Workshop on Essential Abstractions in GCC

A Summary of Essential Abstrations

GCC Resource Center
(www.cse.iitb.ac.in/grc)

Department of Computer Science and Engineering,
Indian Institute of Technology, Bombay

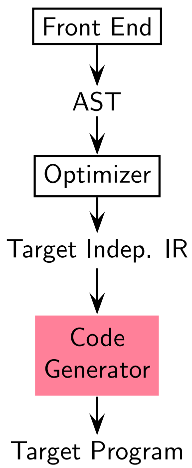
3 July 2012

Workshop Coverage

	External View	Internal View
Compiler Specifications	Machine descriptions	Front end hooks
Compiler Generator	Configuration and building	Retargetability mechanism
Generated Compiler	Gray box probing Pass structure and IR Data Flow Analysis Parallelization, Vectorization	Pass structure Control flow Static and dynamic plugin mechanisms

Compilation Models

Aho Ullman Model



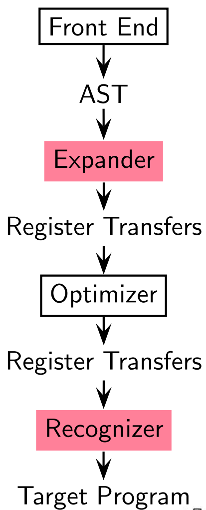
Aho Ullman: Instruction selection

- over optimized IR using
- cost based tree pattern matching

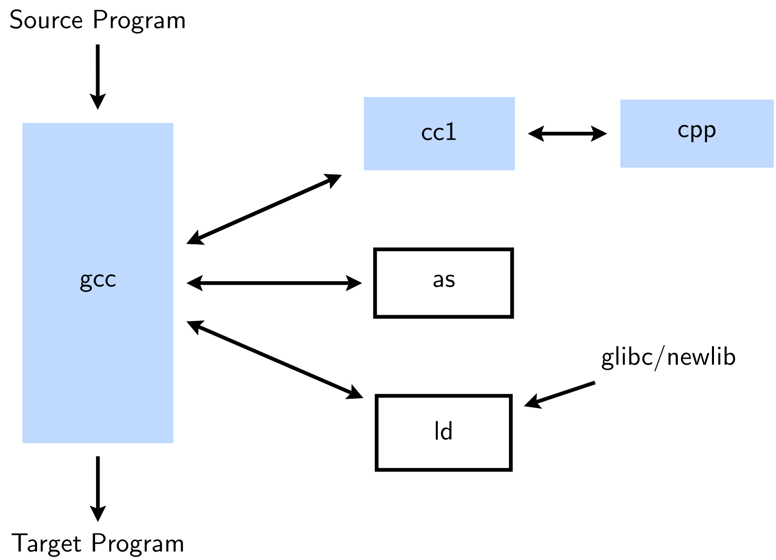
Davidson Fraser: Instruction selection

- over AST using
- structural tree pattern matching
- naive code which is
  - target dependent, and is
  - optimized subsequently

