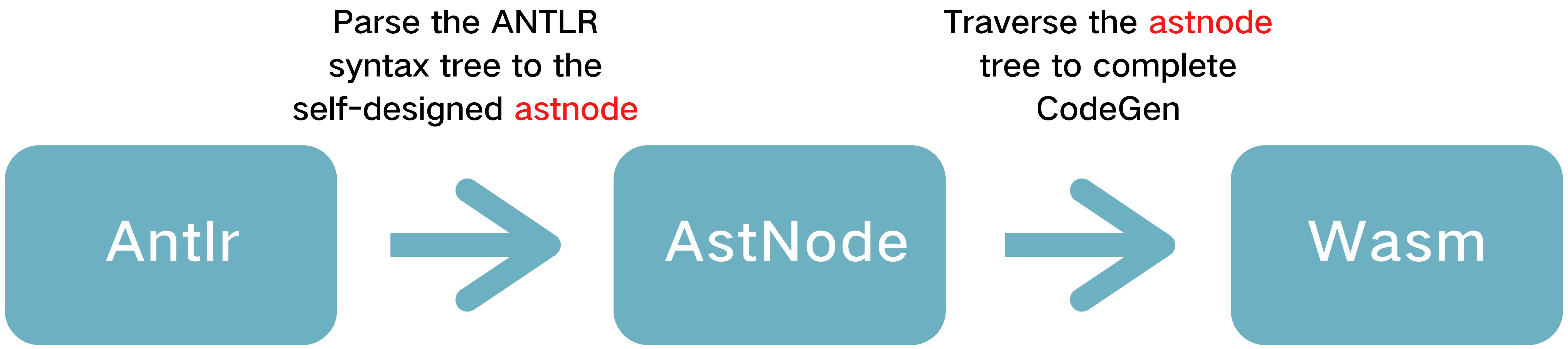
Design of Deeplang FrontEnd

MaHaotian@Deeplang

2020-12-20

Outline

1.Overview

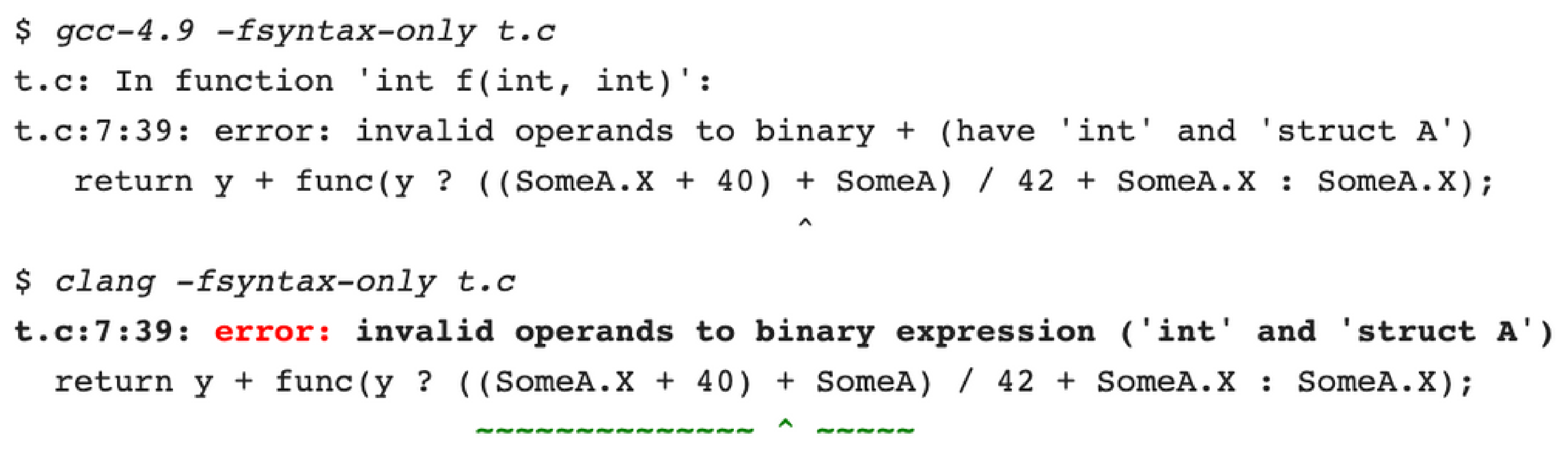

Parse the ANTLR syntax tree to the self-designed astnode
Traverse the astnode tree to complete CodeGen
Antlr
AstNode
Wasm

