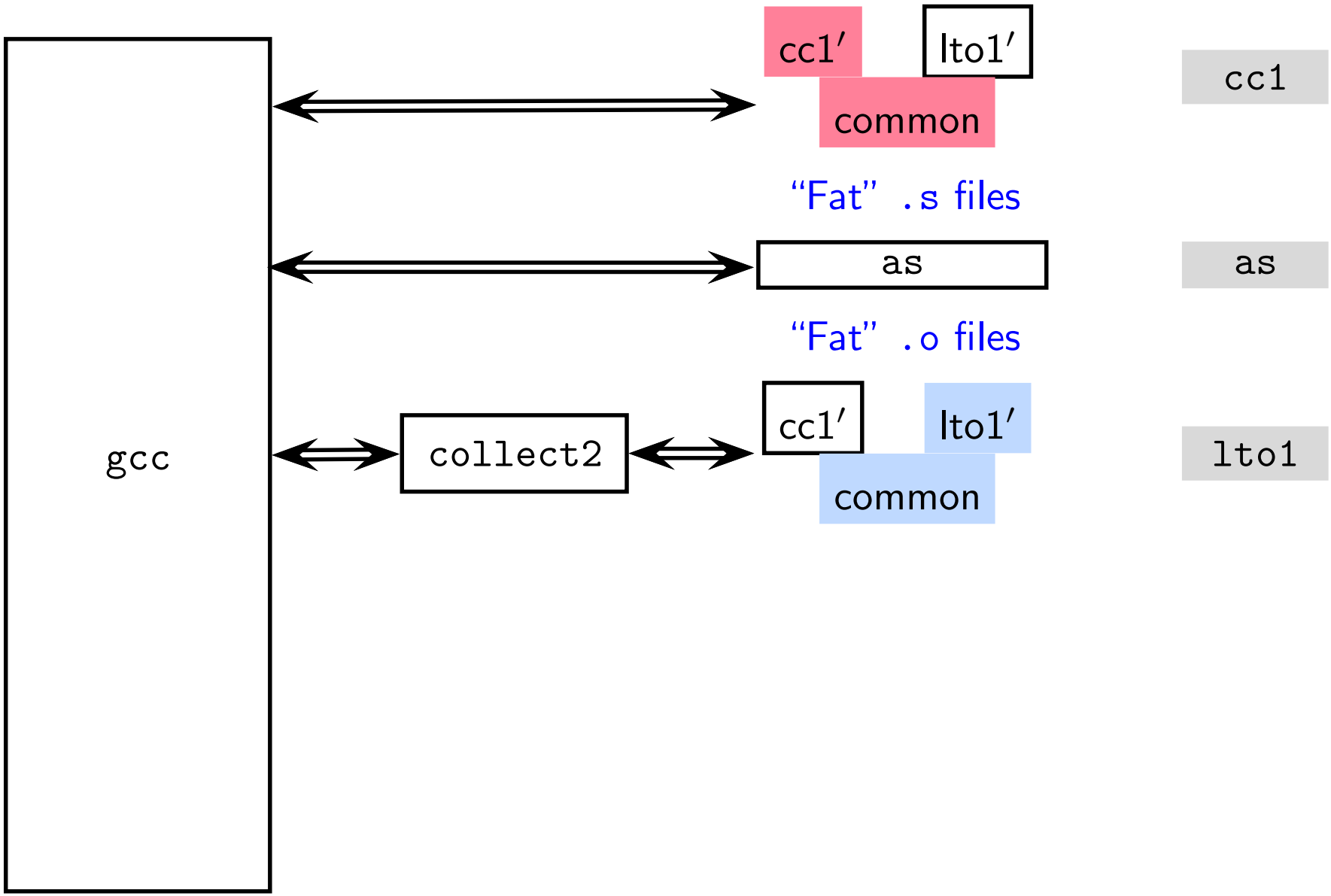
The GNU Tool Chain for Non-Partitioned LTO Support



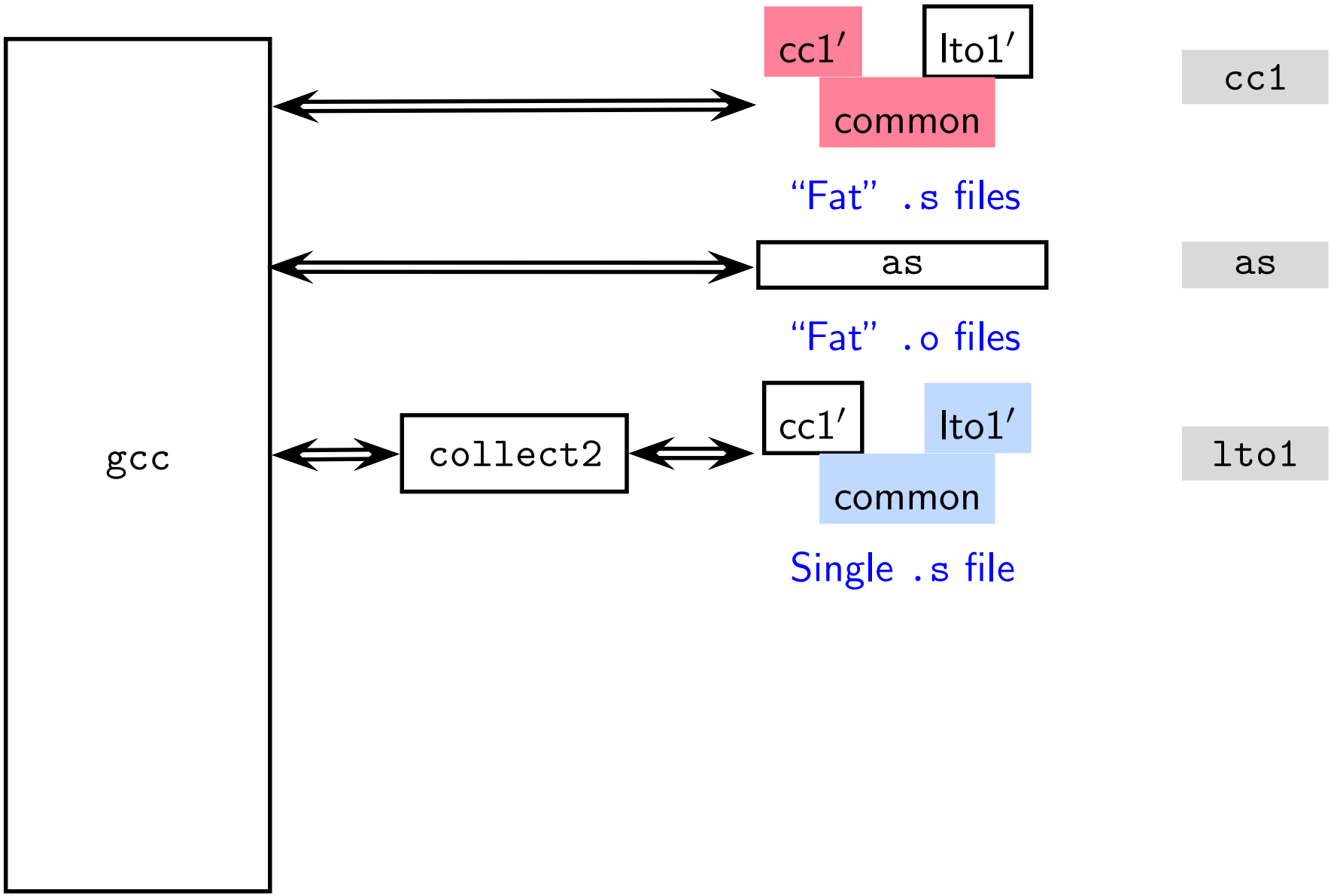
The GNU Tool Chain for Non-Partitioned LTO Support



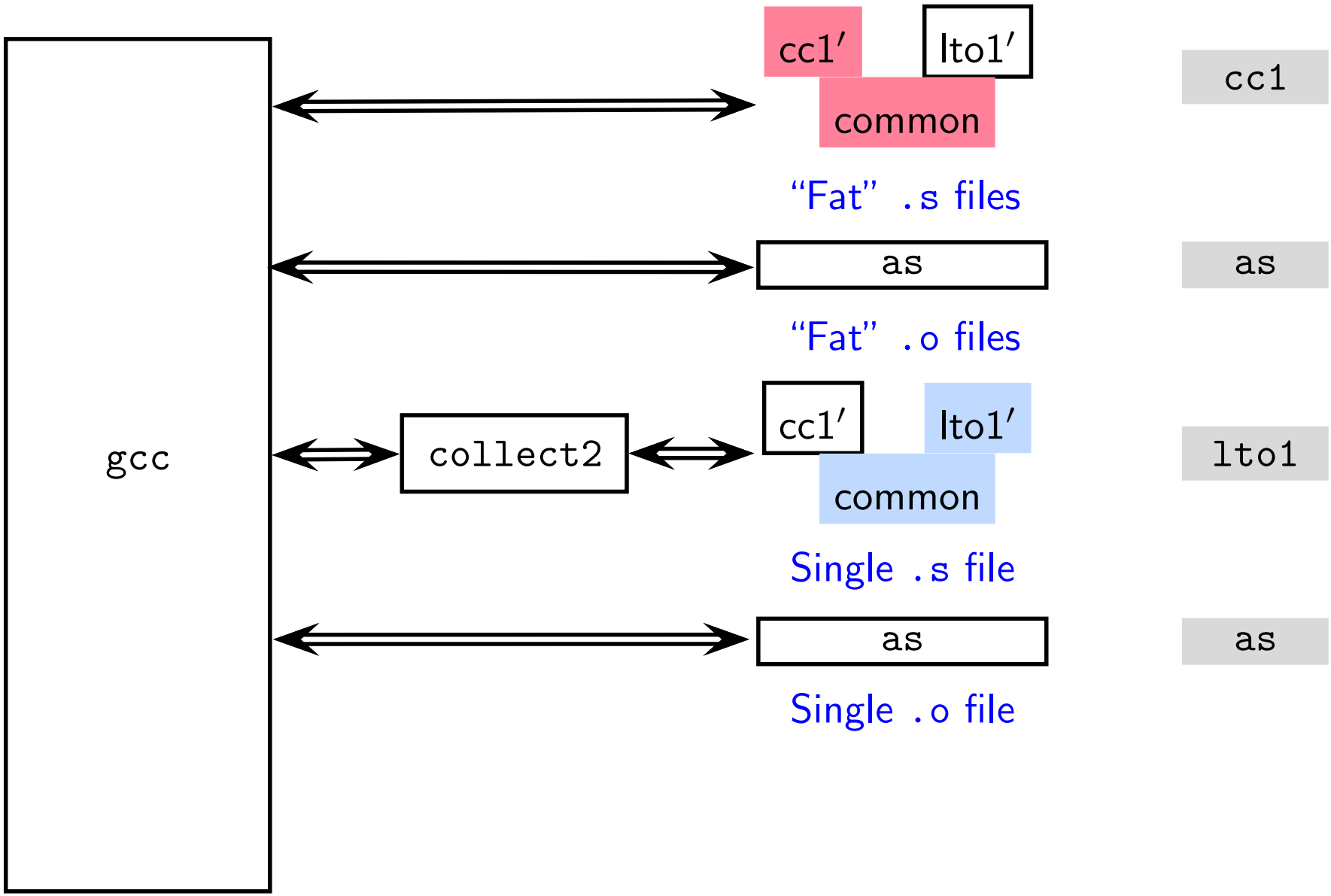
The GNU Tool Chain for Non-Partitioned LTO Support



