Workshop on Essential Abstractions in GCC

Understanding GCC: The Essential Abstractions in GCC

GCC Resource Center

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

Department of Computer Science and Engineering,

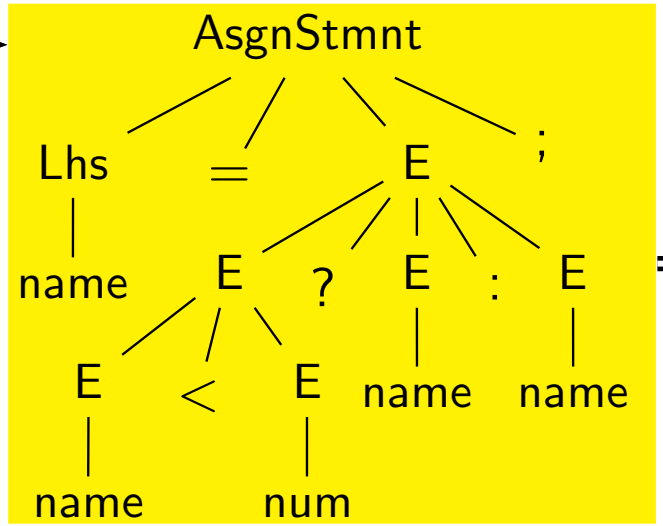
Indian Institute of Technology, Bombay

July 2010

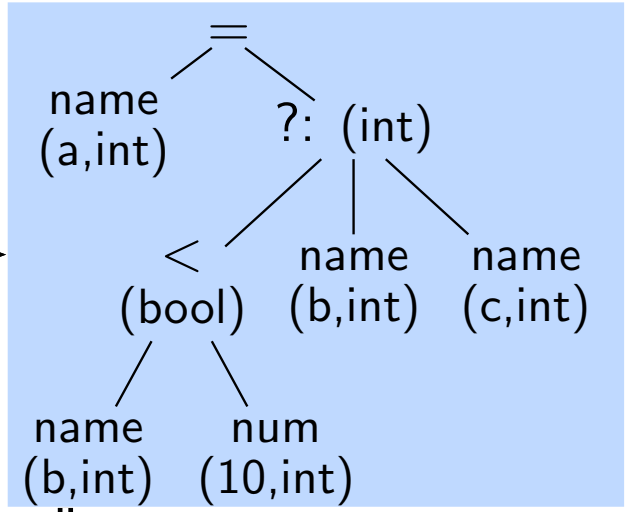
Translation Sequence in Our Toy Compiler

`a=b<10?b:c;`

Input

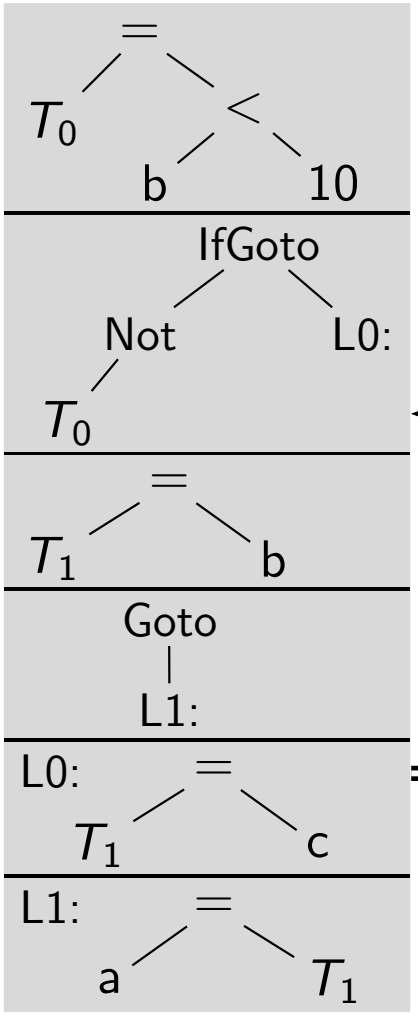


Parse Tree



Abstract Syntax Tree
(with attributes)