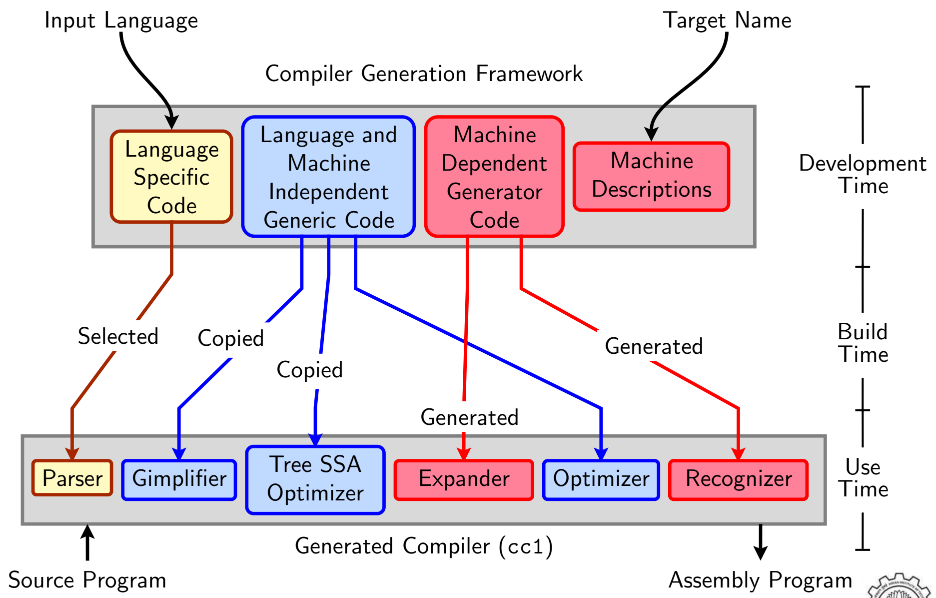
Davidson Fraser Model



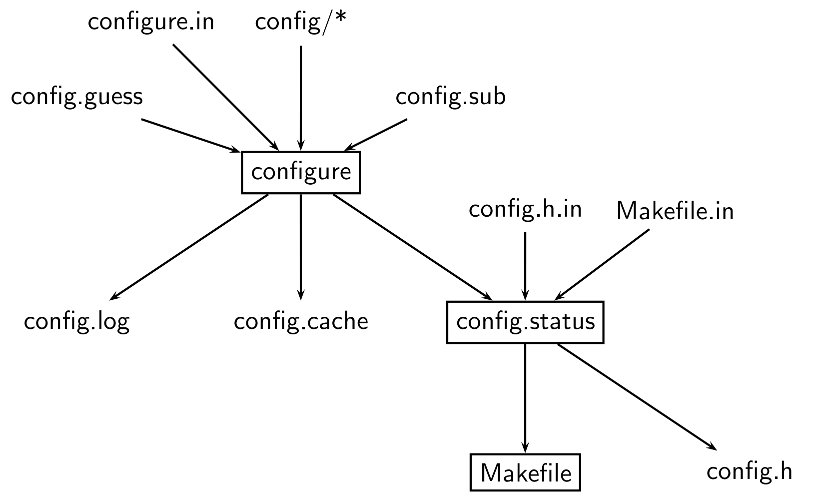
The GNU Tool Chain for C



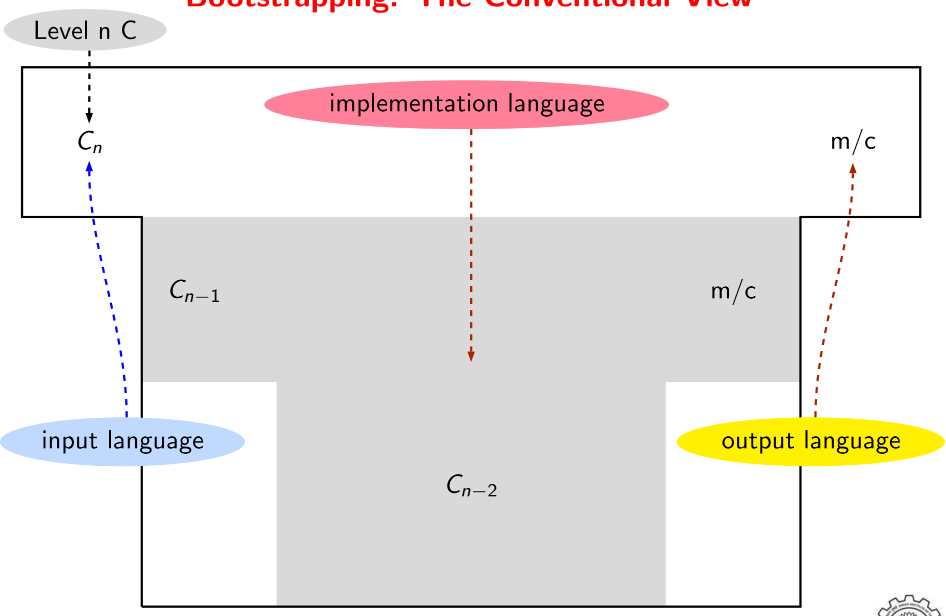
The Architecture of GCC



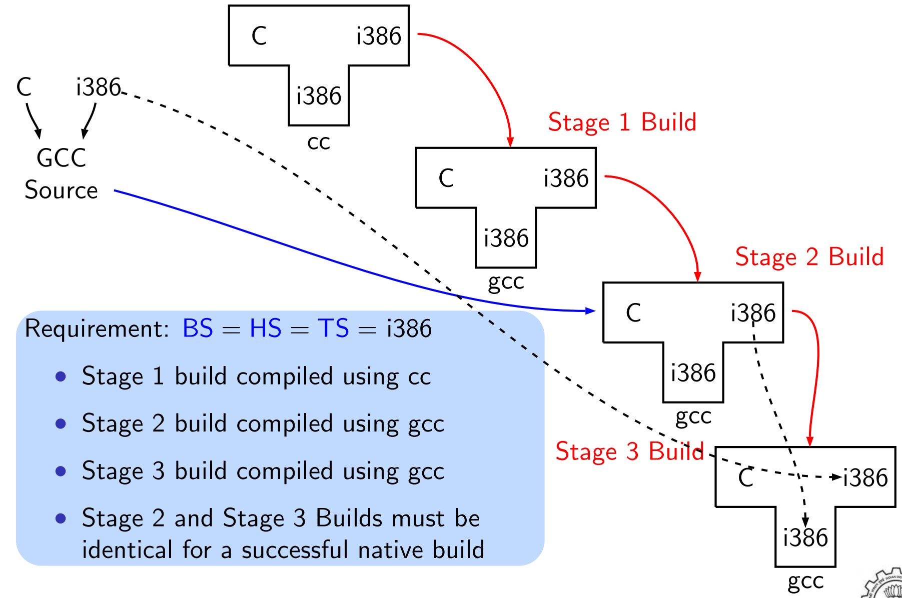
Configuring GCC



Bootstrapping: The Conventional View



A Native Build on i386



Requirement: BS = HS = TS = i386

- Stage 1 build compiled using cc
- Stage 2 build compiled using gcc
- Stage 3 build compiled using gcc
- Stage 2 and Stage 3 Builds must be identical for a successful native build

Build for a Given Machine

This is what actually happens!

