

# GCC Internals and Porting

HelloGcc Workshop  
October 24, 2009



Mingjie Xing  
joefoxreal@gmail.com

# Outline



- Getting Started
- Overview
  - Compilation, Source Tree, Internal Framework
- Front End
  - Language Hooks, C Parser, Tree & GENERIC
- Middle End
  - GIMPLE, Call Graph, Passes
- Back End & Port
  - RTL, MD, Target Macros



# Getting Started

<http://gcc.gnu.org/wiki/GettingStarted>

- Tutorials, HOWTOs
  - GCC Internals Podcast - English listening :)
  - GCC Internals Tutorial - Very detailed !
  - Workshop on GCC Internals - Spim port
- Internal Documentation
  - GCC Internals
- Dealing With the Source Code
  - Debugging, Testing, Writing pass/front-end/back-end
- Structure of GCC
  - Front end, Tree/RTL Optimizers, Passes

# Overview

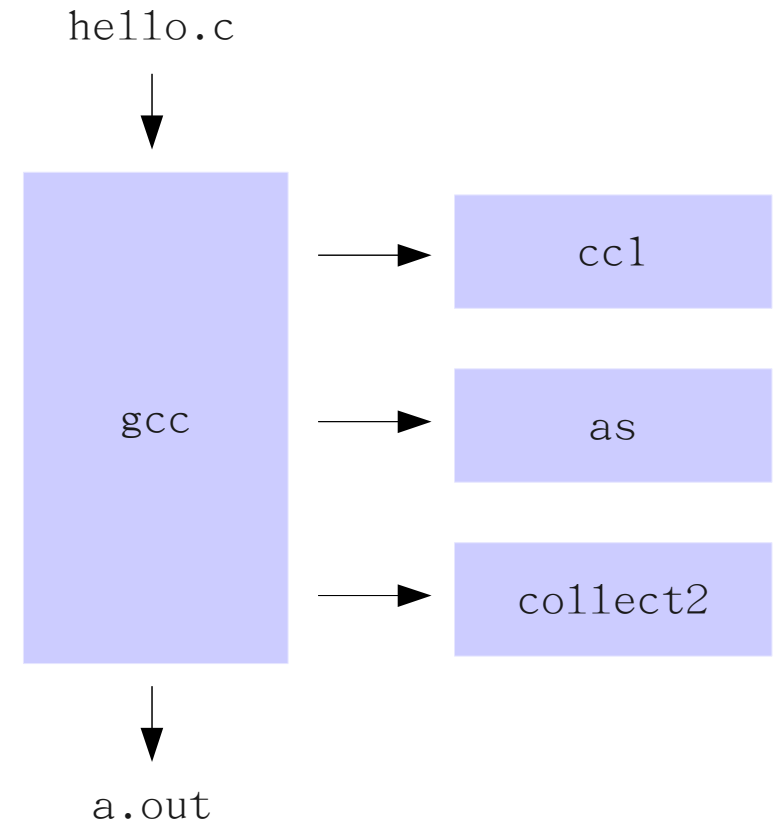


- Compilation
- Source Tree
- Internal Framework

# Compilation

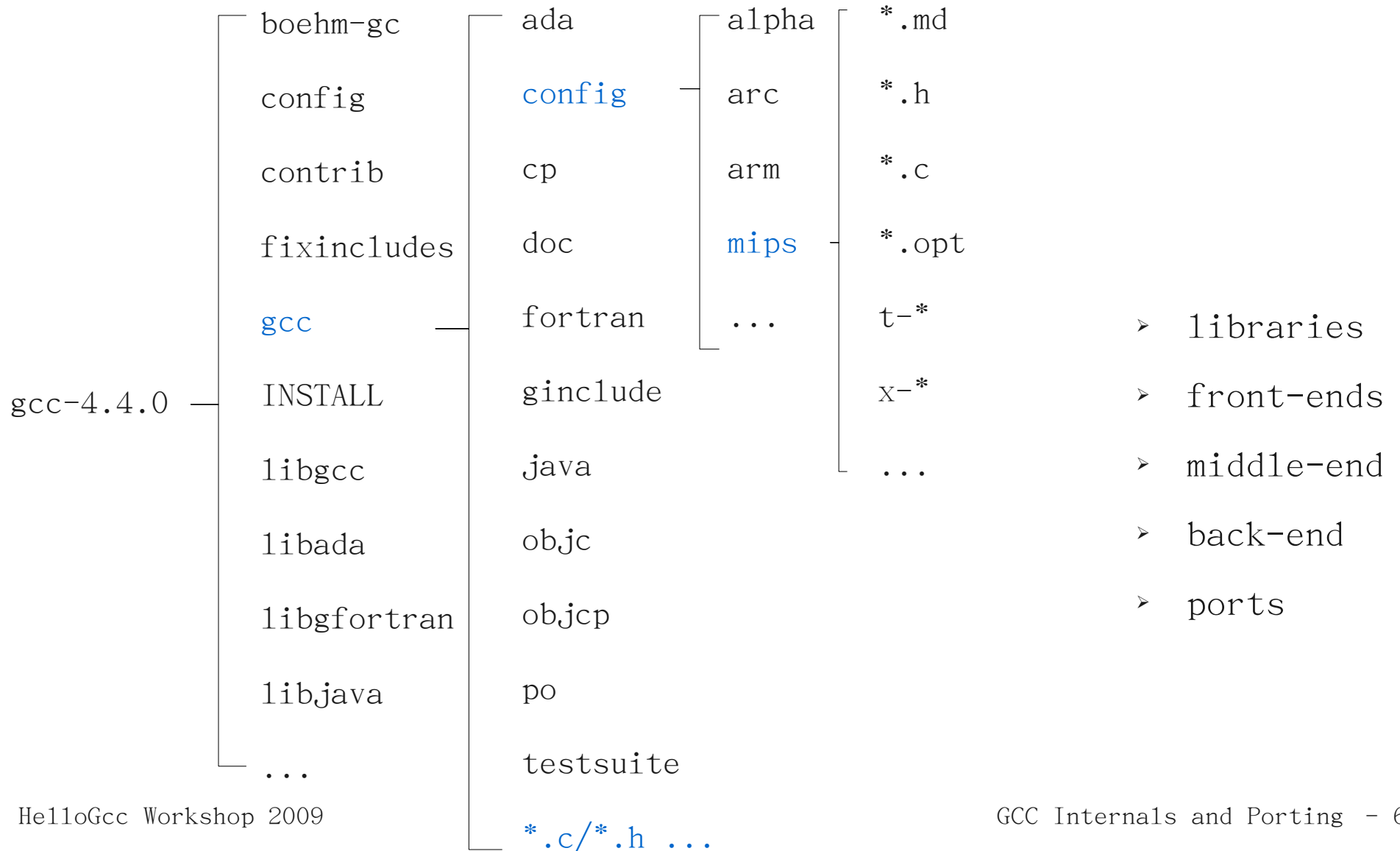


- › gcc driver
  - › Process spec strings: gcc.c
  - › `$ gcc -dumpspecs`
- › ccl
  - › Entry point: `toplev_main`, `toplev.c`
  - › Same to `cclplus`, `jcl`, `f771`, etc.
- › collect2
  - › Real linker: `ld`
  - › Handle initialization functions:  
`main` → `__main` → constructors