Tree List



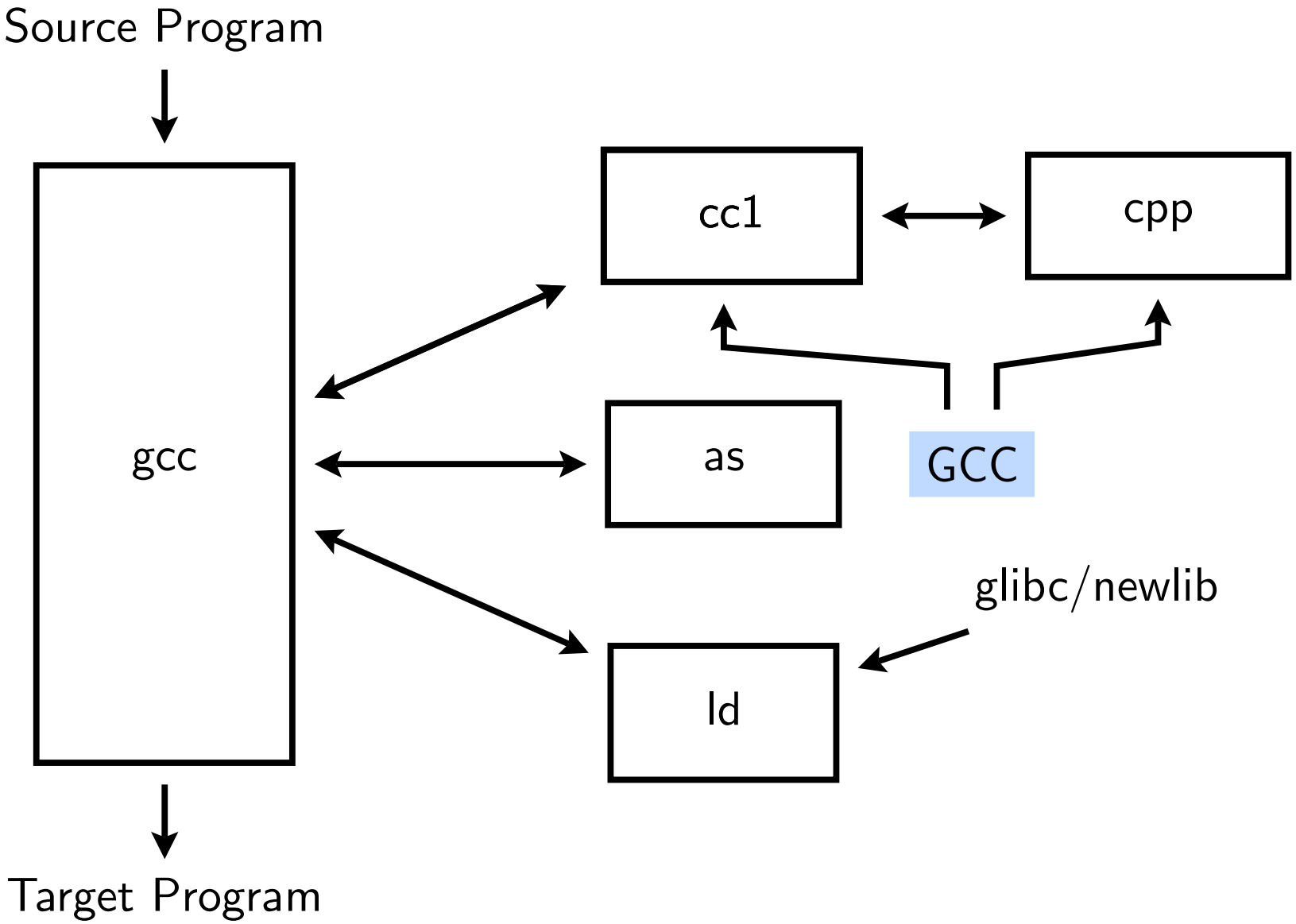
Instruction List

```
     T0 ← b
     T0 ← T0 < 10
     T0 ← ! T0
     if T0 > 0 goto L0:
     T1 ← b
     goto L1:
L0:  T1 ← c
L1:  a ← T1
```

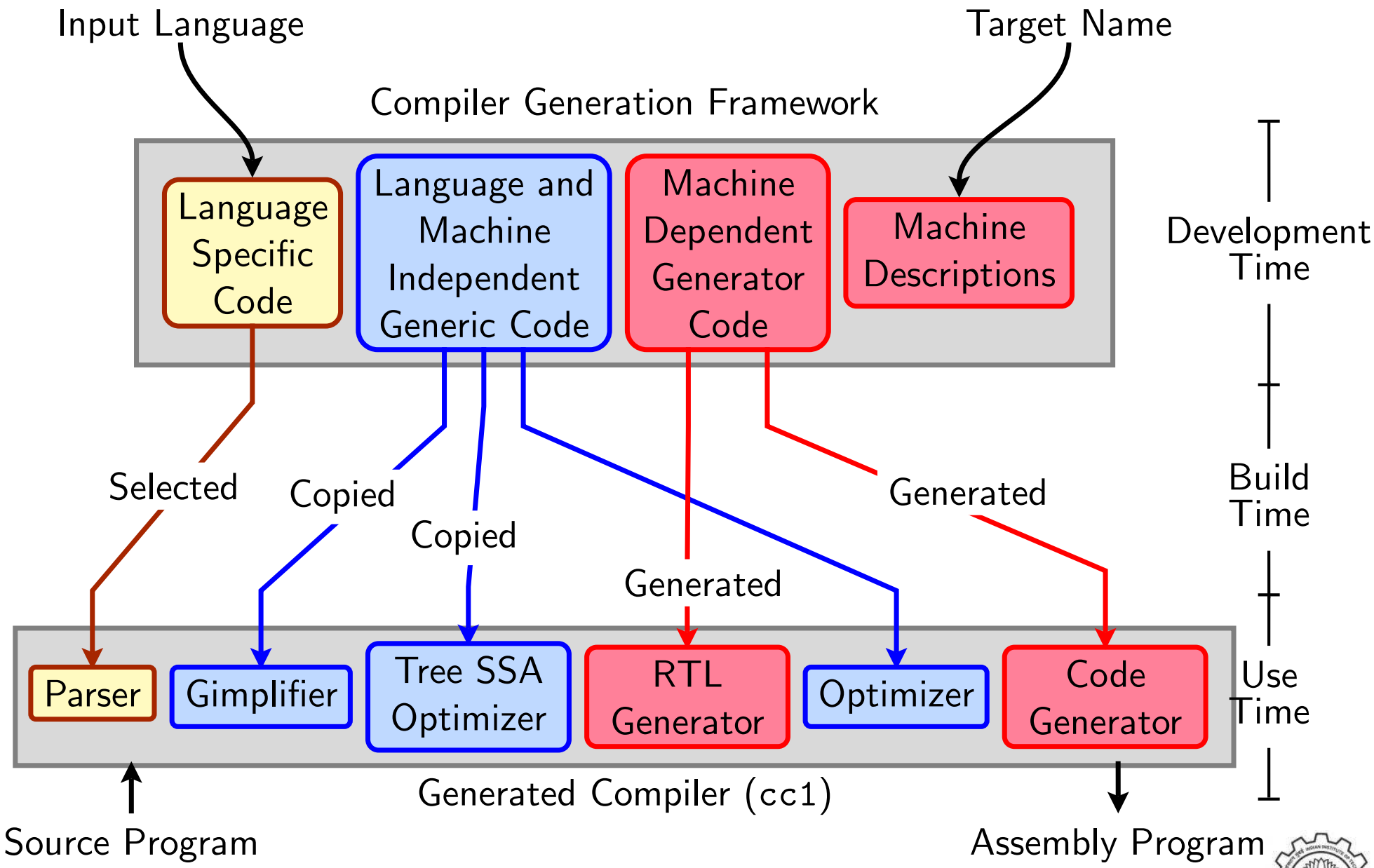
Assembly Code

```
     lw   $t0, 4($fp)
     slti $t0, $t0, 10
     not  $t0, $t0
     bgtz $t0, L0:
     lw   $t0, 4($fp)
     b    L1:
L0:  lw   $t0, 8($fp)
L1:  sw   0($fp), $t0
```