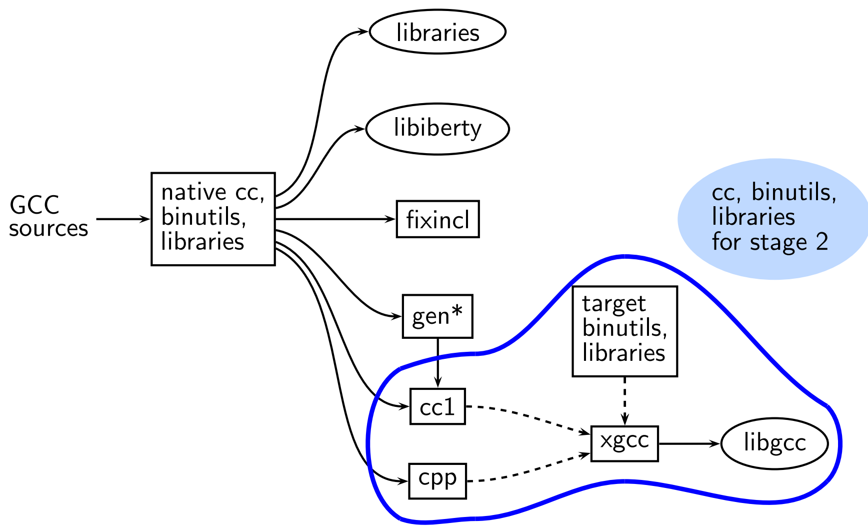
- Generation
  - Generator sources ($(SOURCE_D)/gcc/gen*.c) are read and generator executables are created in $(BUILD)/gcc/build
  - MD files are read by the generator executables and back end source code is generated in $(BUILD)/gcc
- Compilation

  Other source files are read from $(SOURCE_D) and executables created in corresponding subdirectories of $(BUILD)
- Installation

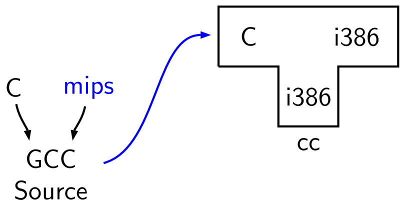
  Created executables and libraries are copied in $(INSTALL)

```
genattr
gencheck
genconditions
genconstants
genflags
genopinit
genpreds
genattrtab
genchecksum
gencondmd
genemit
gengenrtl
genmddeps
genoutput
genrecog
genautomata
gencodes
genconfig
genextract
gengtype
genmodes
genpeep
```


More Details of an Actual Stage 1 Build for C

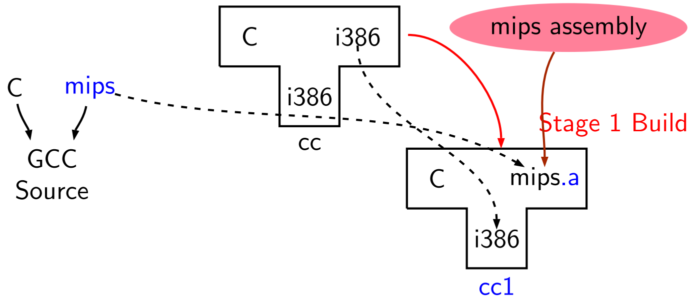


Building a MIPS Cross Compiler on i386: A Closer Look



Requirement: BS = HS = i386, TS = mips

Building a MIPS Cross Compiler on i386: A Closer Look

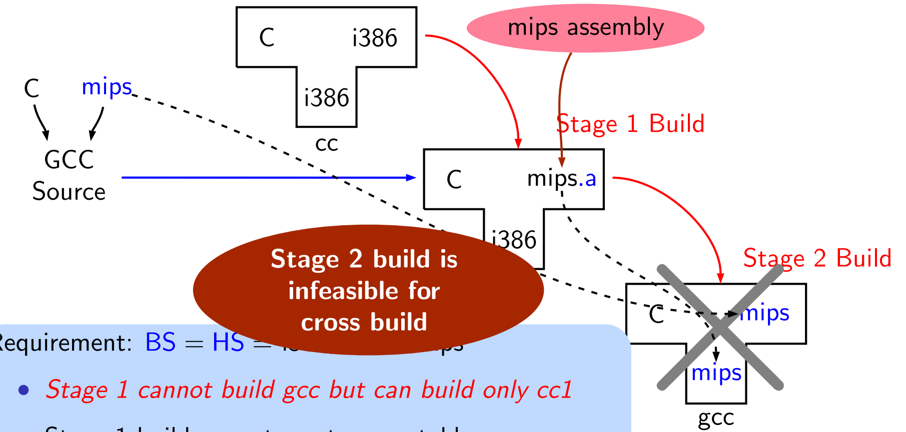


Requirement: BS = HS = i386, TS = mips

- *Stage 1 cannot build gcc but can build only cc1*
- Stage 1 build cannot create executables
- Library sources cannot be compiled for mips using stage 1 build

we have not built libraries for mips

Building a MIPS Cross Compiler on i386: A Closer Look

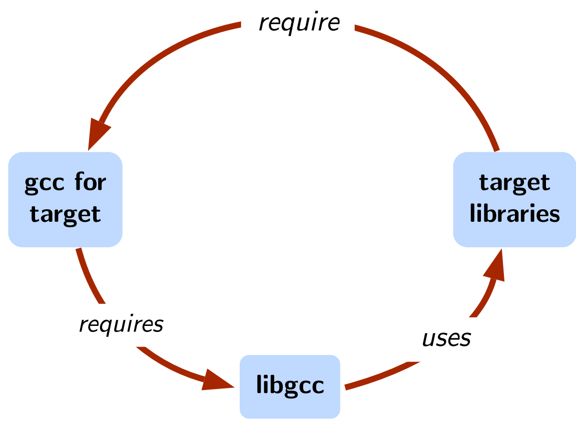


Requirement: BS = HS = i ... ps

- *Stage 1 cannot build gcc but can build only cc1*
- Stage 1 build cannot create executables
- Library sources cannot be compiled for mips using stage 1 build
- Stage 2 build is not possible

