Link Time Optimization Mechanism in GCC-4.7.2

Uday Khedker

(www.cse.iitb.ac.in/~uday)

GCC Resource Center,
Department of Computer Science and Engineering,
Indian Institute of Technology, Bombay

13 June 2014

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 in GCC-4.7.2 (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 in GCC-4.7.2 (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 in GCC-4.7.2 (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 in GCC-4.7.2 (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 in GCC-4.7.2 (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
```


Main Change in GCC-4.9.0

- LTO output does not contain object code but only LTO information

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

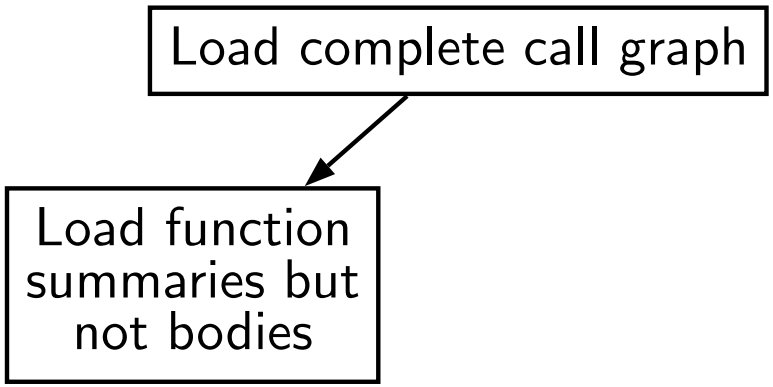
Analysis

Sequential Analysis

Transformation

Partitioned and Non-Partitioned LTO

Analysis

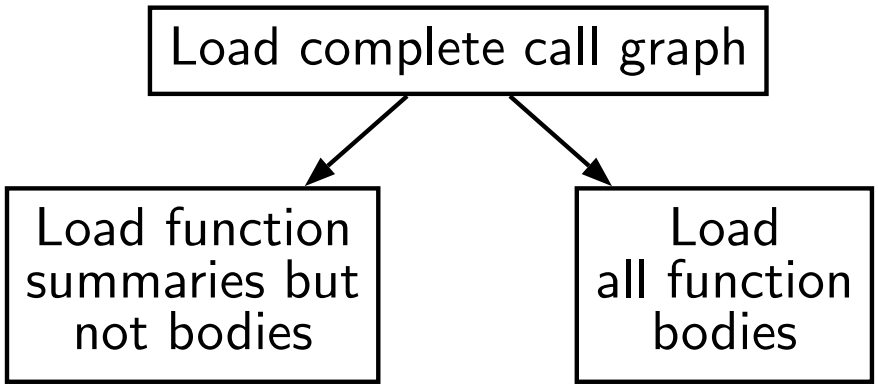


Sequential Analysis

Transformation

Partitioned and Non-Partitioned LTO

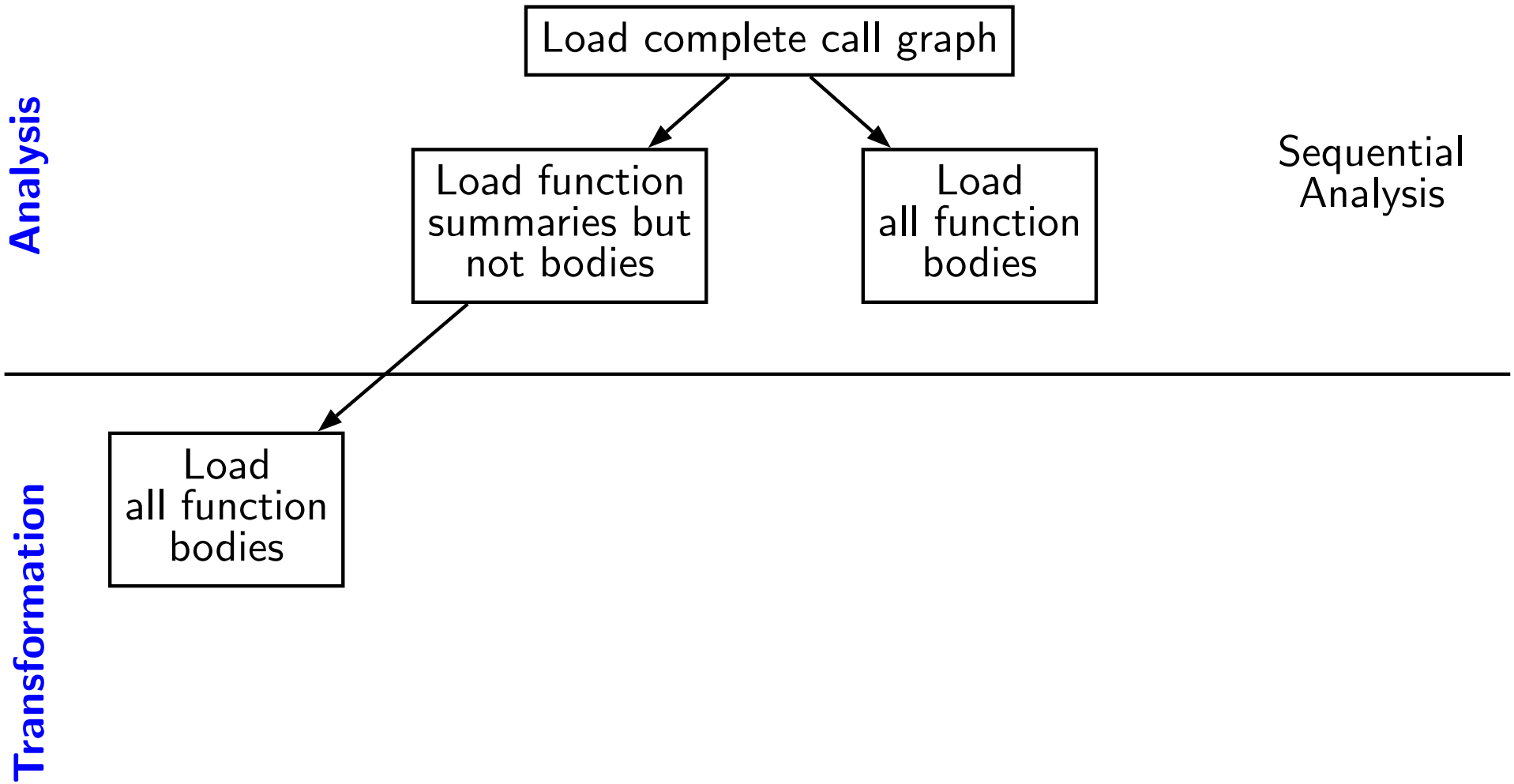