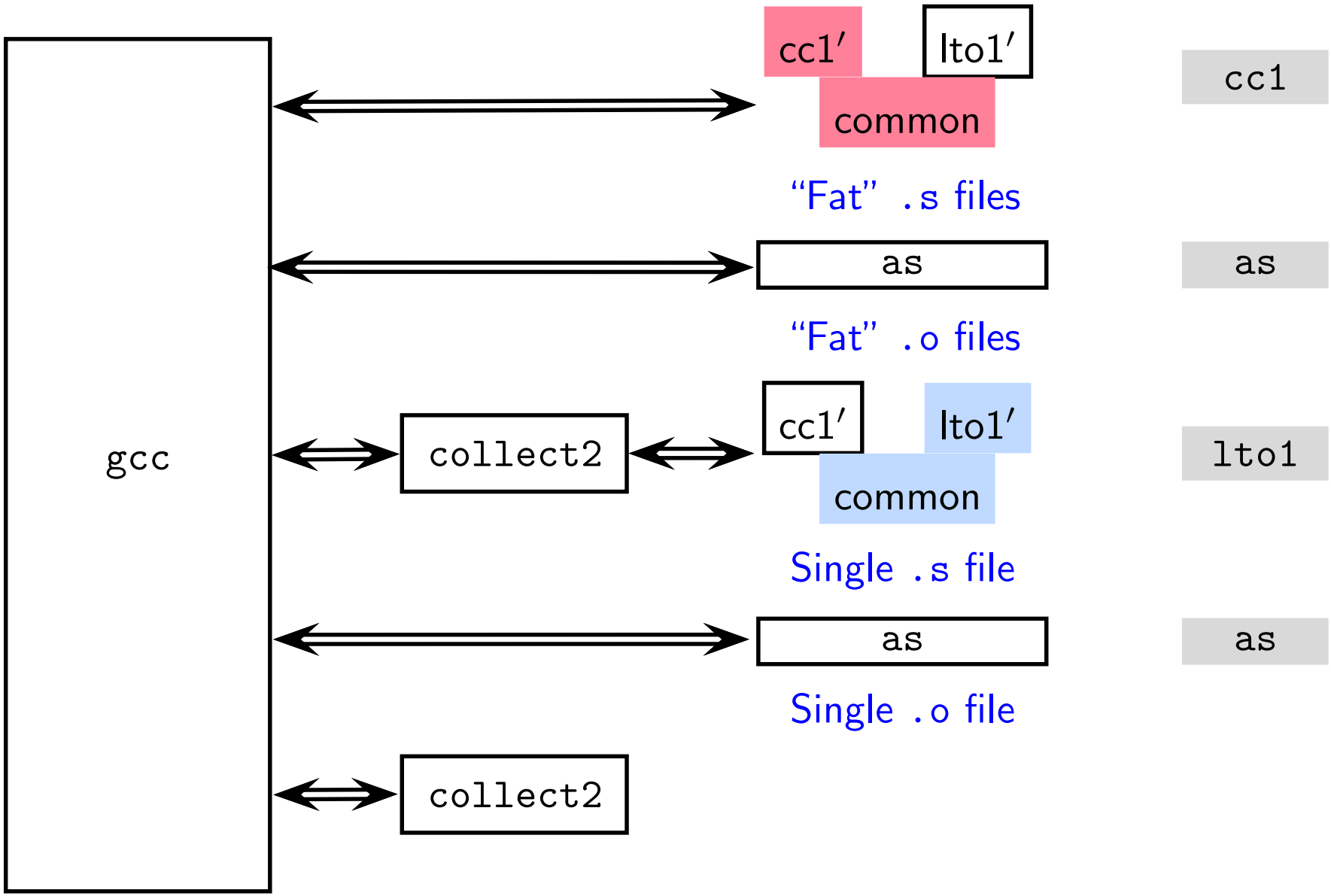
The GNU Tool Chain for Non-Partitioned LTO Support



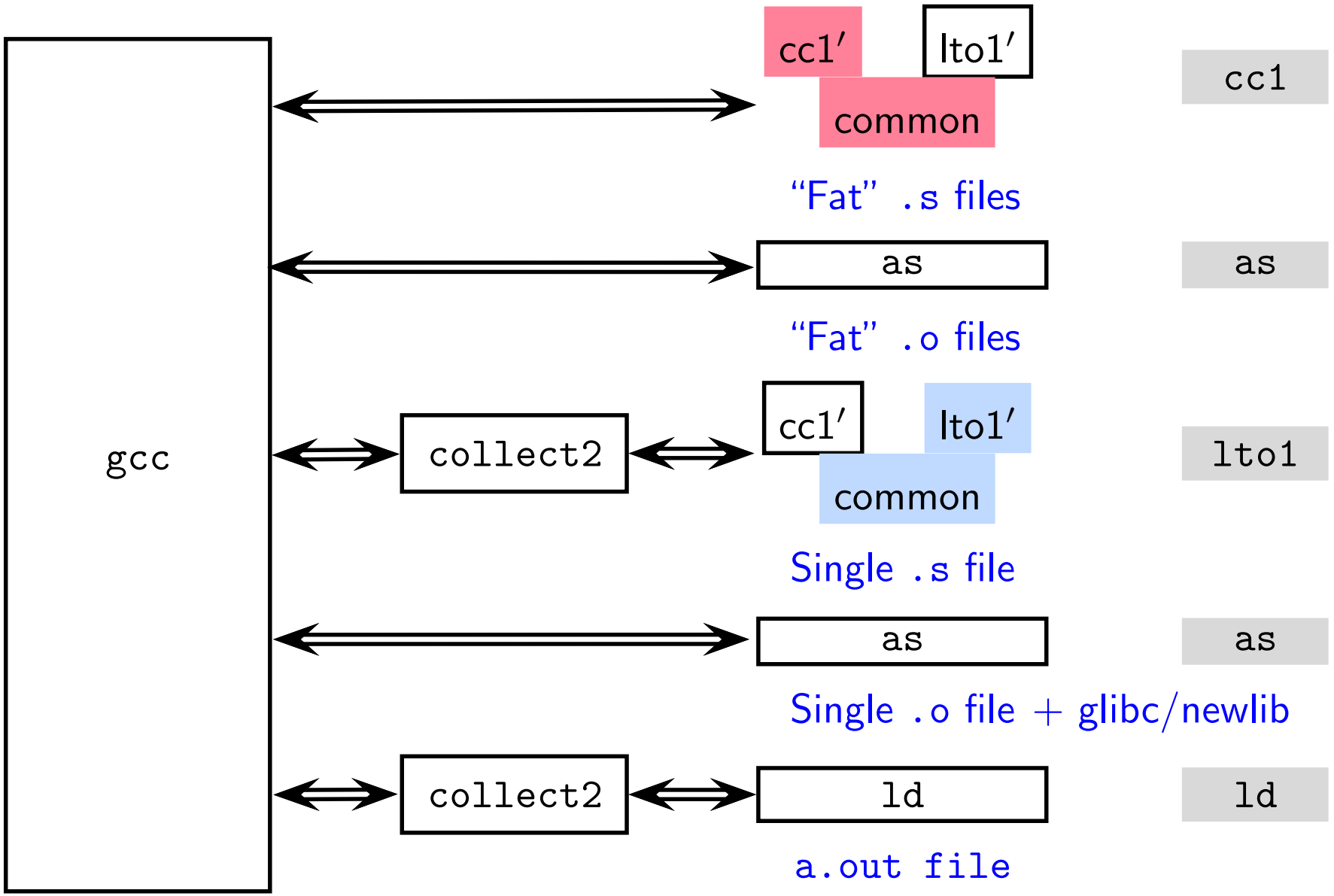
The GNU Tool Chain for Non-Partitioned LTO Support



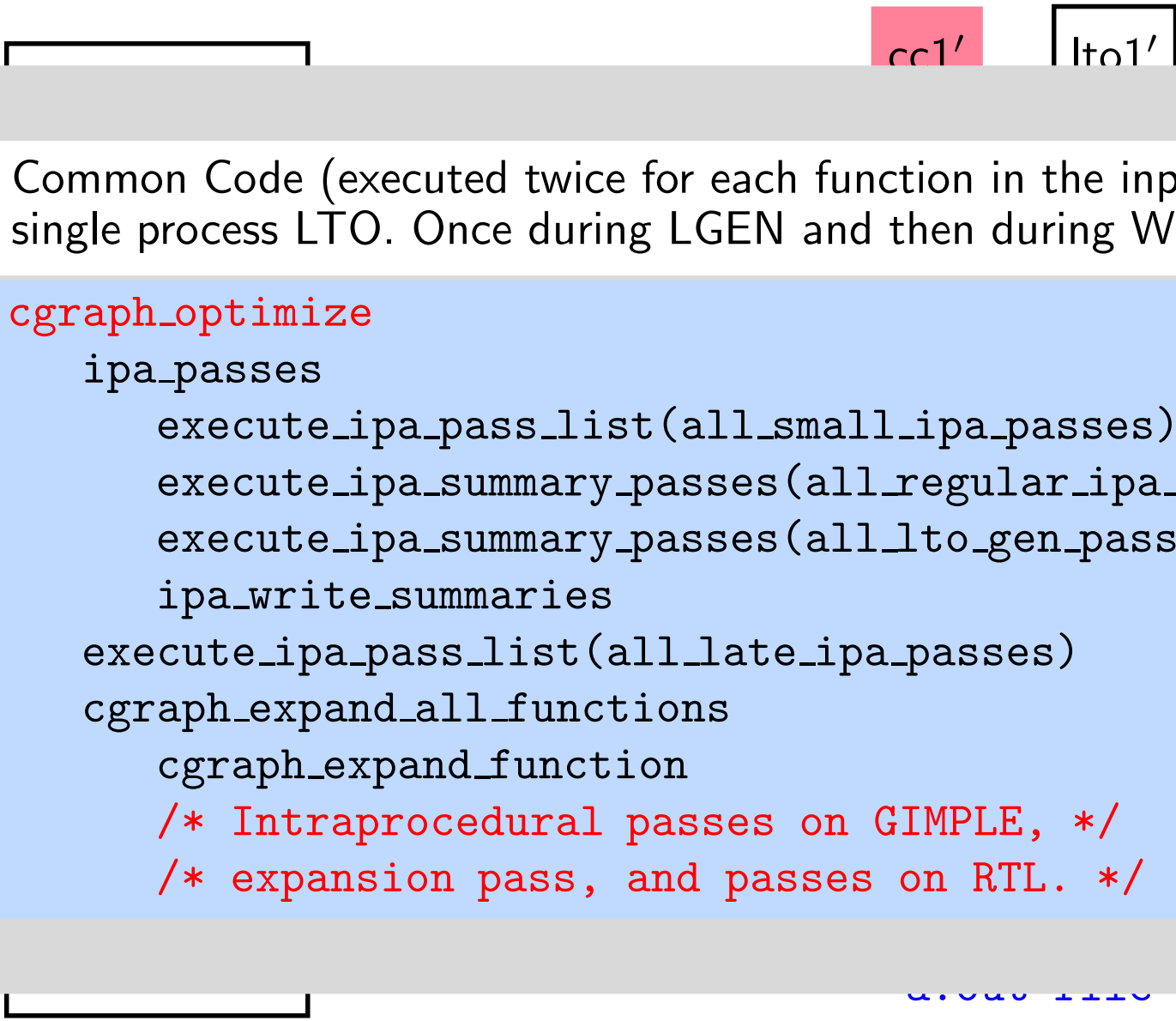
The GNU Tool Chain for Non-Partitioned LTO Support