we have
not built libraries
for mips

Difficulty in Building a Cross Compiler

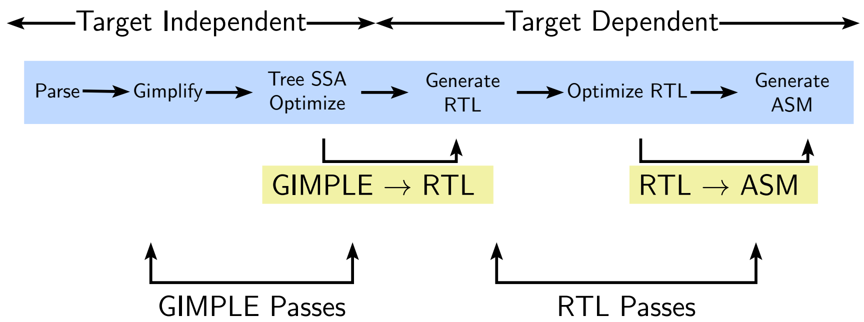


Generated Compiler Executable for All Languages

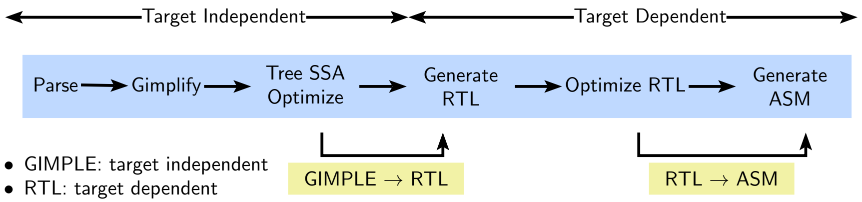
Main driver	`$BUILD/gcc/xgcc`
C compiler	`$BUILD/gcc/cc1`
C++ compiler	`$BUILD/gcc/cc1plus`
Fortran compiler	`$BUILD/gcc/f951`
Ada compiler	`$BUILD/gcc/gnat1`
Java compiler	`$BUILD/gcc/jc1`
Java compiler for generating main class	`$BUILD/gcc/jvgenmain`
LTO driver	`$BUILD/gcc/lto1`
Objective C	`$BUILD/gcc/cc1obj`
Objective C++	`$BUILD/gcc/cc1objplus`

Basic Transformations in GCC

Tranformation from a language to a *different* language



Instruction Specification and Translation: A Recap



- GIMPLE: target independent
- RTL: target dependent
- Need: associate the *semantics*

⇒GCC Solution: Standard Pattern Names

RTL Template

ASM

GIMPLE_ASSIGN

```
(define_insn  "movsi"
   [(set (match_operand 0 "register_operand" "r")
         (match_operand 1 "const_int_operand" "k"))]
   "" /* C boolean expression, if required */
    "li %0, %1"
)
```


Translation Sequence in GCC

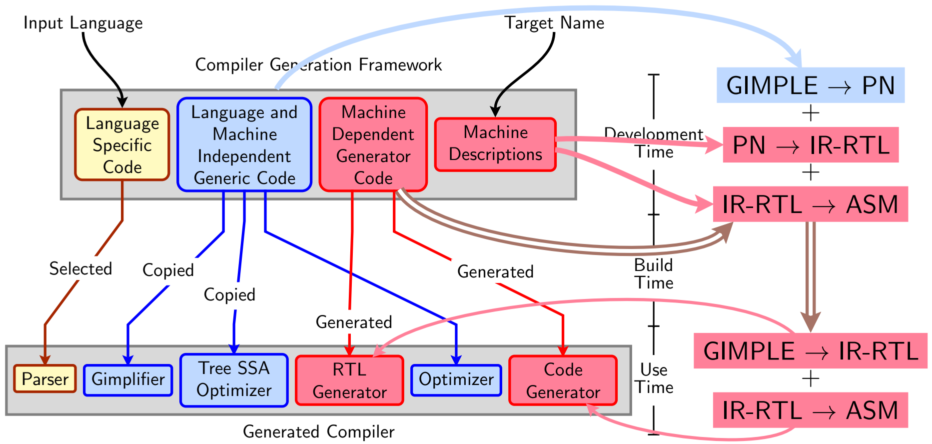
Development

```
(define_insn
   "movsi"
    [(set
      (match_operand 0 "register_operand" "r")
      (match_operand 1 "const_int_operand" "k")
      )]
   "" /*  C boolean expression, if required */
   "li %0, %1"
)
```

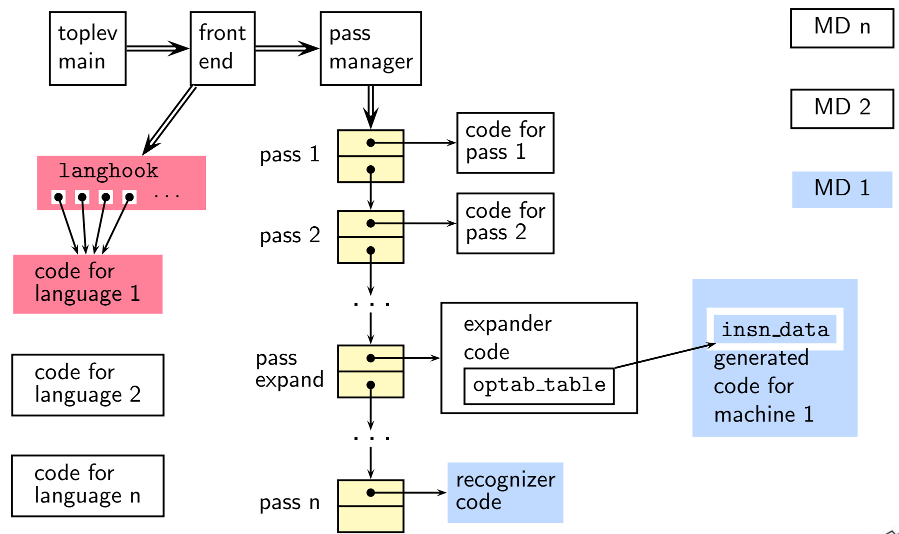
Use

```
D.1283 = 10;  ⇒  (set
                      (reg:SI 58 [D.1283])
                      (const_int 10:  [0xa])
                  )                           ⇒  li $t0, 10
```