Analysis



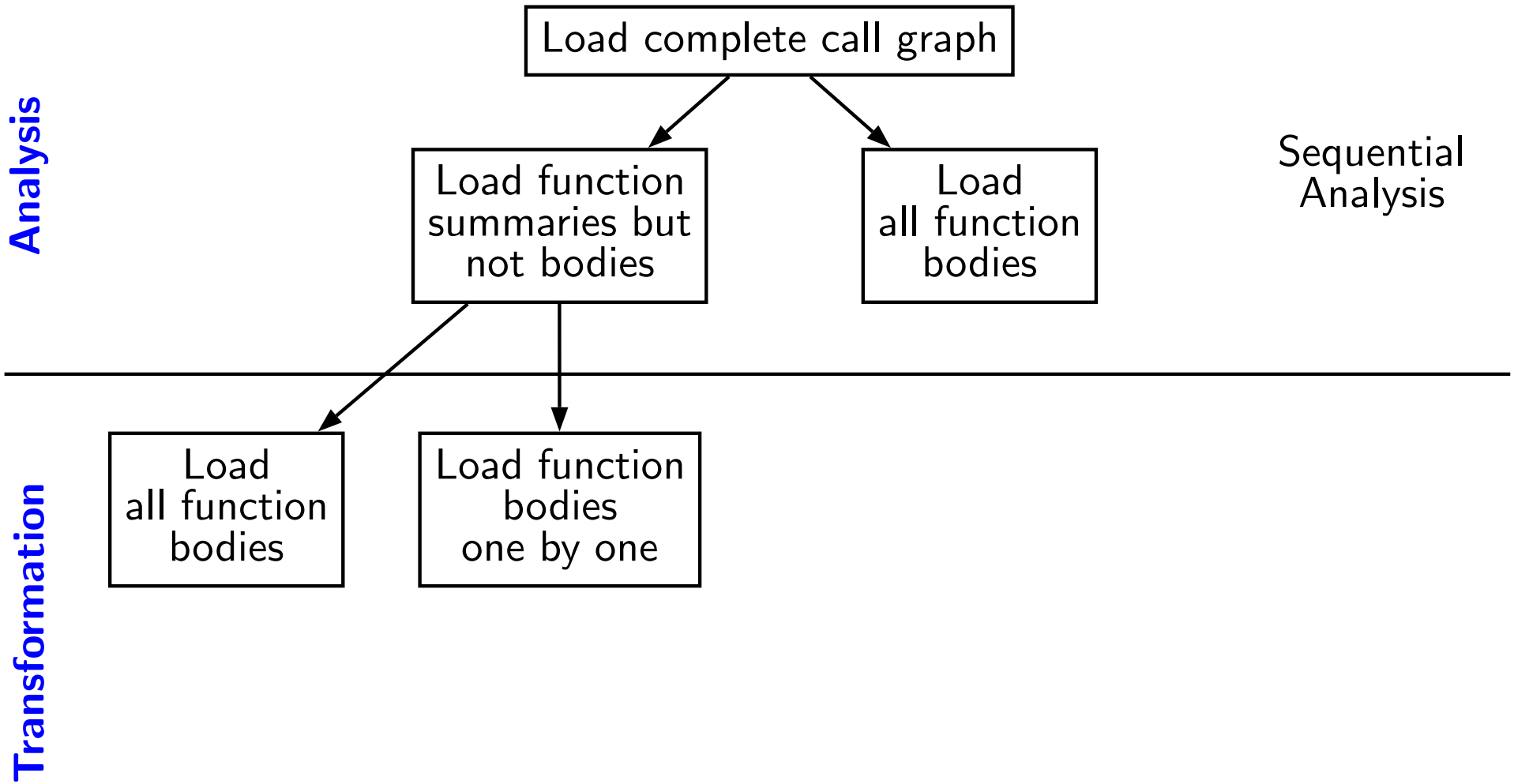
Sequential Analysis

Transformation

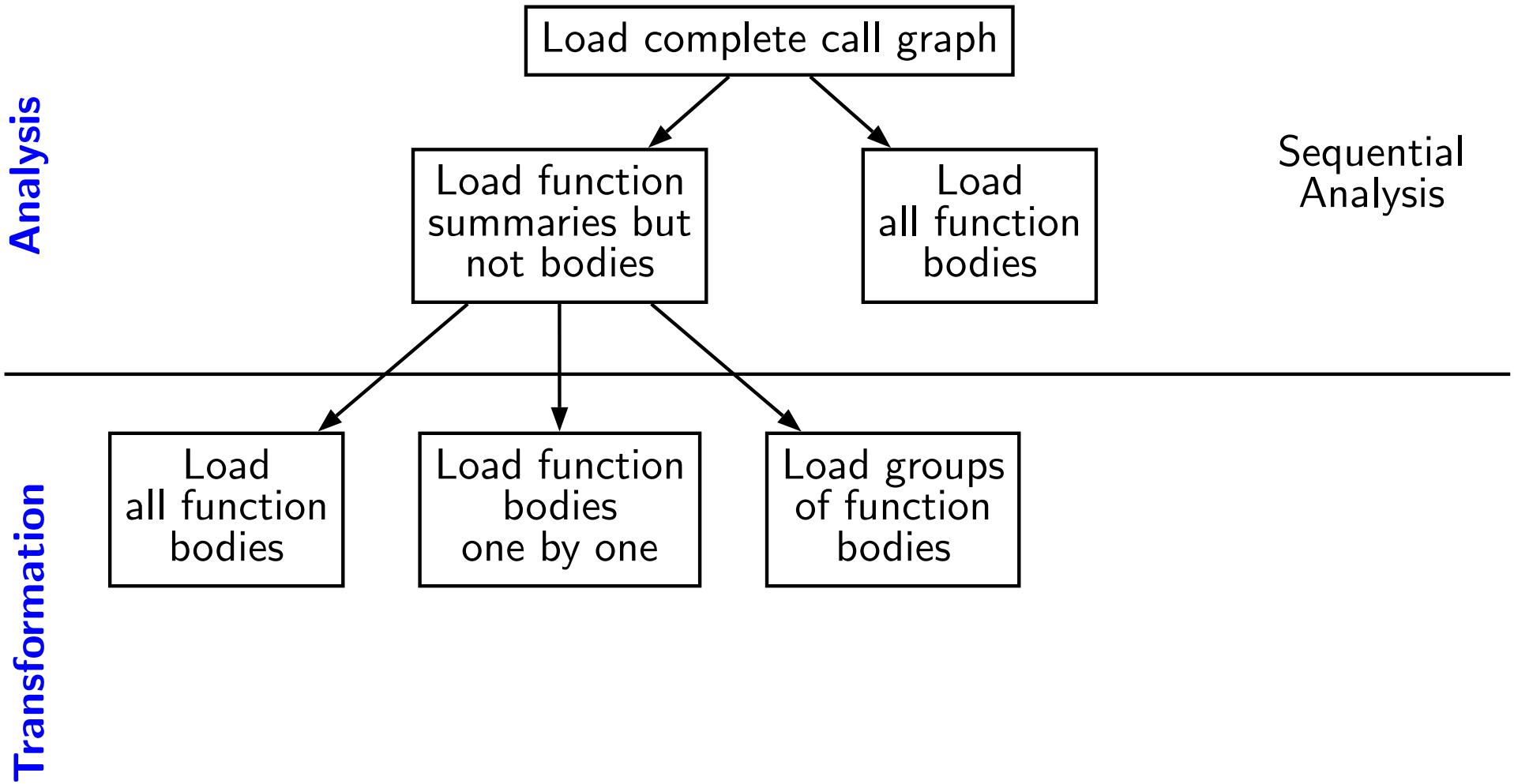
Partitioned and Non-Partitioned LTO



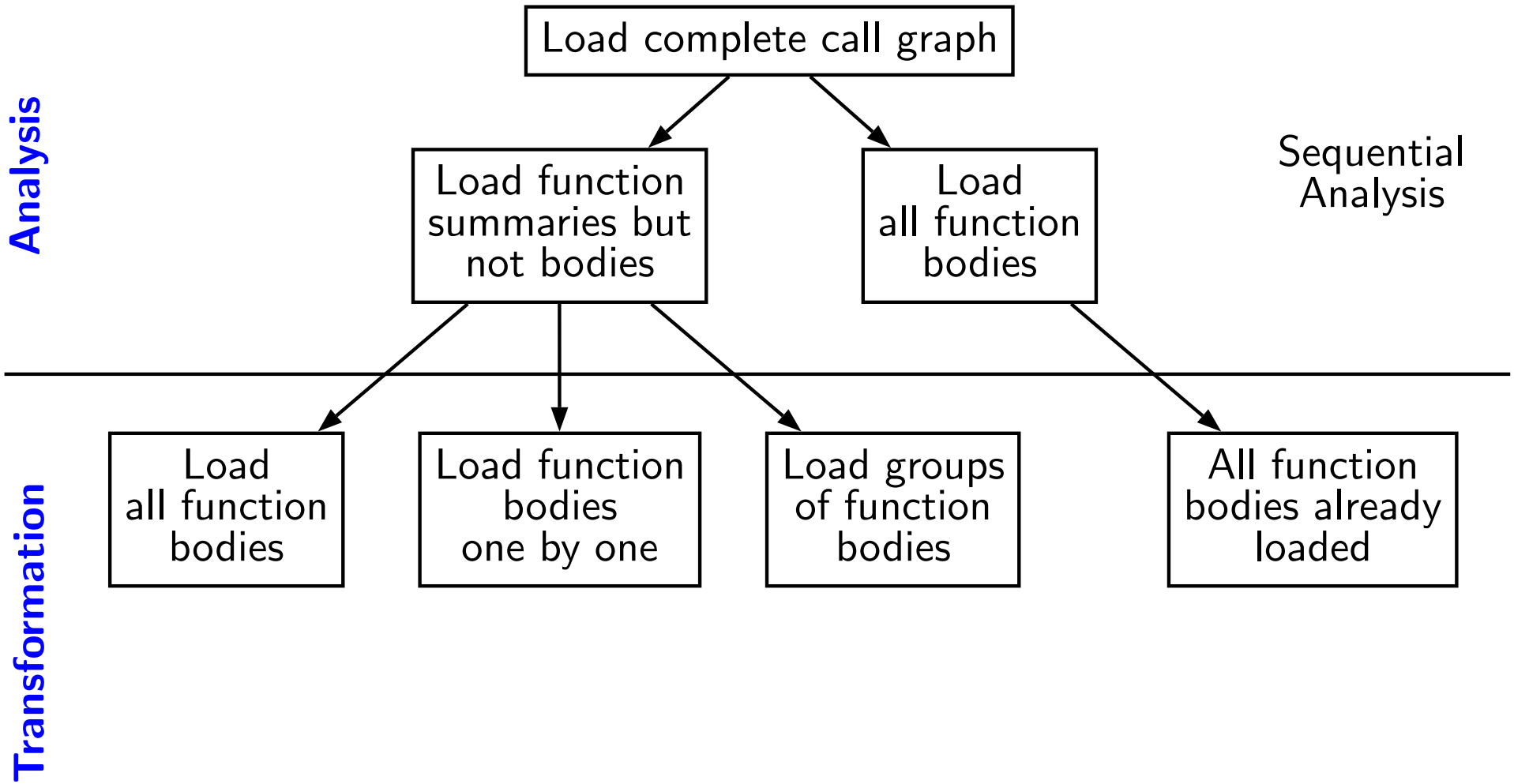
Partitioned and Non-Partitioned LTO



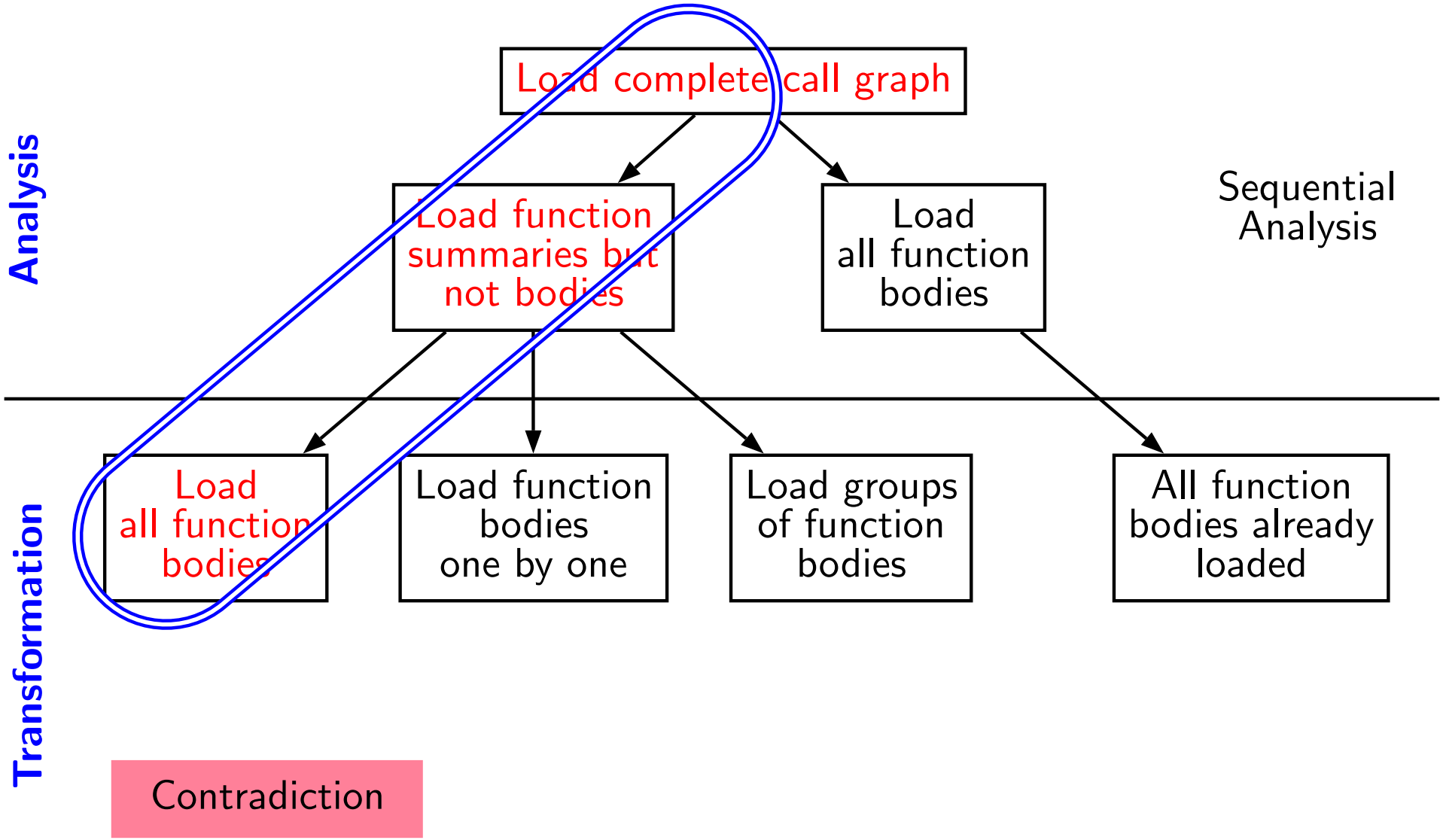
Partitioned and Non-Partitioned LTO



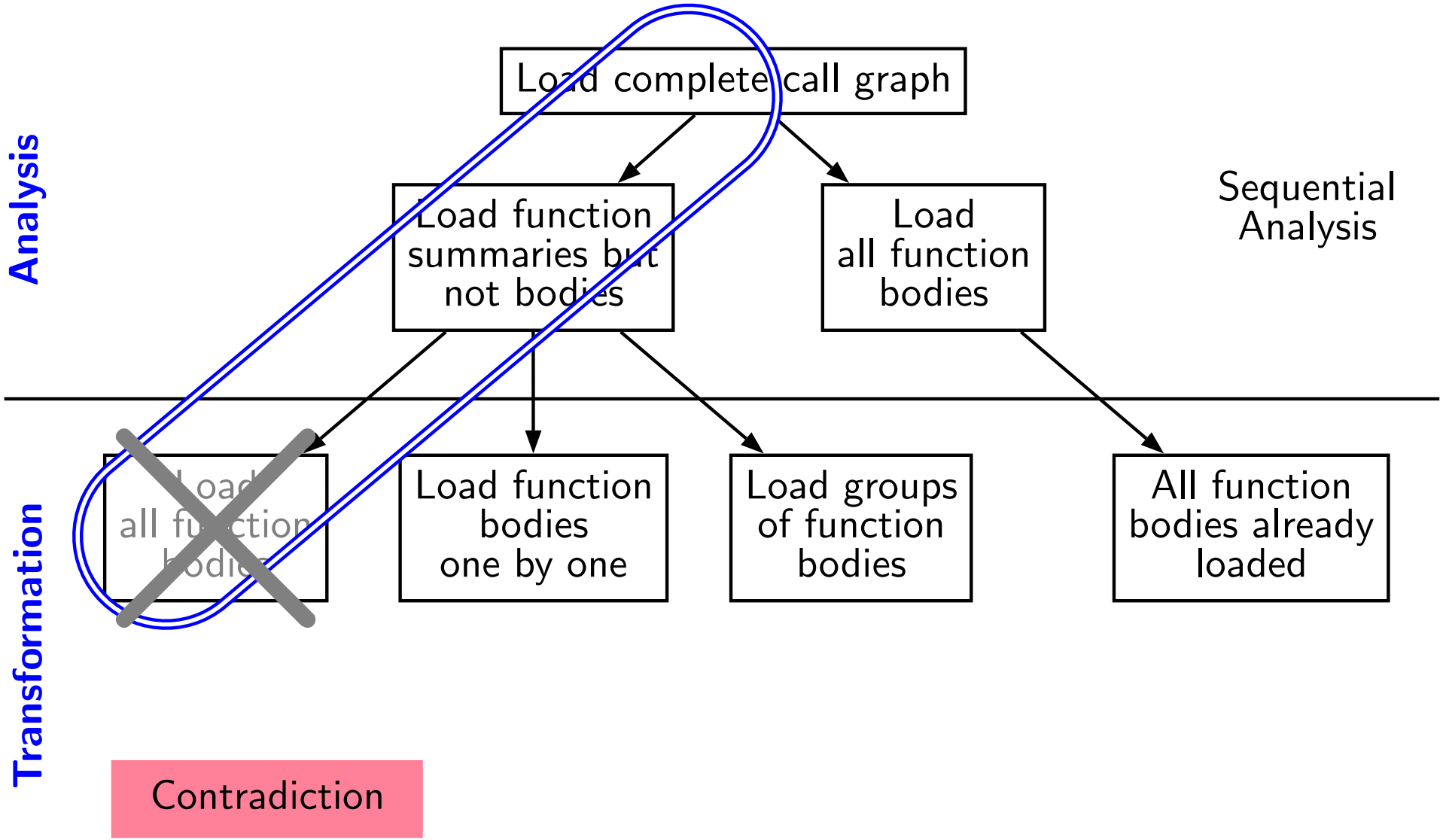
Partitioned and Non-Partitioned LTO



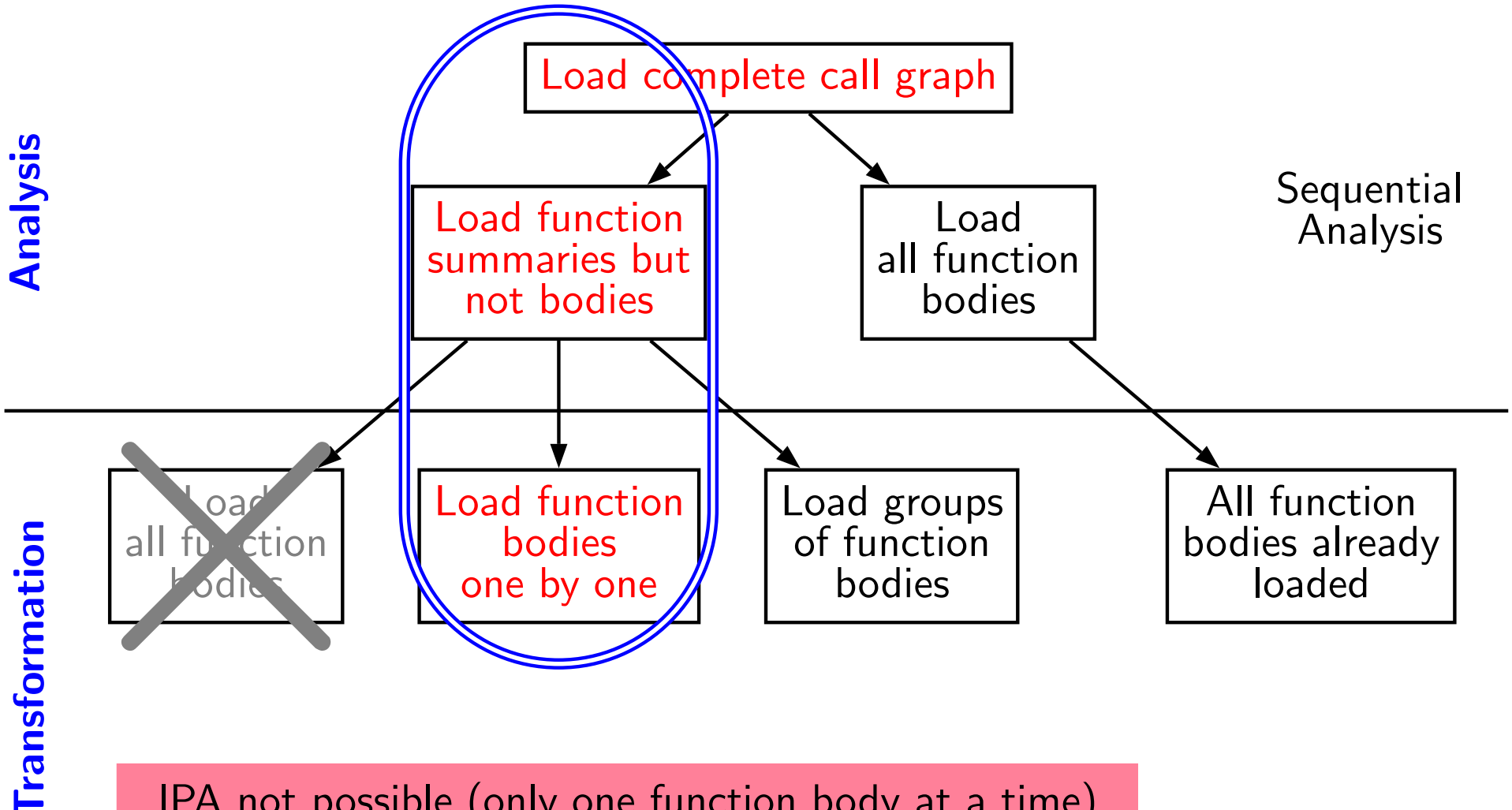
Partitioned and Non-Partitioned LTO



Partitioned and Non-Partitioned LTO

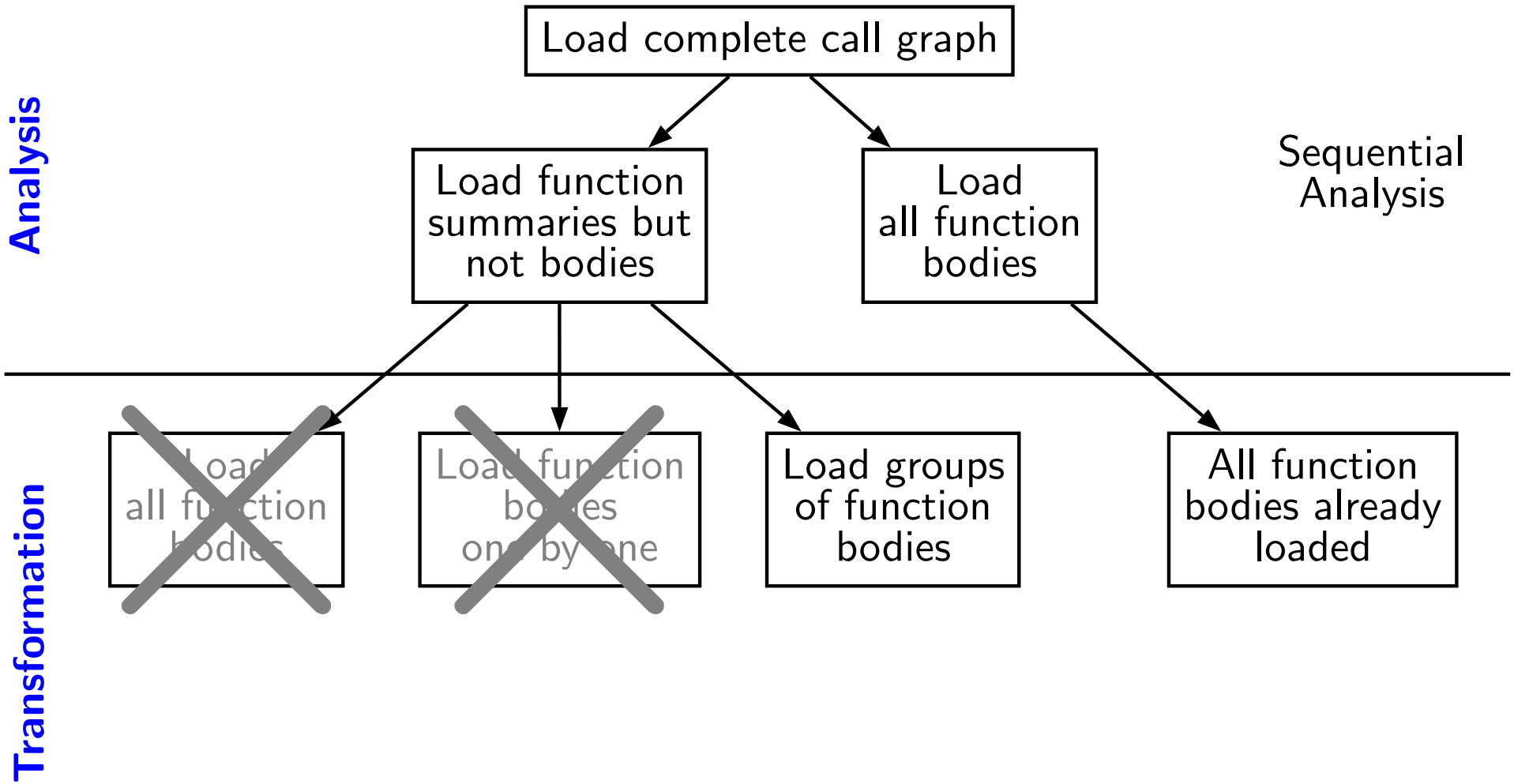


Partitioned and Non-Partitioned LTO

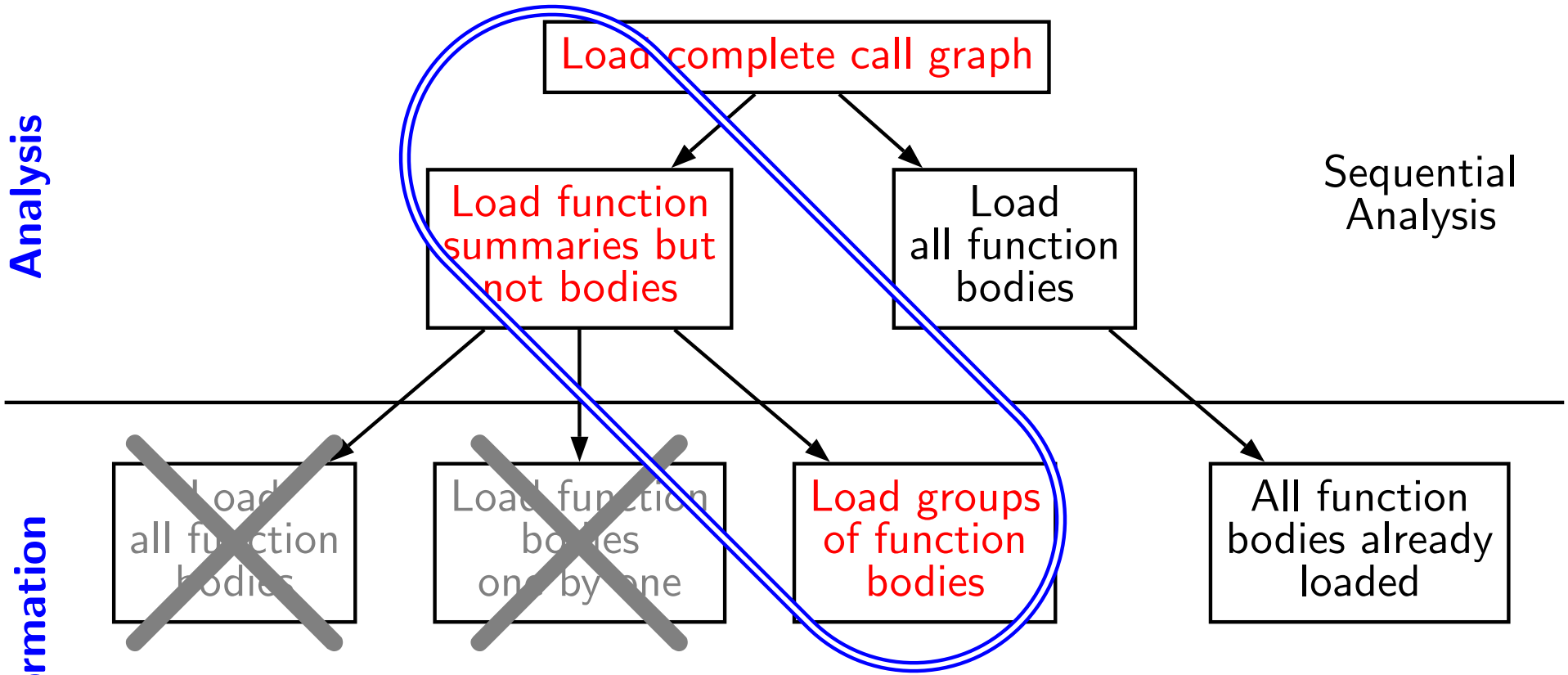


IPA not possible (only one function body at a time)
Strictly sequential transformations

Partitioned and Non-Partitioned LTO

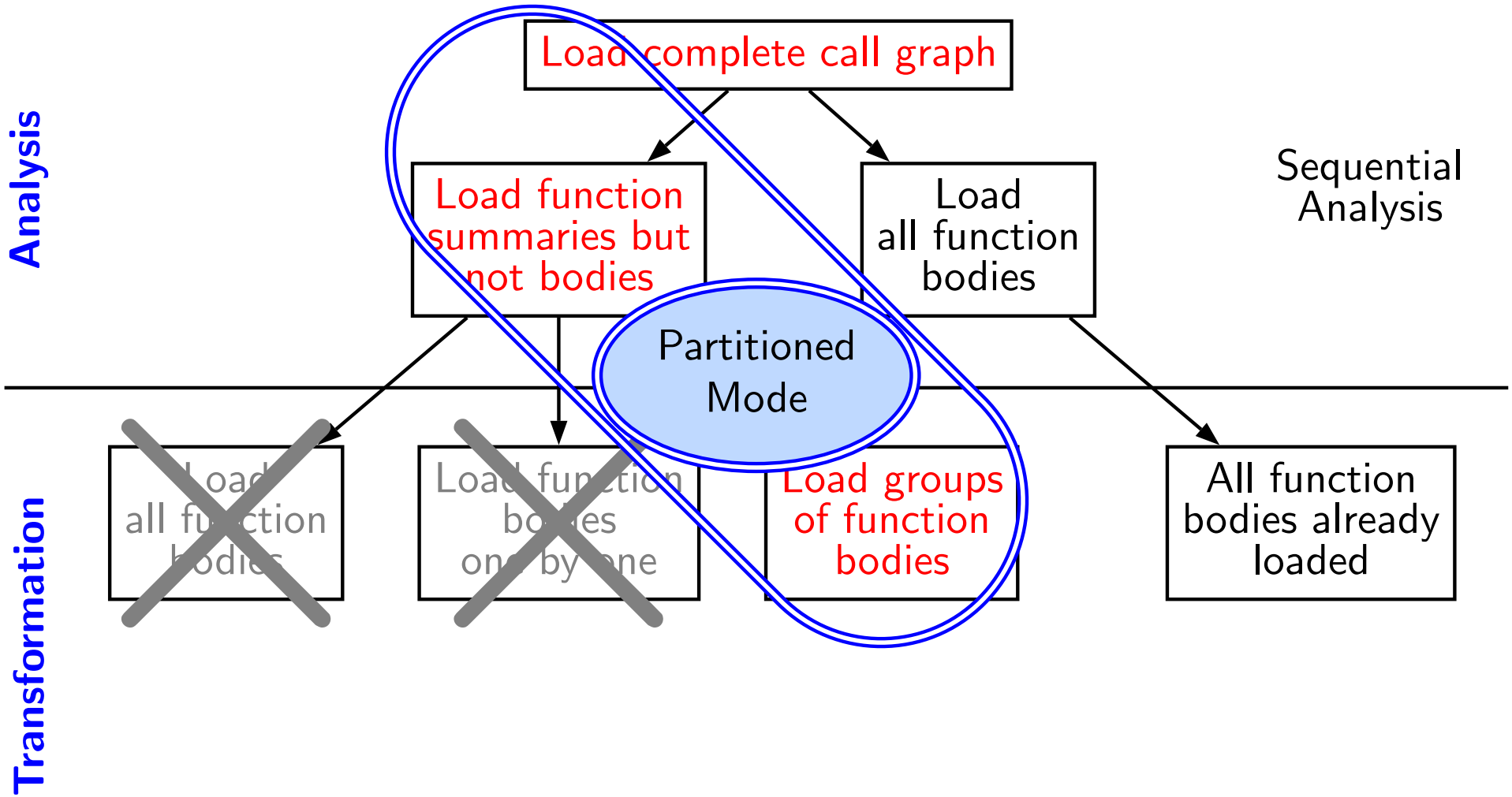


Partitioned and Non-Partitioned LTO

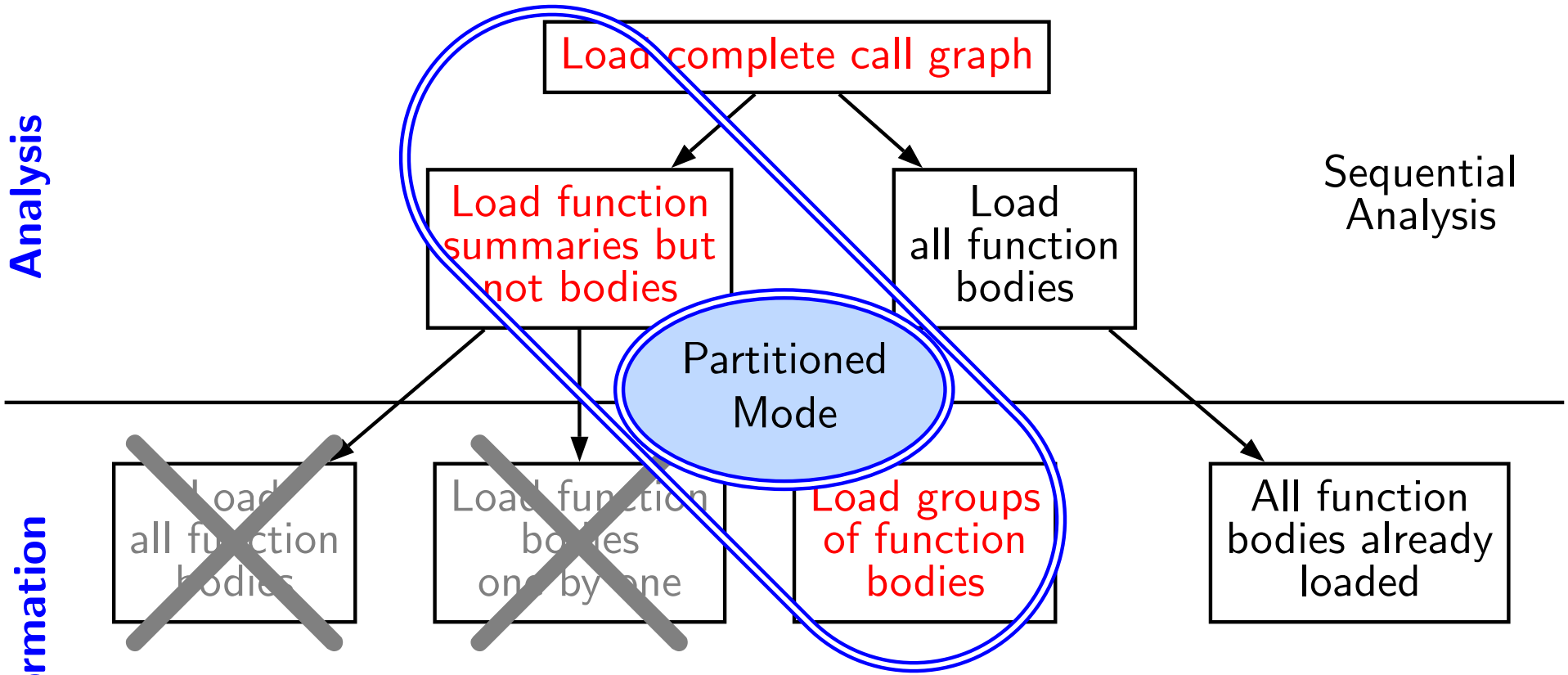


No need to load the entire program in memory
IPA possible (multiple function bodies)
Parallel transformations possible