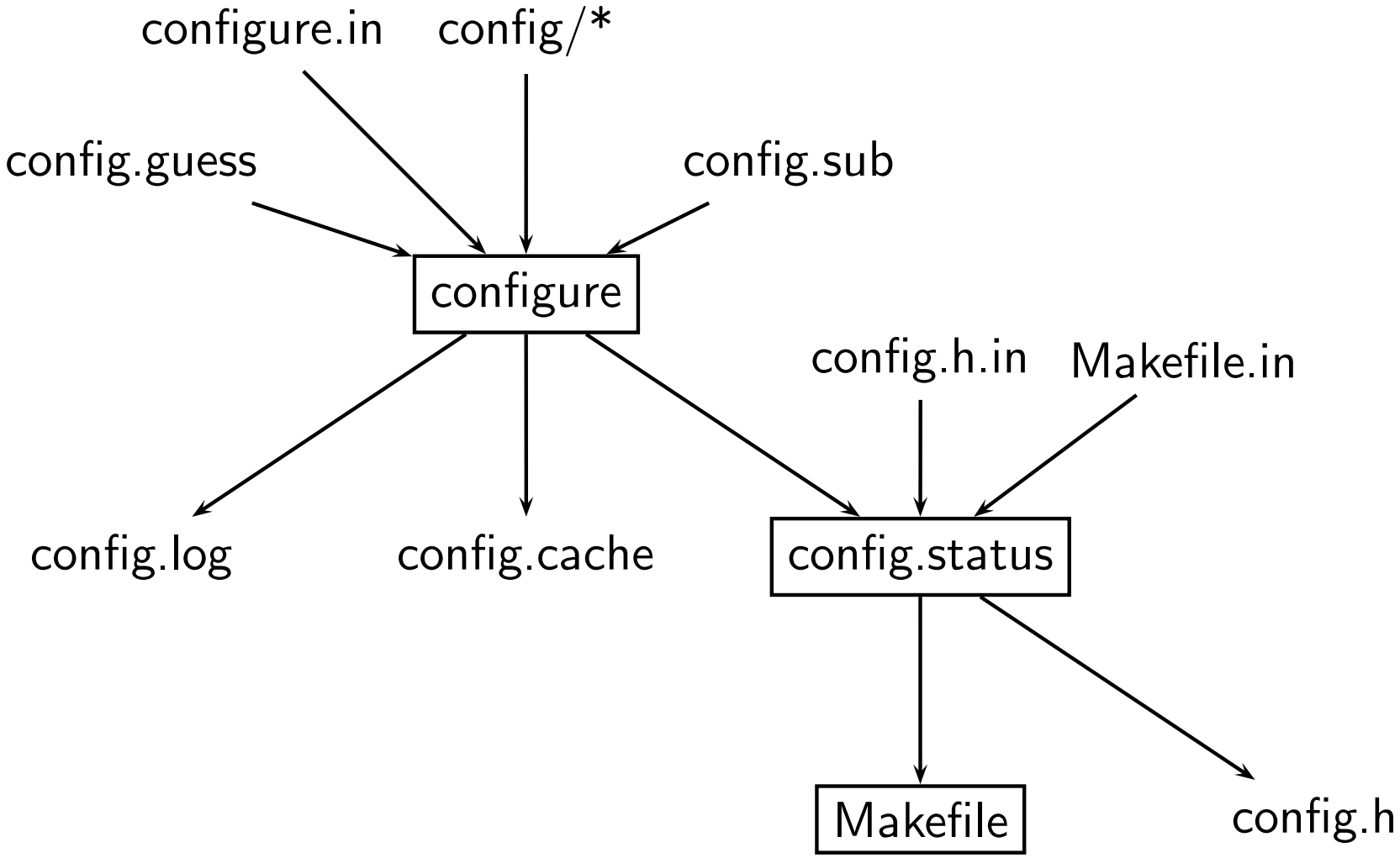
The GNU Tool Chain



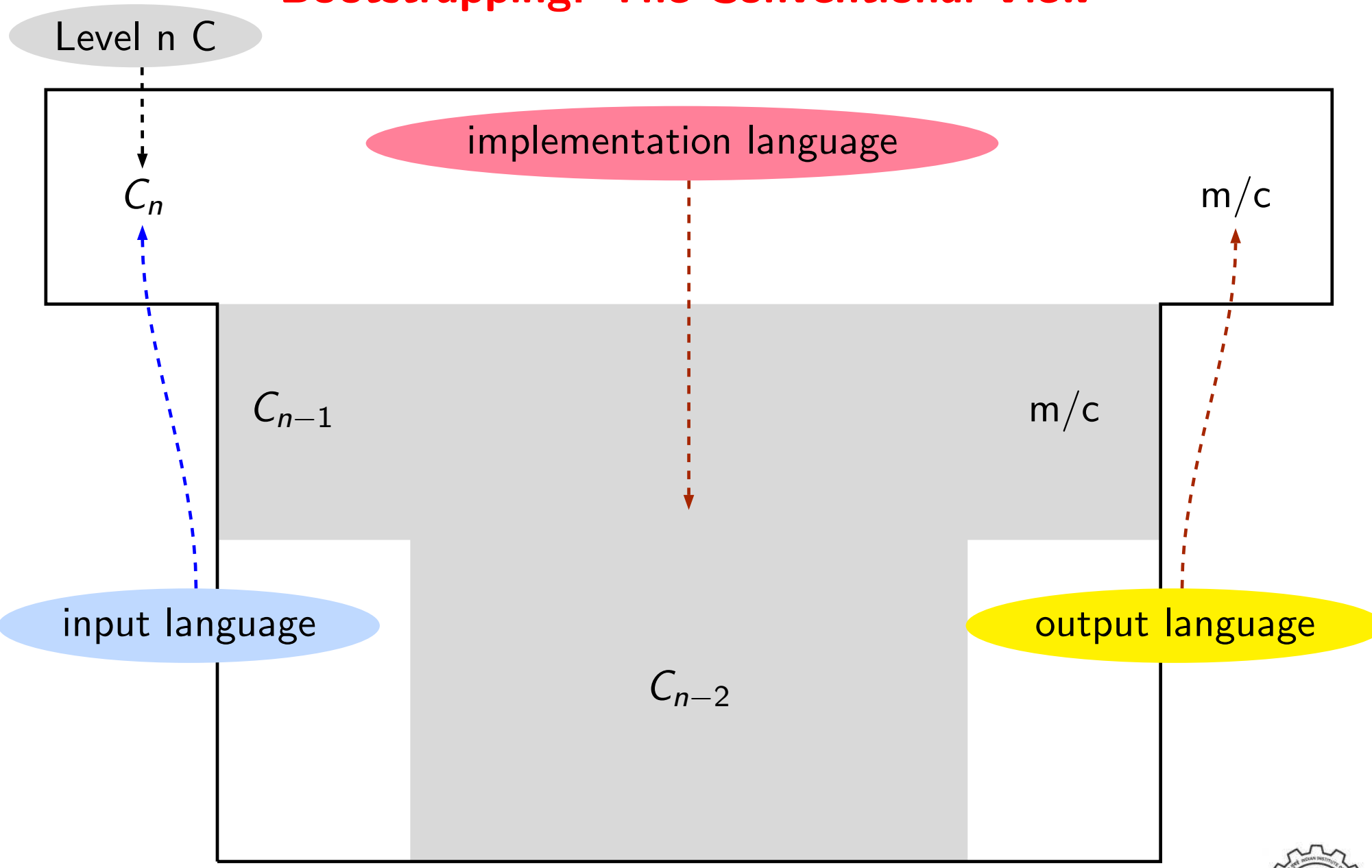
The Architecture of GCC



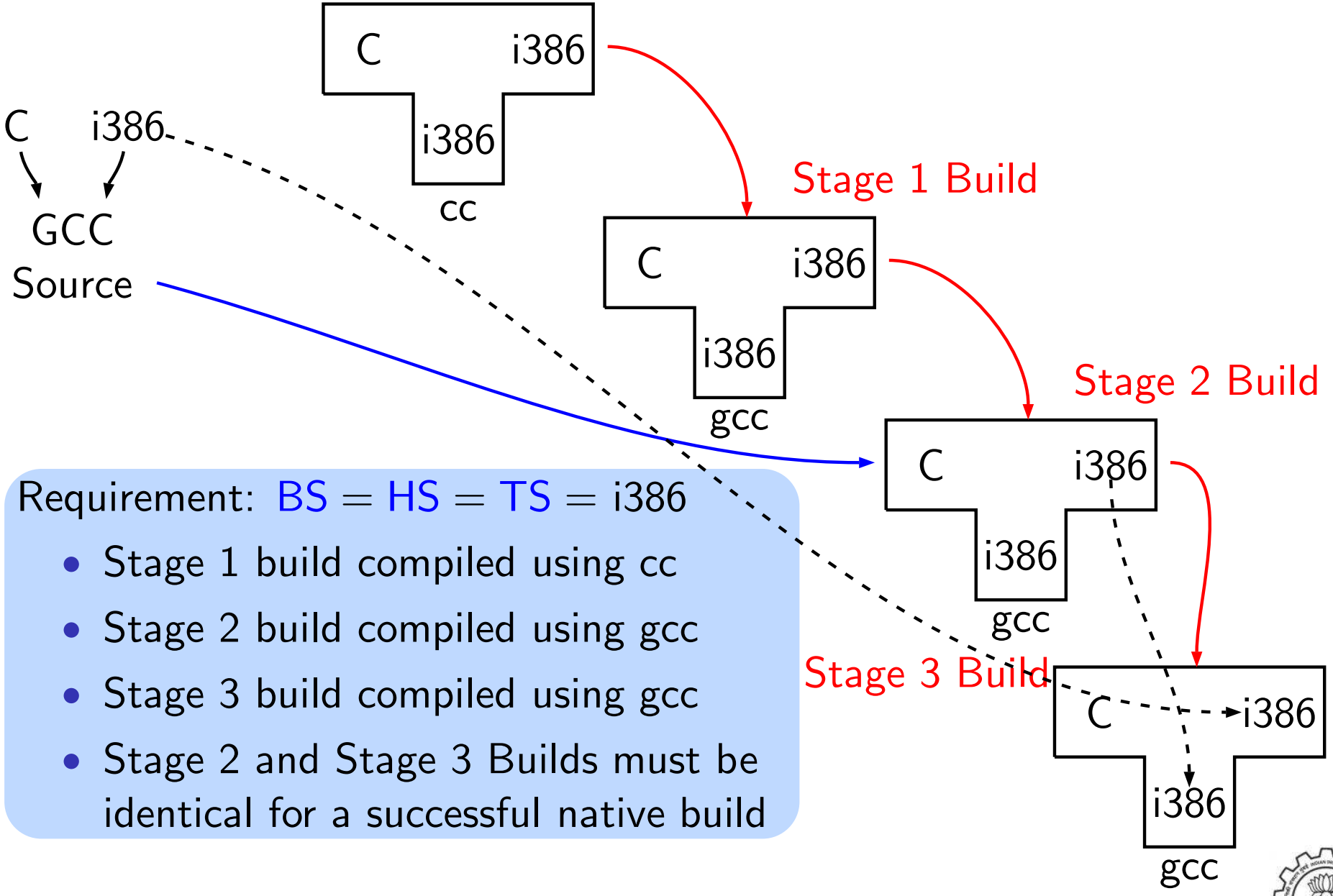