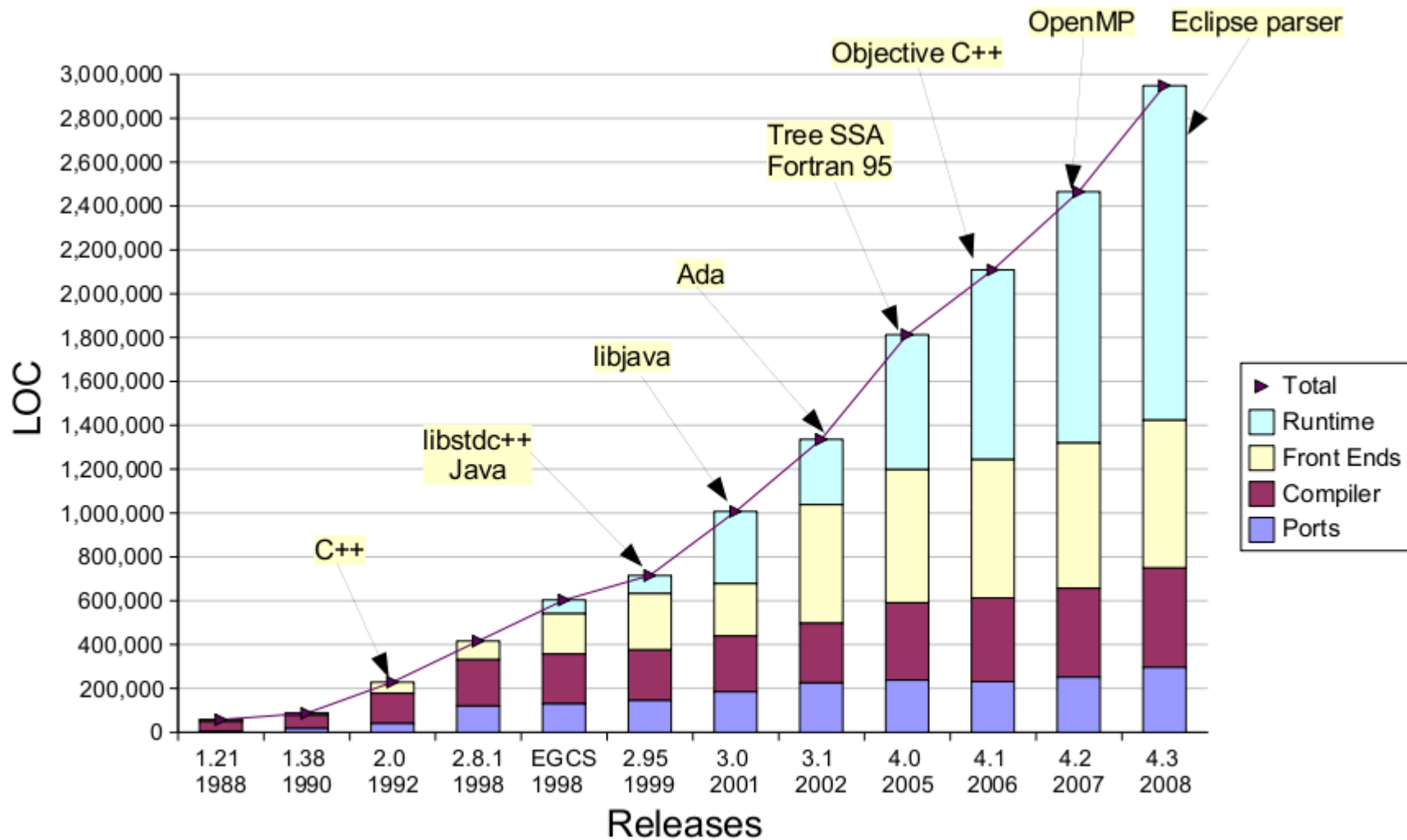


```
$ gcc -v hello.c
```

# Source Tree



# GCC Growth

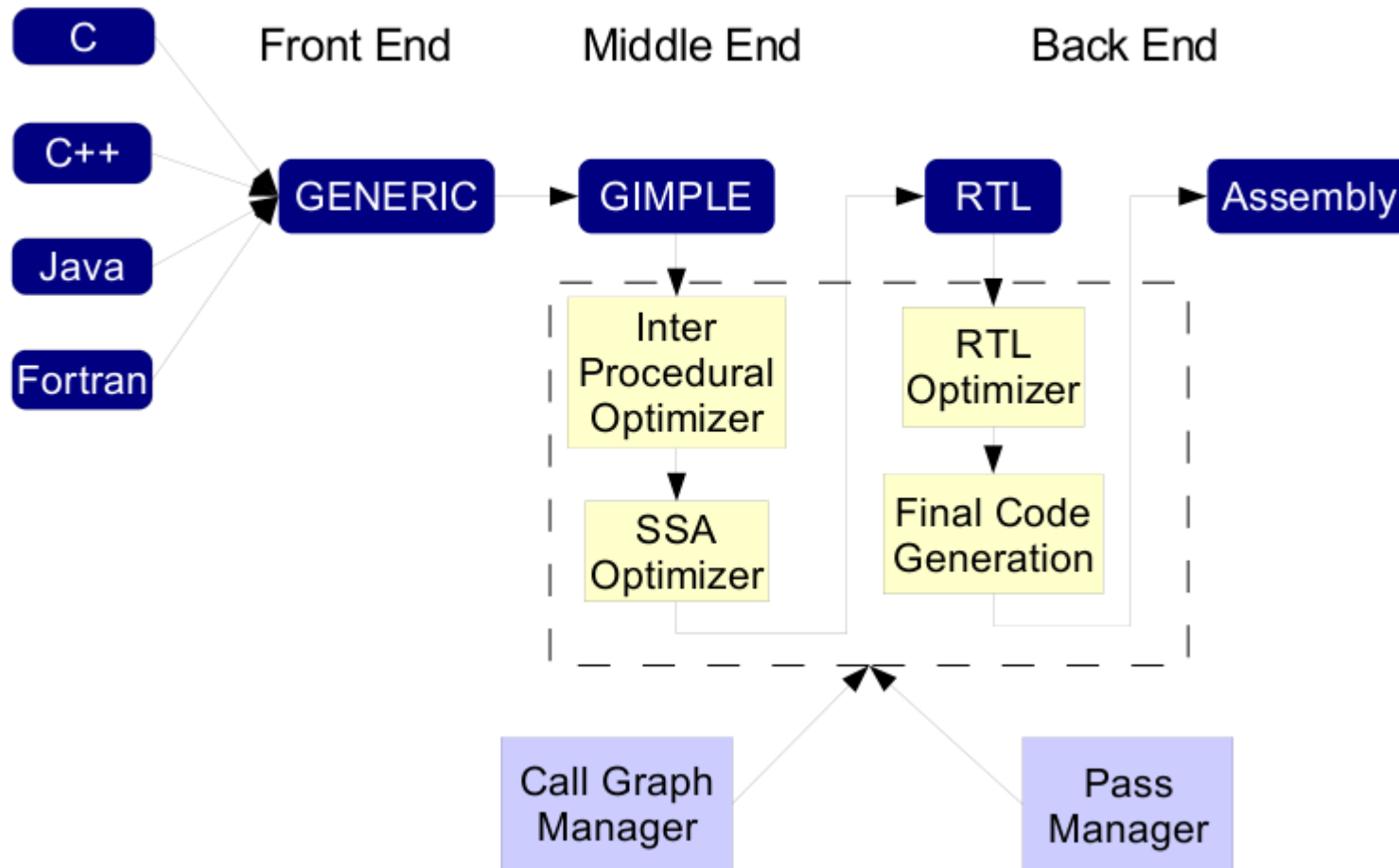


<sup>1</sup>generated using David A. Wheeler's 'SLOCCount'.

From Diego Novillo's Slides



# Internal Framework



From Diego Novillo's Slides

# Front End



- Language Hooks
- C Parser
- Tree & GENERIC

# Language Hooks



## › Define the structure

```
struct lang_hooks
{
    ...
} /* langhooks.h */
```

## › Define default functions

```
bool
hook_bool_void_false (void)
{
    return false;
} /* langhooks.c */
```

## › Define the default initializer

```
#define LANG_HOOKS_NAME "GNU unknown"
#define LANG_HOOKS_INIT \
    hook_bool_void_false
#define LANG_HOOKS_INITIALIZER { \
    ...
} /* langhooks-def.h */
```

## › Define specific functions

```
bool
c_objc_common_init (void)
{
    ...
} /* c-opts.c */