Analysis and transformations in independent processes

Partitioned and Non-Partitioned LTO

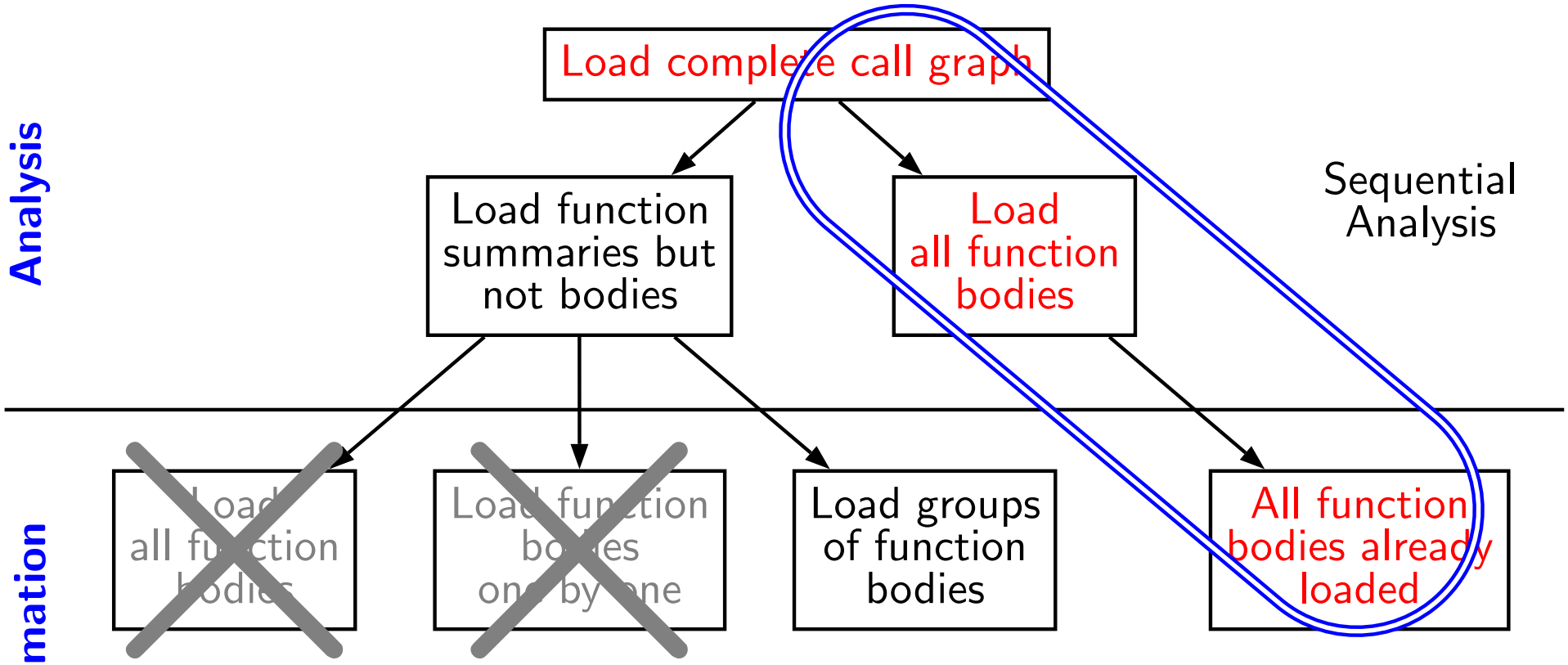


Partitioned and Non-Partitioned LTO



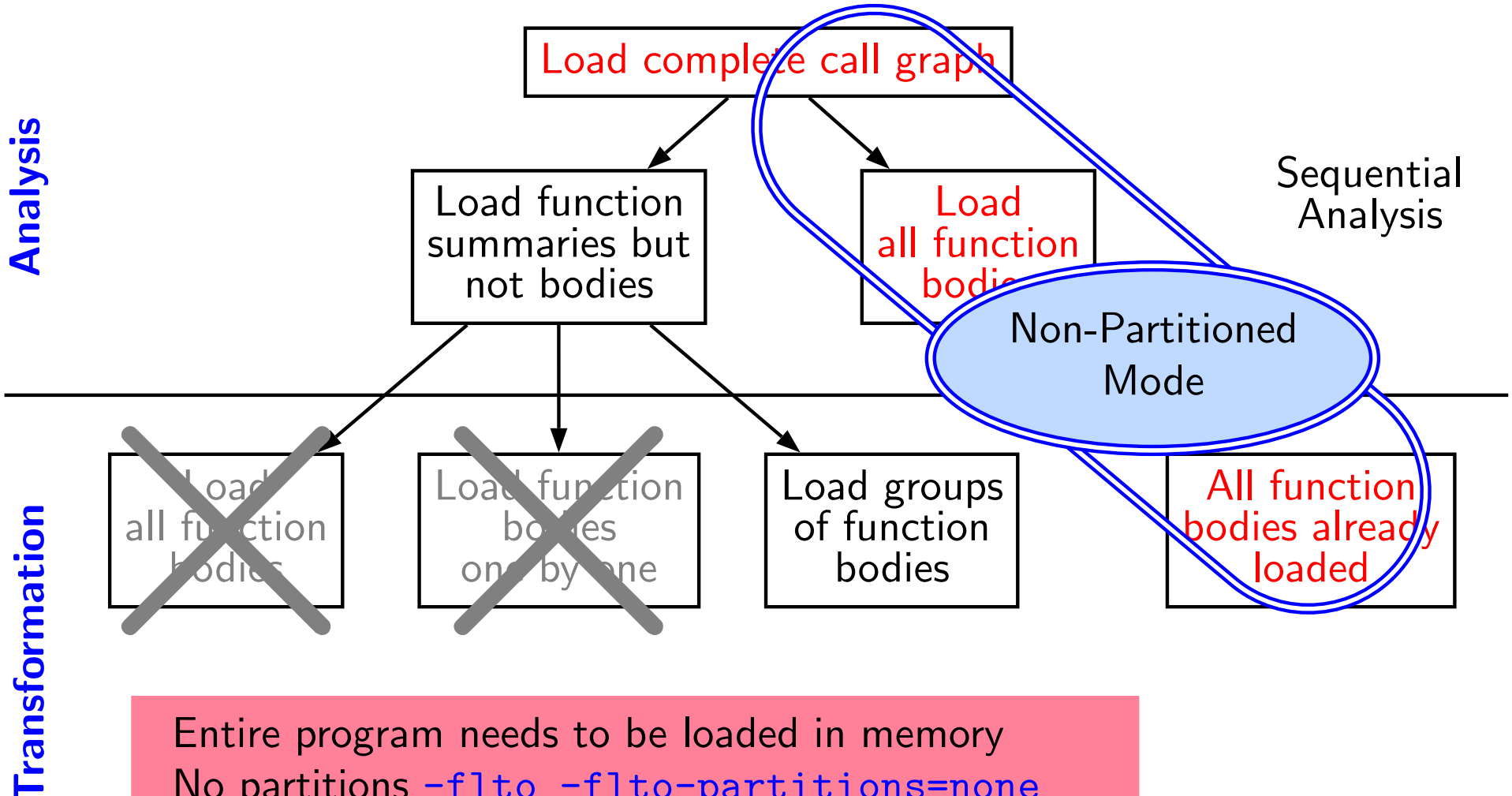
Balanced partitions `-flto -flto-partitions=balanced`
One Partition per file `-flto -flto-partitions=1to1`
Partitions by number `-flto --params lto-partitions=n`
Partitions by size `-flto --params lto-min-partition=s`

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)

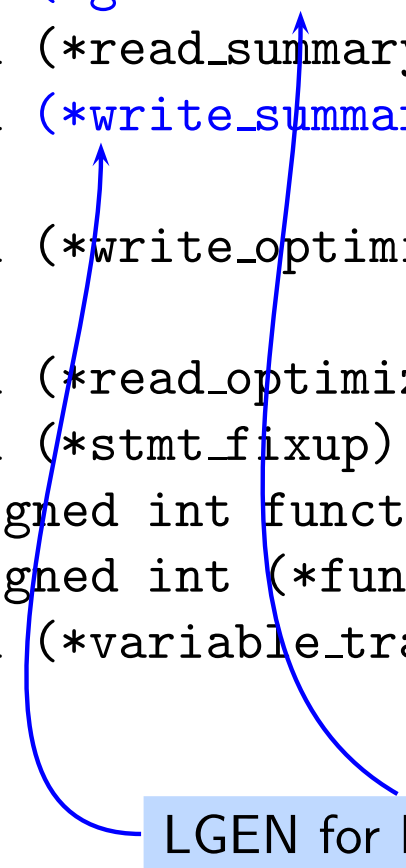
- Processing sequence
  - `gcc` executes LGEN
  - Subsequent process of `lto1` executes IPA

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

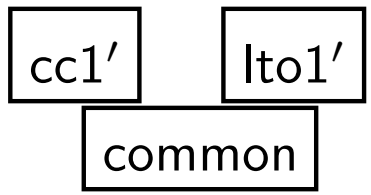
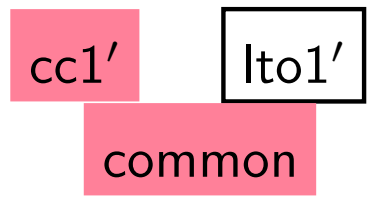
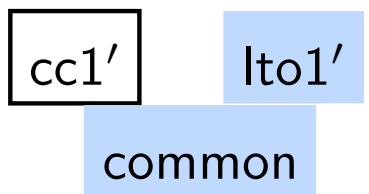
```
toplev_main
...
  compile_file
  ...
    cgraph_analyze_function
```

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

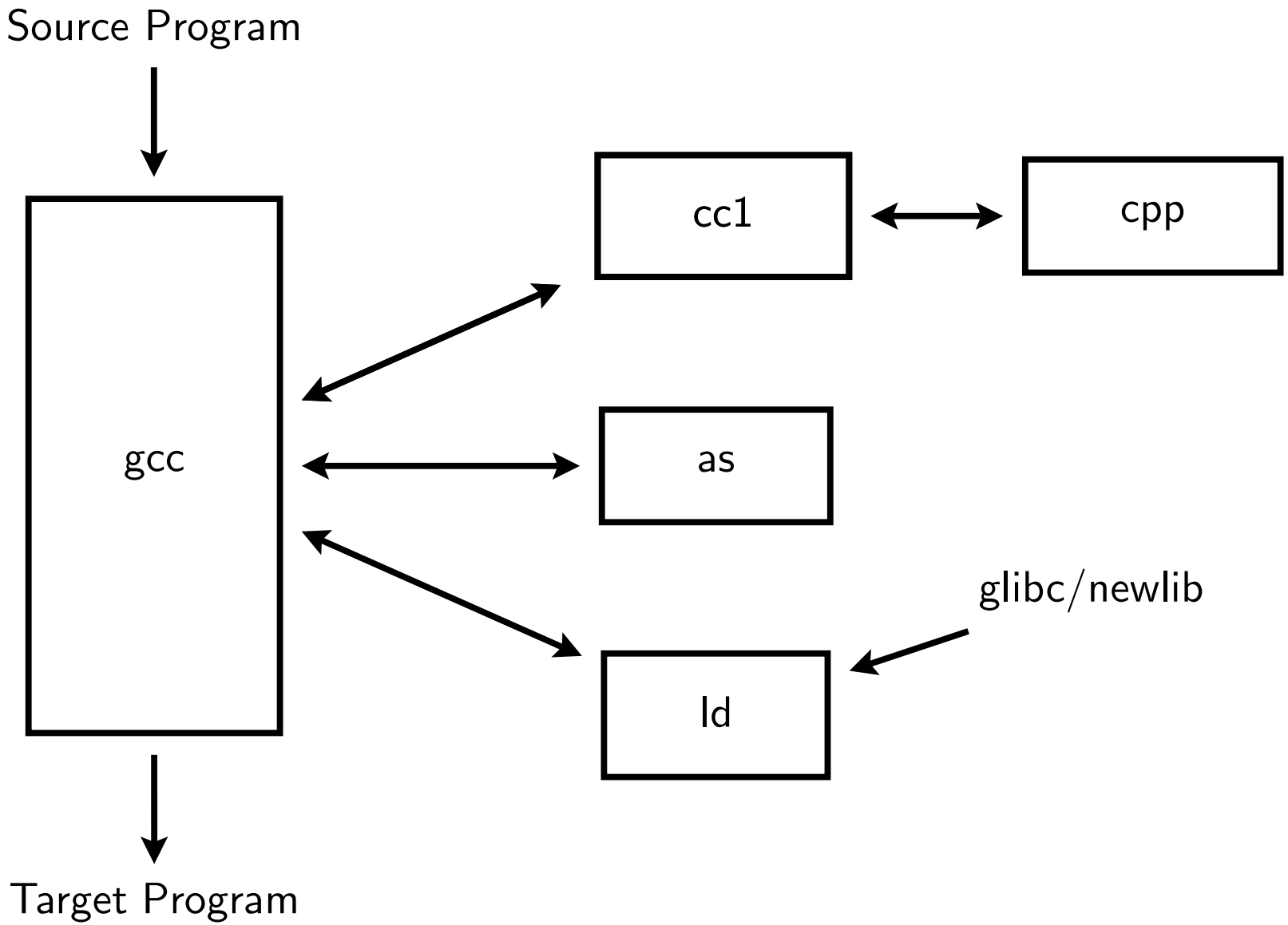
```
cgraph_optimize
...
  ipa_passes
  ...
  cgraph_expand_all_functions
    ...