The GNU Tool Chain for Non-Partitioned LTO Support



The GNU Tool Chain for Non-Partitioned LTO Support

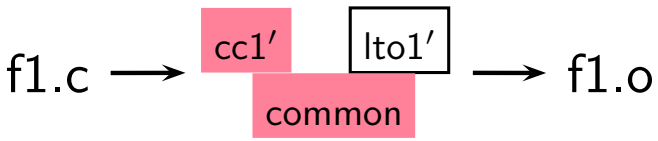


Common Code (executed twice for each function in the input program for single process LTO. Once during LGEN and then during WPA + LTRANS)

```
cgraph_optimize
   ipa_passes
      execute_ipa_pass_list(all_small_ipa_passes)/*!in lto*/
      execute_ipa_summary_passes(all_regular_ipa_passes)
      execute_ipa_summary_passes(all_lto_gen_passes)
      ipa_write_summaries
   execute_ipa_pass_list(all_late_ipa_passes)
   cgraph_expand_all_functions
      cgraph_expand_function
      /* Intraprocedural passes on GIMPLE, */
      /* expansion pass, and passes on RTL. */
```

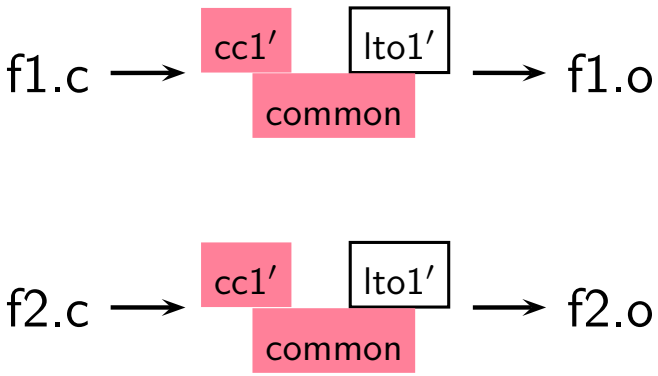
Partitioned LTO (aka WHOPR LTO)

Option `-flto -c`



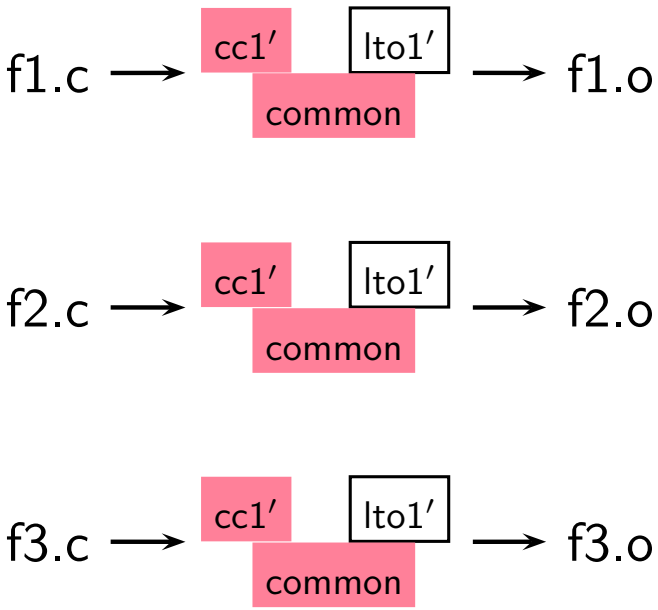
Partitioned LTO (aka WHOPR LTO)

Option `-flto -c`

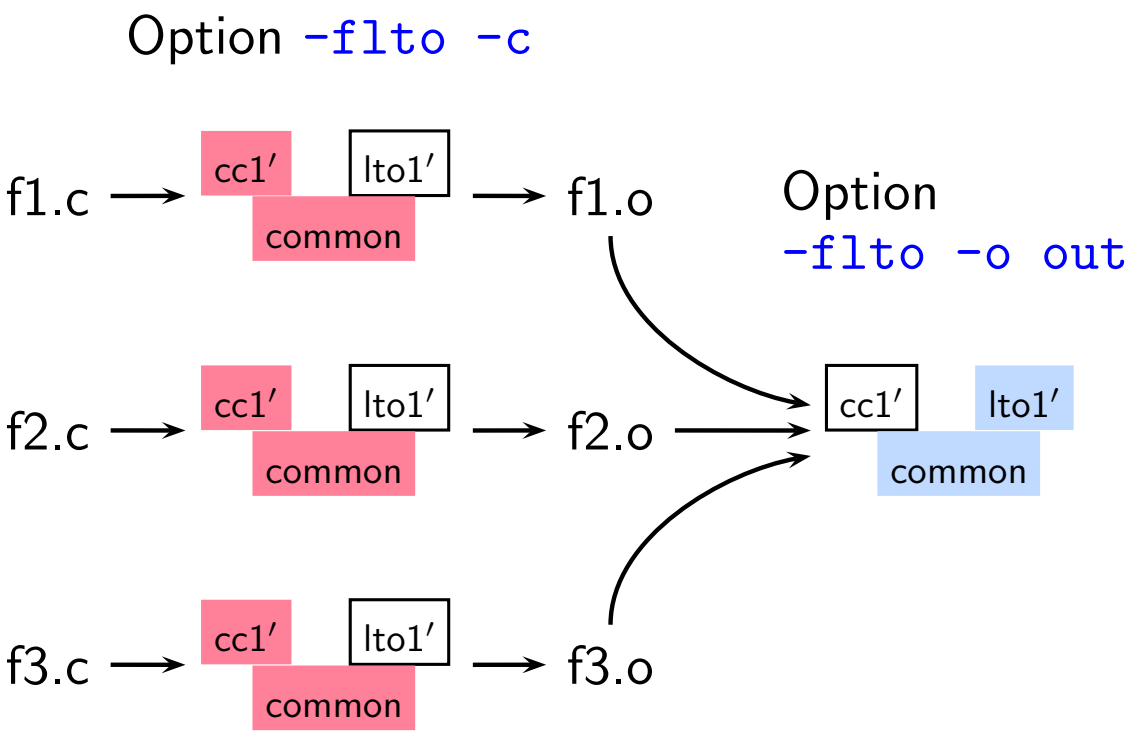


Partitioned LTO (aka WHOPR LTO)