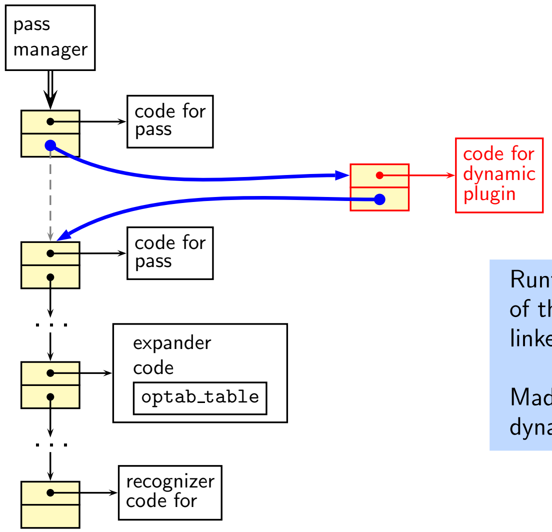

Retargetability Mechanism of GCC



Plugin Structure in cc1



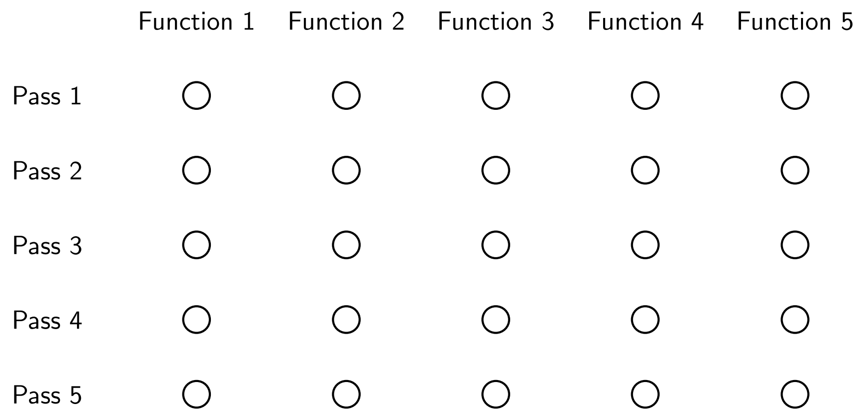
The Mechanism of Dynamic Plugin



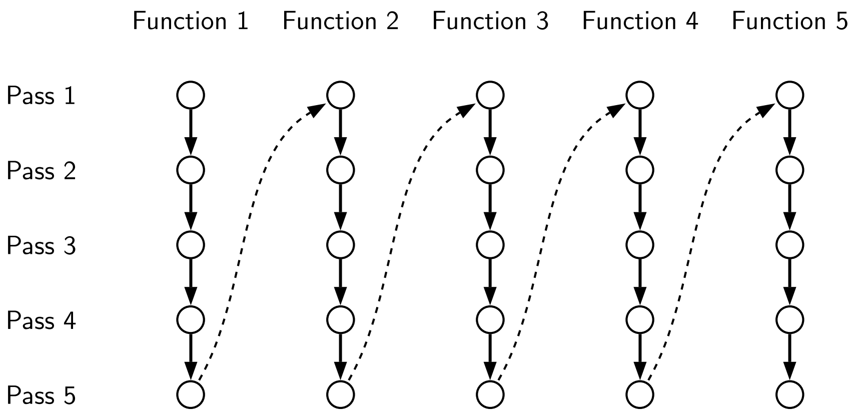
Runtime initialization of the appropriate linked list of passes

Made possible by dynamic linking

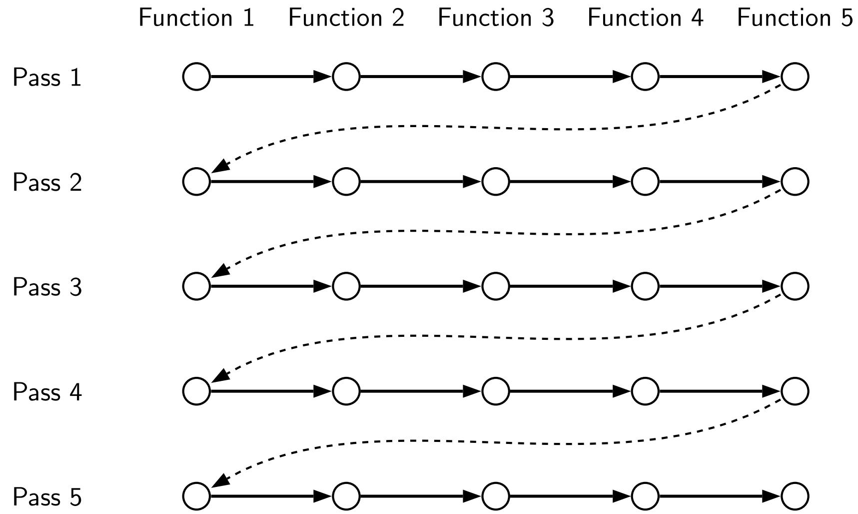
Execution Order in Intraprocedural Passes