    tree_rest_of_compilation
```

lto1

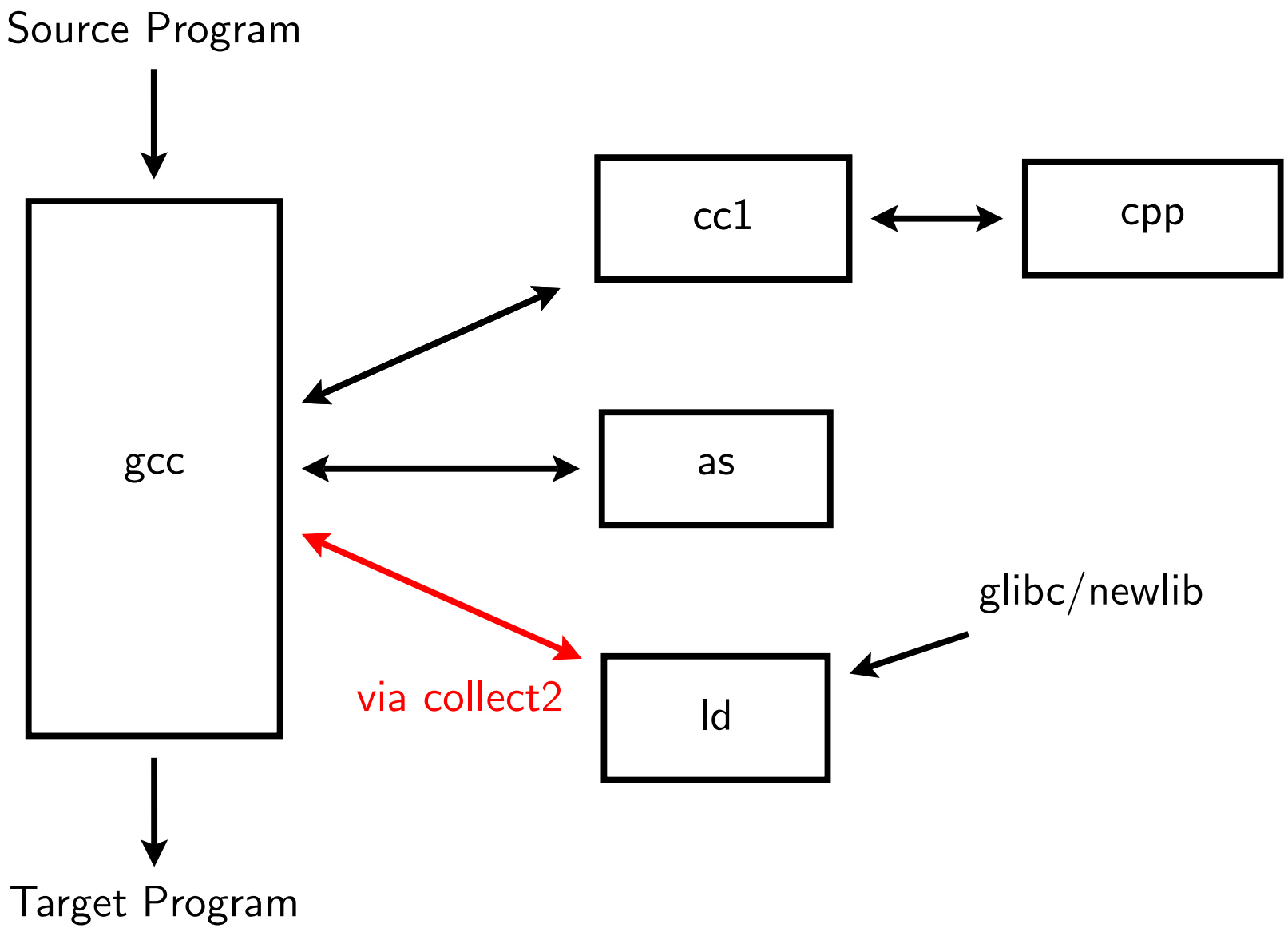
Our Pictorial Convention

Source code	
cc1 executable	
lto1 executable	

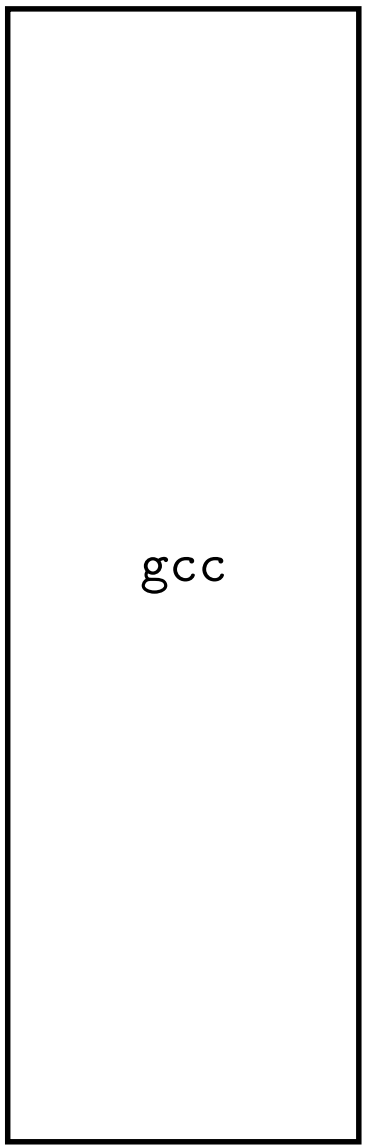
The GNU Tool Chain: Our First Picture



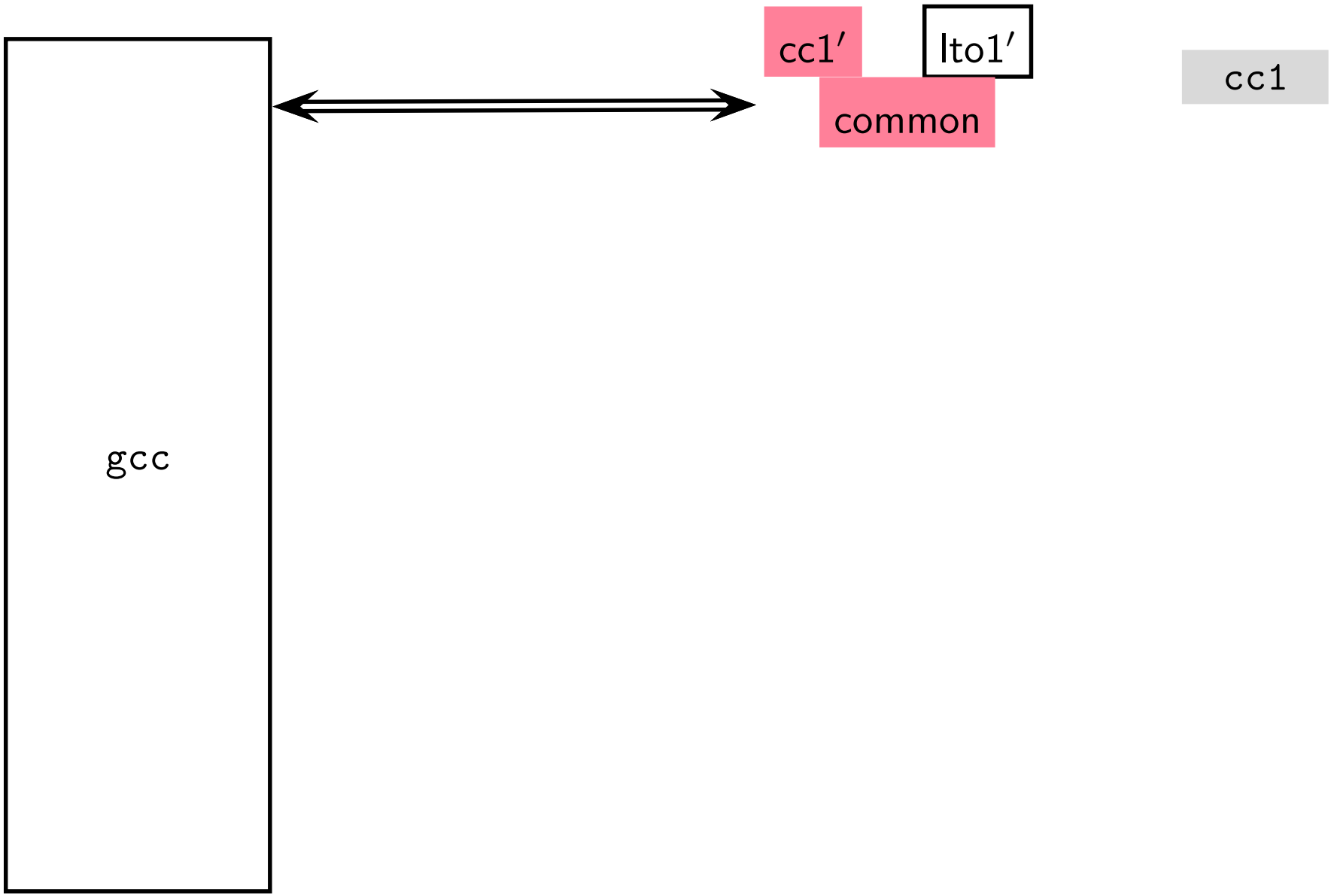
The GNU Tool Chain: Our First Picture



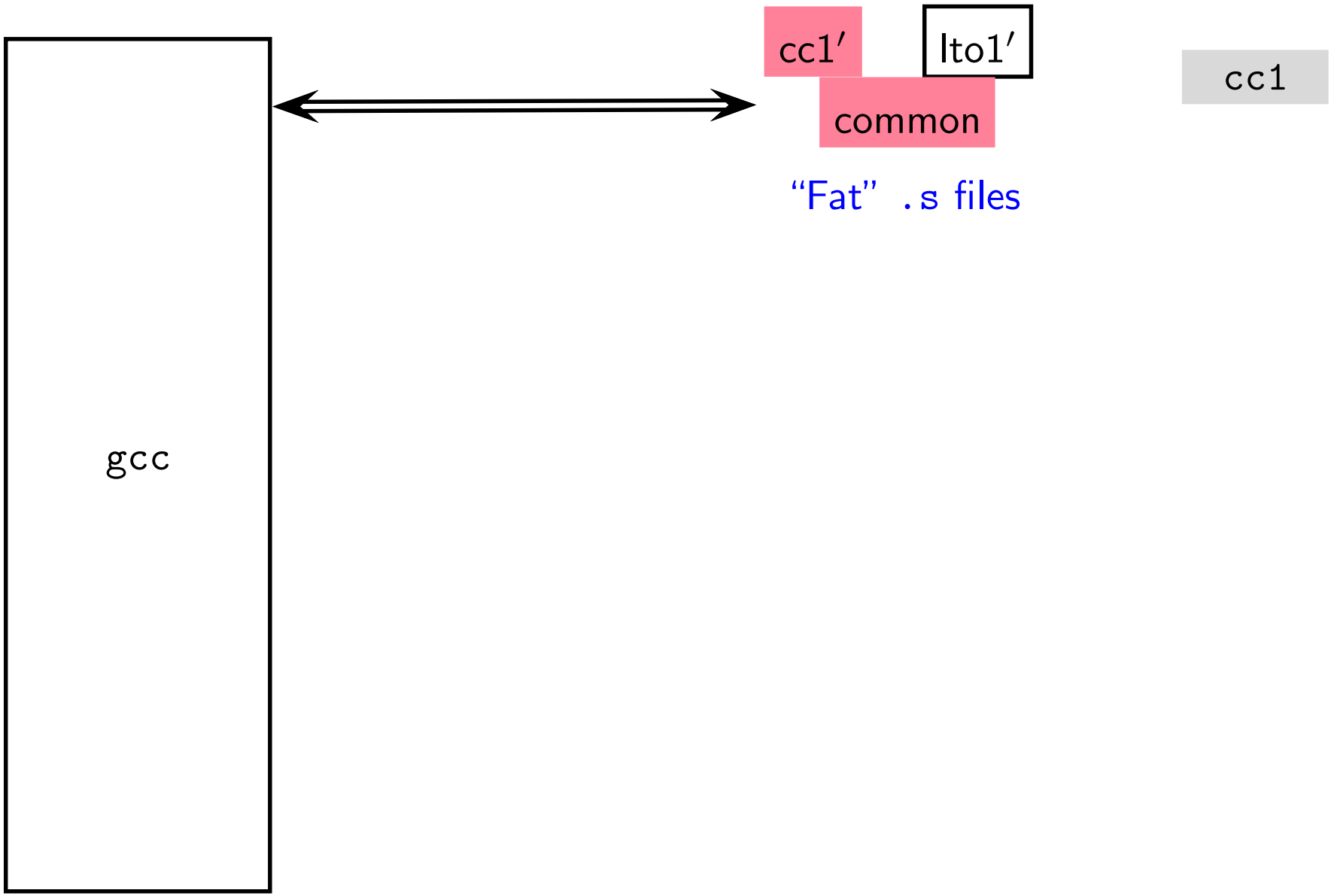
The GNU Tool Chain for Non-Partitioned LTO Support



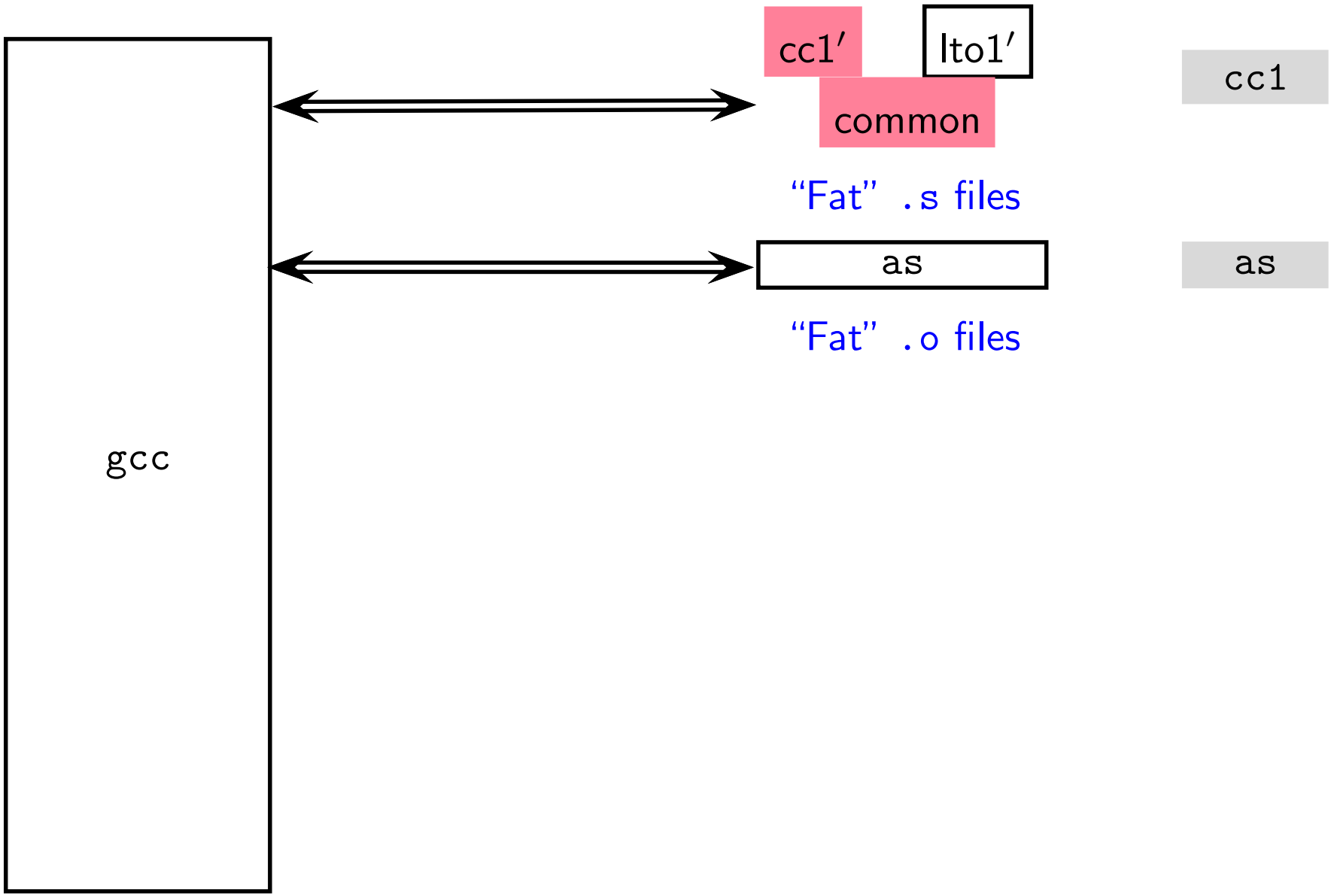
The GNU Tool Chain for Non-Partitioned LTO Support



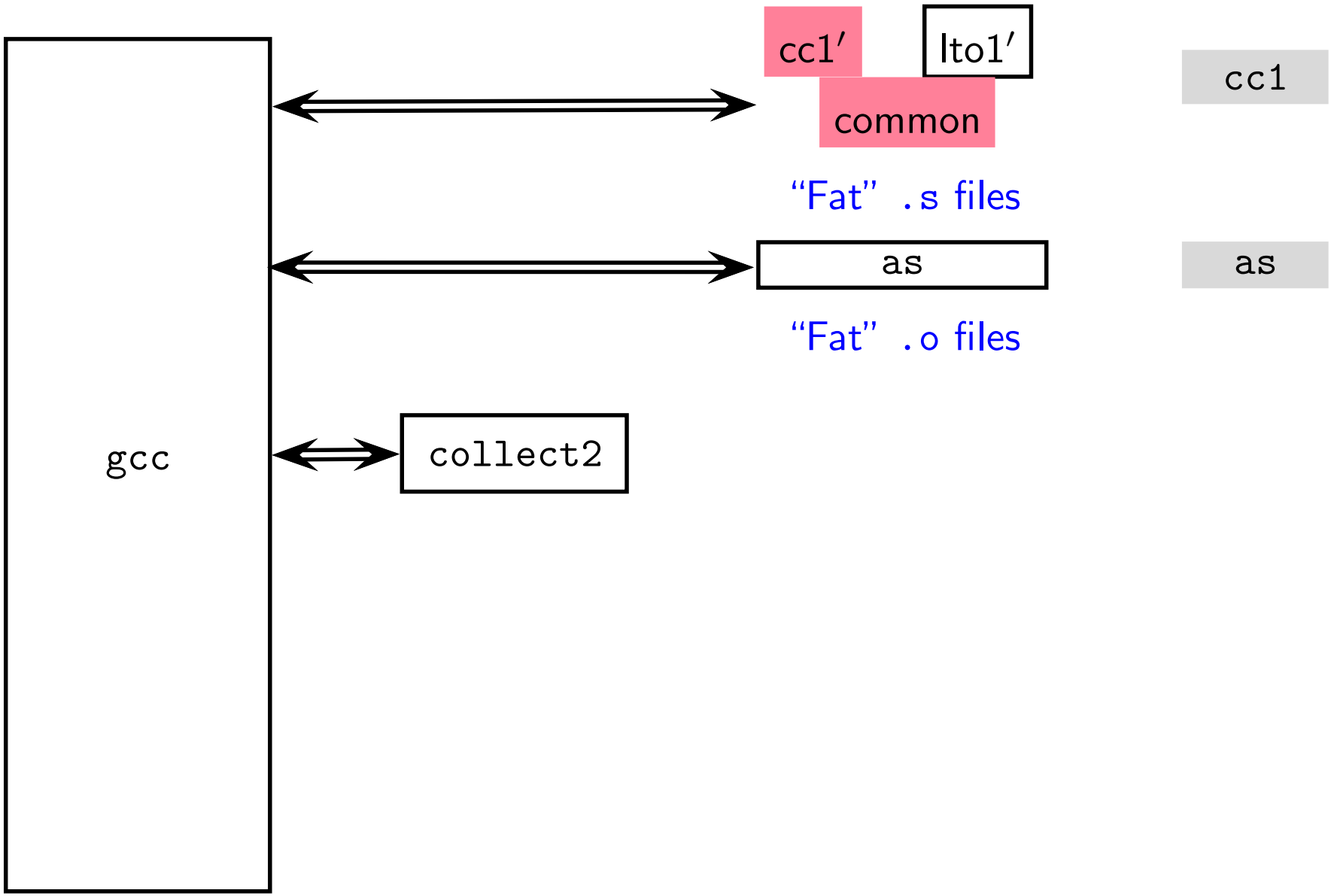
The GNU Tool Chain for Non-Partitioned LTO Support



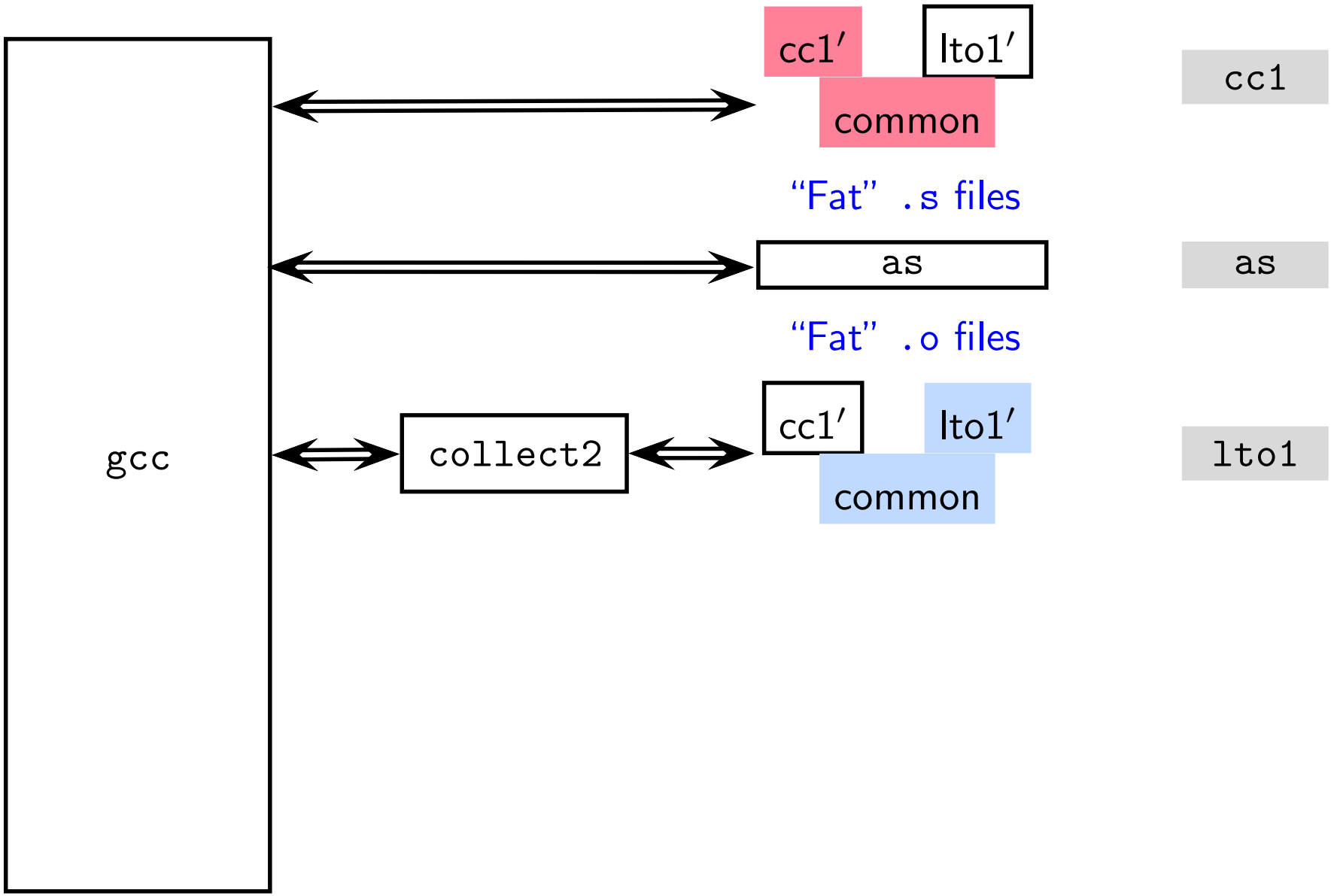