Configuring GCC



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

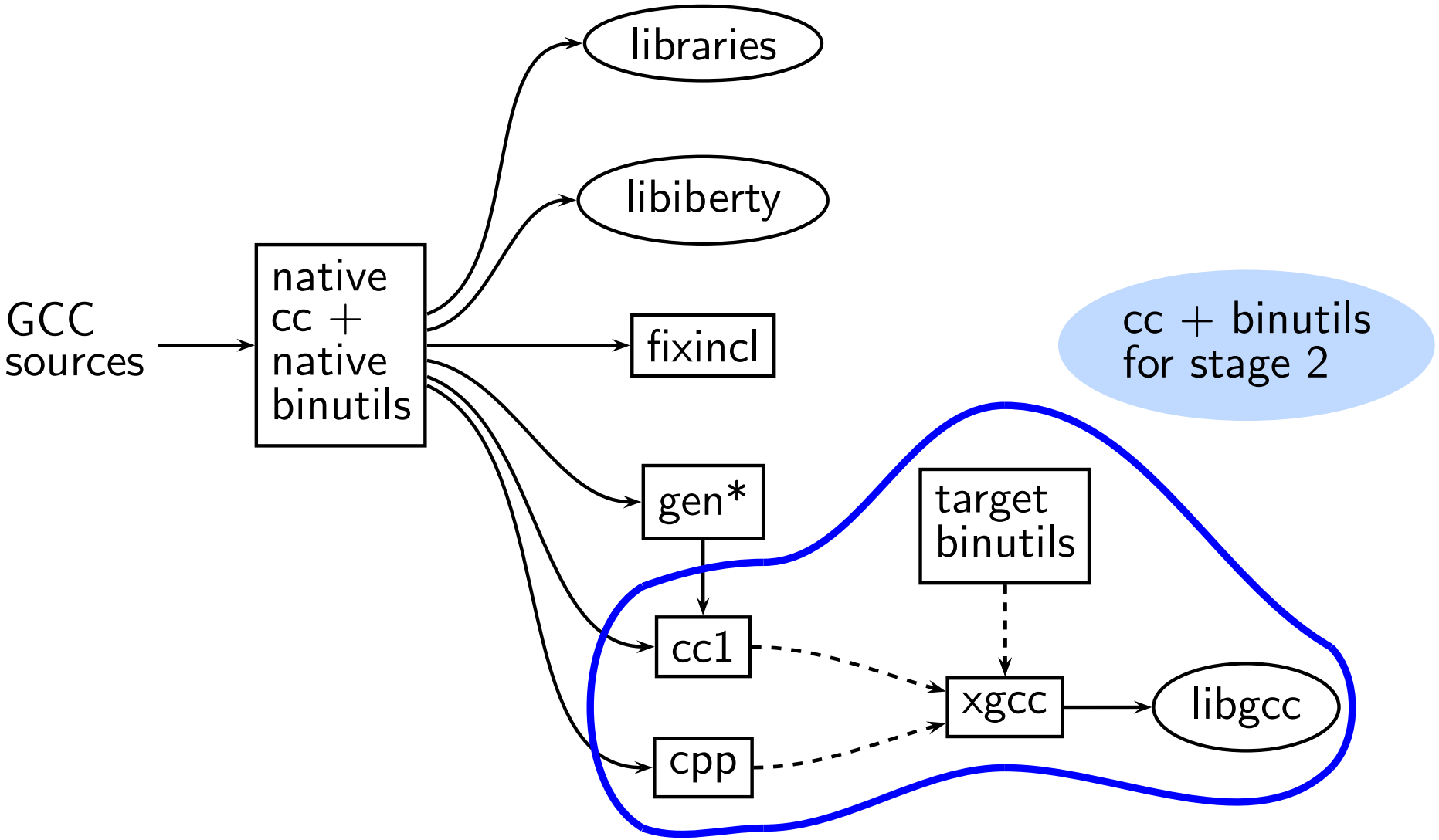
Build for a Given Machine