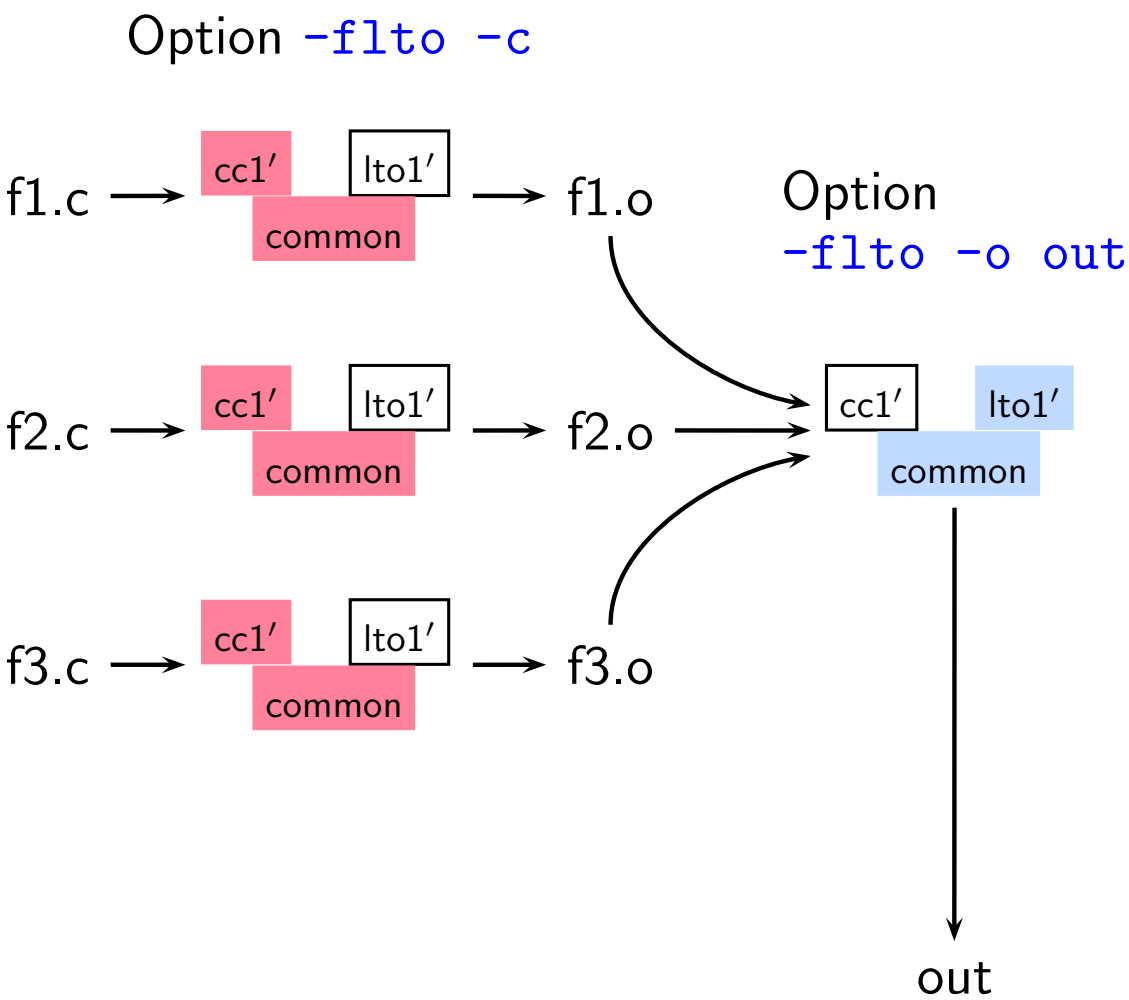
Option `-flto -c`



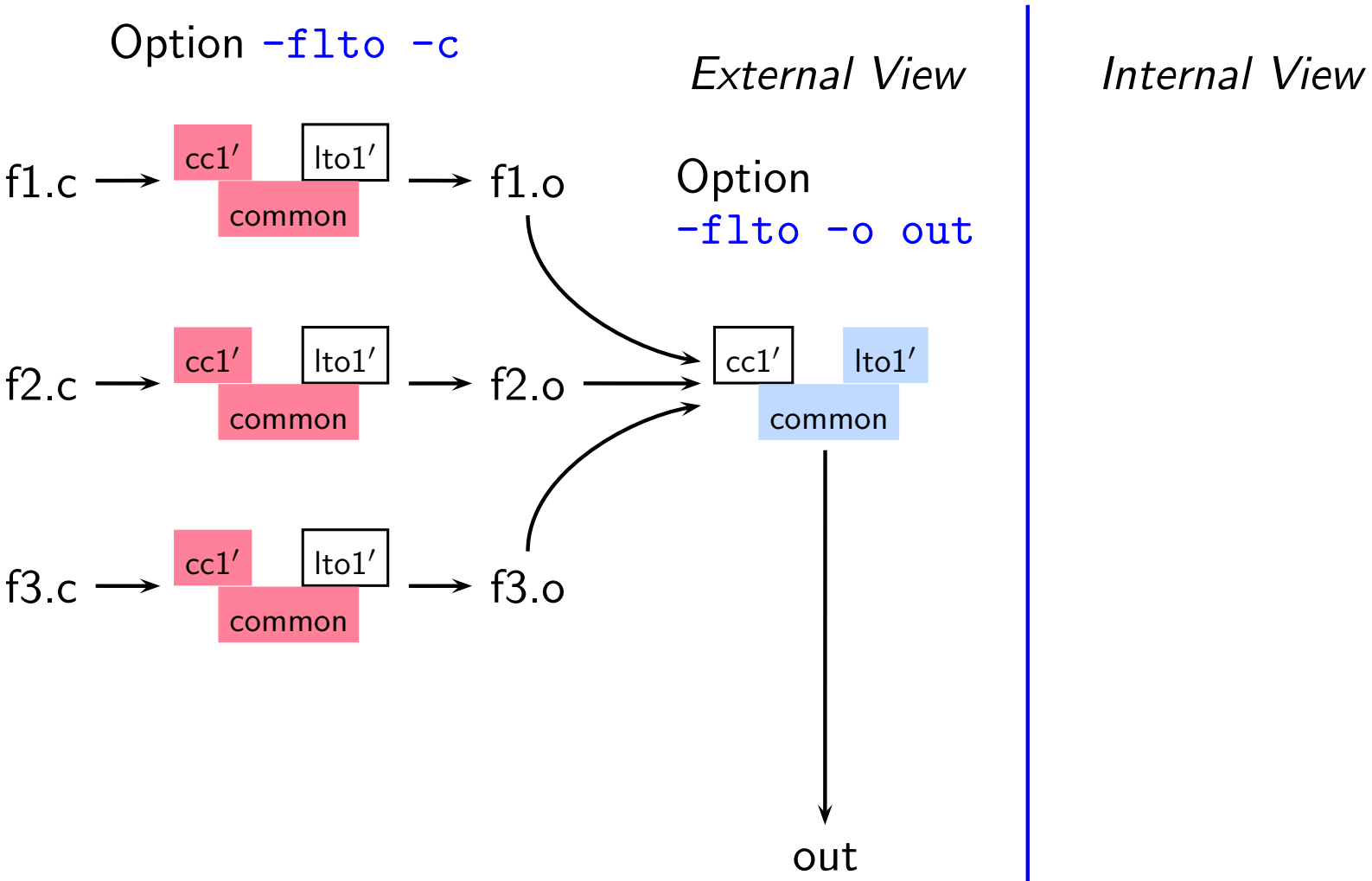
Partitioned LTO (aka WHOPR LTO)



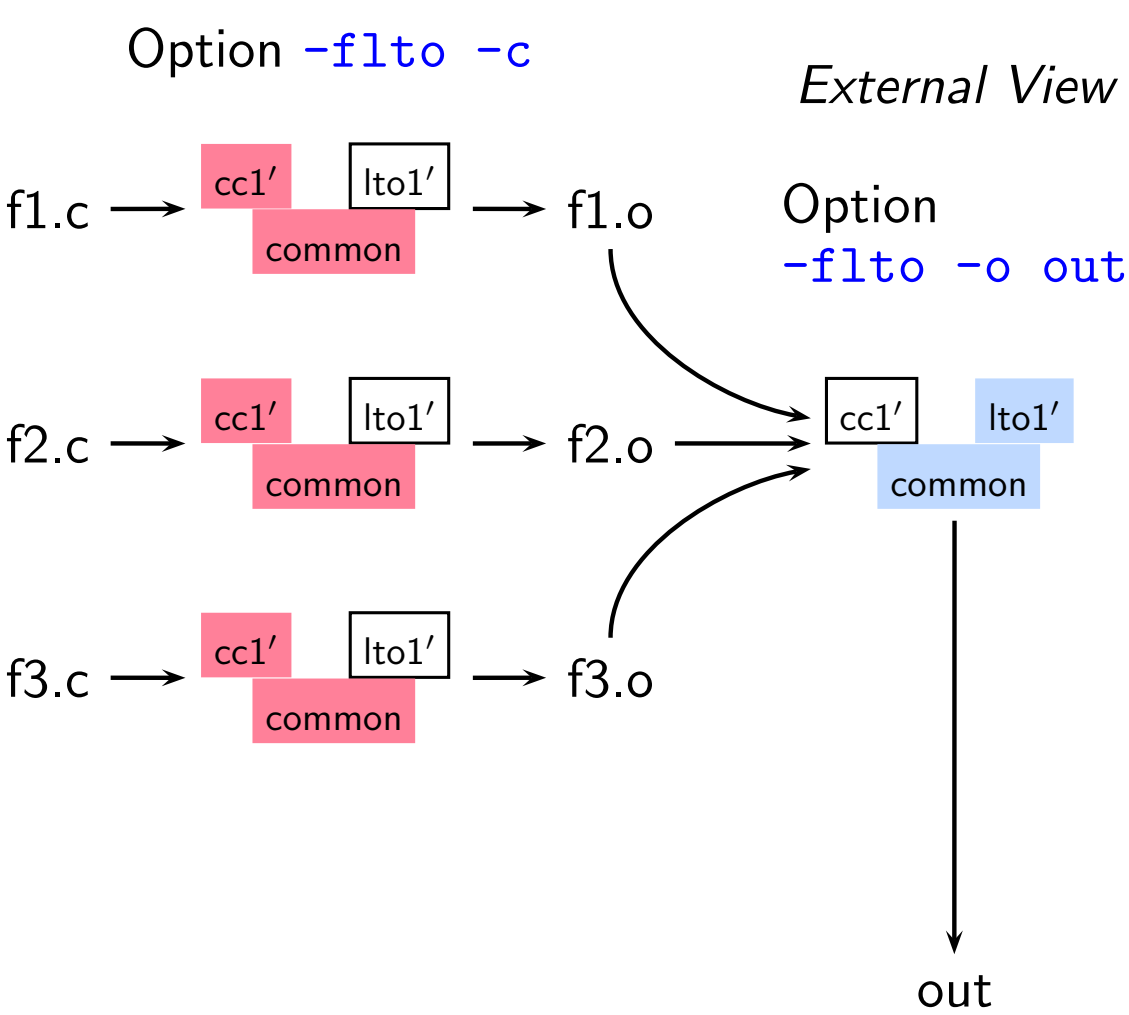
Partitioned LTO (aka WHOPR LTO)



Partitioned LTO (aka WHOPR LTO)



Partitioned LTO (aka WHOPR LTO)