/* also in c-common.c, c-decl.c */
```

## › Declare the variable & Initialize

```
#include "c-objc-common.h"

#undef LANG_HOOKS_NAME
#define LANG_HOOKS_NAME "GNU C"
#undef LANG_HOOKS_INIT
#define LANG_HOOKS_INIT c_objc_common_init
const struct lang_hooks = \
    LANG_HOOKS_INITIALIZER;
/* c-lang.c */
```



# C Parser

➤ `toplev_main` → `c_parser_file`

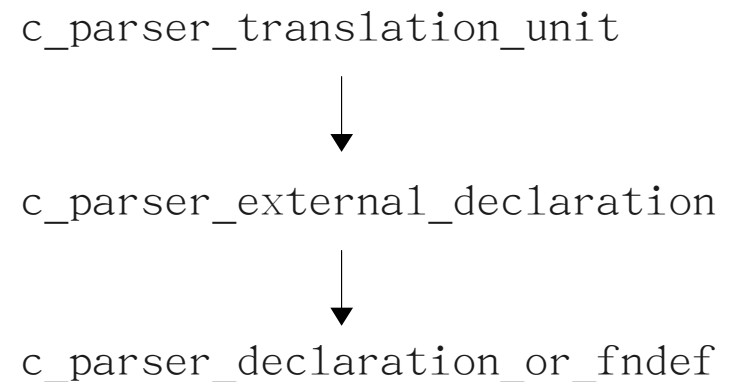
<code>toplev_main</code>	<code>toplev.c</code>
<code>do_compile</code>	<code>toplev.c</code>
<code>compile_file</code>	<code>toplev.c</code>
<code>lang_hooks.parse_file</code>	
<code>(c_common_parse_file)</code>	<code>c-opts.c</code>
<code>c_parser_file</code>	<code>c-parser.c</code>

➤ Hand-written Recursive-descent Parser

`translation-unit:`  
  `external-declarations`

`external-declarations:`  
  `external-declaration`  
  `external-declarations external-declaration`

