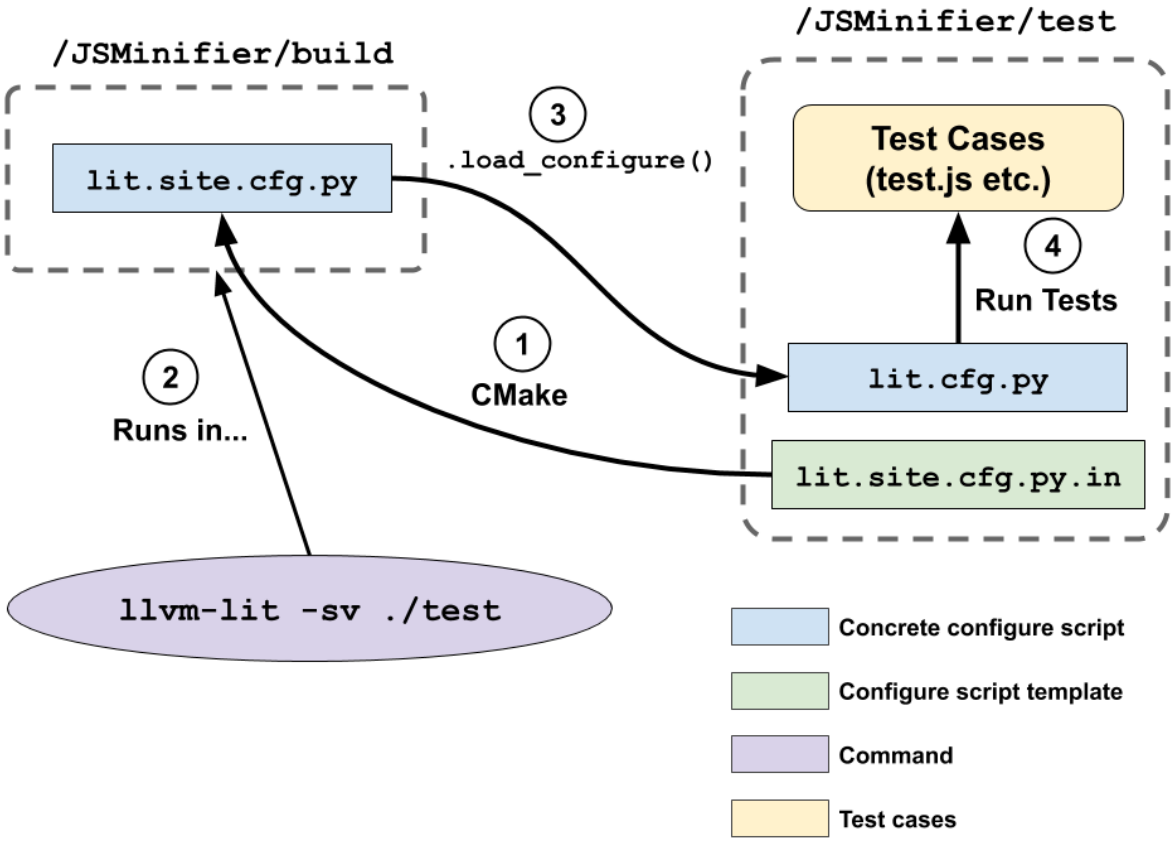
Chapter 1: Saving resources when building LLVM from source

No images…

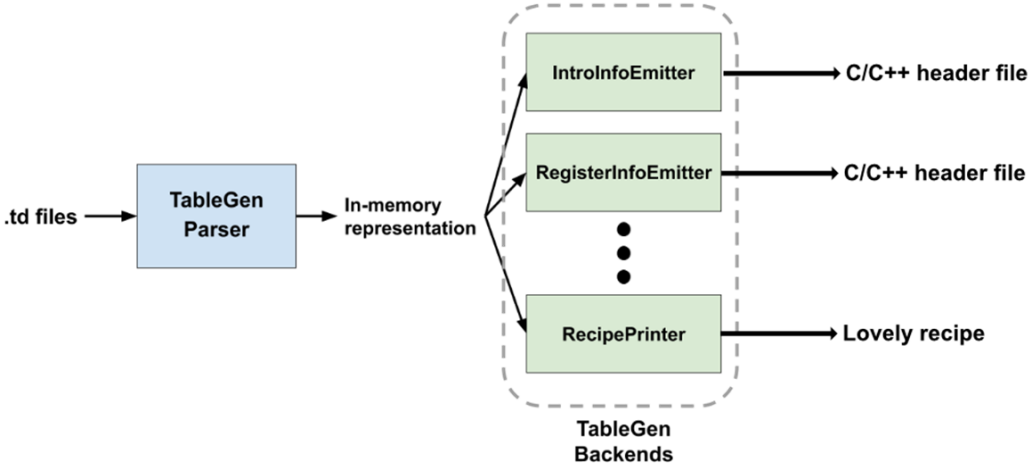
Chapter 2: Exploring LLVM's Build System Features