The GNU Tool Chain for Non-Partitioned LTO Support



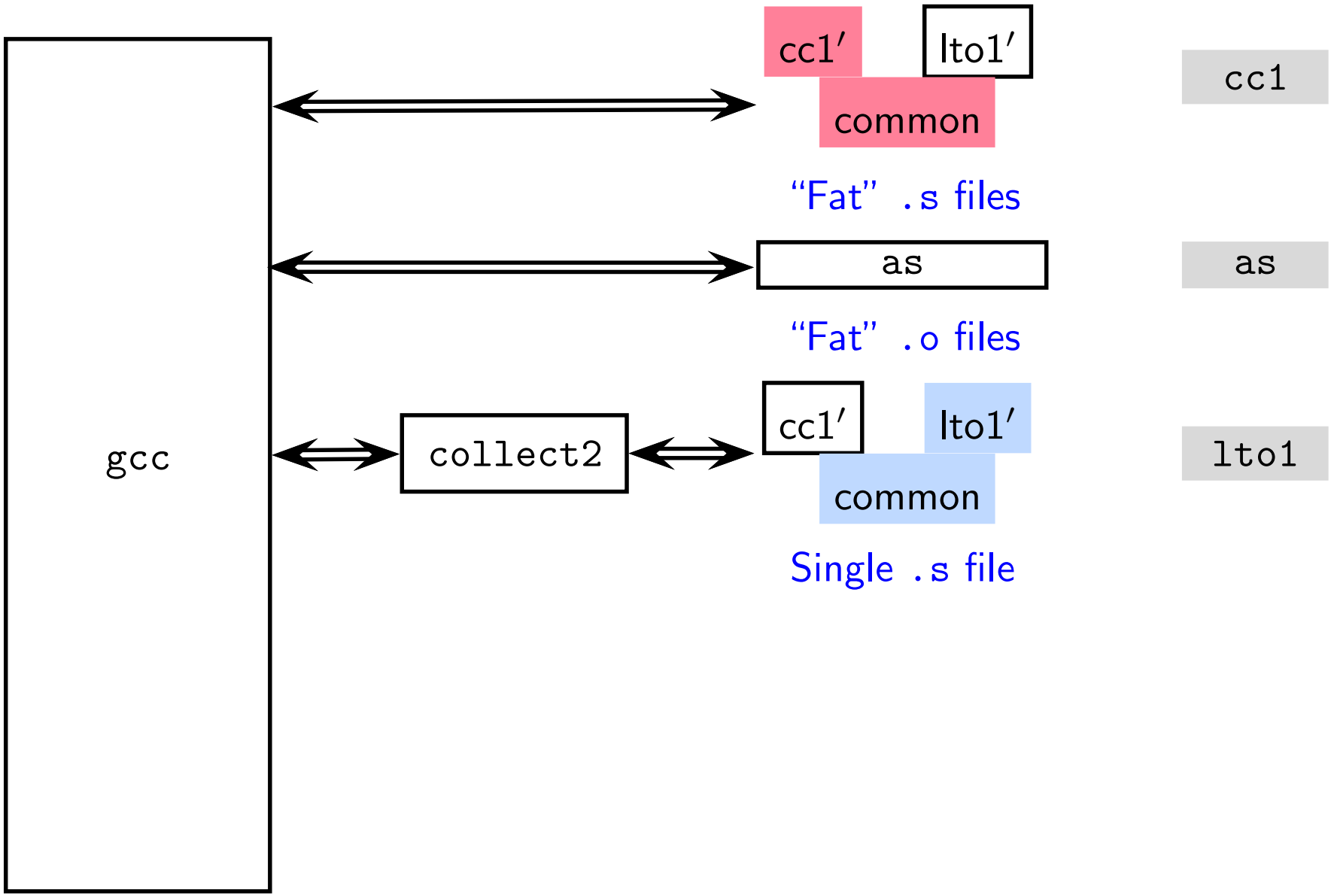
The GNU Tool Chain for Non-Partitioned LTO Support



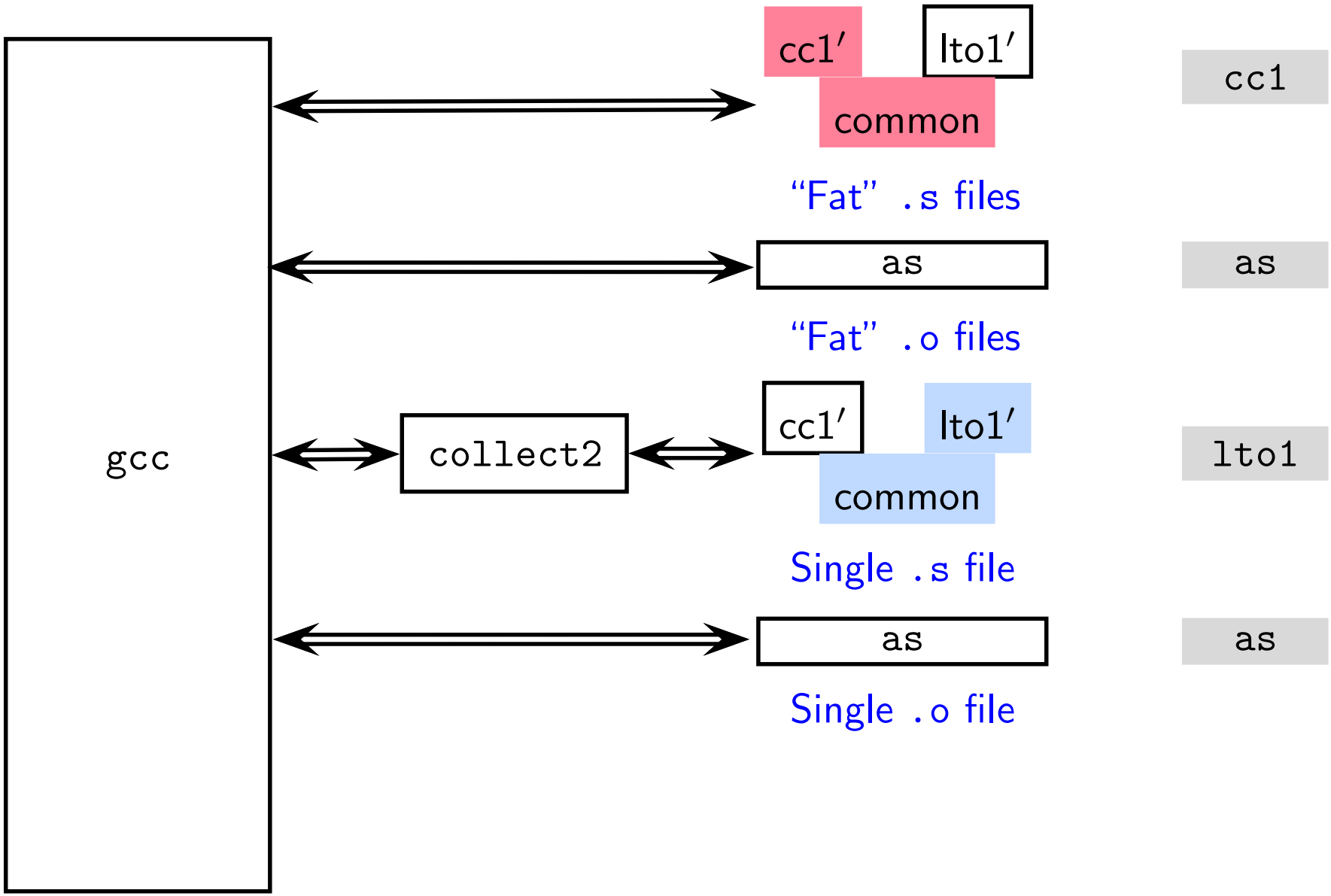
The GNU Tool Chain for Non-Partitioned LTO Support



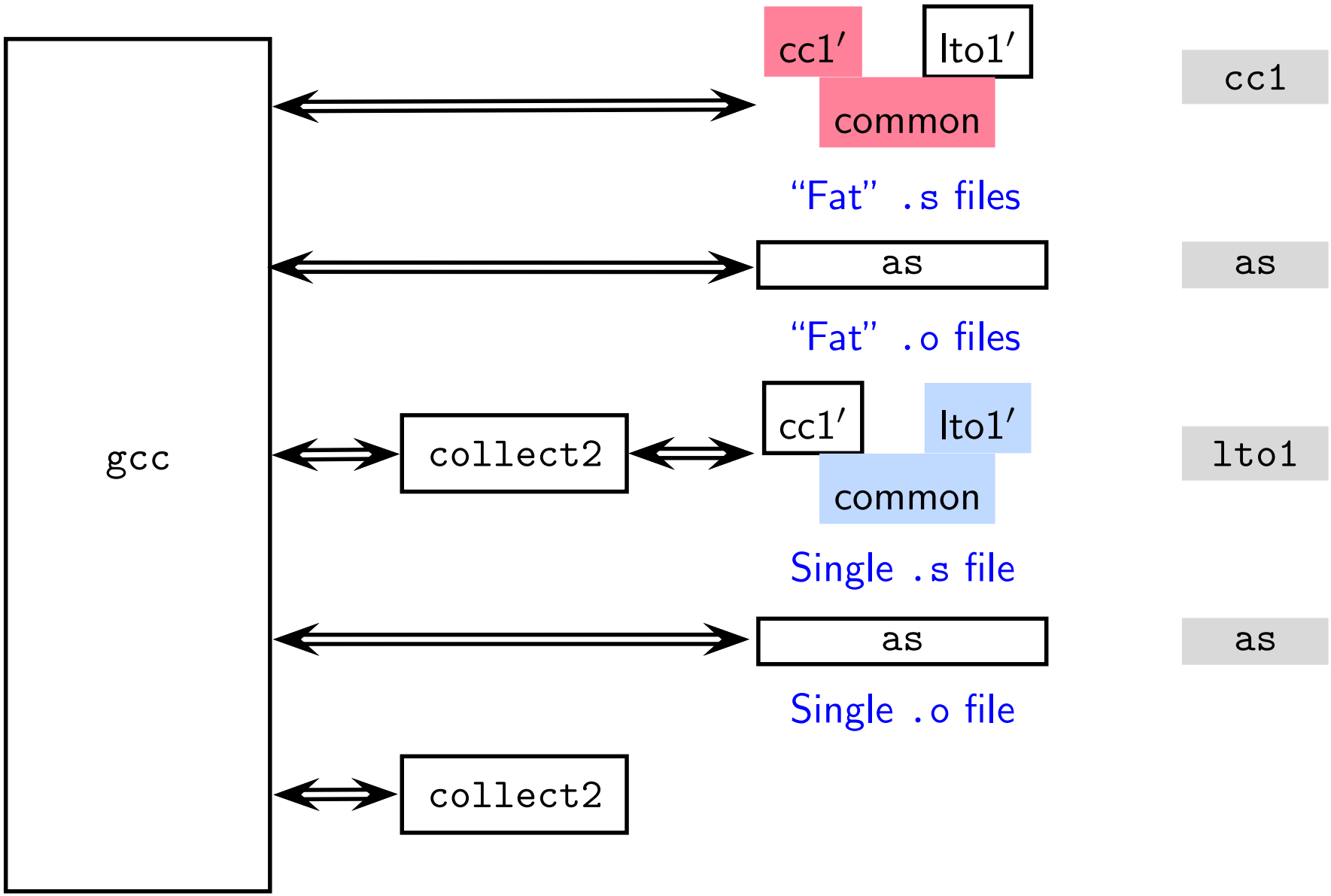
The GNU Tool Chain for Non-Partitioned LTO Support



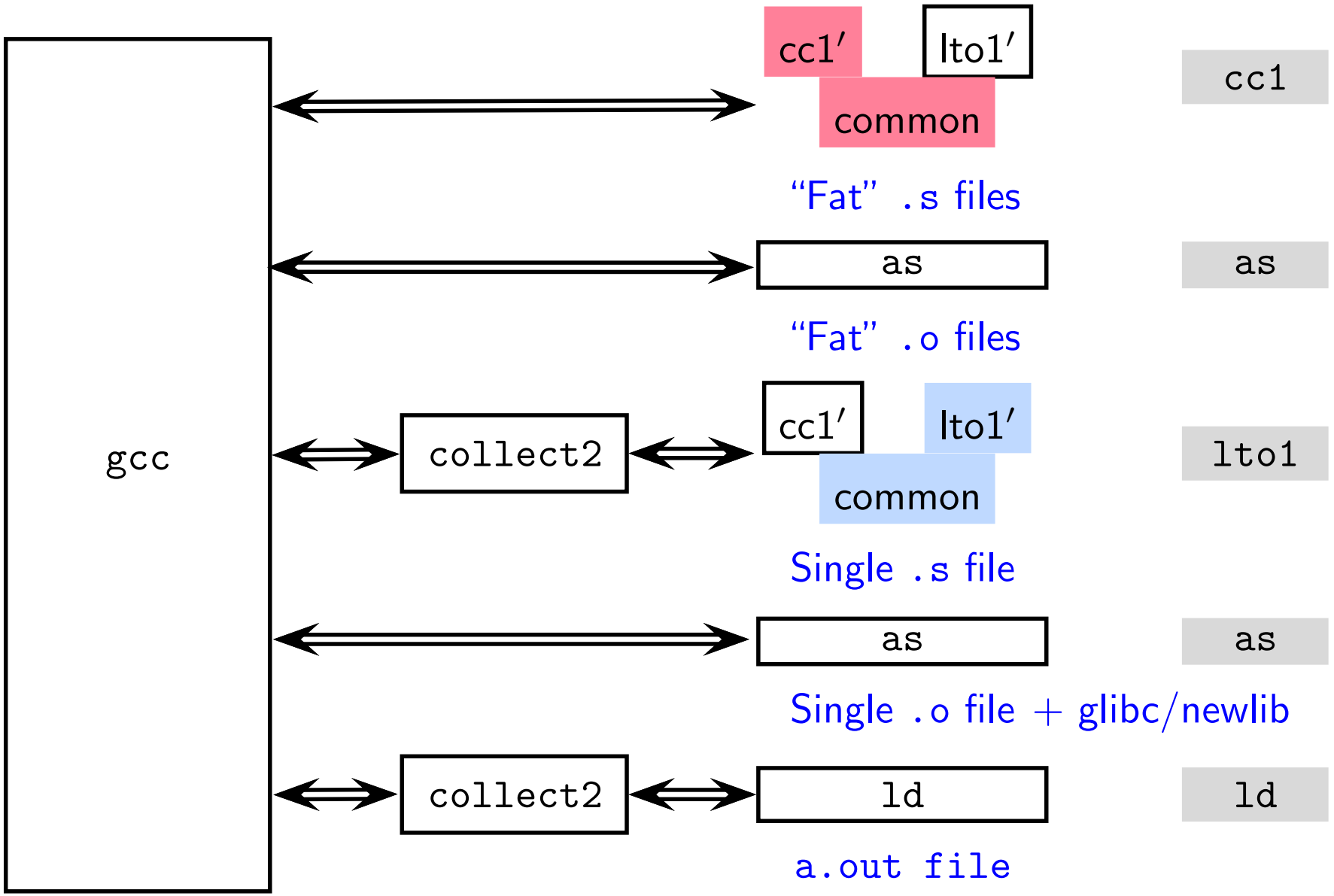
The GNU Tool Chain for Non-Partitioned LTO Support



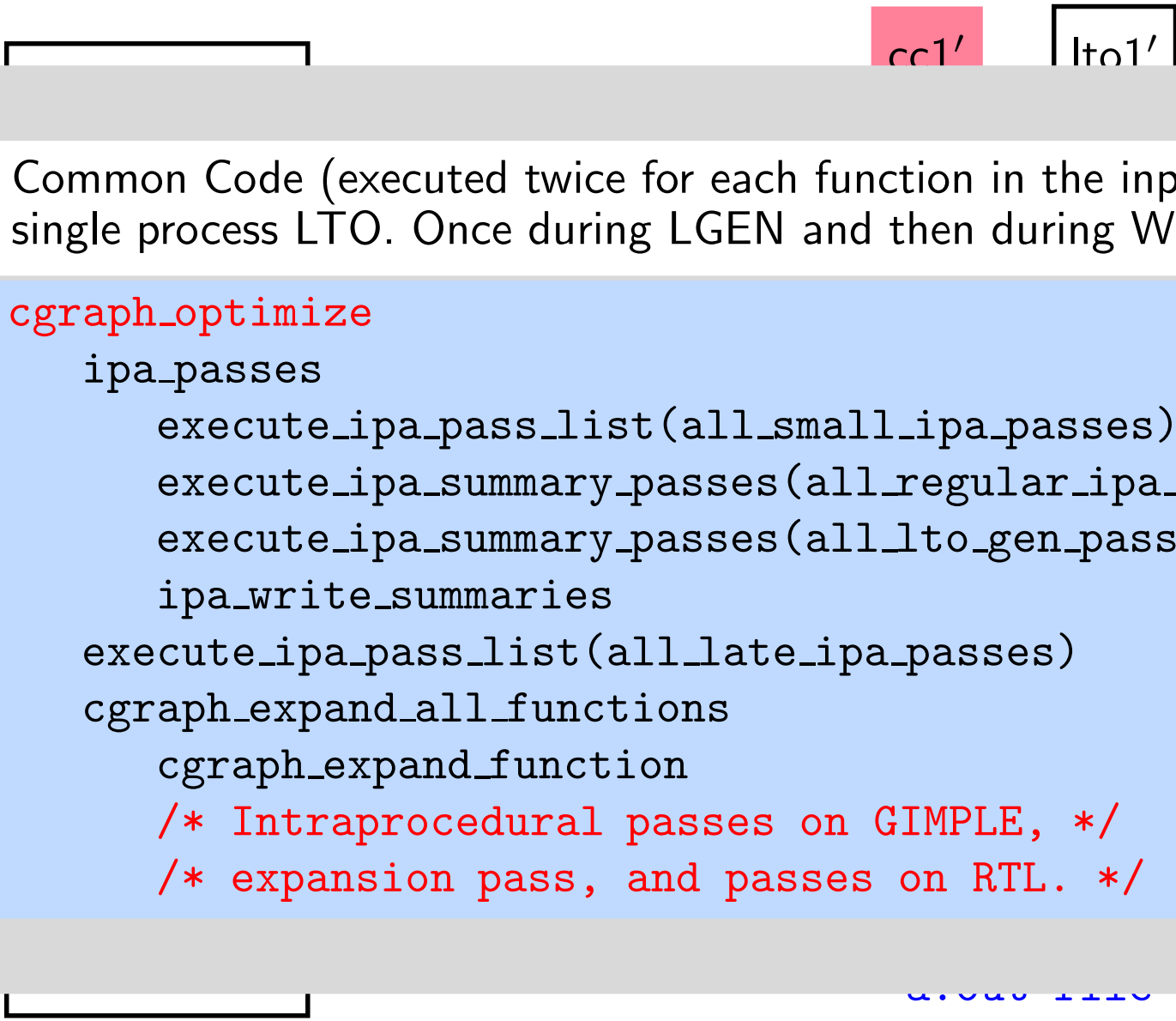
The GNU Tool Chain for Non-Partitioned LTO Support



The GNU Tool Chain for Non-Partitioned LTO Support



The GNU Tool Chain for Non-Partitioned LTO Support

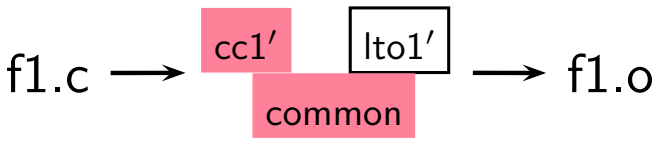