`external-declaration:`  
  `function-definition`  
  `declaration`





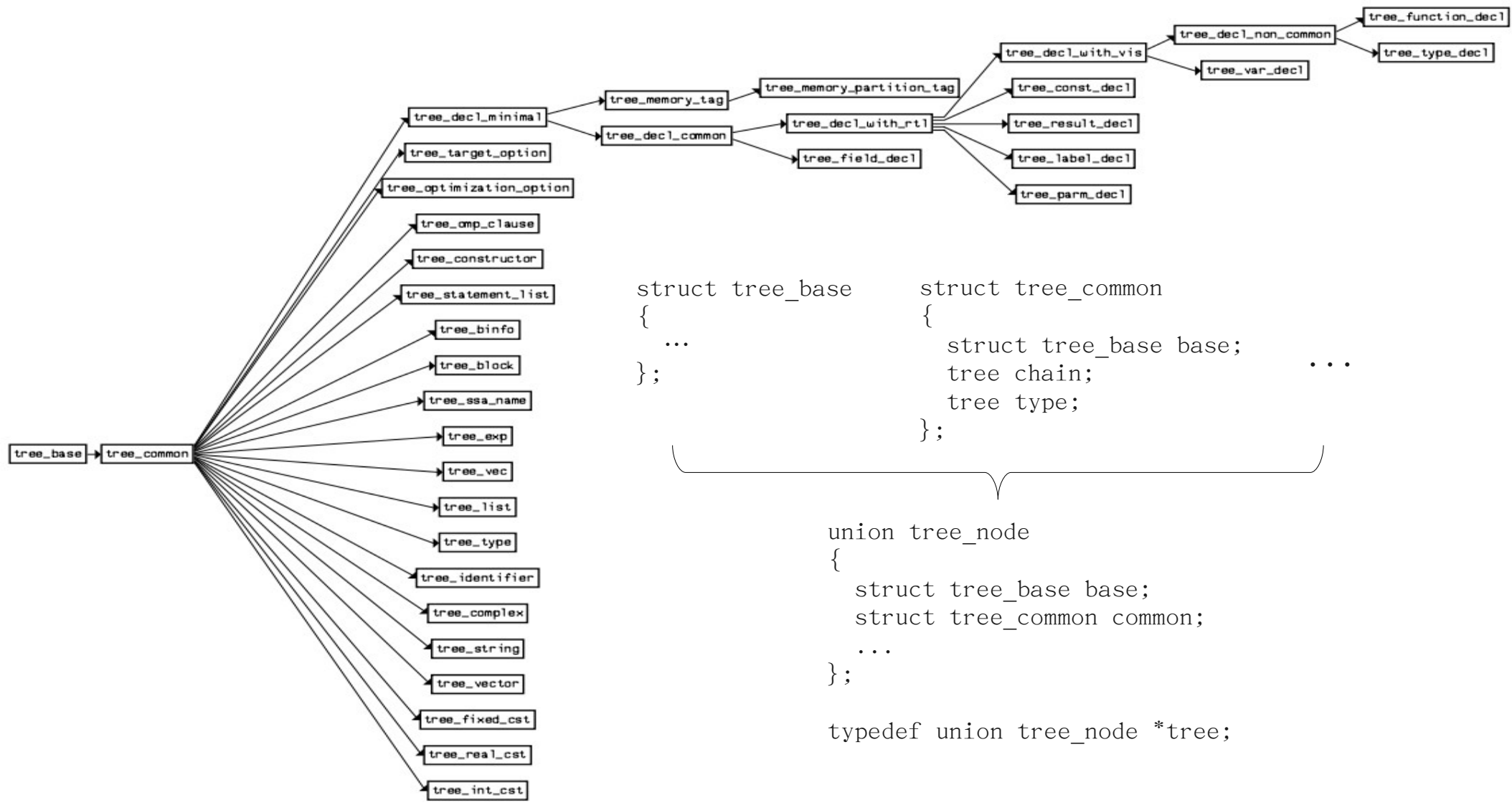
# Tree & GENERIC

- Tree
  - Language-dependent IR
  - `tree.def`, `c-common.def`, `java-tree.def` ...
- GENERIC
  - Language-independent IR
  - `tree.def`
- Genericize
  - Tree → GENERIC (currently dose nothing)

```
c_parser_declaration_or_fndef  
  finish_function  
    c_genericize
```

```
c-parser.c  
c-decl.c  
c-gimplify.c
```