No images…

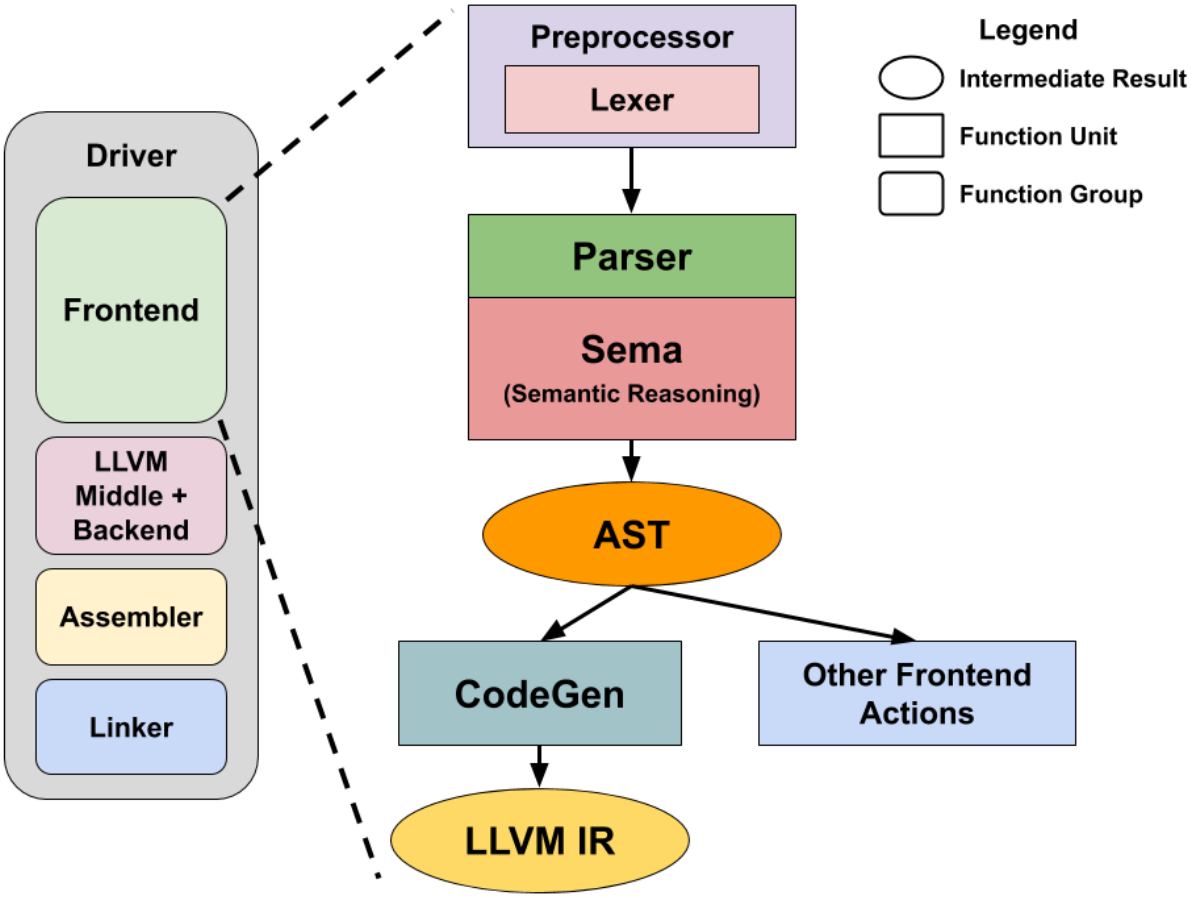
Chapter 3: Advanced usages of LLVM LIT



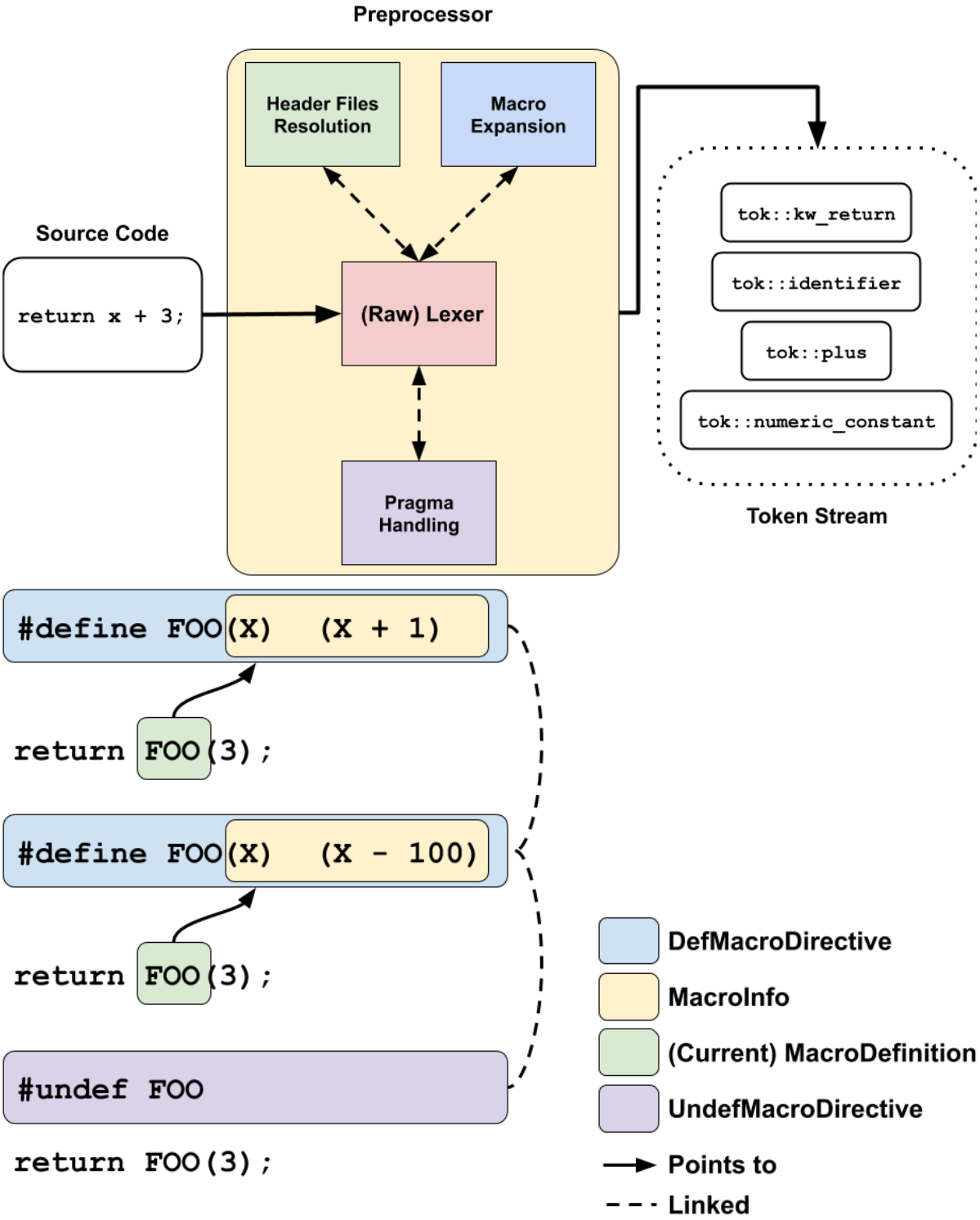
Chapter 4: TableGen Development: For Fun and For Profit