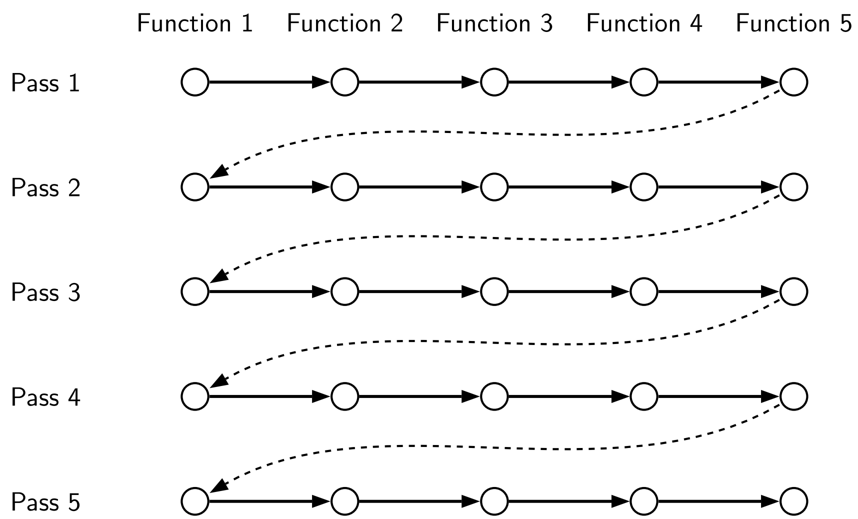
Execution Order in Intraprocedural Passes



Execution Order in Interprocedural Passes



Execution Order in Interprocedural Passes



LTO Support in GCC

		Transformation		
		In the same process as that of analysis	In an independent process (possibly multiple processes)	
		Single partition of the program	Single partition of the program	Multiple partitions of the program
Whole Program Analysis	Call graph without function bodies	Not suppported	Suppported in GCC-4.6.0 ⇒	Will be suppported in future
	Call graph with function bodies	Suppported in GCC-4.6.0	Not suppported	Not suppported

`-flto`

`-flto -flto-partition=none`

WHOPR mode

cc1 and Single Process lto1

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

cc1

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


cc1 and Single Process lto1

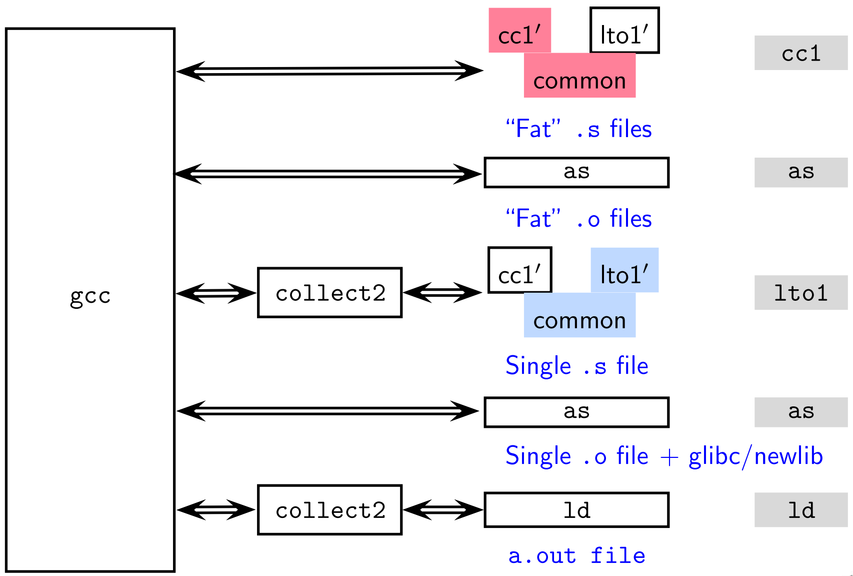
```
toplev_main
...
  compile_file
  ...
    cgraph_analyze_function
```

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

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

lto1

The GNU Tool Chain for Single Process LTO Support



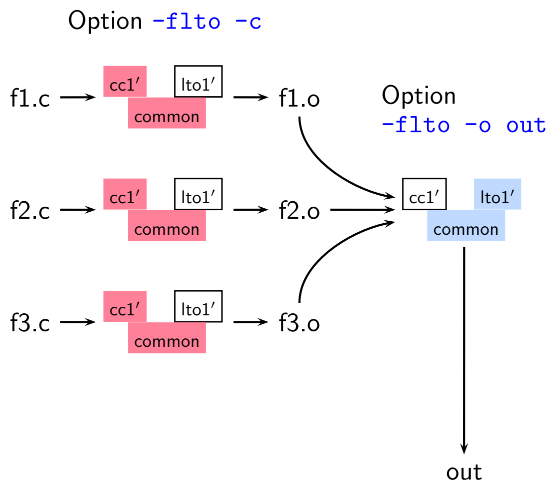
The GNU Tool Chain for Single Process LTO Support