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)

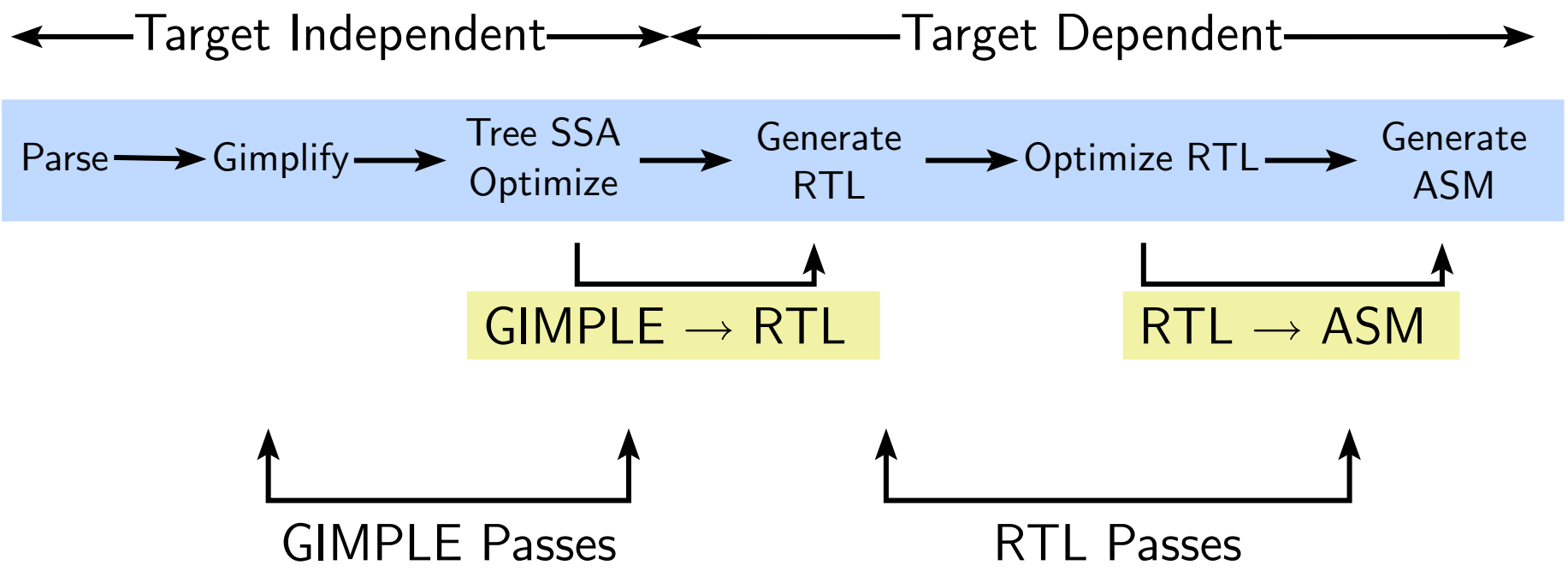
```
genattr
gencheck
genconditions
genconstants
genflags
genopinit
genpreds
genattrtab
genchecksum
gencondmd
genemit
gengenrtl
genmddeps
genoutput
genrecog
genautomata
gencodes
genconfig
genextract
gengtype
genmodes
genpeep
```