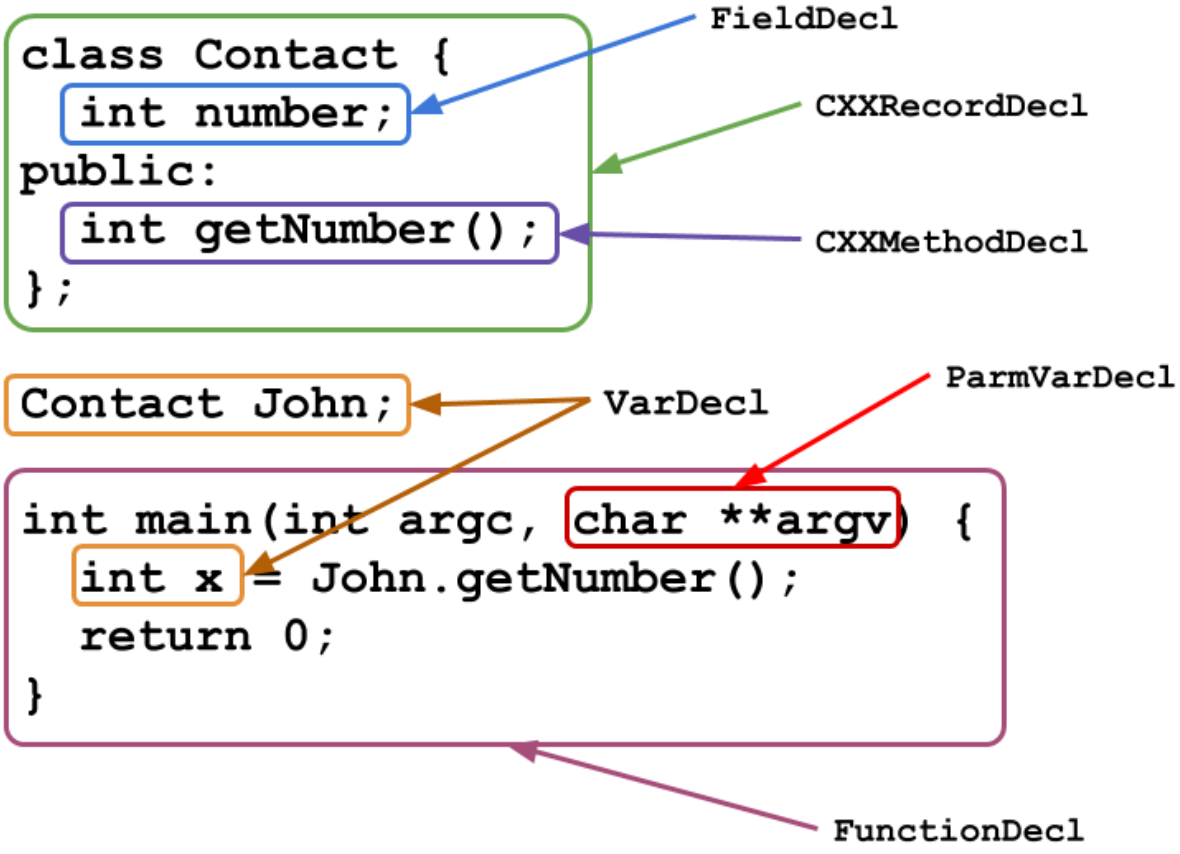
Chapter 5: Exploring Clang's Architecture