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

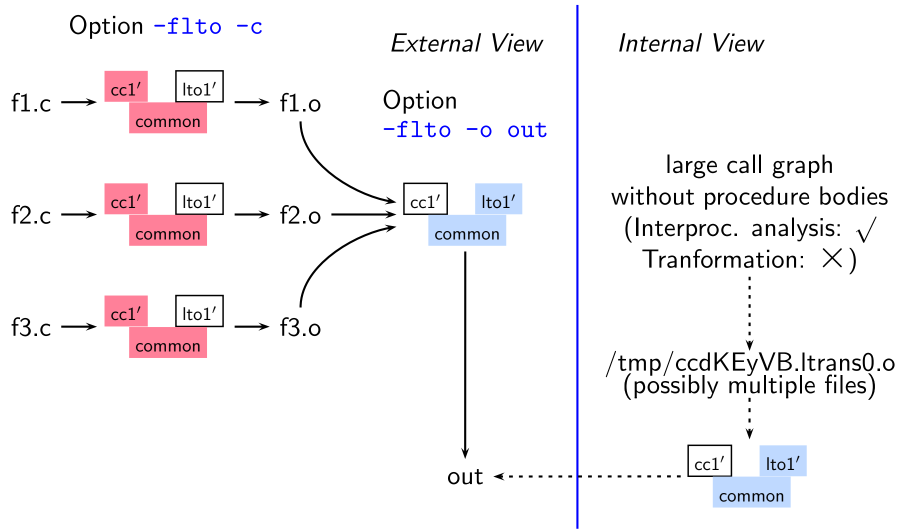
```
cgraph_optimize
   ipa_passes
      execute_ipa_pass_list(all_small_ipa_passes)/*!in lto*/
         execute_ipa_summary_passes(all_regular_ipa_passes)
         execute_ipa_summary_passes(all_lto_gen_passes)
         ipa_write_summaries
      cgraph_expand_all_functions
         cgraph_expand_function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

a.out file

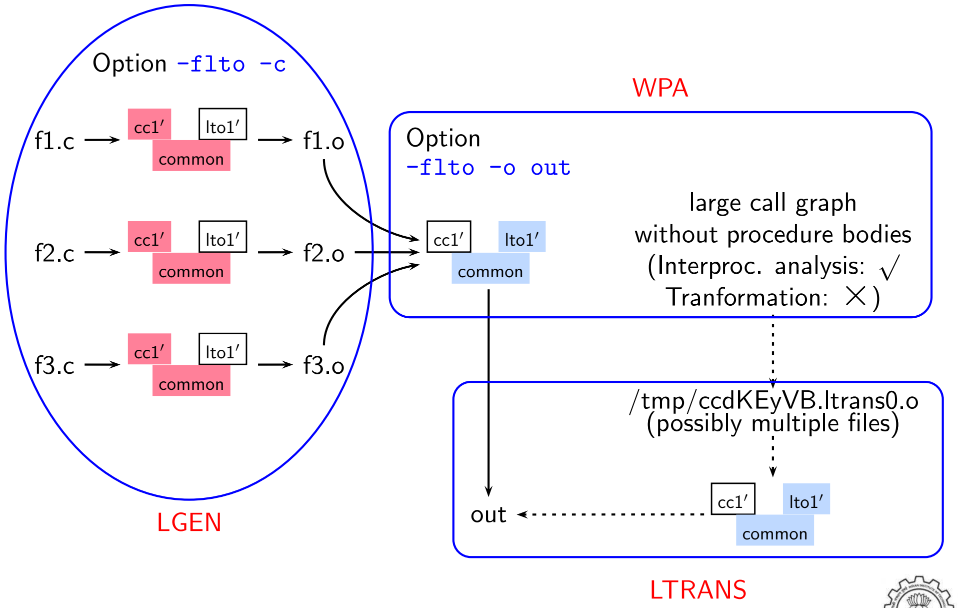
Multi Process LTO (aka WHOPR LTO)



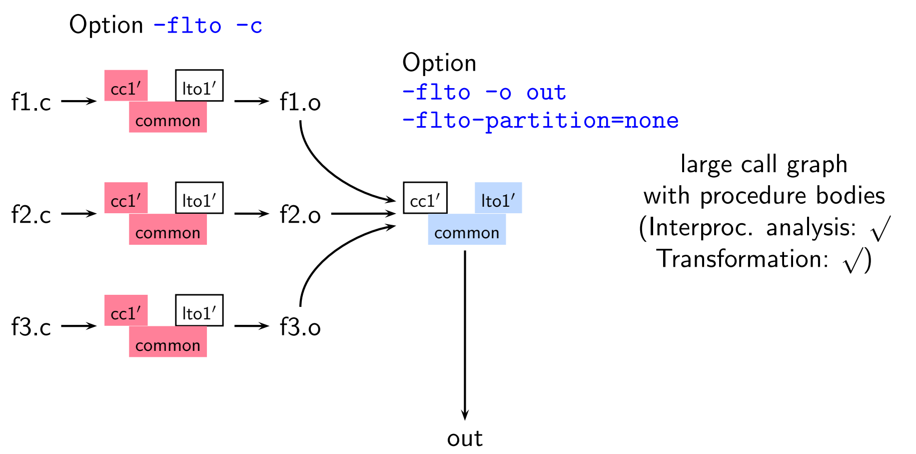
Multi Process LTO (aka WHOPR LTO)



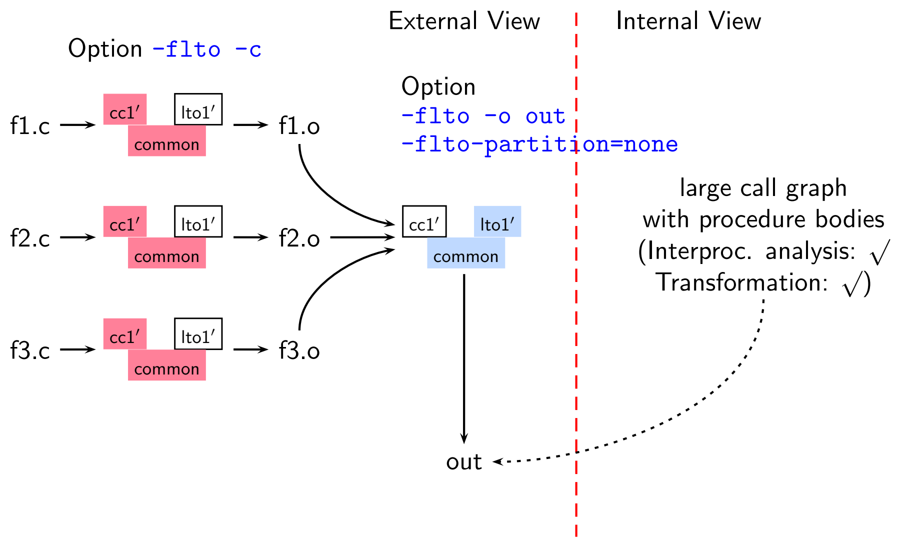
Multi Process LTO (aka WHOPR LTO)



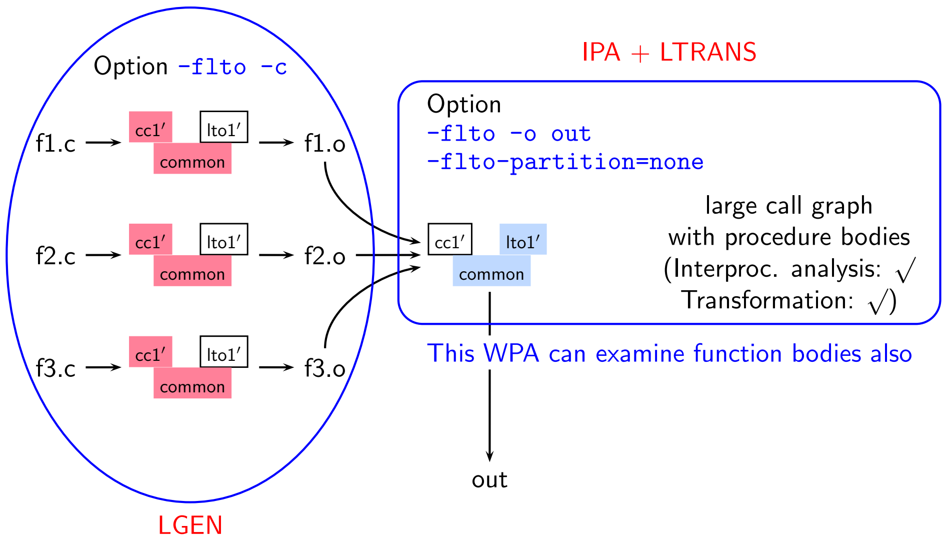
Single Process LTO



Single Process LTO



Single Process LTO

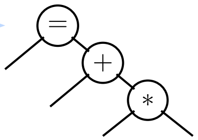


Redundancy in MIPS Machine Descriptions: Example 3