Common Code (executed twice for each function in the input program for single process LTO. Once during LGEN and then during WPA + LTRANS)

```
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. */
```

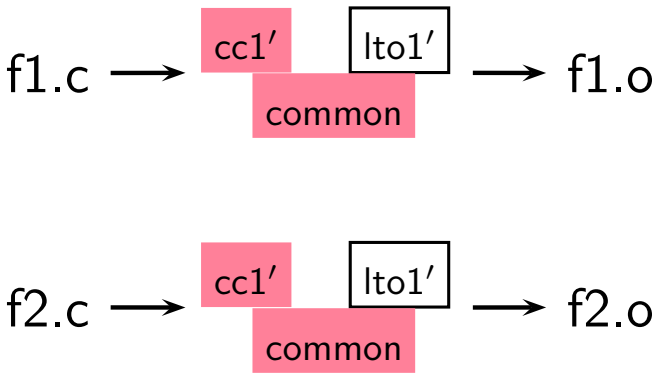
Partitioned LTO (aka WHOPR LTO)

Option `-flto -c`



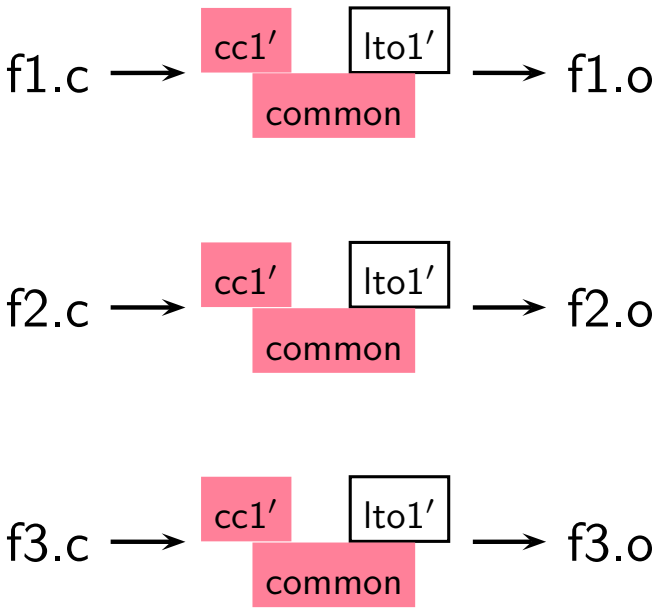
Partitioned LTO (aka WHOPR LTO)

Option `-flto -c`

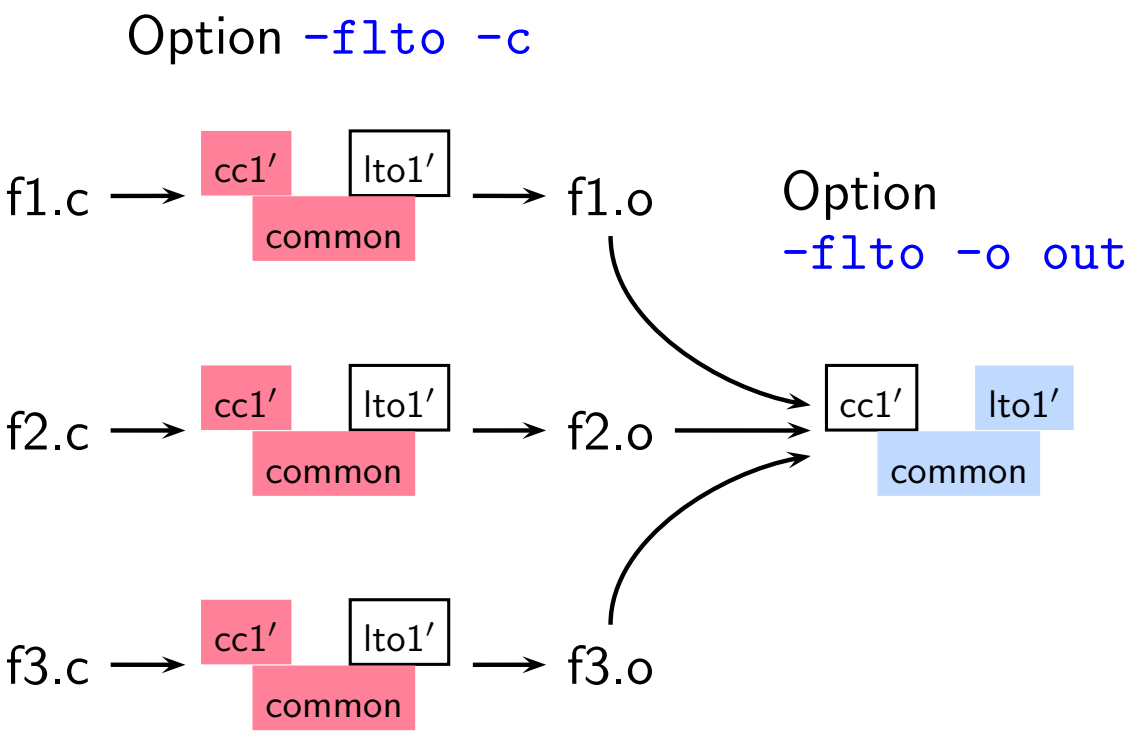


Partitioned LTO (aka WHOPR LTO)

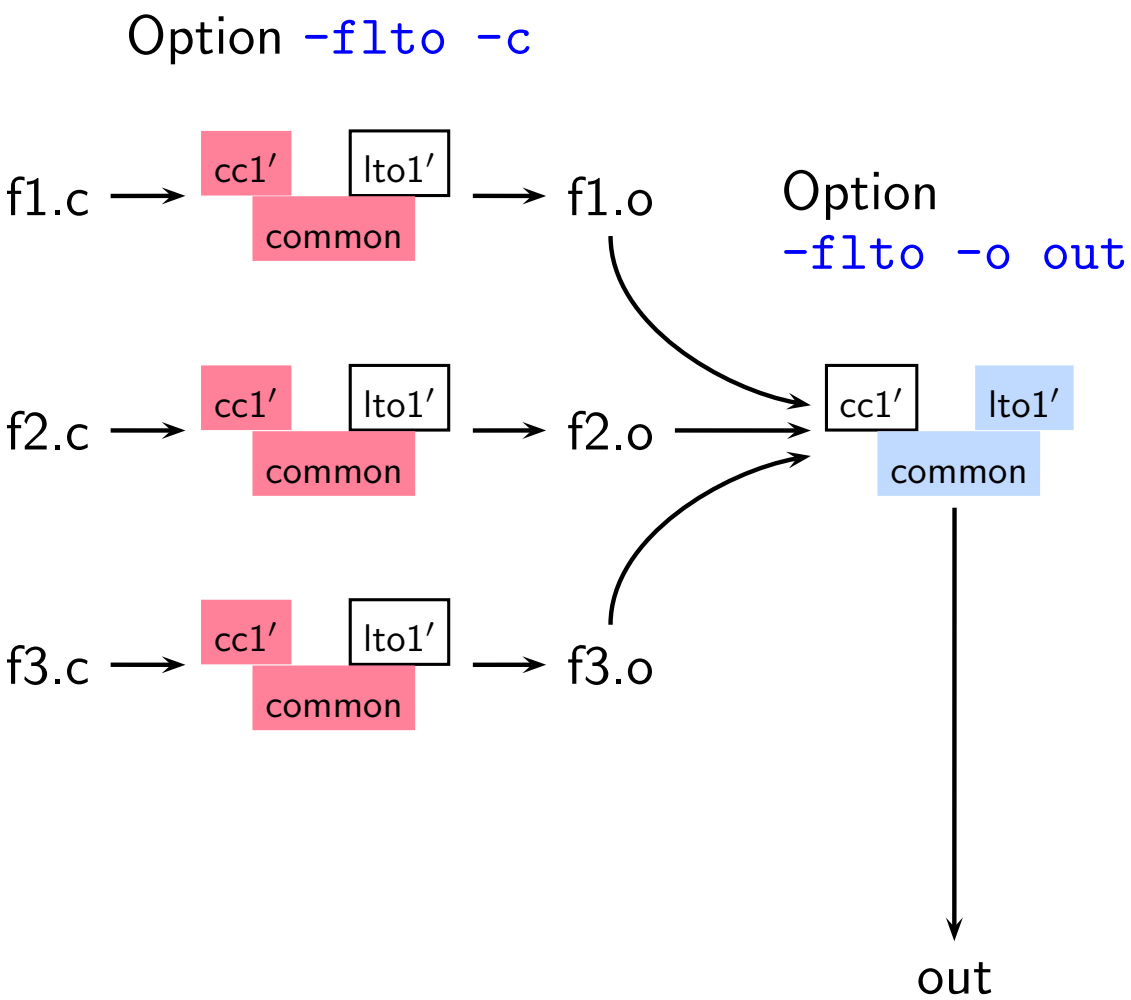
Option `-flto -c`



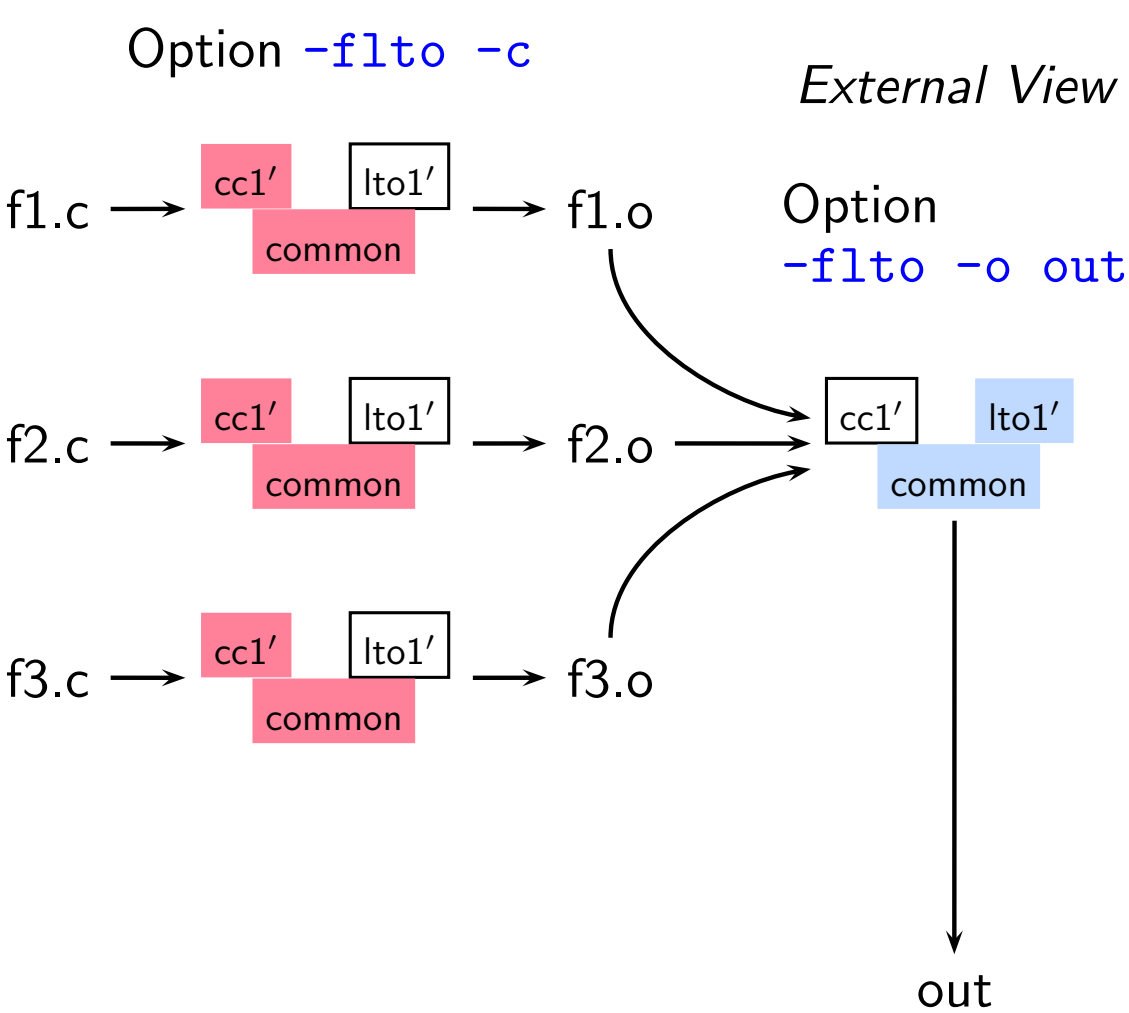
Partitioned LTO (aka WHOPR LTO)



Partitioned LTO (aka WHOPR LTO)



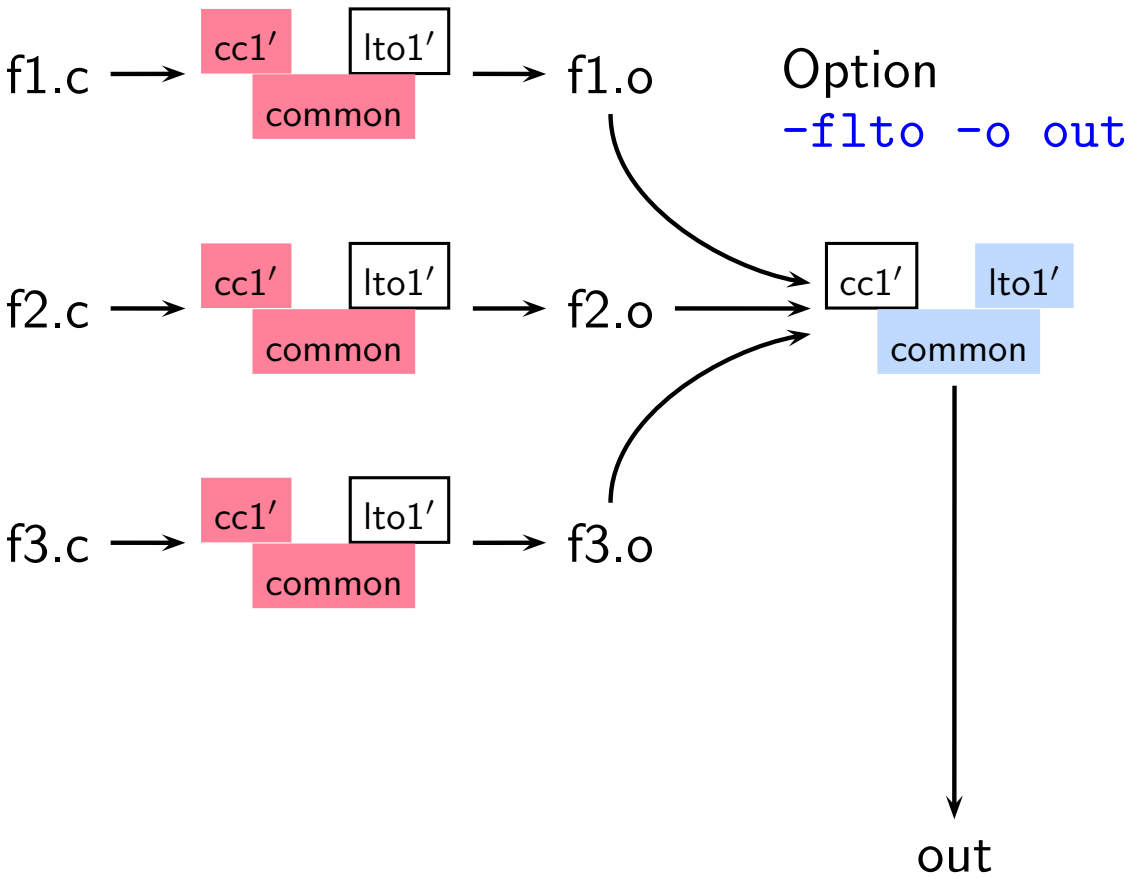