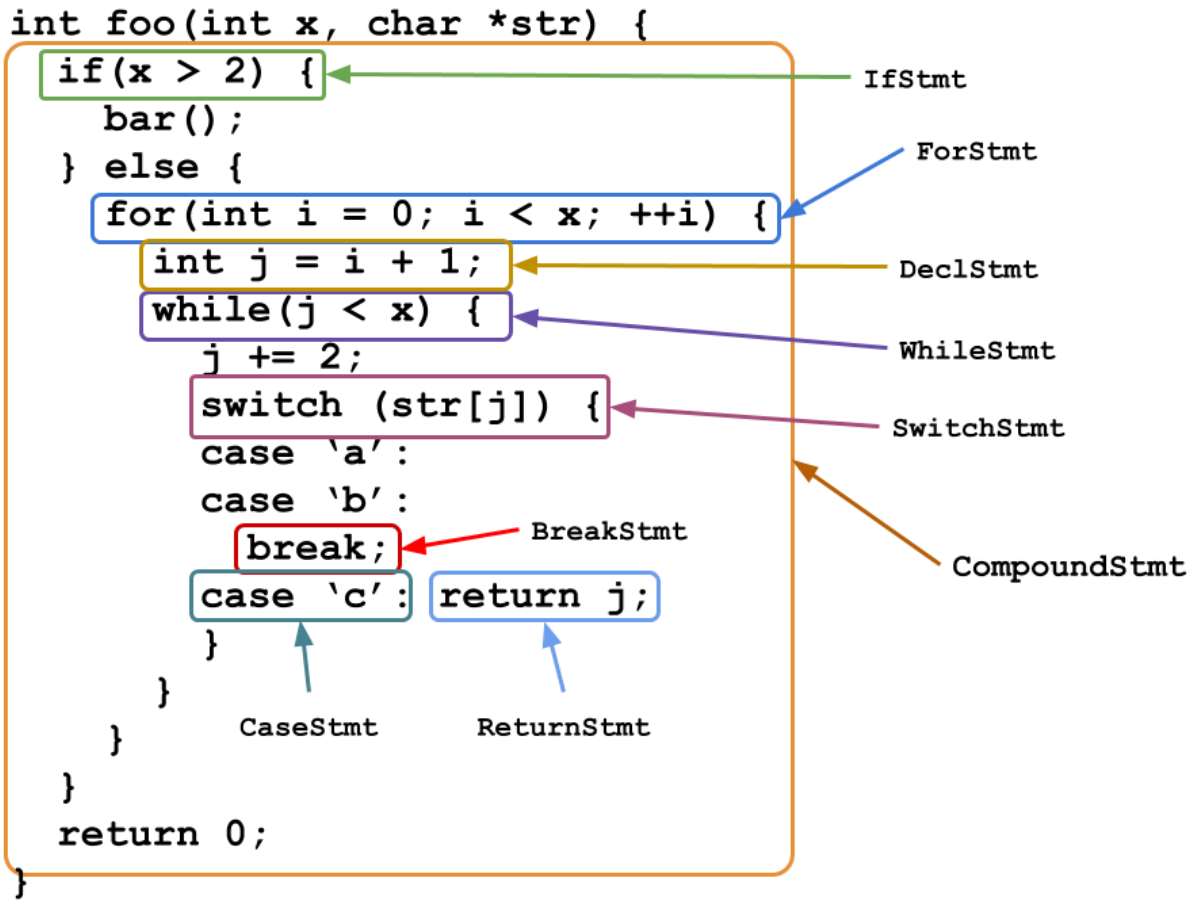


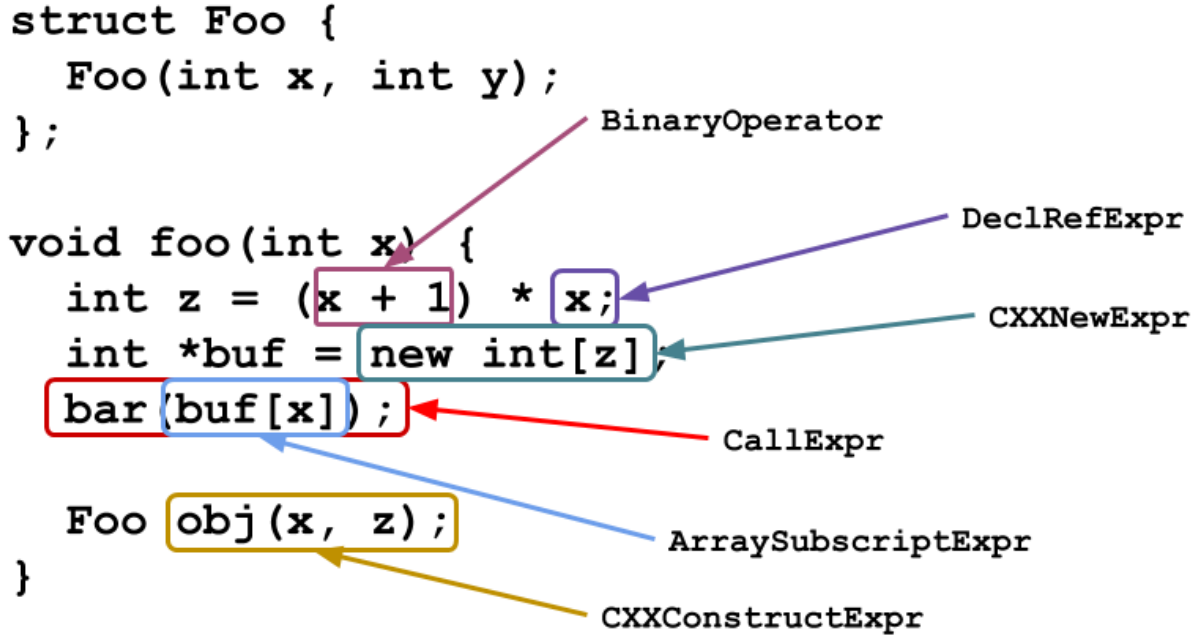
Chapter 6: Extending the preprocessors



Chapter 7: Handling AST




int foo(int x, char *str) {
if(x > 2) {
bar();
} else {
for(int i = 0; i < x; ++i) {
int j = i + 1;
while(j < x) {
j += 2;
switch (str[j]) {
case 'a':
case 'b':
break;
case 'c':
return j;
}
}
}
}
return 0;
}
IfStmt
ForStmt
DeclStmt
WhileStmt
SwitchStmt
CompoundStmt
BreakStmt
CaseStmt
ReturnStmt

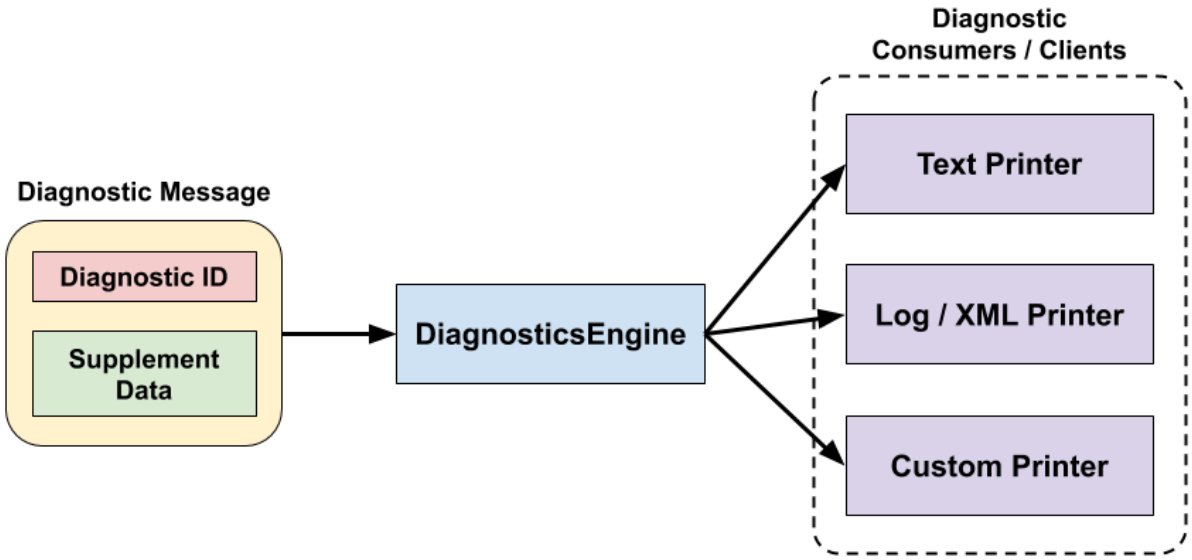

struct Foo {
Foo(int x, int y);
};
void foo(int x) {
int z = (x + 1) * x;
int *buf = new int[z];
bar(buf[x]);
Foo obj(x, z);
}
BinaryOperator
DeclRefExpr
CXXNewExpr
CallExpr
ArraySubscriptExpr
CXXConstructExpr

Matcher<Stmt>	expr	Matcher<Expr>...
Matcher<Stmt>	exprWithCleanups	Matcher<ExprWithCleanups>...
Matcher<Stmt>	fixedPointLiteral	Matcher<FixedPointLiteral>...
Matcher<Stmt>	floatLiteral	Matcher<FloatingLiteral>...
Matcher<Stmt>	forStmt	Matcher<ForStmt>...