# Tree Node



```

struct tree_base
{
  ...
};

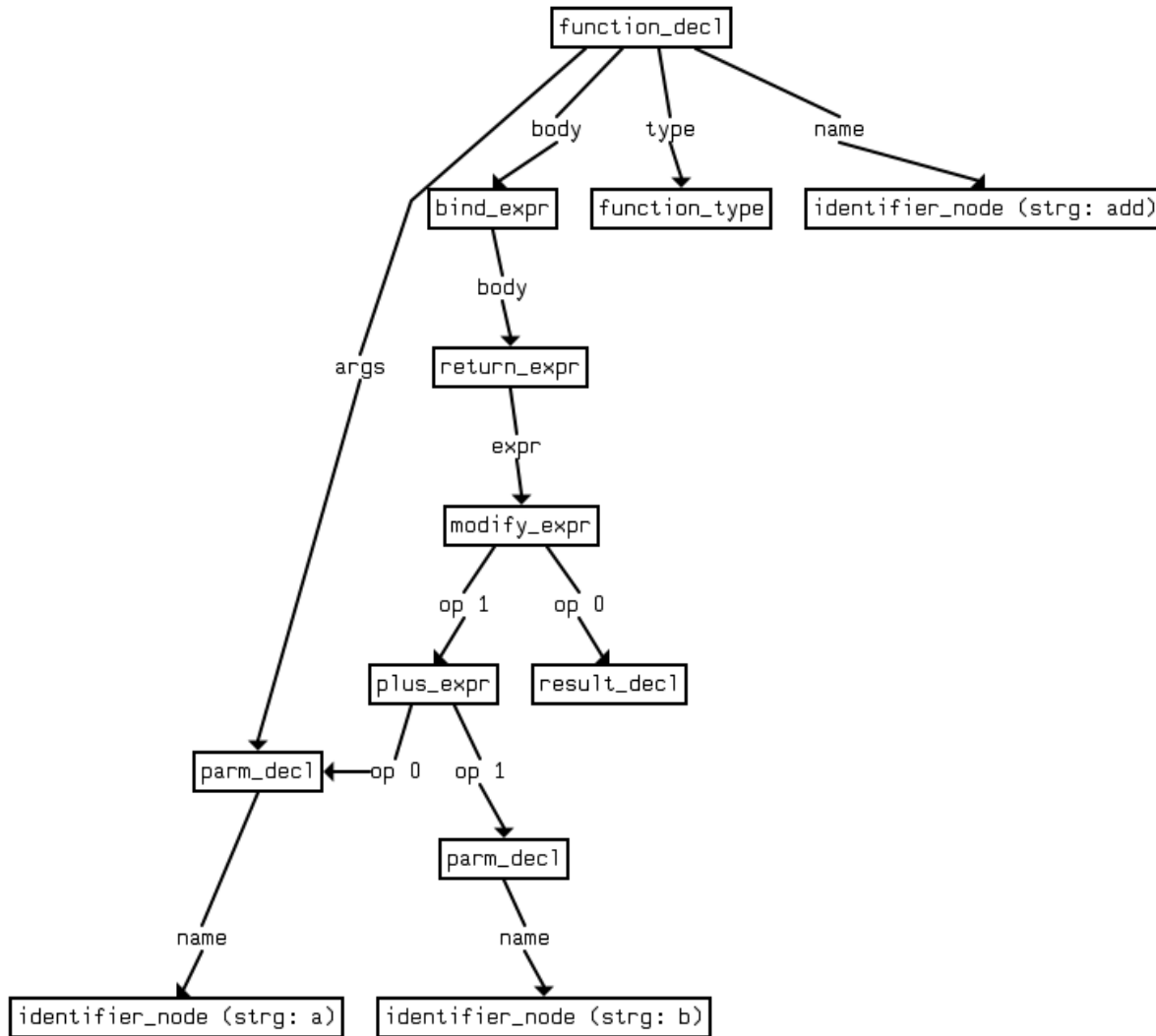
struct tree_common
{
  struct tree_base base;
  tree chain;
  tree type;
  ...
};

union tree_node
{
  struct tree_base base;
  struct tree_common common;
  ...
};

typedef union tree_node *tree;
  
```



# Example: Tree



```
$ cat test.c
int
add (int a, int b)
{
    return a + b;
}
```

```
(gdb) b c_generic
(gdb) r
(gdb) p debug_function (fndecl, 4)
```

# Middle End



- GIMPLE
- Call Graph (also for back-end)
- Passes (also for back-end)

# Proceed to Middle End



```
compile_file                                toplev.c
  lang_hooks.parse_file                    toplev.c
  lang_hooks.final_write_globals           toplev.c
    c_write_global_declarations            c-decl.c
      cgraph_finalize_compilation_unit     c-decl.c
        cgraph_analyze_functions           cgraphunit.c
          cgraph_optimize                  cgraphunit.c
```

```
/* We're done parsing; proceed to optimize and emit assembly.
   FIXME: shouldn't be the front end's responsibility to call this. */
cgraph_finalize_compilation_unit ();
```

# GIMPLE



- Derived from GENERIC
  - Tuple representation, no more than 3 operands
  - `typedef union gimple_statement_d *gimple;`
- Gimplifier
  - `GENERIC → GIMPLE` (currently `tree → GIMPLE`)

<code>cgraph_analyze_function</code>	<code>cgraphunit.c</code>
<code>gimplify_function_tree</code>	<code>gimplify.c</code>
<code>...</code>	
<code>gimplify_expr</code>	<code>gimplify.c</code>
<code>lang_hooks.gimplify_expr</code>	
<code>(c_gimplify_expr)</code>	<code>c-gimplify.c</code>

# Example: GIMPLE



```
test (int a, int b, int c)
{
  if (foo (a + b, c) != 0)
    {
      c = b++ / a;
    }
  return c;
}
```