Partitioned LTO (aka WHOPR LTO)



Partitioned LTO (aka WHOPR LTO)

Option `-flto -c`

External View



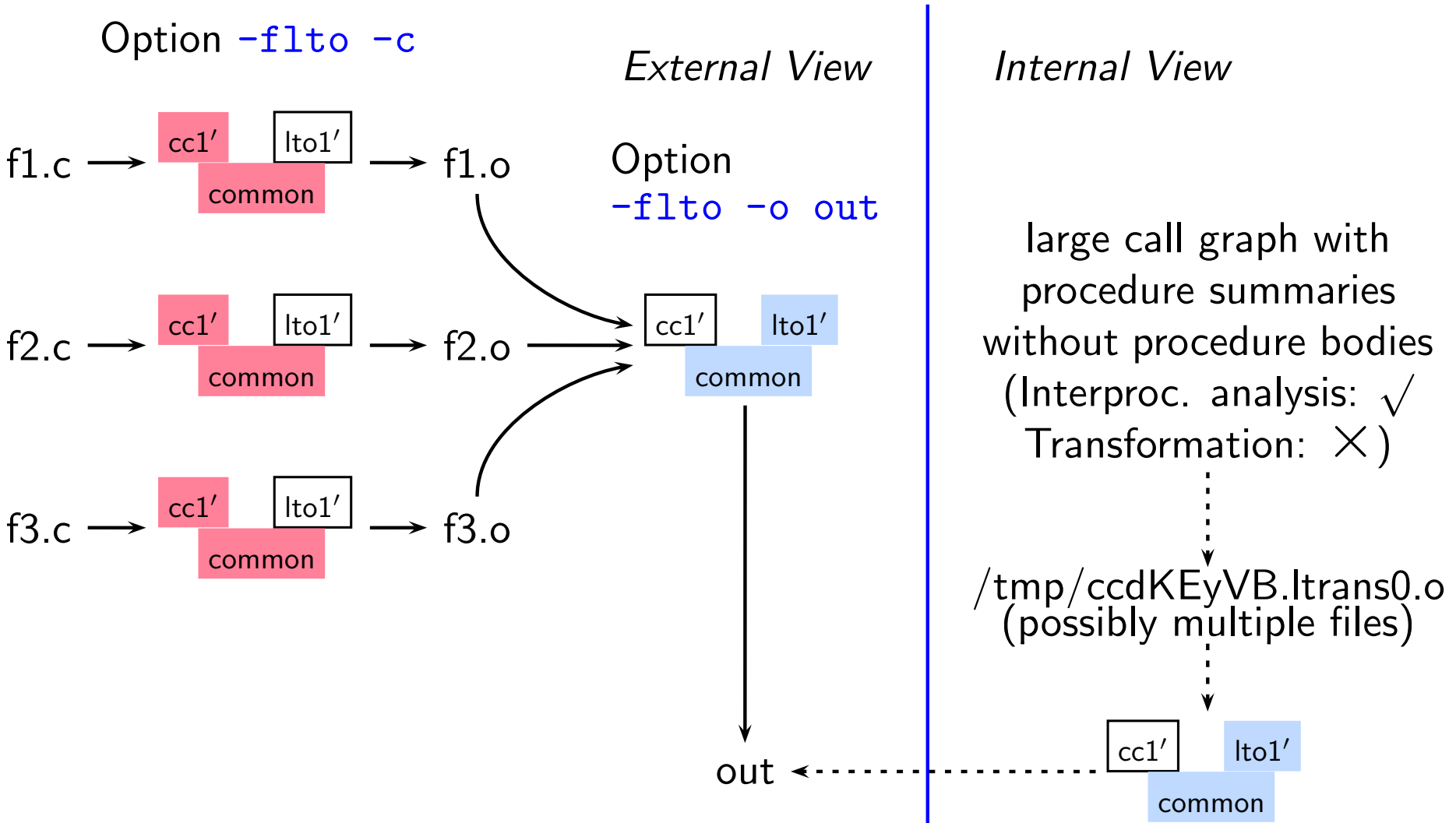
Option
`-flto -o out`

Internal View

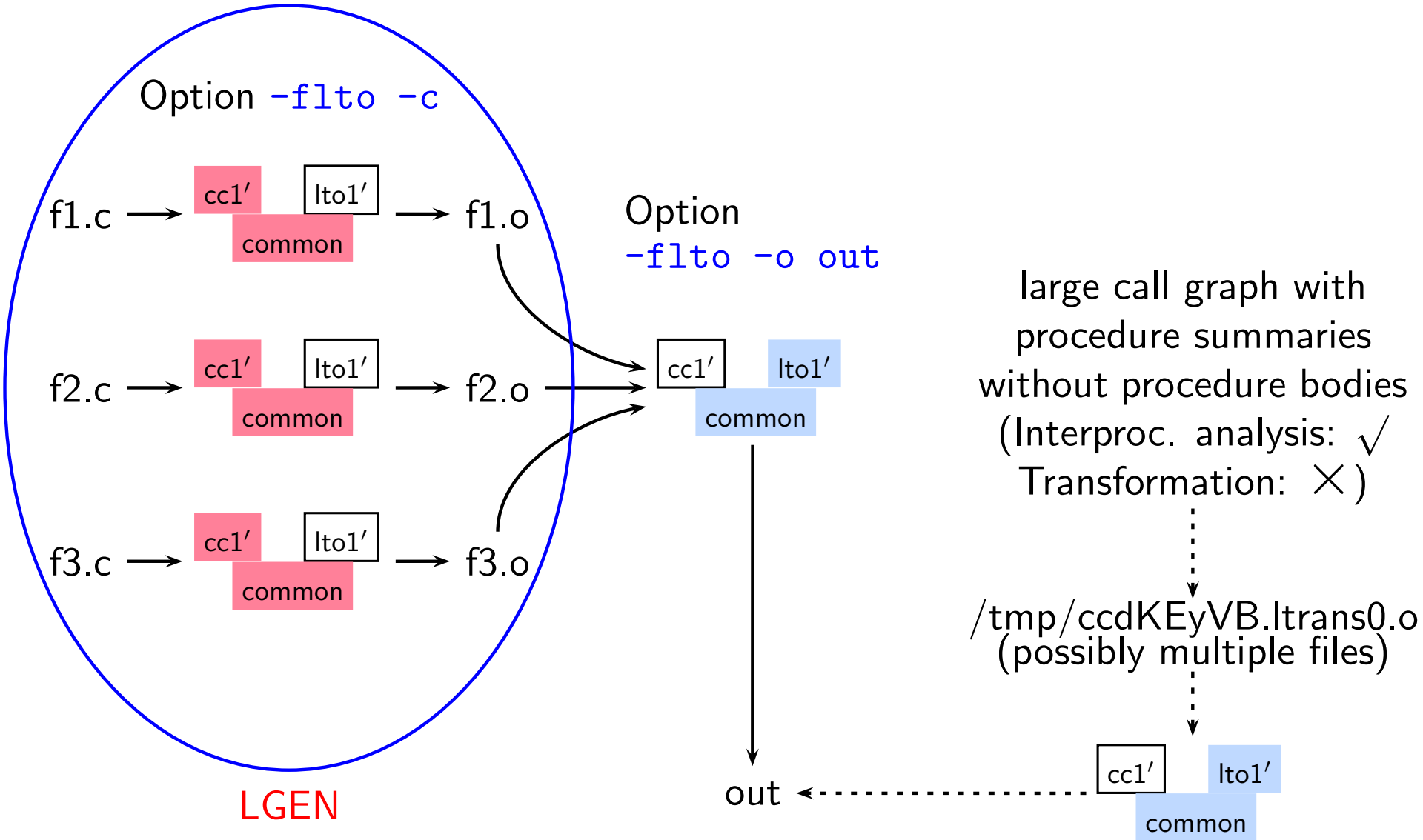
large call graph with
procedure summaries
without procedure bodies
(Interproc. analysis: $\surd$
Transformation: $\times$)

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

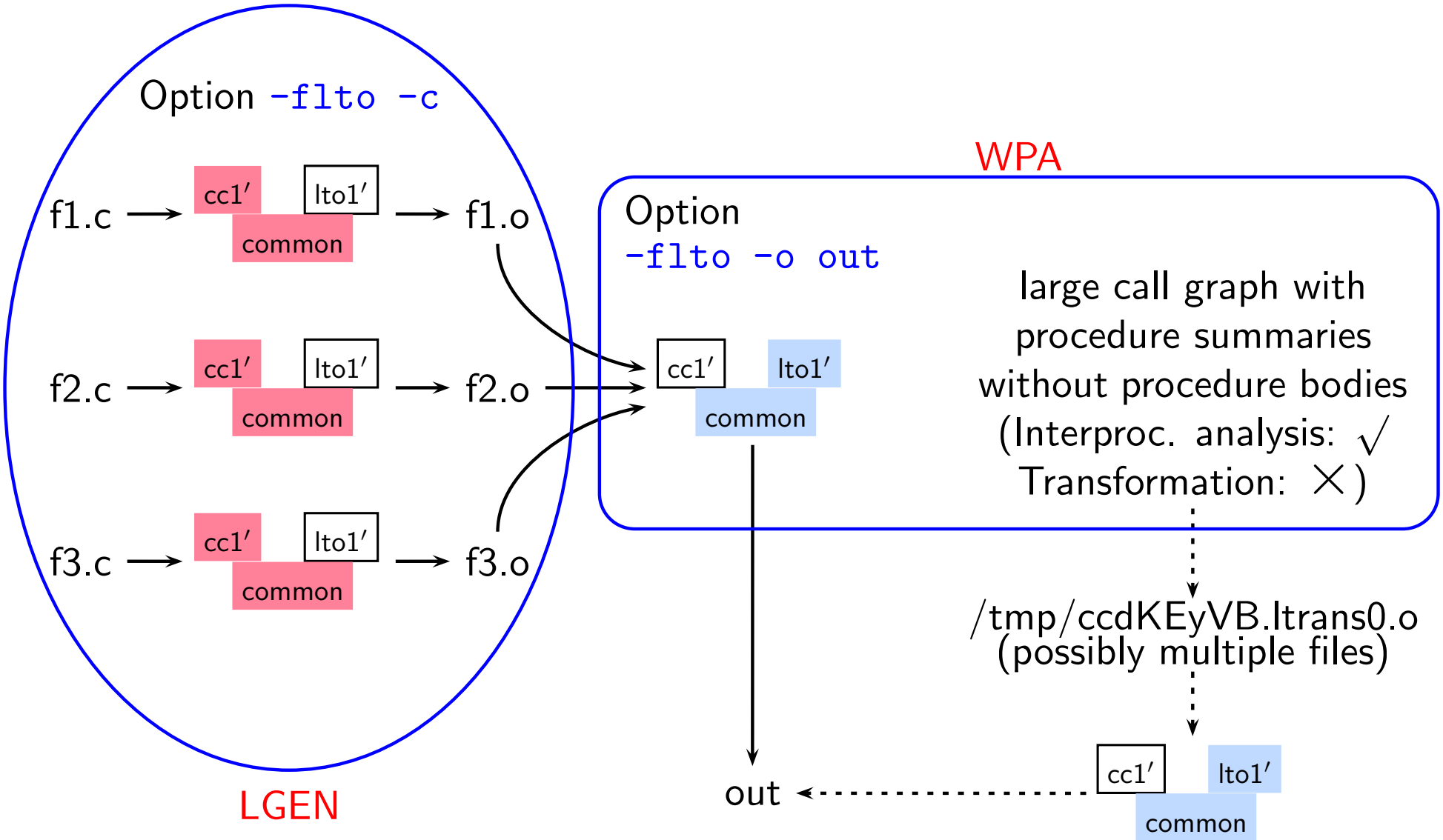
Partitioned LTO (aka WHOPR LTO)



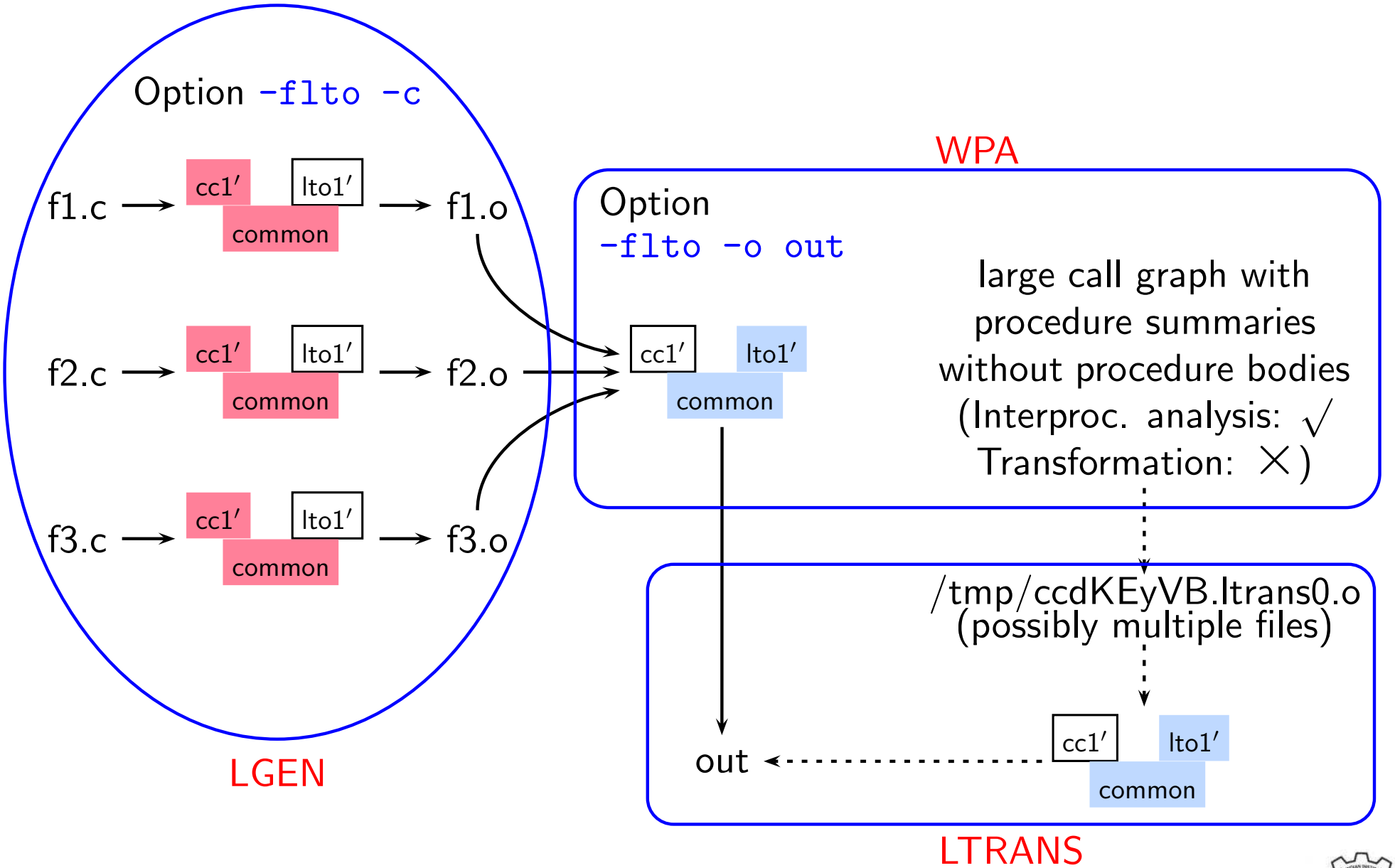
Partitioned LTO (aka WHOPR LTO)



Partitioned LTO (aka WHOPR LTO)

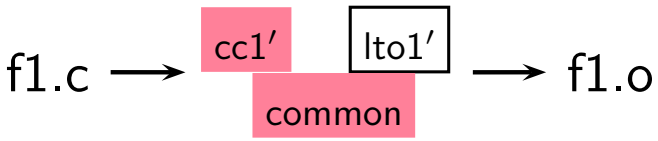


Partitioned LTO (aka WHOPR LTO)