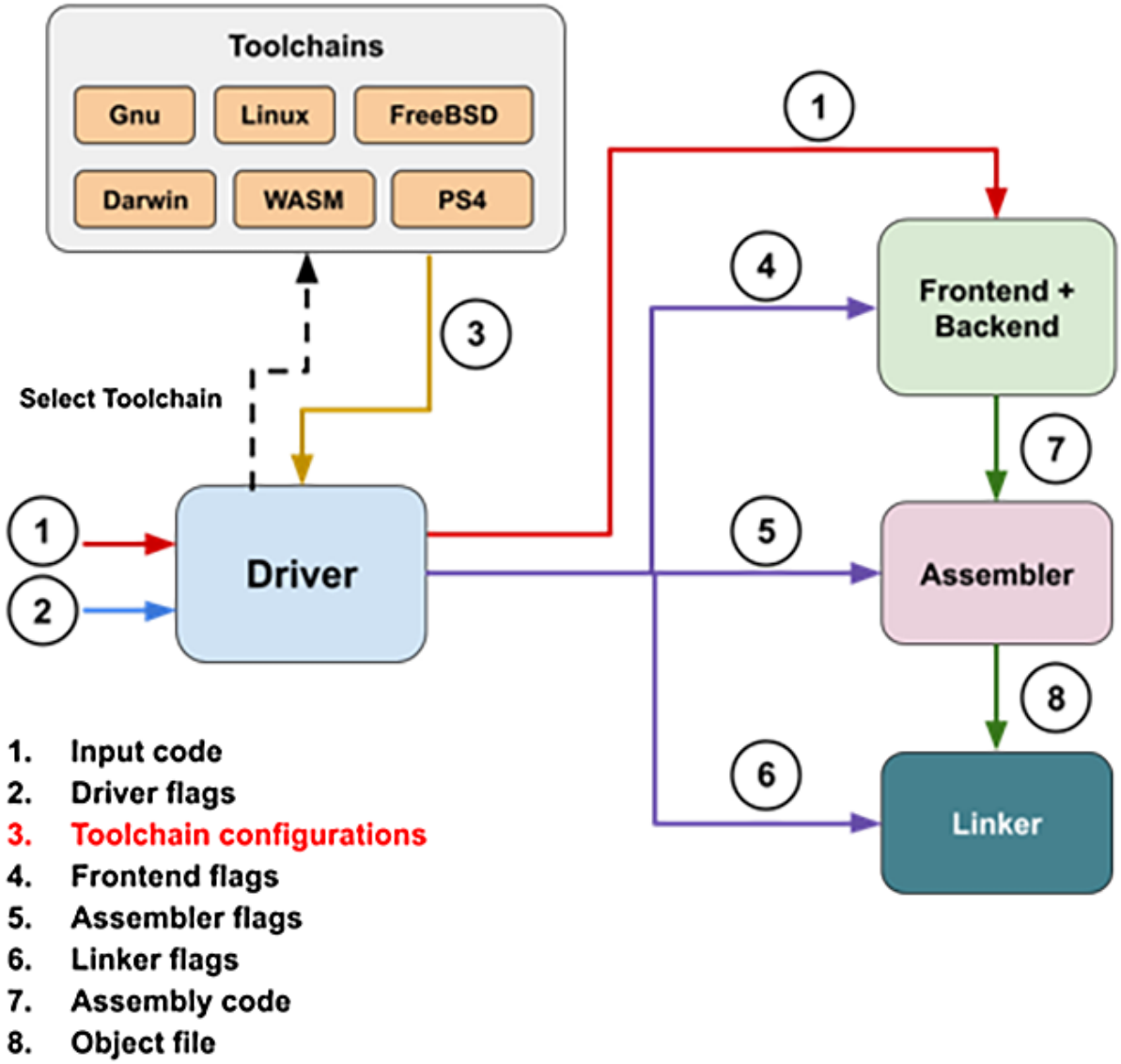
Return type

DSL directive

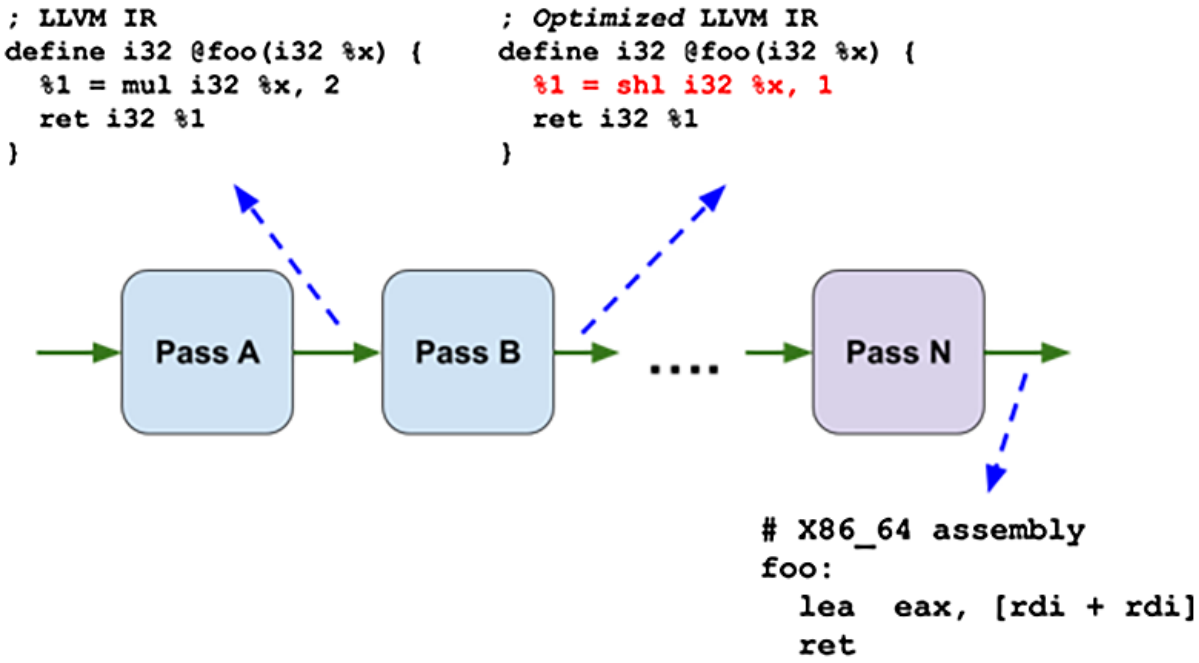
Argument type(s)



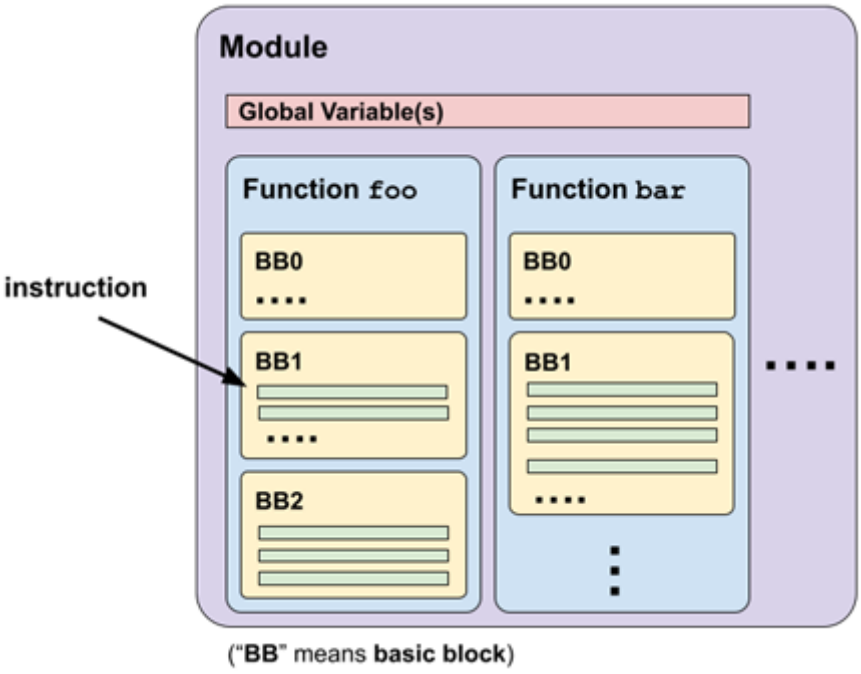
Chapter 8: Working with Compiler Flags and Toolchains



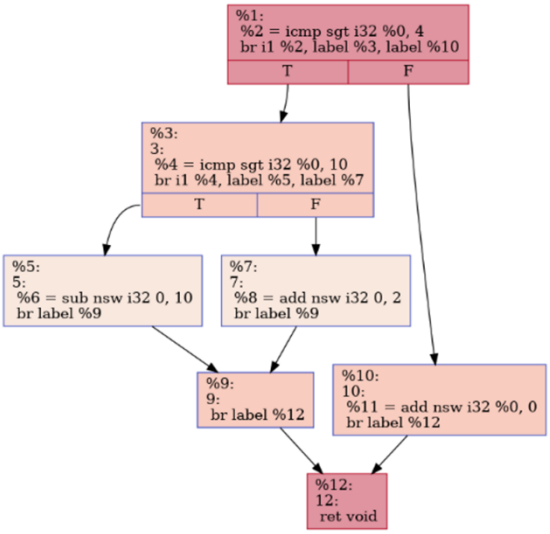
Chapter 9: All you need to know about the new PassManager and AnalysisManager