GENERIC

```
test (int a, int b, int c)
{
  int D.1239;
  int D.1240;
  int D.1243;

  D.1239 = a + b;
  D.1240 = foo (D.1239, c);
  if (D.1240 != 0) goto <D.1241>; else goto <D.1242>;
<D.1241>:
  c = b / a;
  b = b + 1;
<D.1242>:
  D.1243 = c;
  return D.1243;
}
```

GIMPLE



# Call Graph

- Call graph is a directed multigraph
  - Nodes are functions
  - Edges are call sites

```
struct cgraph_node
{
    tree decl;
    struct cgraph_edge *callees;
    struct cgraph_edge *callers;
    struct cgraph_node *next;
    struct cgraph_node *previous;
    ...
};
```

- Build cgraph
  - `cgraph_create_node`
  - `cgraph_finalize_function`
  - `cgraph_finalize_compilation_unit`

# A Pass



```
struct gimple_opt_pass pass_remove_useless_stmts =
{
  {
    GIMPLE_PASS,
    "useless",          /* name */
    NULL,               /* gate */
    remove_useless_stmts, /* execute */
    NULL,              /* sub */
    NULL,              /* next */
    0,                 /* static_pass_number */
    0,                 /* tv_id */
    PROP_gimple_any,  /* properties_required */
    0,                 /* properties_provided */
    0,                 /* properties_destroyed */
    0,                 /* todo_flags_start */
    TODO_dump_func    /* todo_flags_finish */
  }
};
```

# Pass List



- › Lowering passes
- › Inter-procedural passes
  - › Early inline passes
  - › Early local passes
  - › IPA passes
- › Intra-procedural passes
  - › GIMPLE passes
  - › RTL passes

```
void
init_optimization_passes (void)
{
    ...
    p = &all_lowering_passes;
    NEXT_PASS (pass_remove_useless_stmts);
    NEXT_PASS (pass_mudflap_1);
    NEXT_PASS (pass_lower_omp);
    NEXT_PASS (pass_lower_cf);
    NEXT_PASS (pass_refactor_eh);
    NEXT_PASS (pass_lower_eh);
    NEXT_PASS (pass_build_cfg);
    NEXT_PASS (pass_lower_complex_00);
    NEXT_PASS (pass_lower_vector);
    NEXT_PASS (pass_warn_function_return);
    NEXT_PASS (pass_build_cgraph_edges);
    NEXT_PASS (pass_inline_parameters);
    *p = NULL;
    ...
}          /* passes.c */
```



# Execute Passes

## ➤ Lowering passes

```
cgraph_finalize_compilation_unit ()  
  for each node N in the cgraph  
    cgraph_analyze_function (N)  
    cgraph_lower_function (N) -> all_lowering_passes
```

## ➤ Inter-procedural & Intra-procedural passes

```
cgraph_optimize ()  
  ipa_passes () -> all_ipa_passes  
  cgraph_expand_all_functions ()  
  for each node N in the cgraph  
    cgraph_expand_function (N)  
    tree_rest_of_compilation (DECL (N)) -> all_passes
```

# Back End & Port



- RTL
- Machine Description
- Target Macros

# RTL



- Low-level IR

  - Variable → Register
  - Reference → Memory
  - Type → Machine Mode

- Insns

  - RTL representation for a function, double-linked chain

```
/* An instruction that cannot jump. */  
DEF_RTL_EXPR(INSN, "insn", "iuuBieie", RTX_INSN)
```

- GIMPLE → Tree → RTL

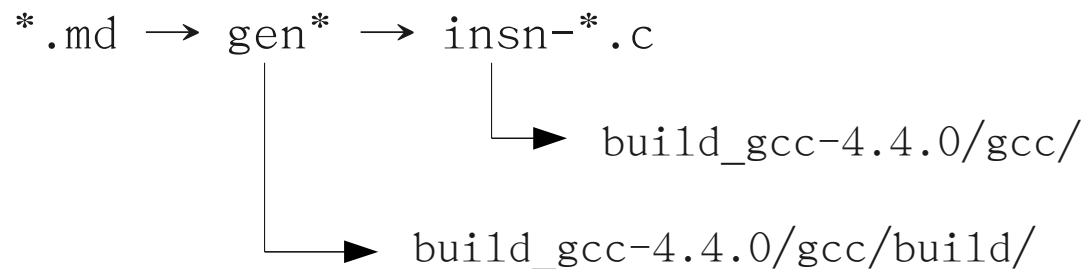
  - pass\_expand



# Machine Description

- Describe Instructions, etc.
  - Define Instruction Patterns
  - Define Predicates
  - Define Constraints
  - Define Attributes
  - Define Delay Slot
  - Define Processor Pipeline
  - Define Peephole
  - Define Iterators