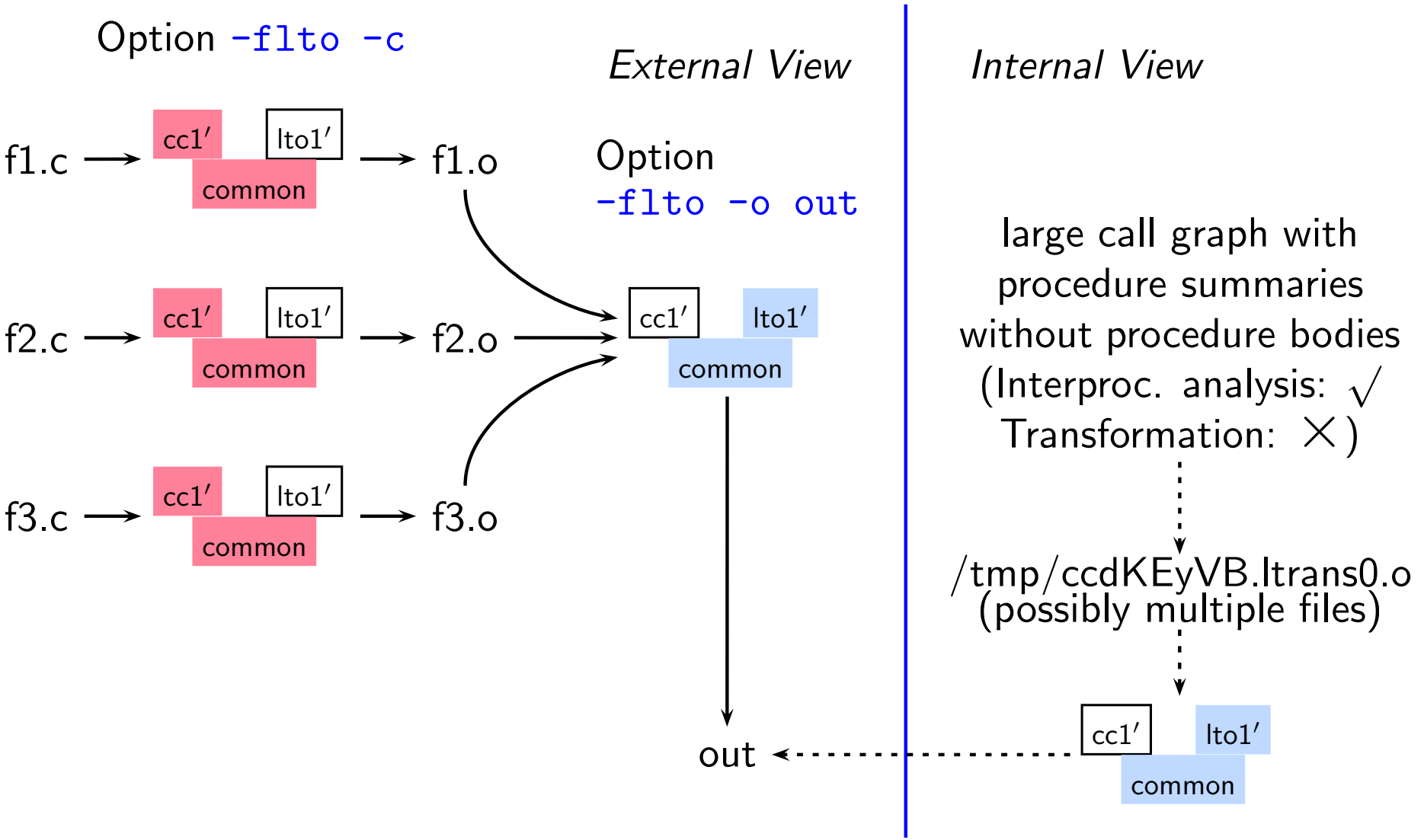


Internal View

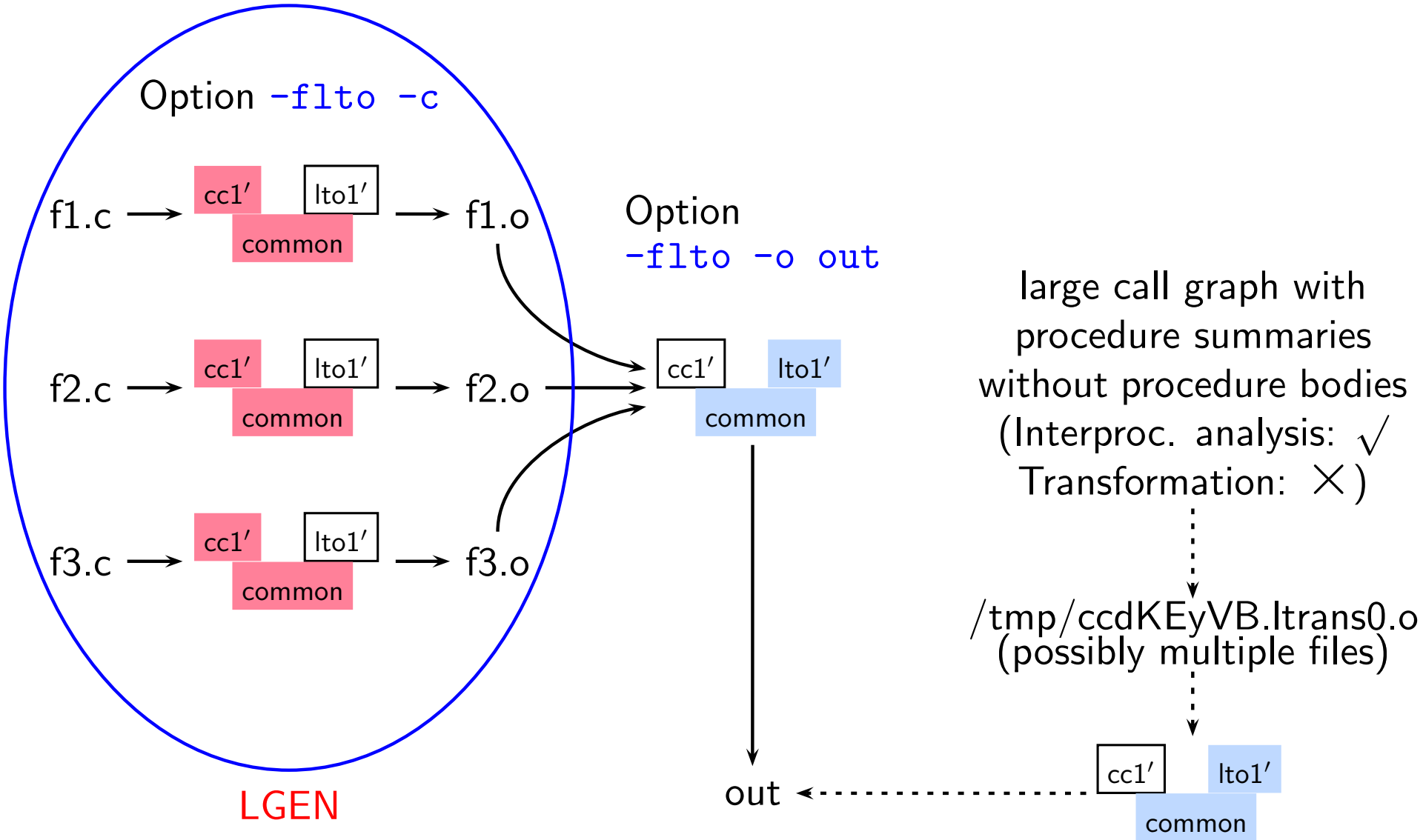
large call graph with procedure summaries without procedure bodies (Interproc. analysis: $\surd$ Transformation: $\times$)

/tmp/ccdKEyVB.ltrans0.o (possibly multiple files)

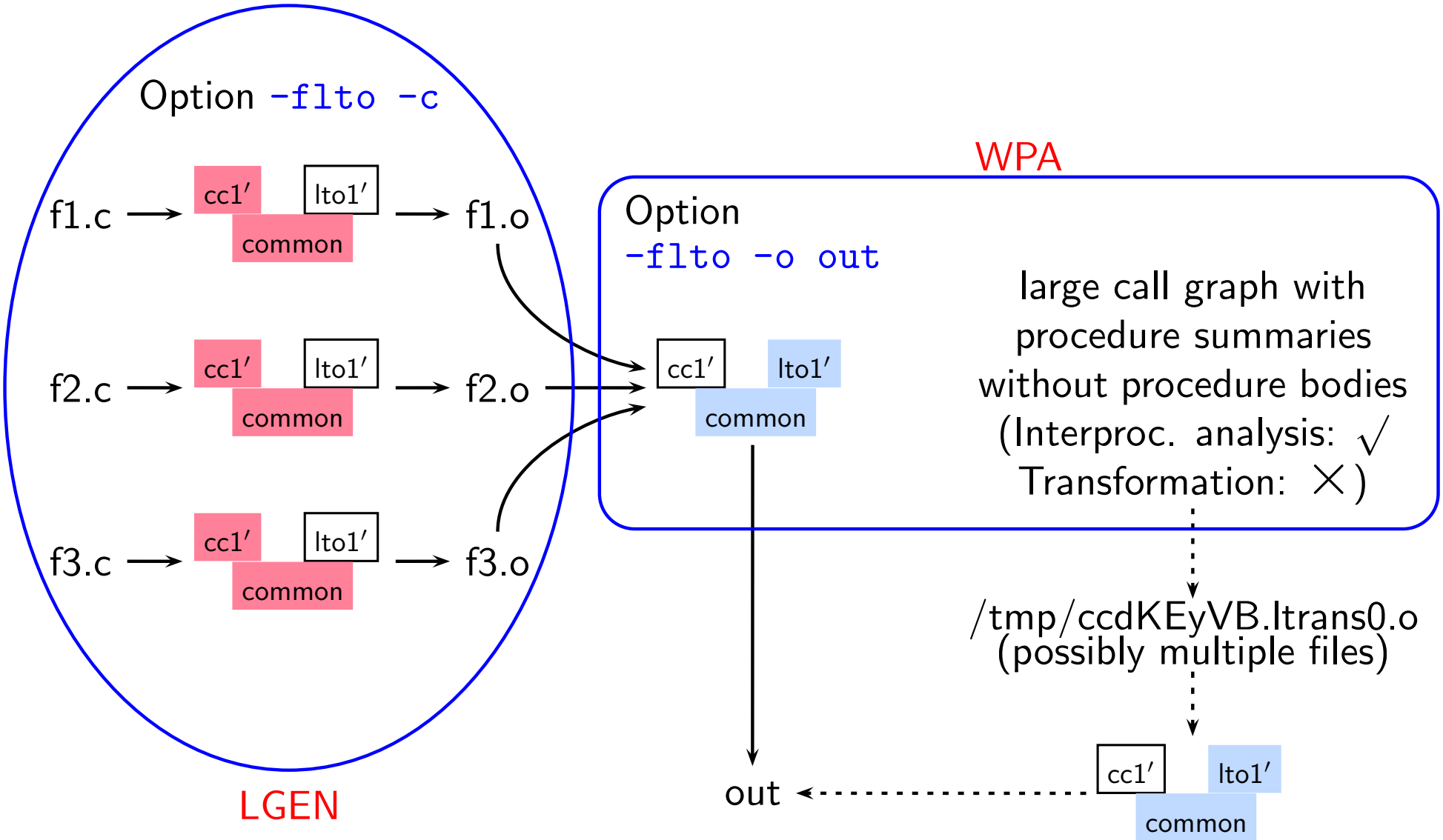
Partitioned LTO (aka WHOPR LTO)



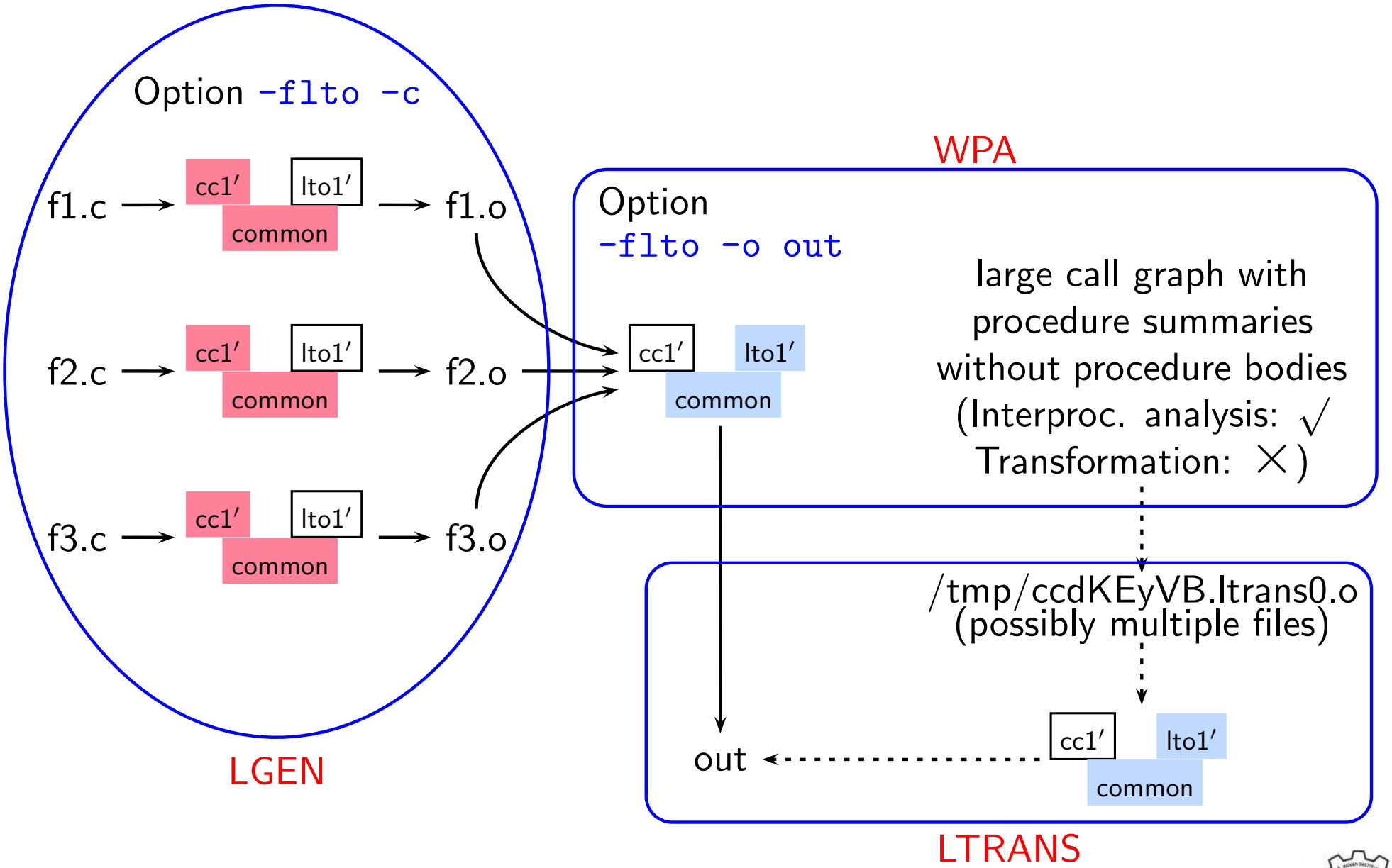
Partitioned LTO (aka WHOPR LTO)