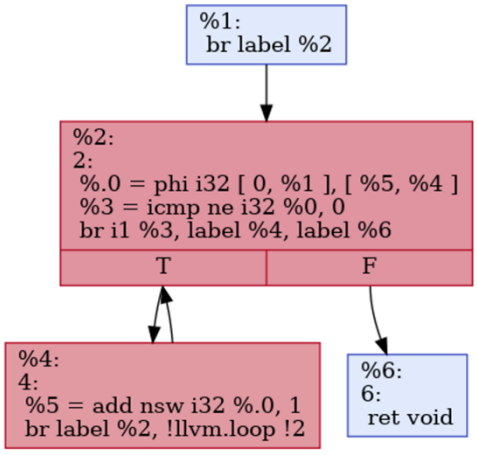
Chapter 10: Processing IR in a proper way



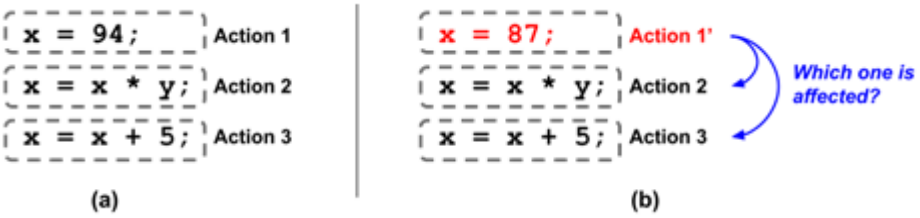

```
target datalayout = "..."
target triple = "x86_64-..."

; Function Attrs: ...
define i32 @foo(i32 %0, i32 %1) #0 {
  %3 = alloca i32, align 4
  %4 = alloca i32, align 4
  store i32 %0, i32* %3, align 4
  store i32 %1, i32* %4, align 4
  %5 = load i32, i32* %3, align 4
  %6 = icmp sgt i32 %5, 0
  br i1 %6, label %7, label %11

7:                                                ; preds = %2
  %8 = load i32, i32* %3, align 4
  %9 = load i32, i32* %4, align 4
  %10 = sub nsw i32 %8, %9
  br label %15

11:                                               ; preds = %2
  %12 = load i32, i32* %3, align 4
  %13 = load i32, i32* %4, align 4
  %14 = add nsw i32 %12, %13
  br label %15

15:                                               ; preds = %11, %7
  %16 = phi i32 [ %10, %7 ], [ %14, %11 ]
  ret i32 %16
}
```

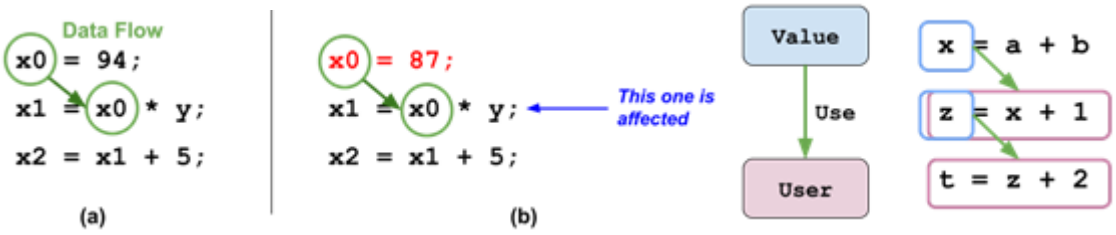


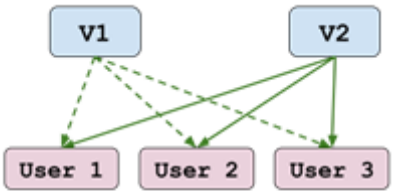
CFG for 'foo' function



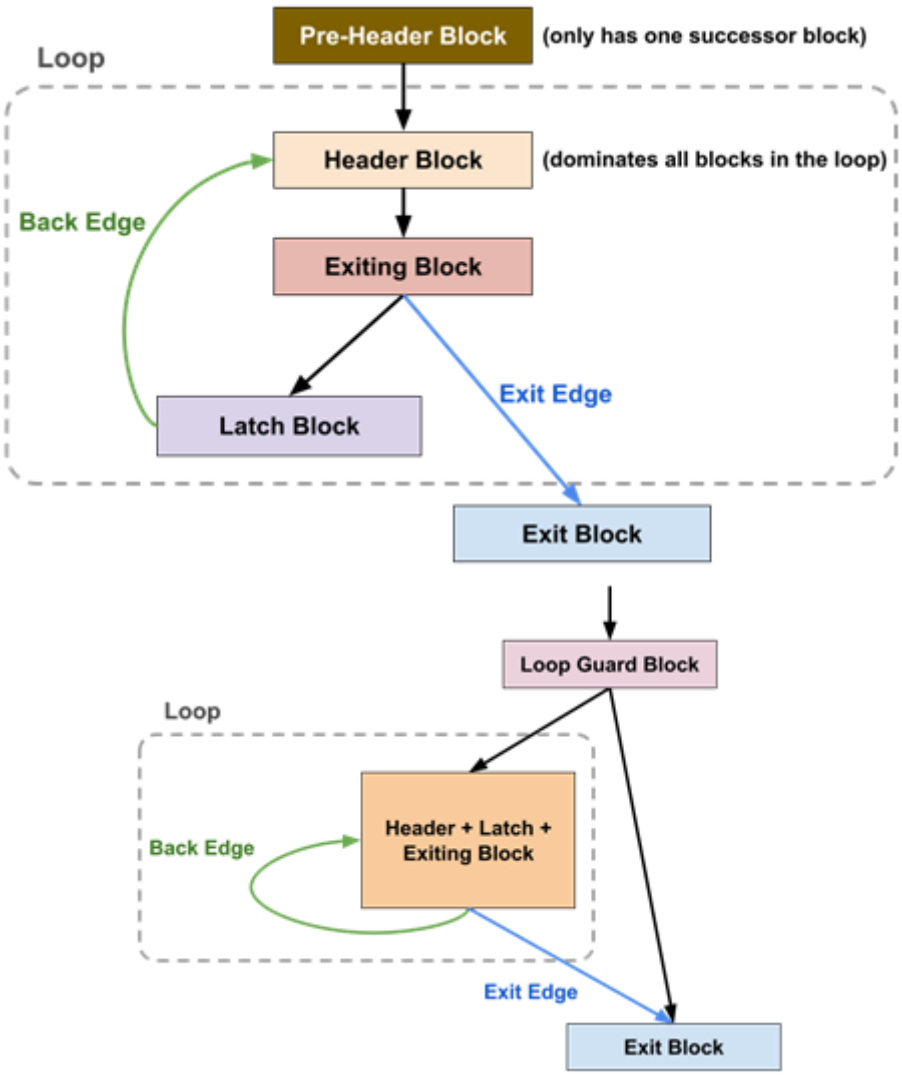
CFG for 'foo' function



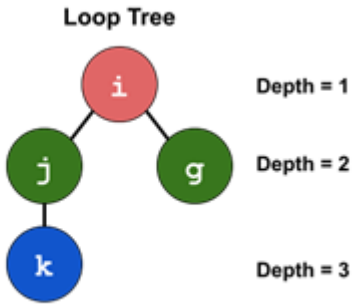


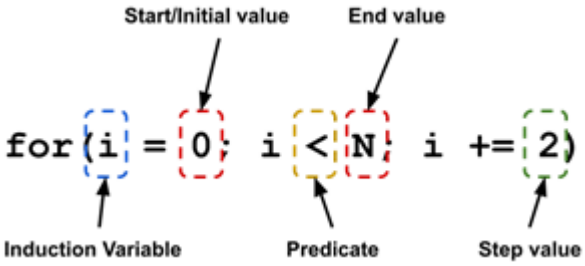



```
Value *V1, *V2;
...
V1->replaceAllUsesWith(V2);
```




```
for(int i = 0; i < N; ++i) {
  for(int j = 0; j < M; ++j) {
    bar(j + N);
    for(int k = 0; k < K; ++k) {
      zoo(k - i);
    }
  }
  for(int g = 0; g <G; ++g) {
    zoo(g + i);
  }
}
```