```
[(set (match_operand:m 0 "register_operand" "c0") (plus:m
     (mult:m (match_operand:m 1 "register_operand" "c1")
                (match_operand:m 2 "register_operand" "c2")))]
     (match_operand:m 3 "register_operand" "c3")))]
```

RTL Template

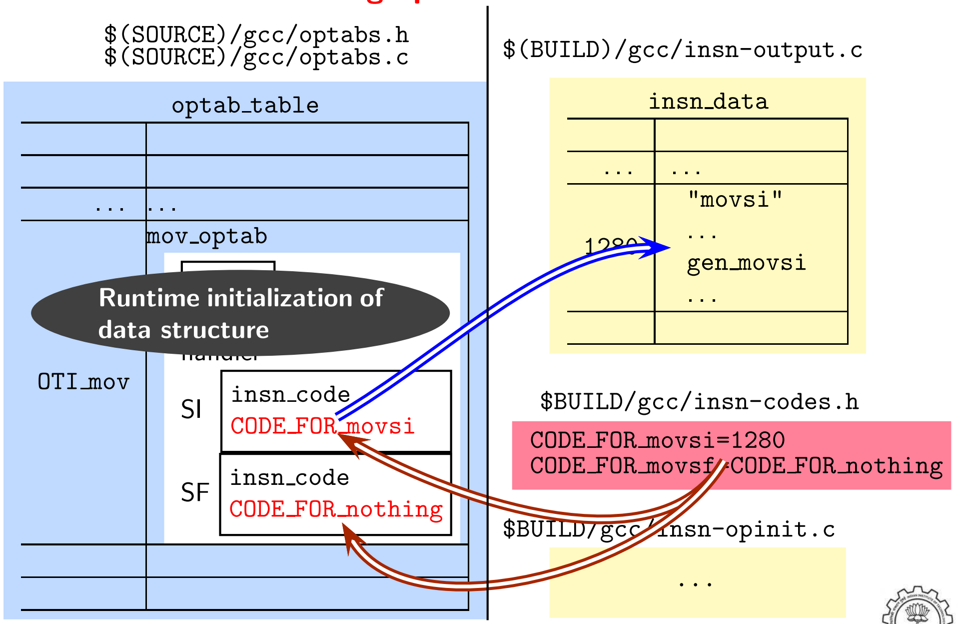
Structure



Details

Pattern name	*m*	*c0*	*c1*	*c2*	*c3*
*mul_acc_si	SI	=l*?*?,d?	d,d	d,d	0,d
*mul_acc_si_r3900	SI	=l*?*?,d*?,d?	d,d,d	d,d,d	0,1,d
*macc	SI	=l,d	d,d	d,d	0,1
*madd4<mode>	ANYF	=f	f	f	f
*madd3<mode>	ANYF	=f	f	f	0

Hooking up Back End Details



And the final realization . . .