Partitioned LTO (aka WHOPR LTO)

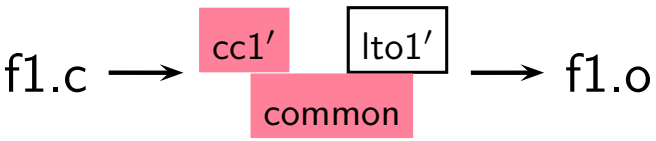


Partitioned LTO (aka WHOPR LTO)



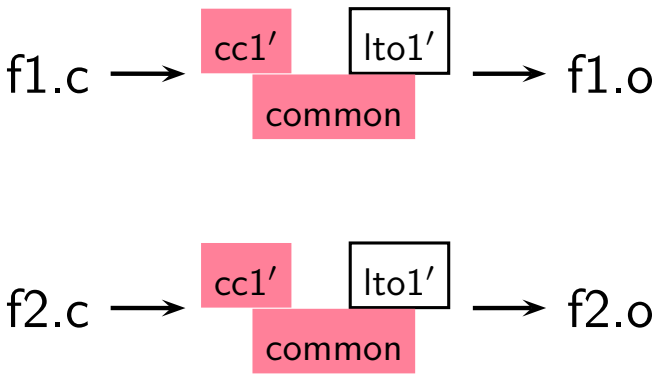
Non-Partitioned LTO

Option `-flto -c`



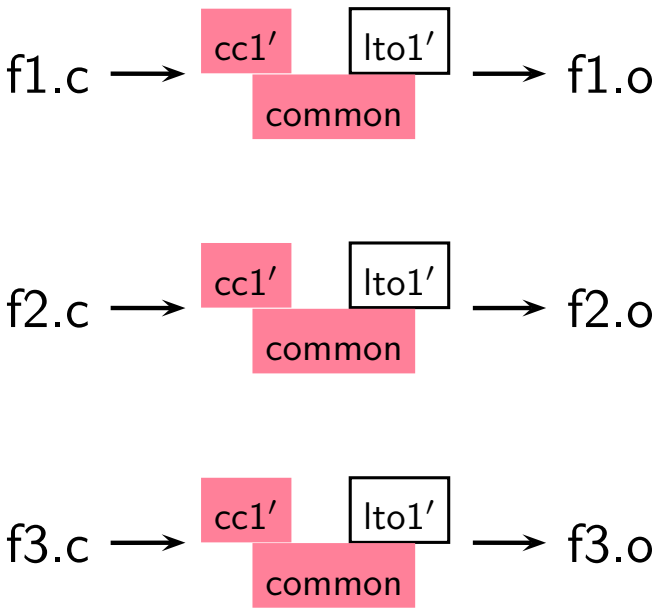
Non-Partitioned LTO

Option `-flto -c`

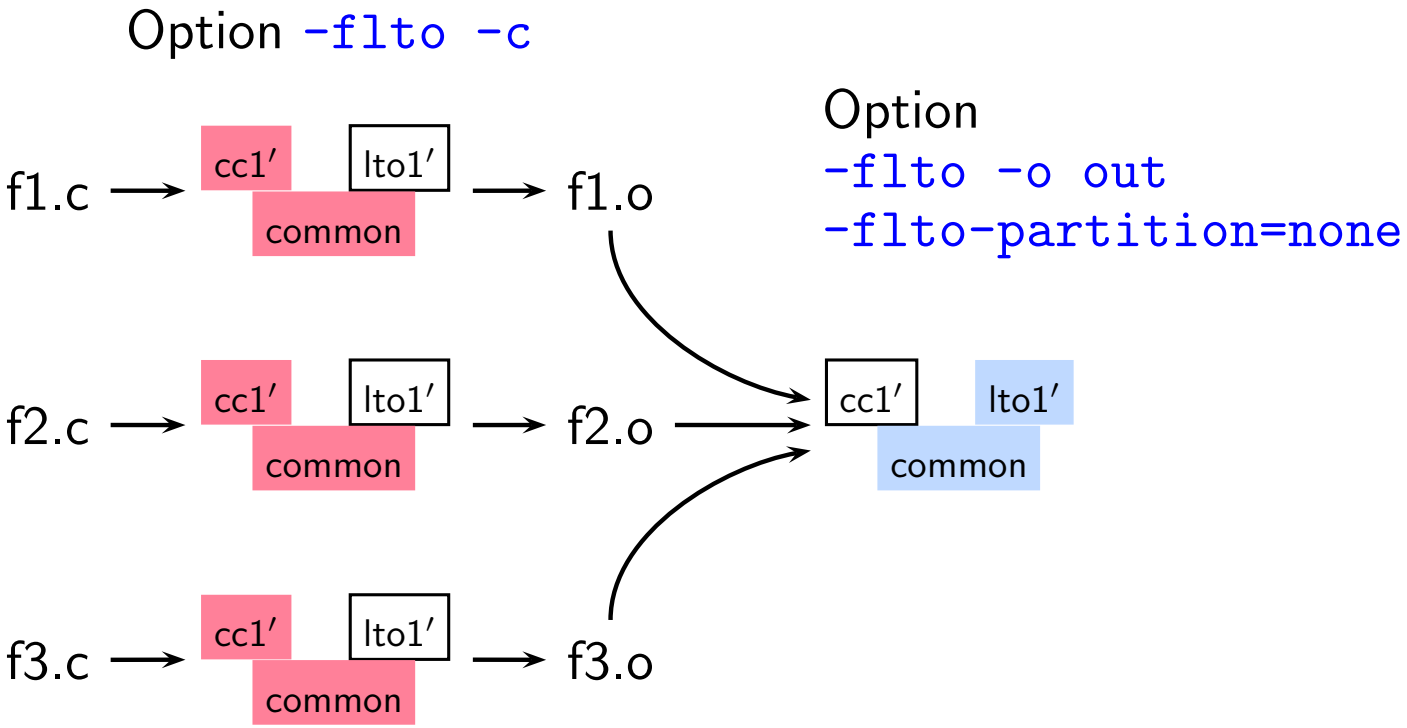


Non-Partitioned LTO

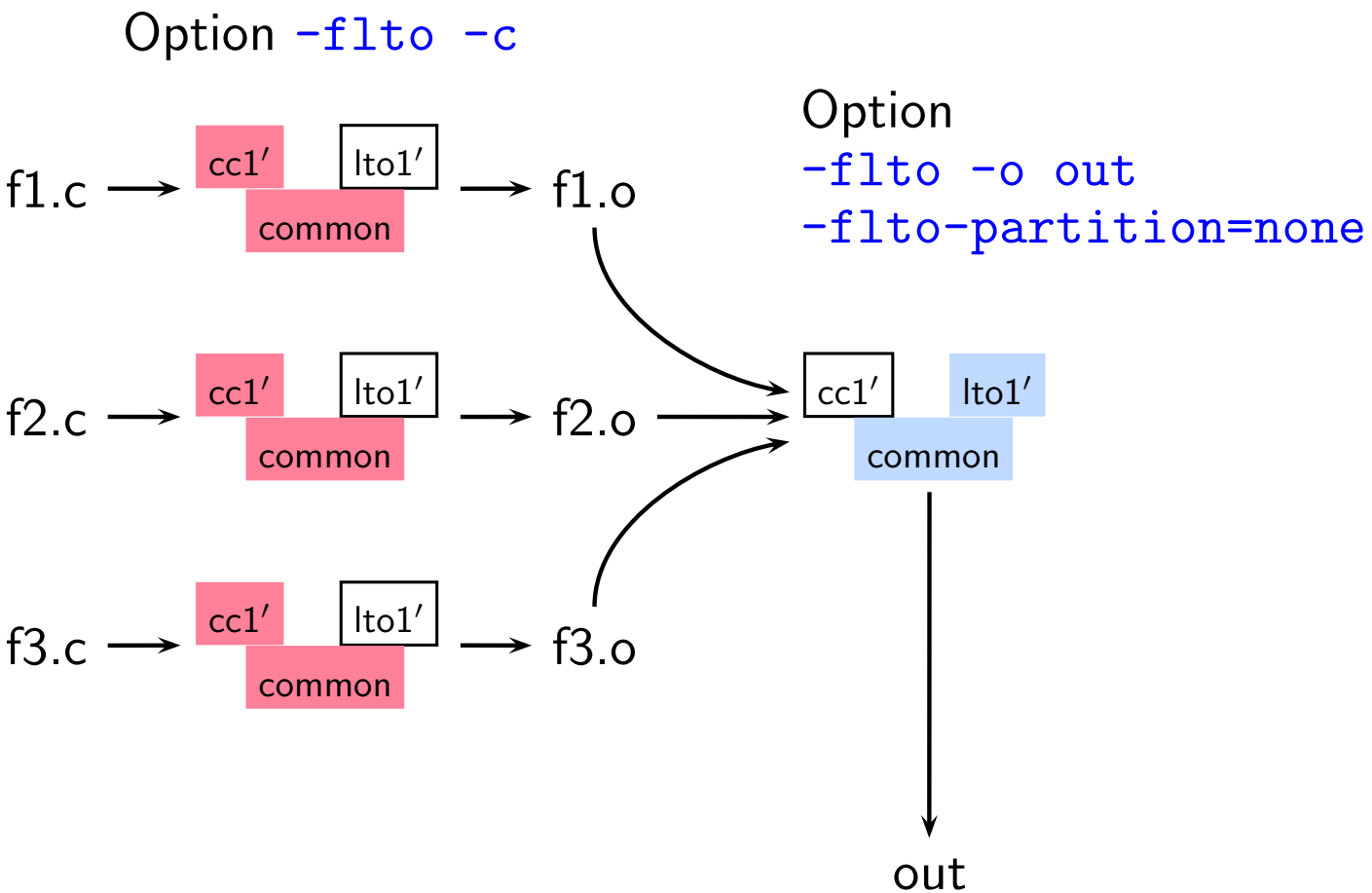
Option `-flto -c`