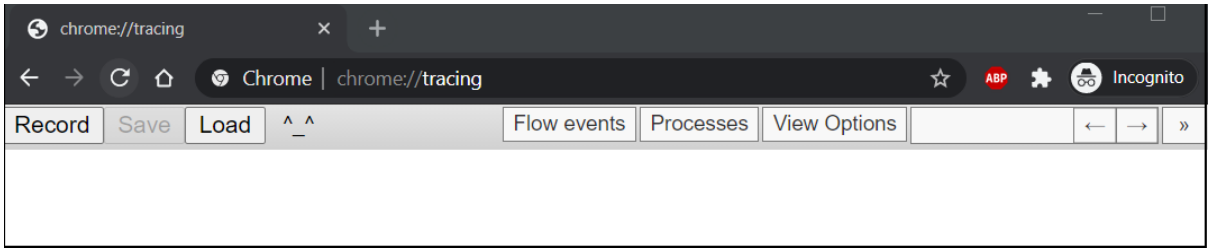


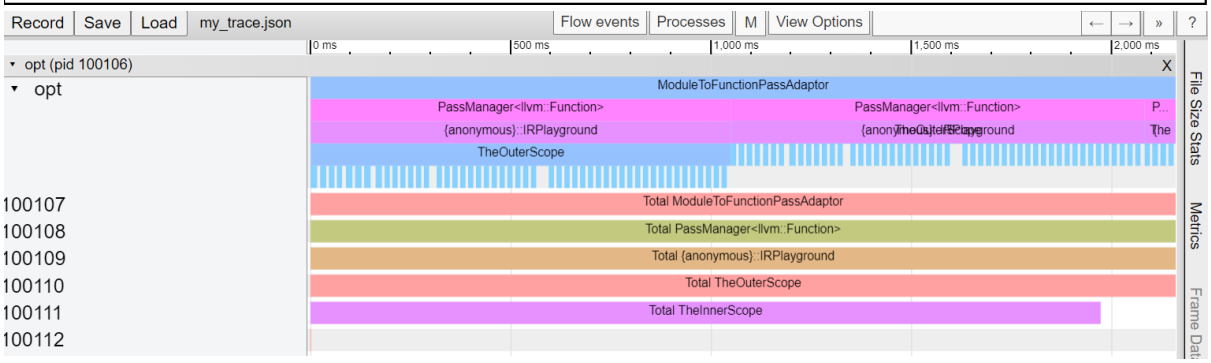


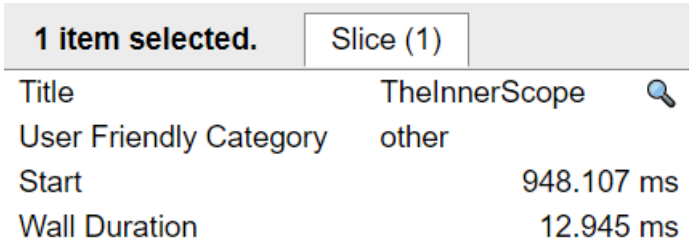
Chapter 11: Gearing up with support utilities

Source Location	Hotness	Function	Pass
./opt_remark_licm.c:0:0		foo	asm-printer
./opt_remark_licm.c:0:0		foo	gvn
./opt_remark_licm.c:1:0		foo	asm-printer
./opt_remark_licm.c:1:0		foo	prologepilog
./opt_remark_licm.c:3:3		foo	asm-printer
./opt_remark_licm.c:3:3		foo	asm-printer
./opt_remark_licm.c:3:3		foo	asm-printer
./opt_remark_licm.c:3:3		foo	asm-printer
./opt_remark_licm.c:3:3		foo	loop-unroll
./opt_remark_licm.c:3:3		foo	loop-vectorize
./opt_remark_licm.c:3:3		foo	loop-vectorize
./opt_remark_licm.c:3:21		foo	asm-printer
./opt_remark_licm.c:4:5		foo	licm
./opt_remark_licm.c:4:5		foo	licm
./opt_remark_licm.c:4:10		foo	asm-printer
./opt_remark_licm.c:4:10		foo	asm-printer

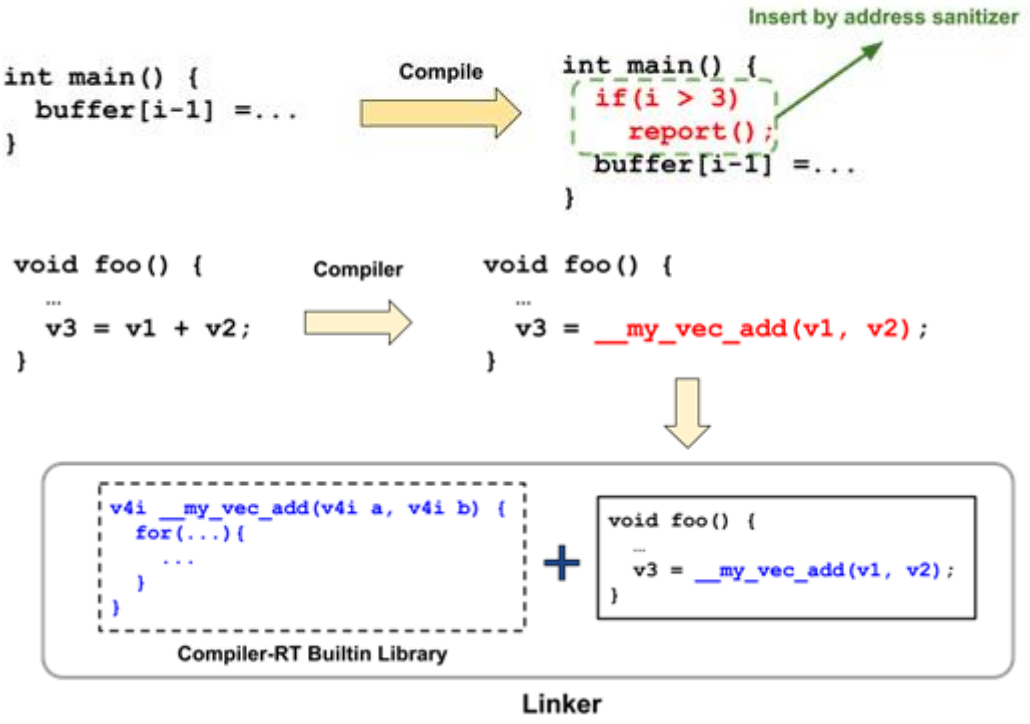
Line	Hotness	Optimization	Source	Inline Context
1			`int foo(int *a, int N) {`	
		prologepilog	0 stack bytes in function	foo
		asm-printer	37 instructions in function	foo
2			`  int x = a[5];`	
3			`  for (int i = 0; i < N; i += 3) {`	
		loop-vectorize	the cost-model indicates that vectorization is not beneficial	foo
		loop-vectorize	the cost-model indicates that interleaving is not beneficial	foo
		loop-unroll	unrolled loop by a factor of 4 with run-time trip count	foo
		asm-printer	+ BasicBlock:	foo
		asm-printer	+ BasicBlock:	foo
		asm-printer	+ BasicBlock:	foo
		asm-printer	+ BasicBlock:	foo
		asm-printer	+ BasicBlock:	foo
4			`    a[i] += 2;`	
		licm	sinking getelementptr	foo
		licm	sinking zext	foo
		asm-printer	+ BasicBlock:	foo
		asm-printer	+ BasicBlock:	foo
5			`  x = a[5];`	
6			`  }`	
7			`  return x;`	
8			`}`	
9				
10				