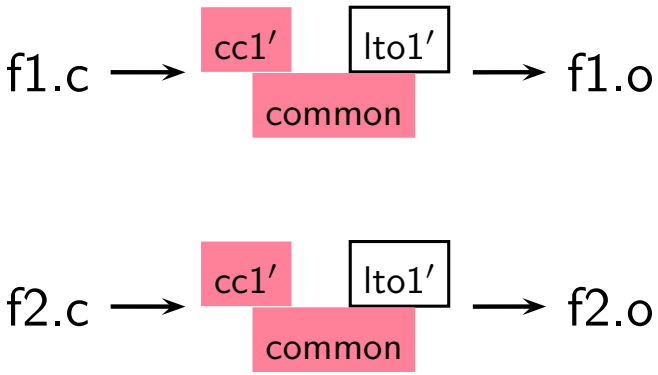
Non-Partitioned LTO

Option `-flto -c`



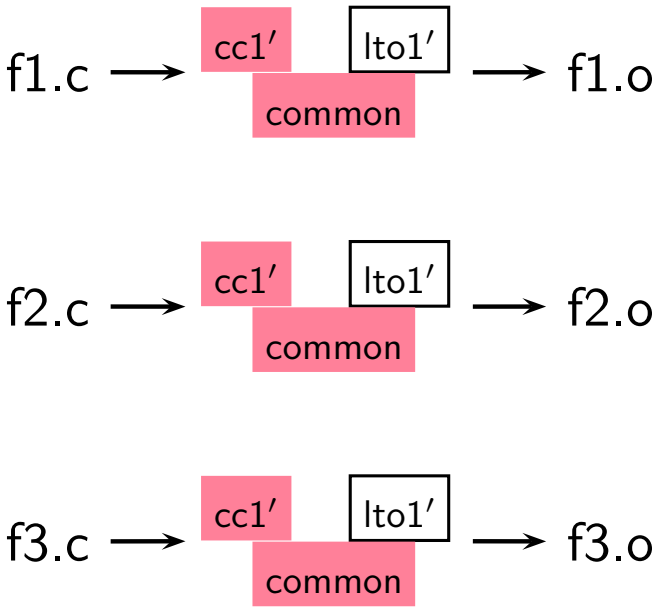
Non-Partitioned LTO

Option `-flto -c`

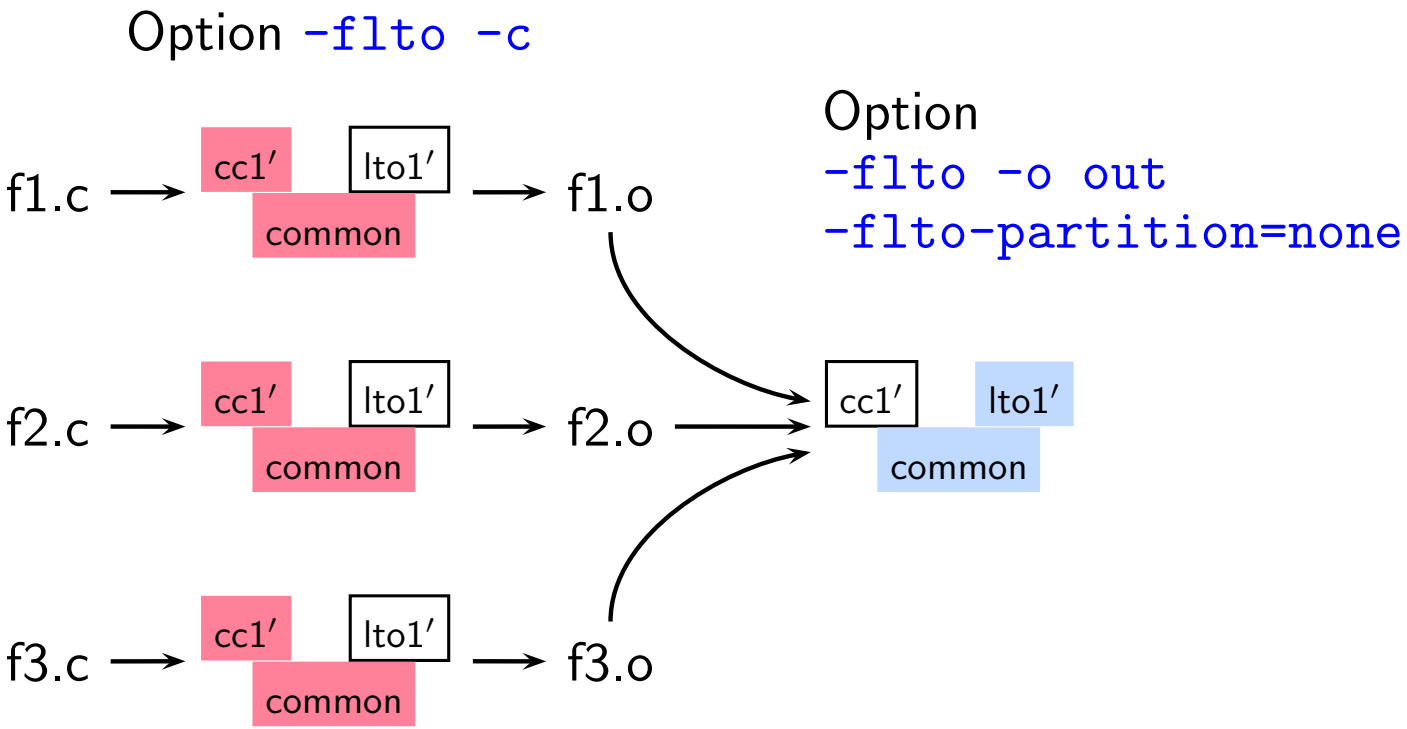


Non-Partitioned LTO

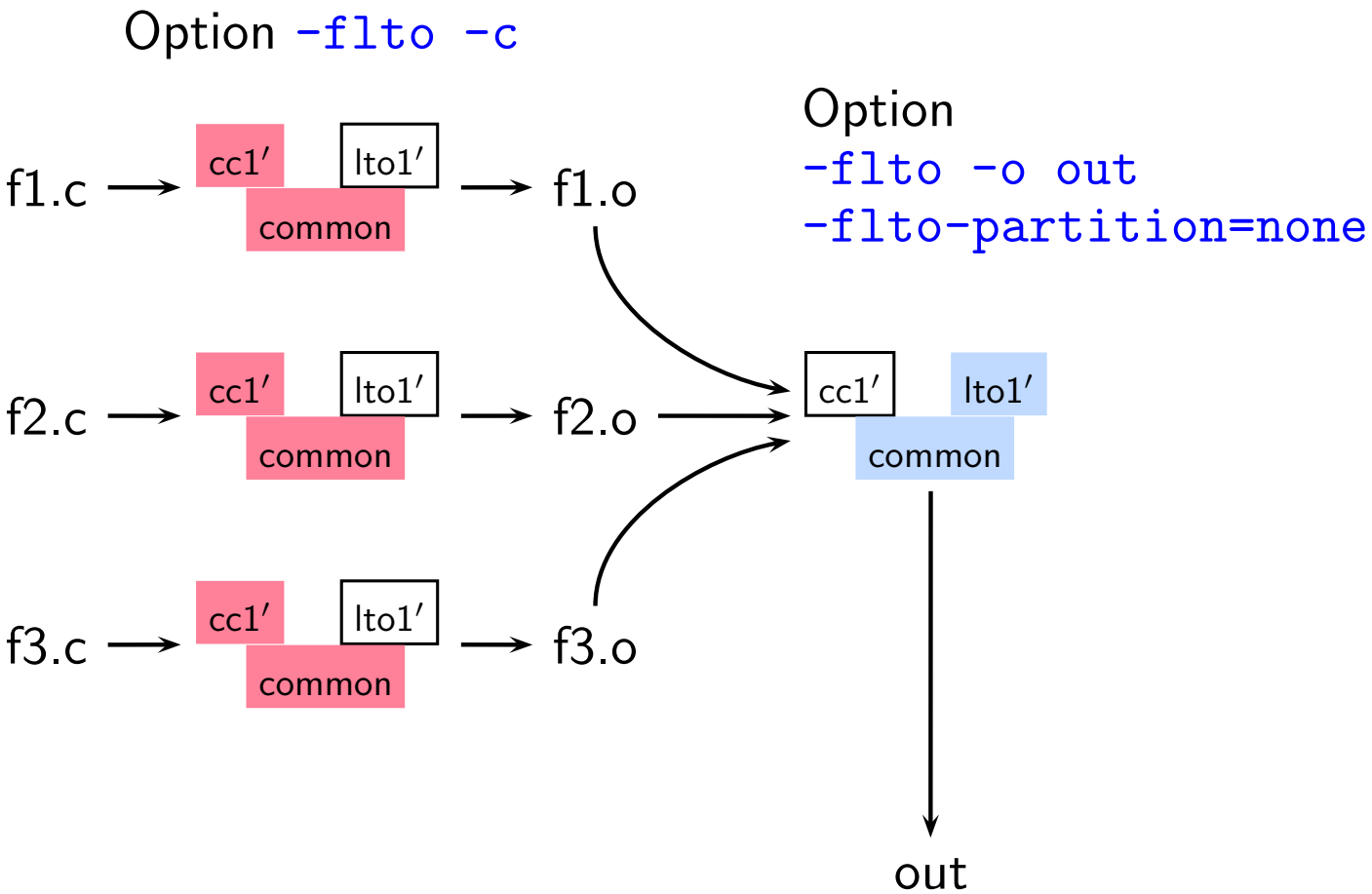
Option `-flto -c`



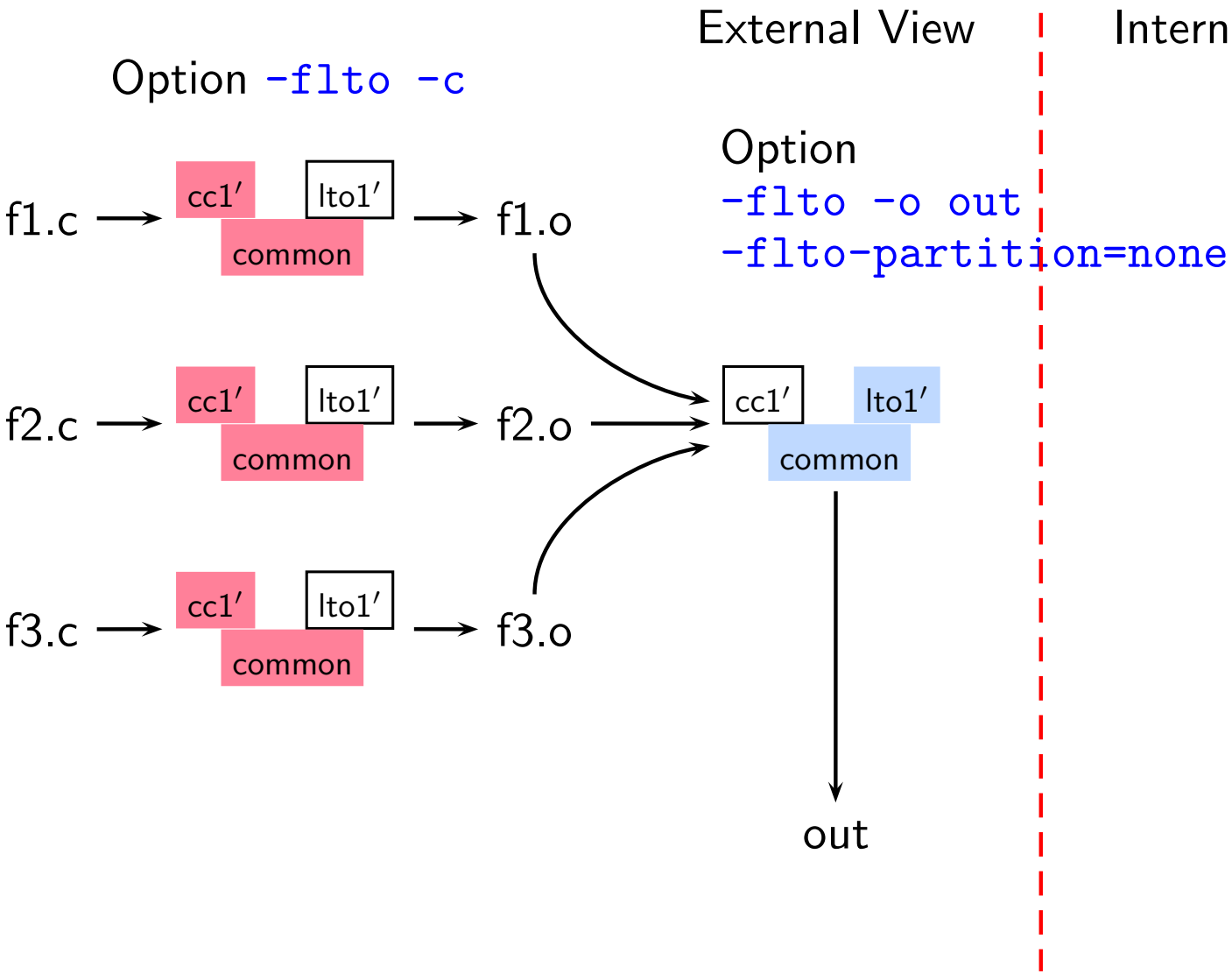
Non-Partitioned LTO



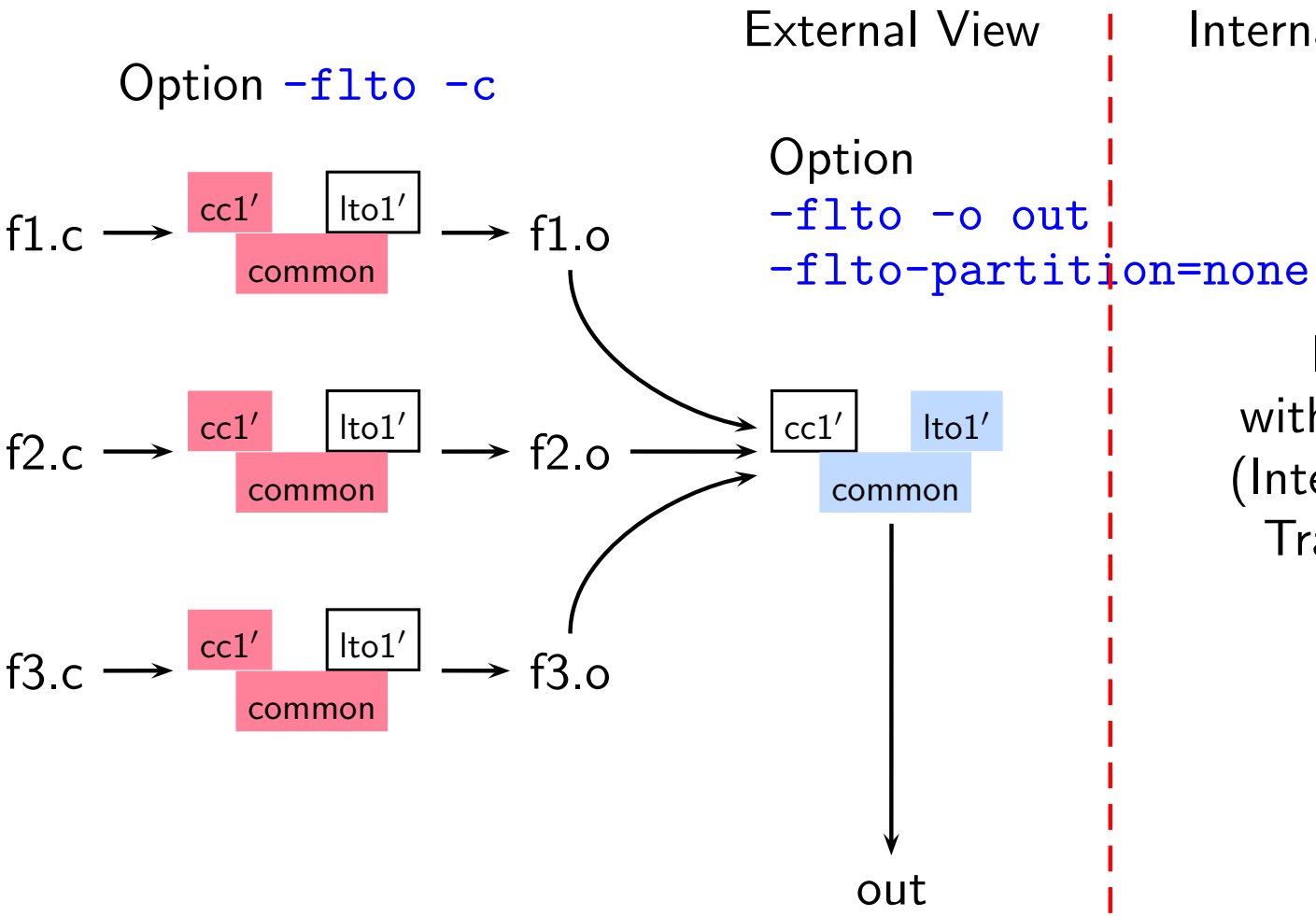
Non-Partitioned LTO



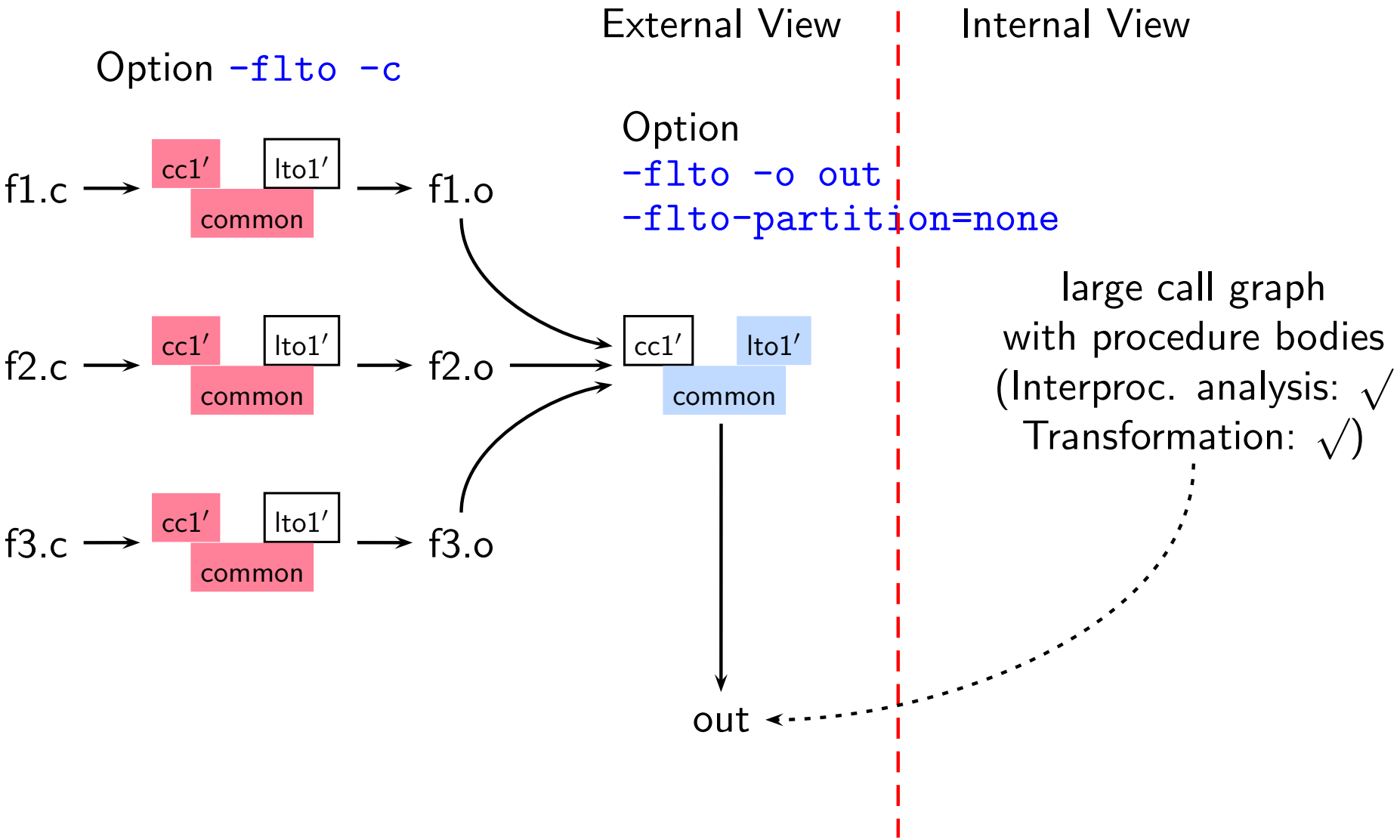
Non-Partitioned LTO



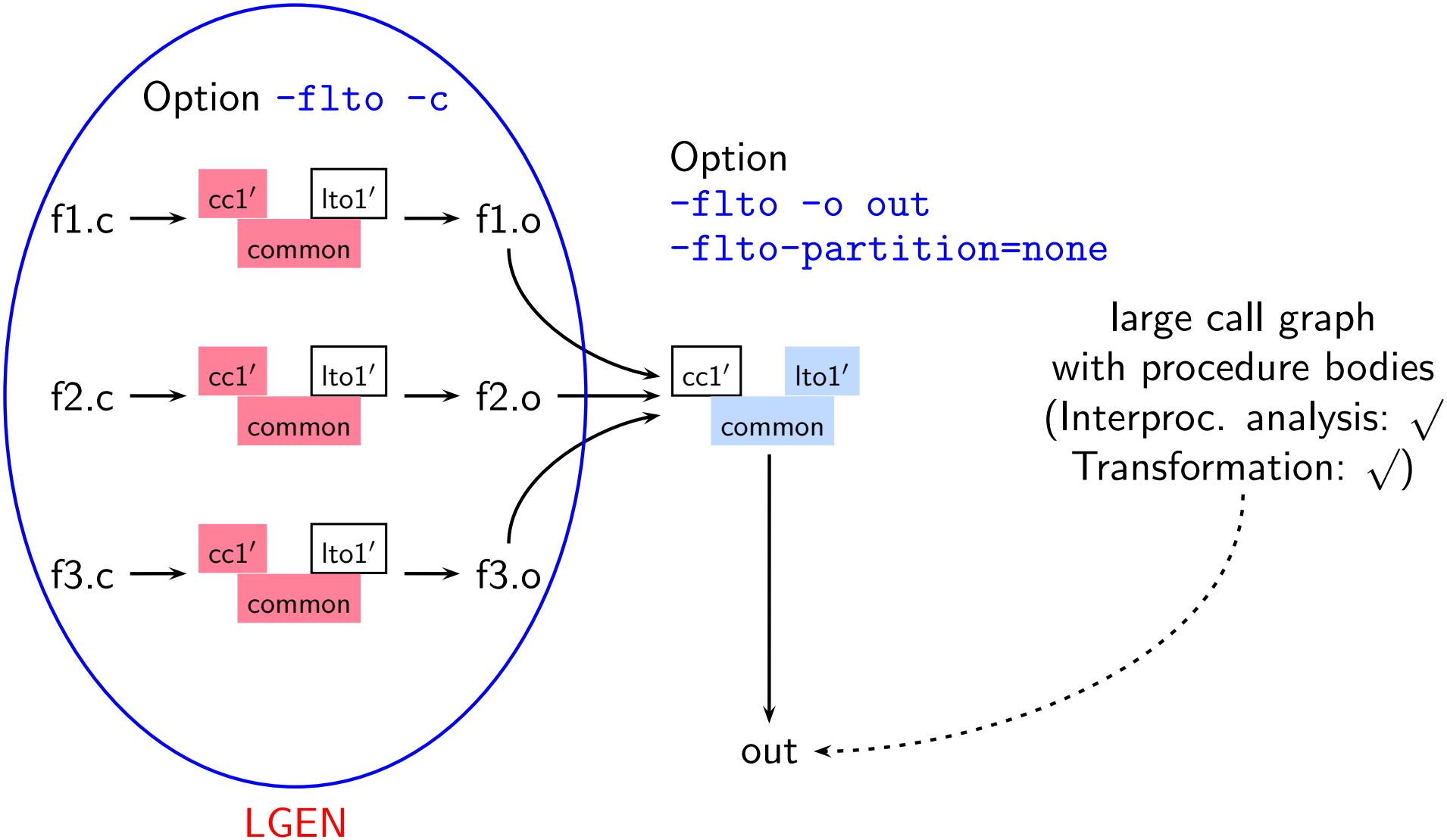
Non-Partitioned LTO



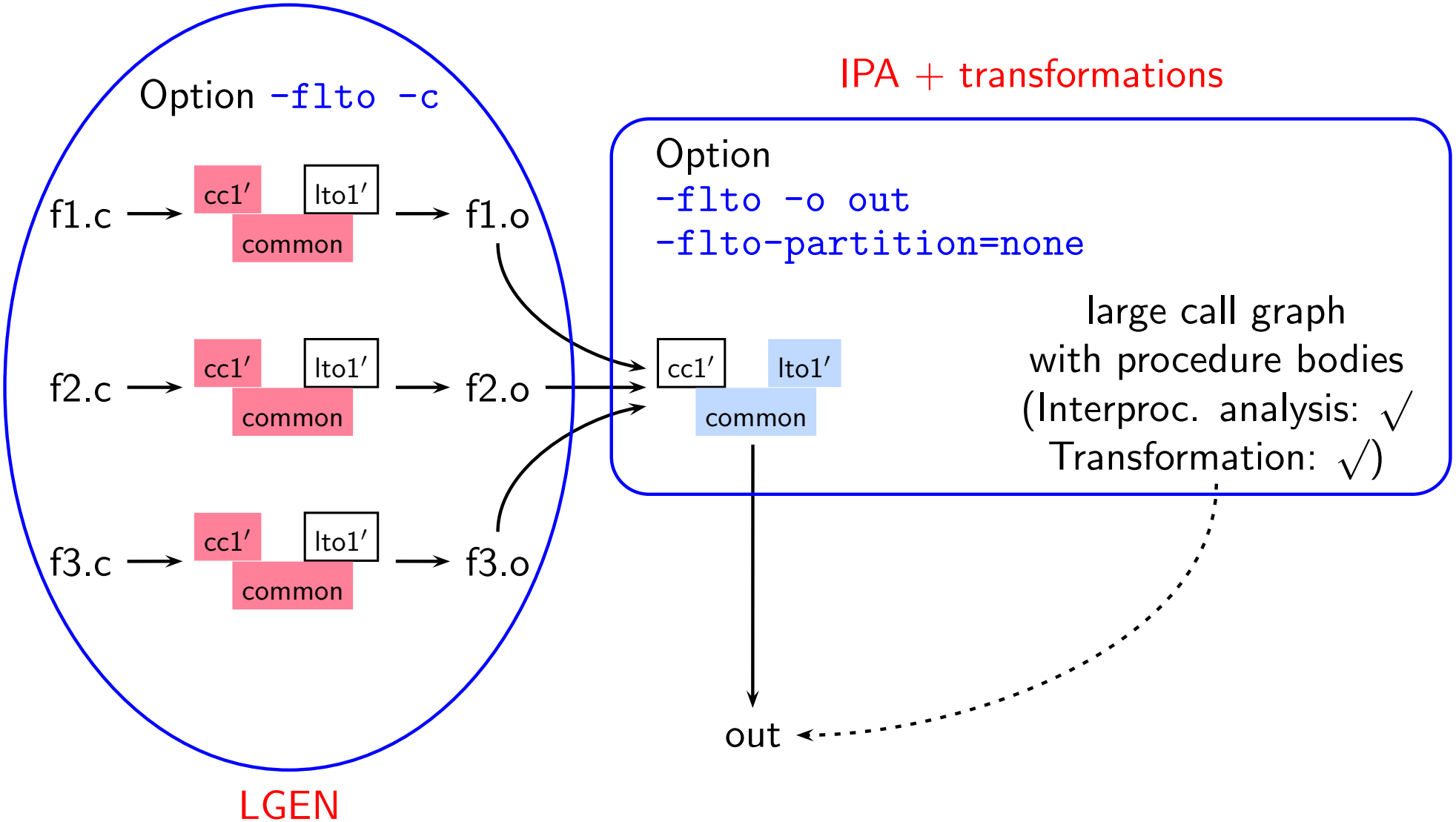
Non-Partitioned LTO



Non-Partitioned LTO



Non-Partitioned LTO



Non-Partitioned LTO