More Details of an Actual Stage 1 Build for C

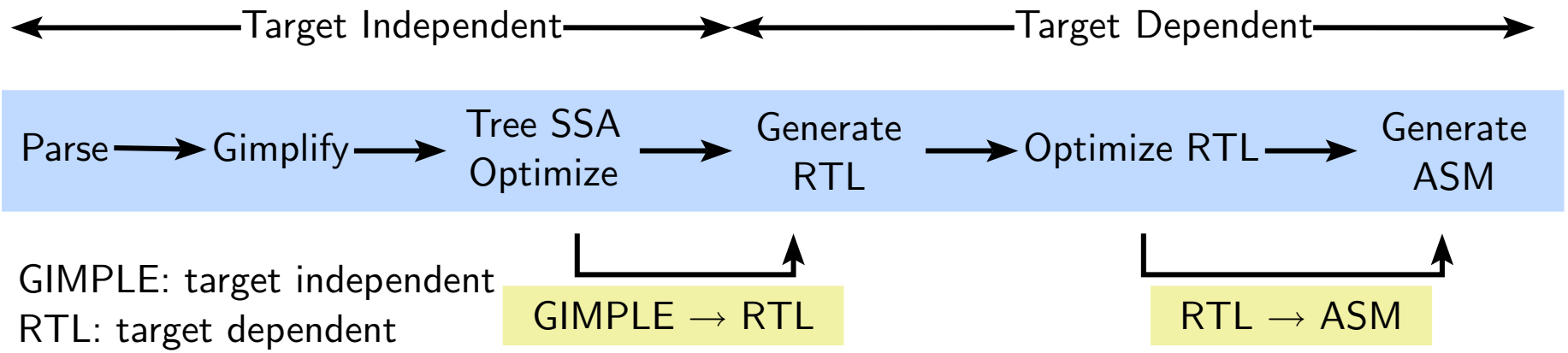


Basic Transformations in GCC

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

GIMPLE_ASSIGN

RTL Template

ASM

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

Translation Sequence in GCC

```
(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"
)
```

Development

```
D.1283 = 10;
```

⟹

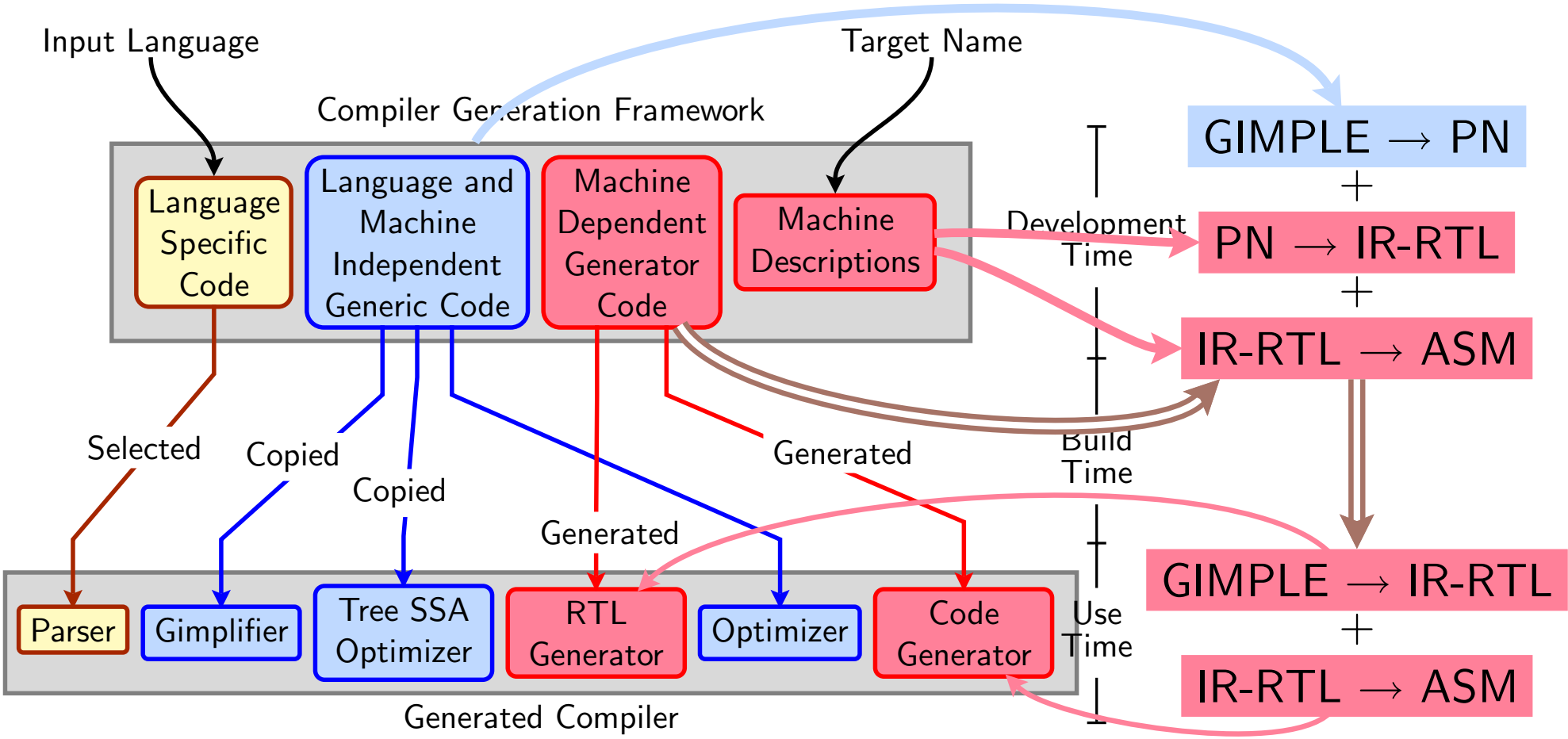
```
(set
    (reg:SI 58 [D.1283])
    (const_int 10:  [0xa])
)
```