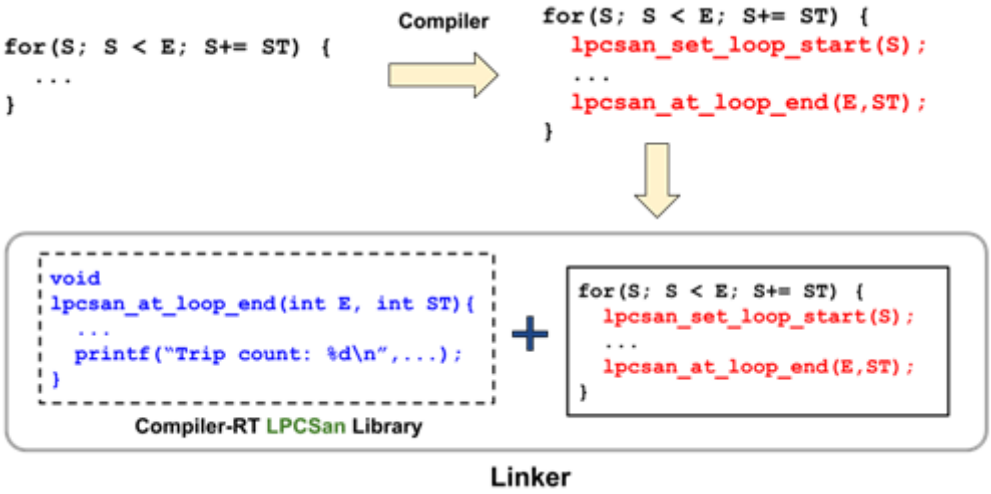


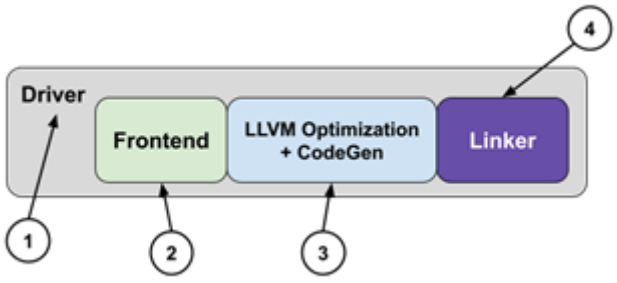


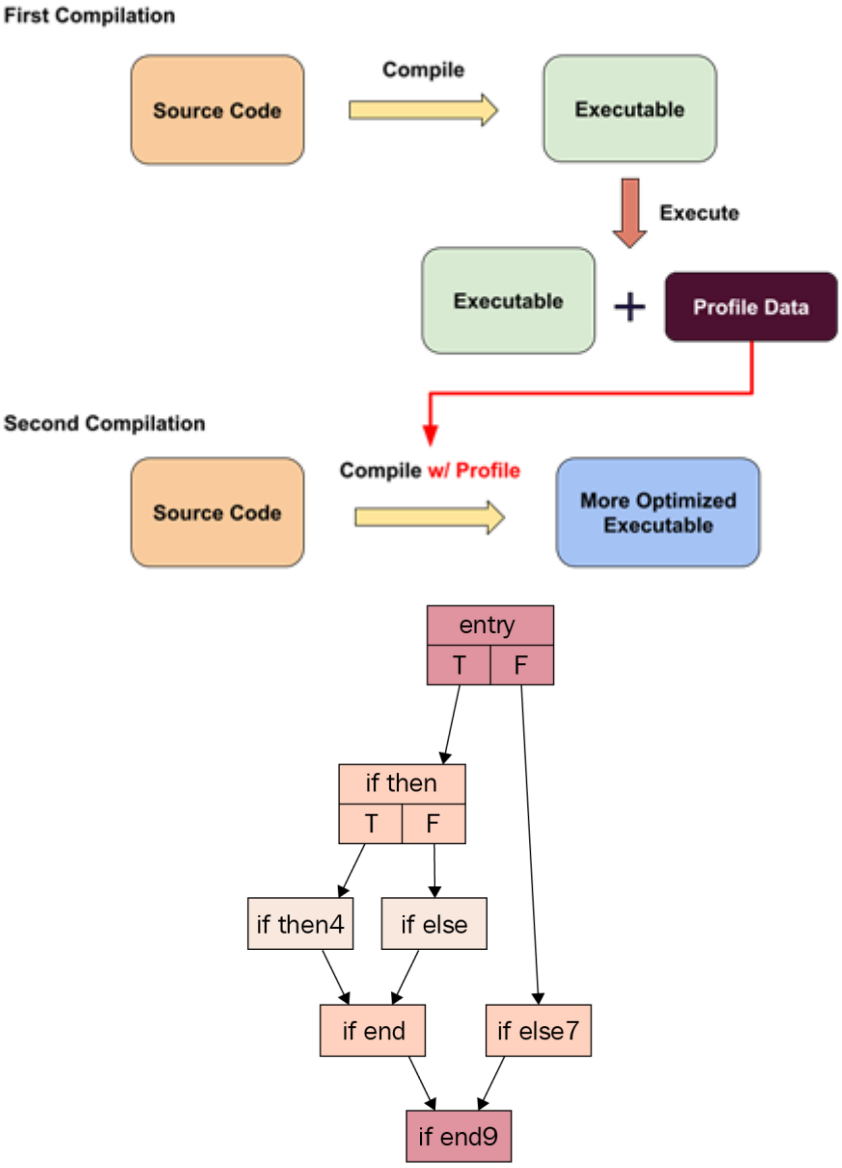


Chapter 12: PGO and Sanitizers developments








First Compilation
Source Code
Compile
Executable
Execute
Executable
Profile Data
Second Compilation
Source Code
Compile w/ Profile
More Optimized Executable
entry
T
F
if then
T
F
if then4
if else
if end
if else7
if end9