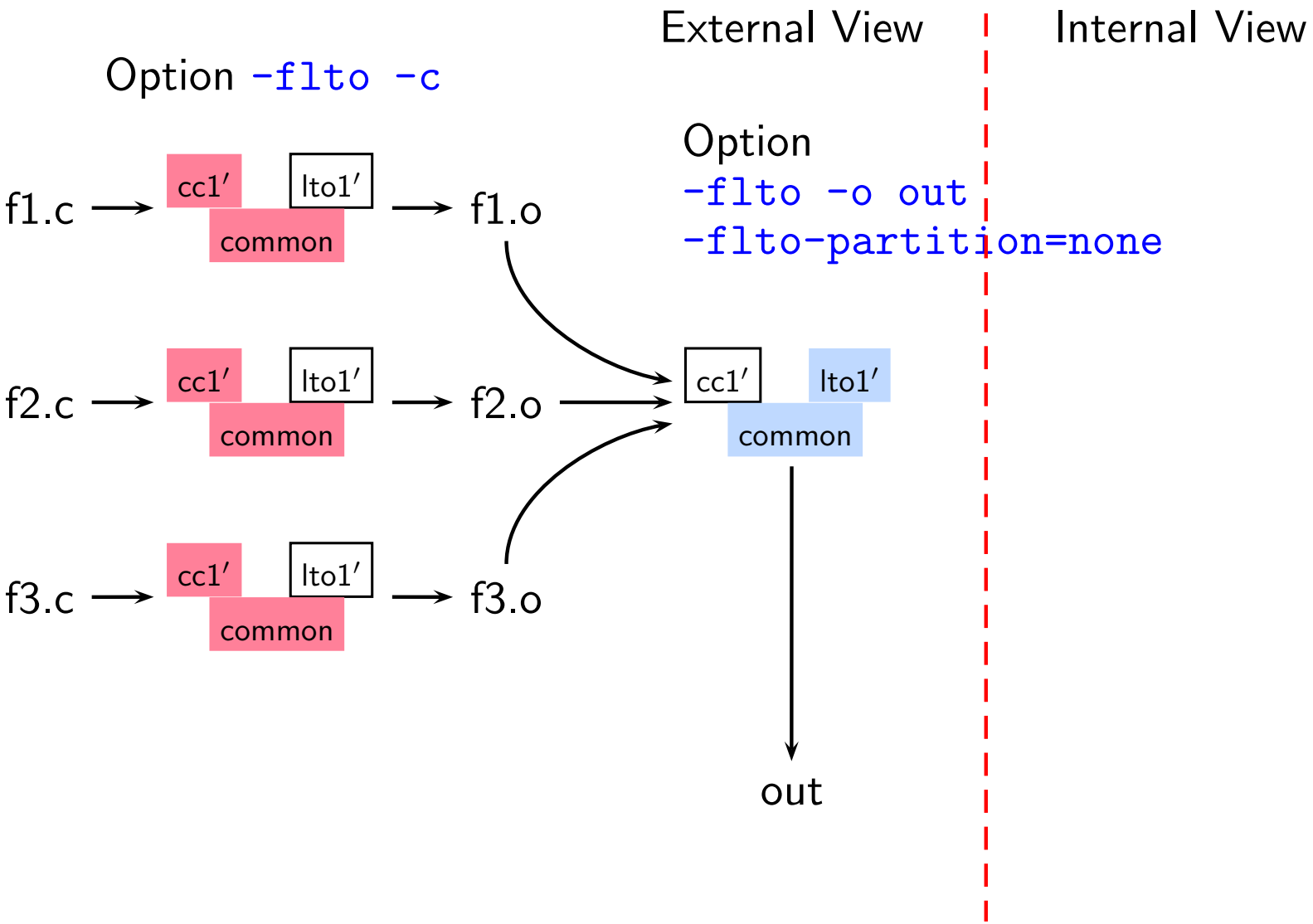
Non-Partitioned LTO



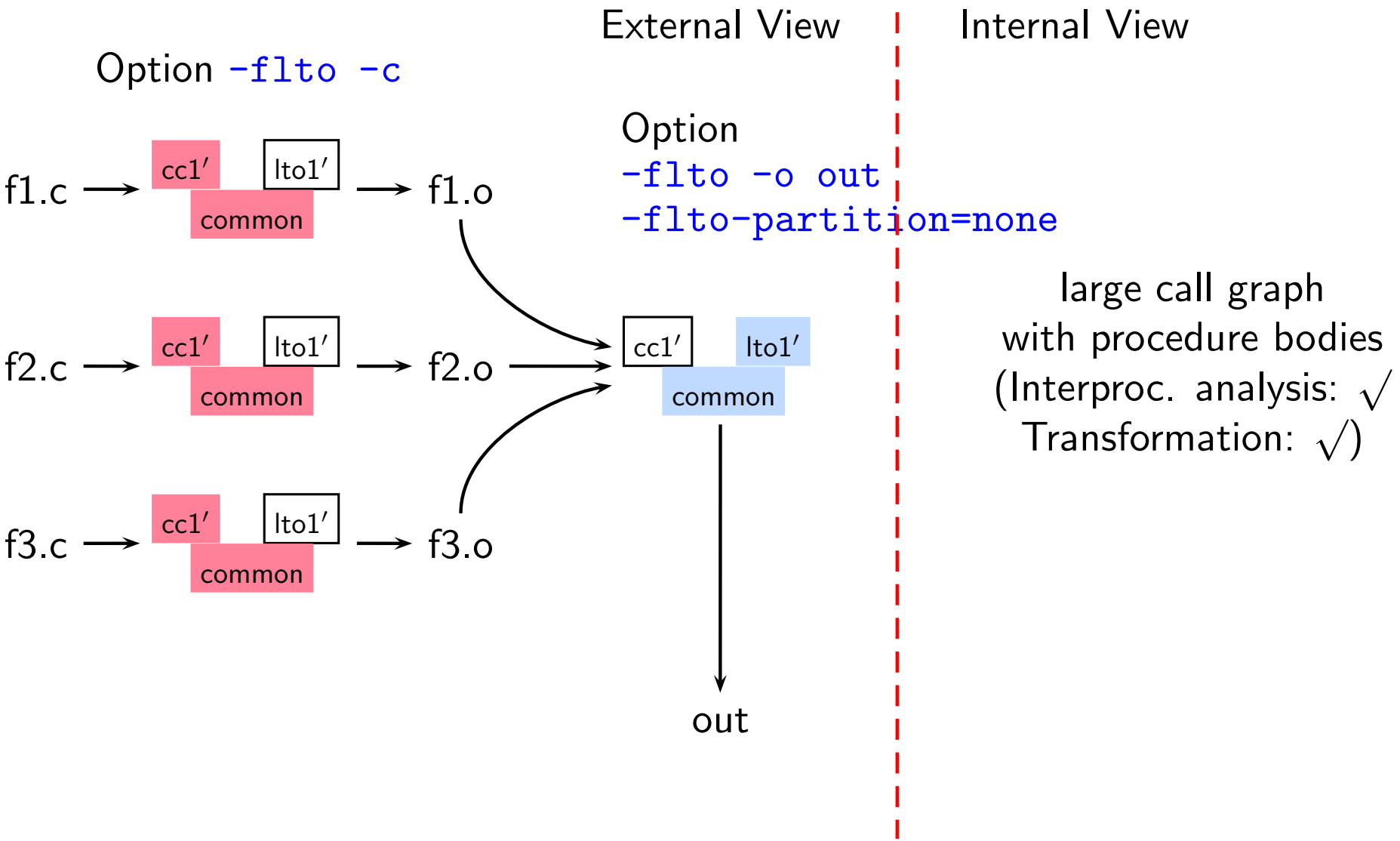
Non-Partitioned LTO



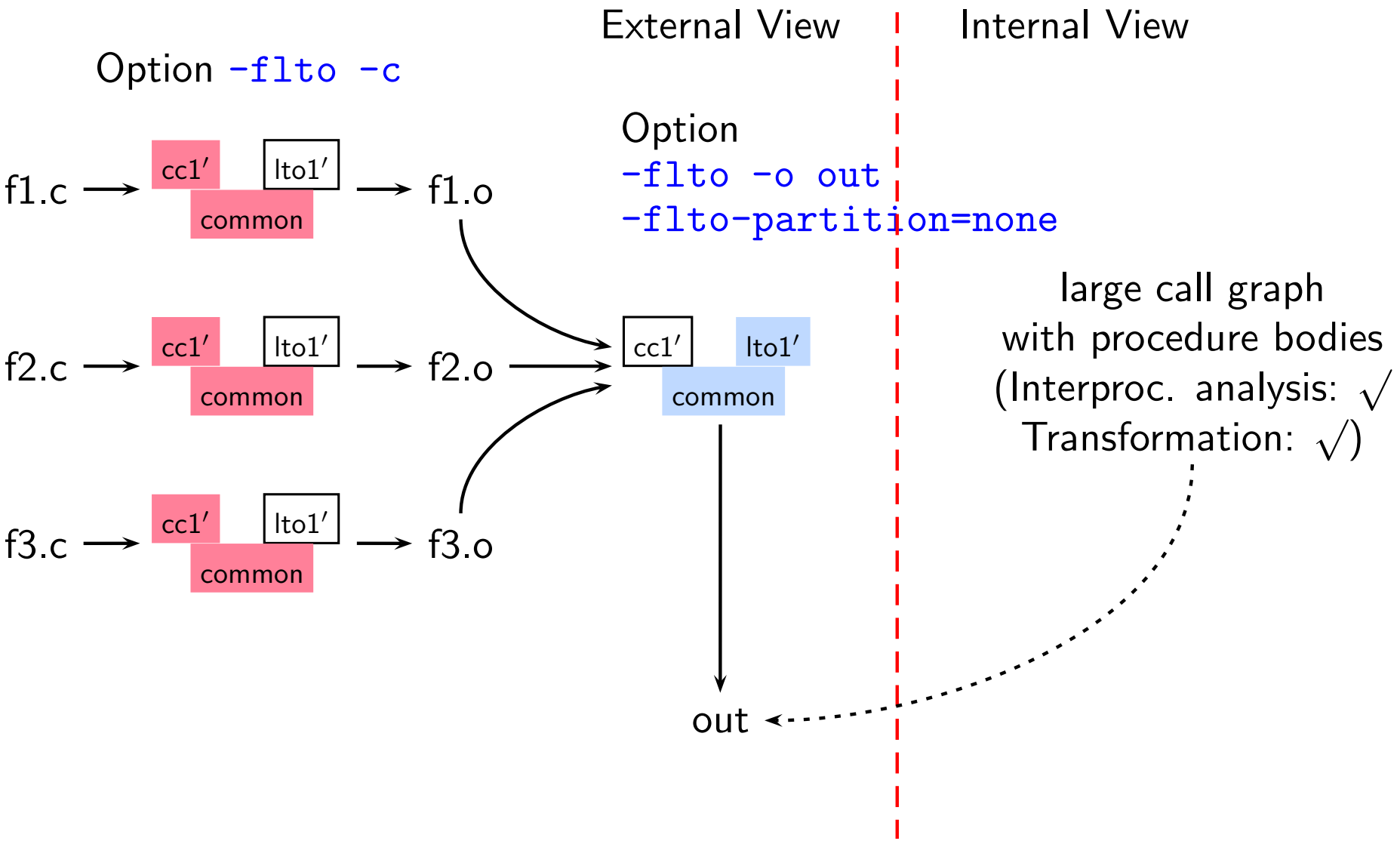
Non-Partitioned LTO



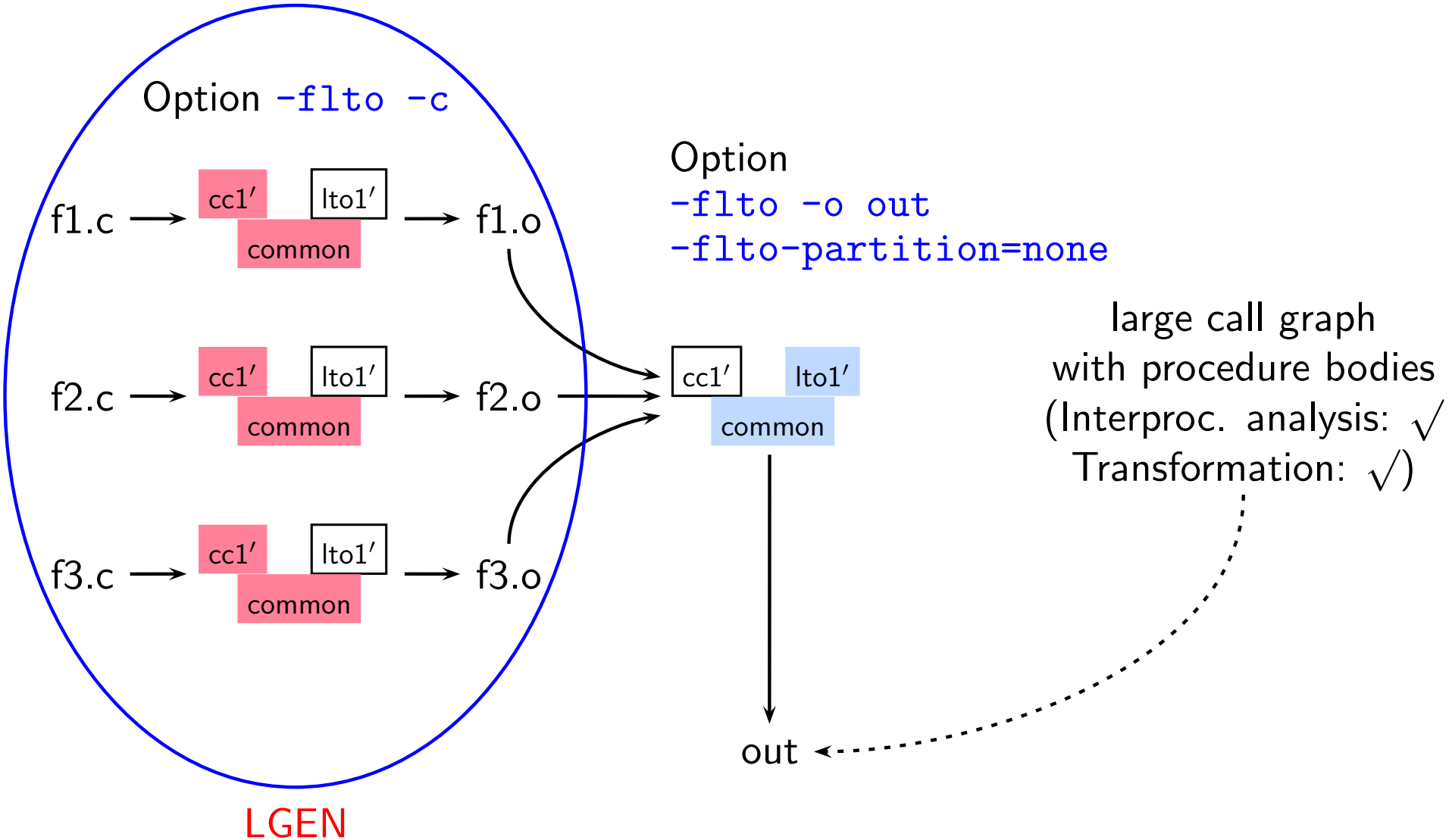
Non-Partitioned LTO



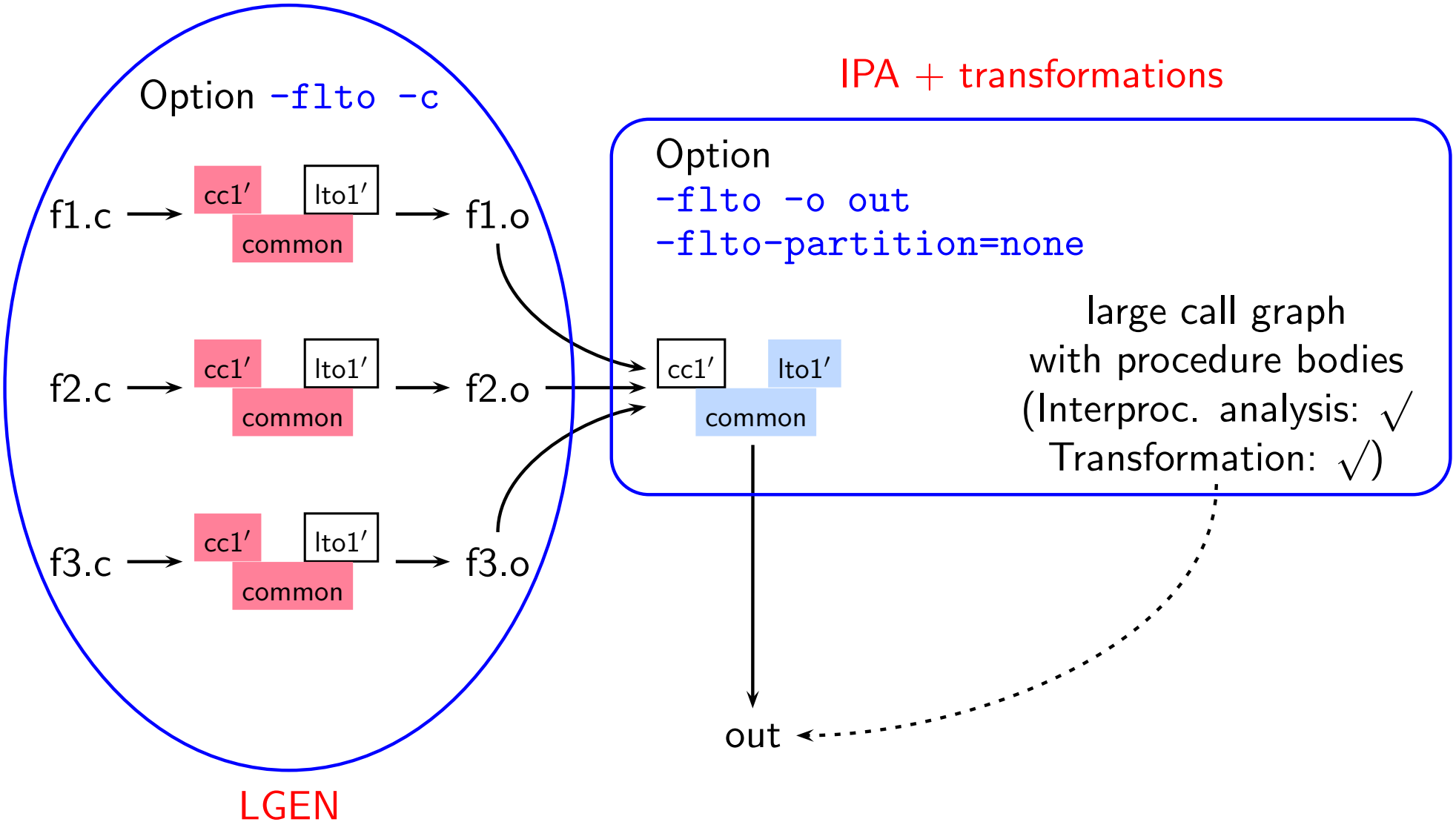
Non-Partitioned LTO



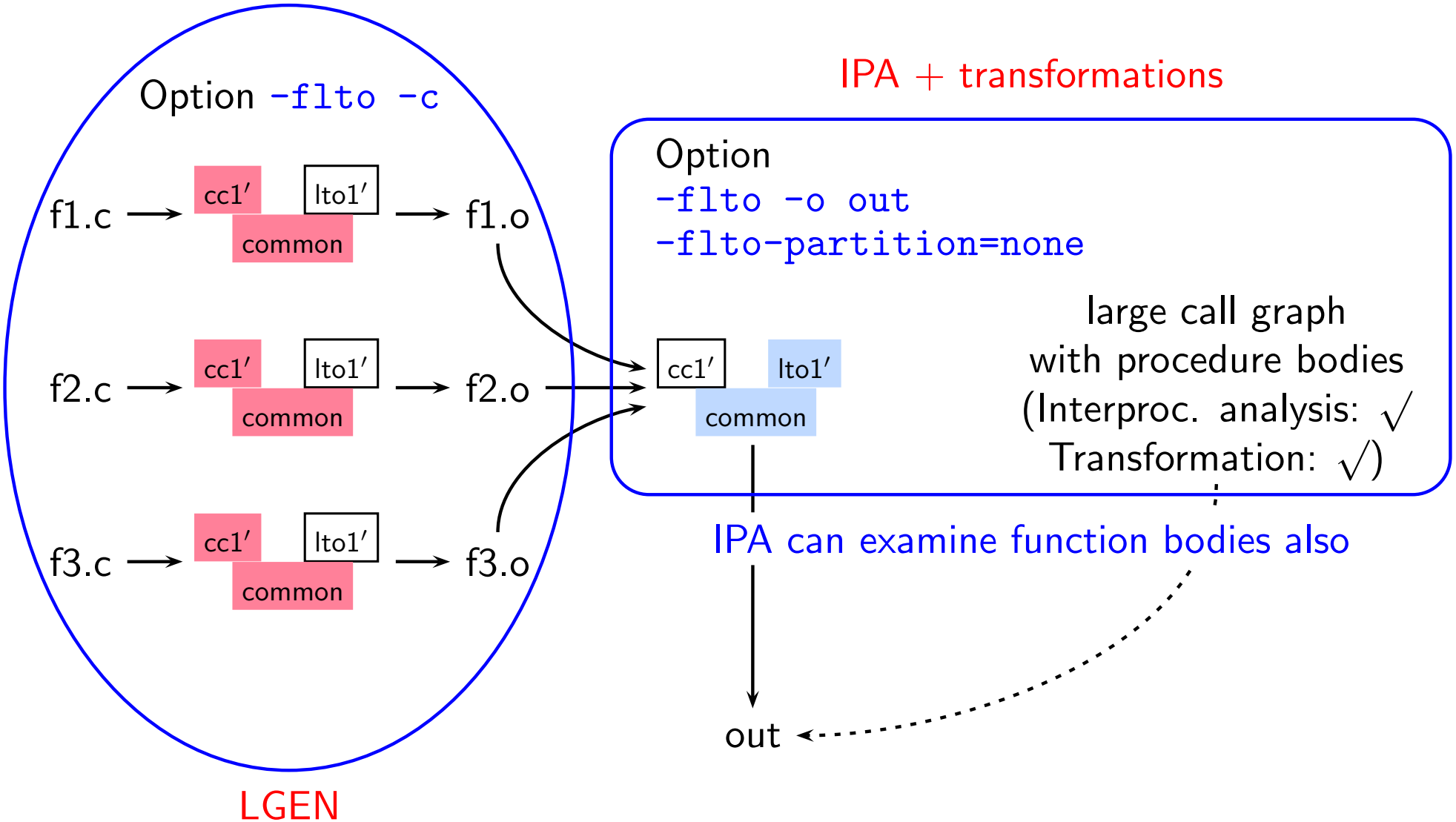
Non-Partitioned LTO



Non-Partitioned LTO



Non-Partitioned LTO



Part 6

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
- LTO is a good support for interprocedural analysis and optimization

  It would be useful to support an LTO mode that
  - creates a large call graph of the entire program with
  - on-demand loading of procedure bodies for
  - enabling examining procedure bodies for interprocedural analysis