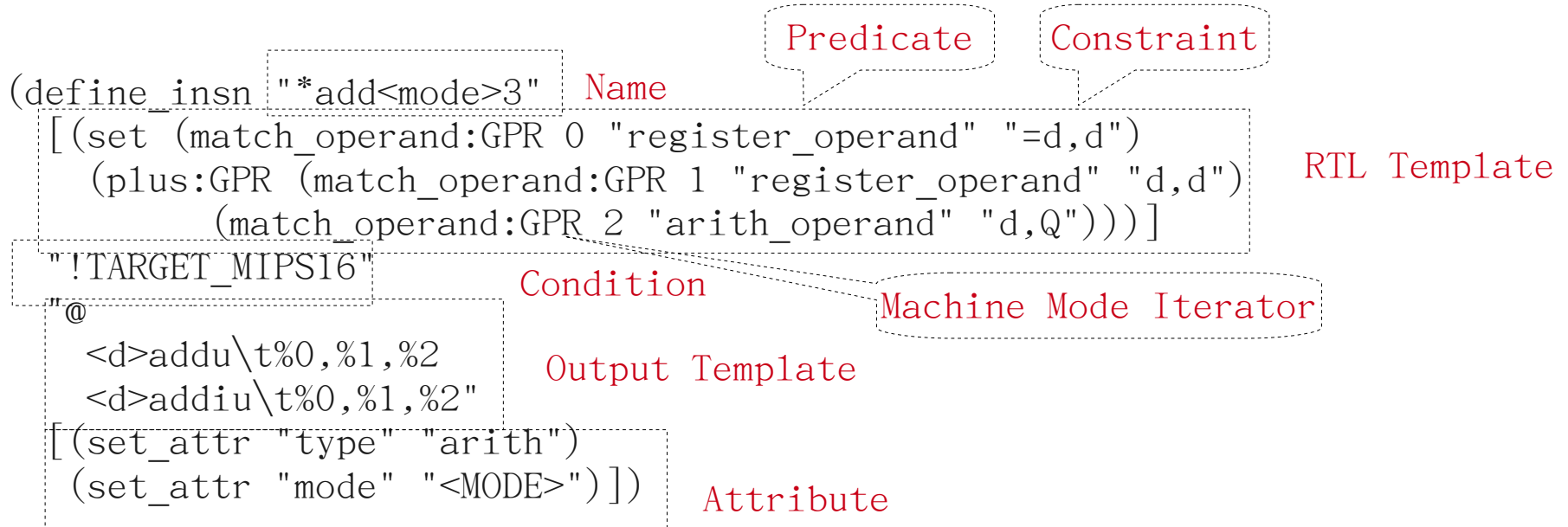
➤ MD → c





# Example: MD

```
(define_expand "add<mode>3"
  [(set (match_operand:GPR 0 "register_operand")
        (plus:GPR (match_operand:GPR 1 "register_operand")
                  (match_operand:GPR 2 "arith_operand")))]
  "")
```





# Example: MD

Gimple Statement

```
D.1237 = a + b;
```

RTL insn generated according to `addsi3` pattern

```
(insn 9 8 10 3 add.c:4 (set (reg:SI 193 [ D.1237 ])
  (plus:SI (reg:SI 195)
    (reg:SI 196))) -1 (nil))
```

RTL insn recognized according to `*addsi3` pattern

```
(insn 9 8 10 2 add.c:4 (set (reg:SI 193 [ D.1237 ])
  (plus:SI (reg:SI 195)
    (reg:SI 196))) 10 {*addsi3} (nil))
```



# Target Macros

- Target Hooks
  - `target.h`, `target-def.h`, `targhooks.c`, `targethooks.h`,  
`machine.h`, `machine.c`
- Macros
  - Storage Layout
  - Type Layout
  - Registers
  - Stack & Calling
  - Addressing Modes
  - Costs
  - Scheduling
  - Sections
  - Assembler Format
  - Debugging Info
- ...

# How to Port



- It depends
  - What does your target need to describe?
  - From scratch or base on the existing?
  - Contribute or not?
- Suggestions
  - Be familiar with gcc internals
  - Refer to an example
  - From simple to complex

# Example: Picochip



- Story about contribution

03 Mar 2003, Dan Towner:

For the last 18 months, I have been developing a port of...

10 Mar 2008 David Edelsohn:

I am pleased to announce that the GCC Steering Committee has...

- `gcc-4.4.0/gcc/config/picochip/`

`constraints.md dfa_space.md dfa_speed.md picochip.md predicates.md`

`picochip.c picochip.h picochip.opt picochip-protos.h t-picochip`

`libgccExtras/`

- Build cross gcc

`--target=picochip-unknown-none`



Thanks !