⟹

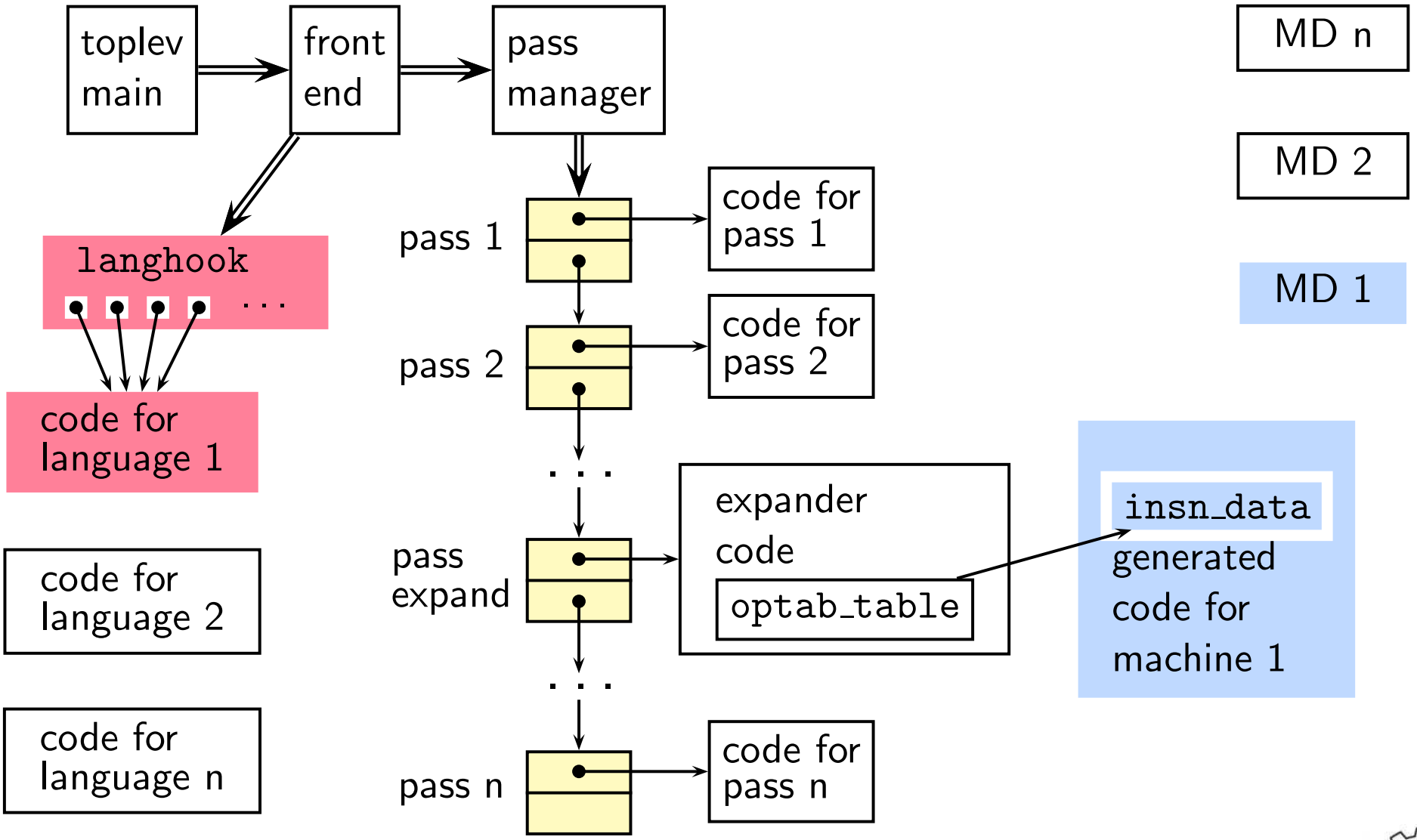
```
li $t0, 10
```

Use

Retargetability Mechanism of GCC



Plugin Structure in cc1



Hooking up Back End Details

$(SOURCE)/gcc/optabs.h
$(SOURCE)/gcc/optabs.c

optab_table

...	...
	mov_optab

OTI_mov

SI	insn_code CODE_FOR_movsi
SF	insn_code CODE_FOR_nothing

Runtime initialization of data structure

$(BUILD)/gcc/insn-output.c

insn_data

...	...
1280	"movsi" ... gen_movsi ...

$BUILD/gcc/insn-codes.h

CODE_FOR_movsi=1280
CODE_FOR_movsf=CODE_FOR_nothing

$BUILD/gcc/insn-opinit.c

...