- Confirm the deepLang syntax;
- Manually write the recursive descent analyzer;

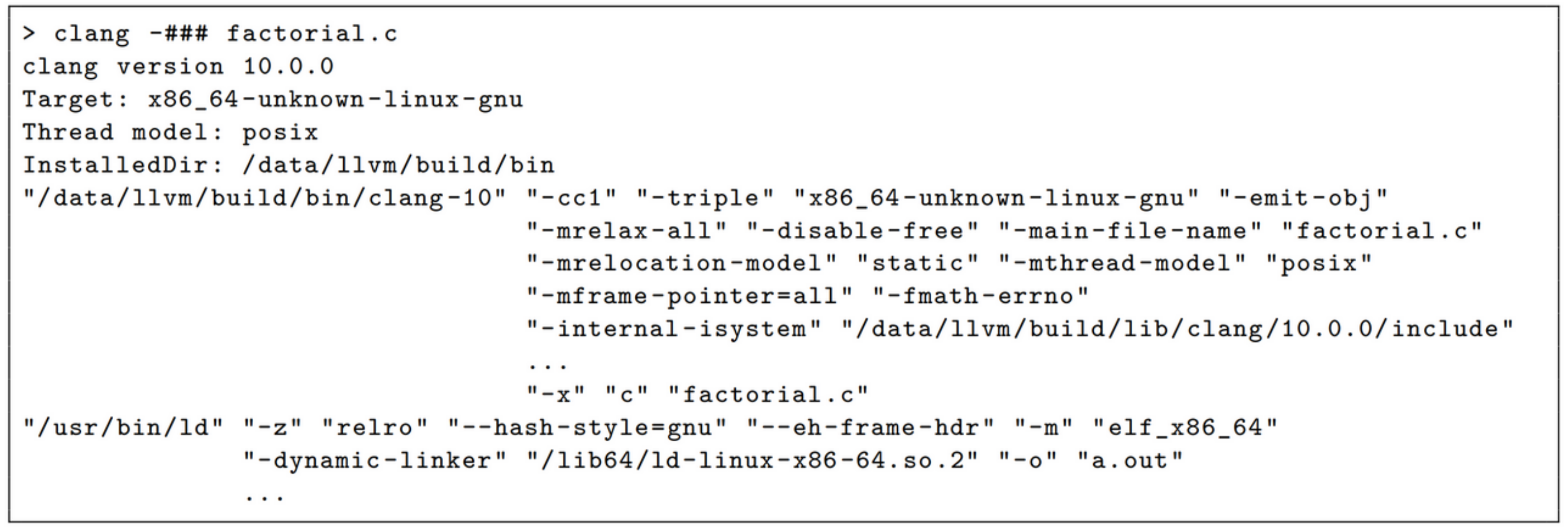
1. Overview

Besides AST building.....

- Driver
- Error diagnosis and handling
- Application
- Optimization of IR level
- ....



```
$ gcc-4.9 -fsyntax-only t.c
t.c: In function 'int f(int, int)':
t.c:7:39: error: invalid operands to binary + (have 'int' and 'struct A')
   return y + func(y ? ((SomeA.X + 40) + SomeA) / 42 + SomeA.X : SomeA.X);
                                       ^

$ clang -fsyntax-only t.c
t.c:7:39: error: invalid operands to binary expression ('int' and 'struct A')
  return y + func(y ? ((SomeA.X + 40) + SomeA) / 42 + SomeA.X : SomeA.X);
                       ~~~~~~~~~~~~~~ ^ ~~~~~
```



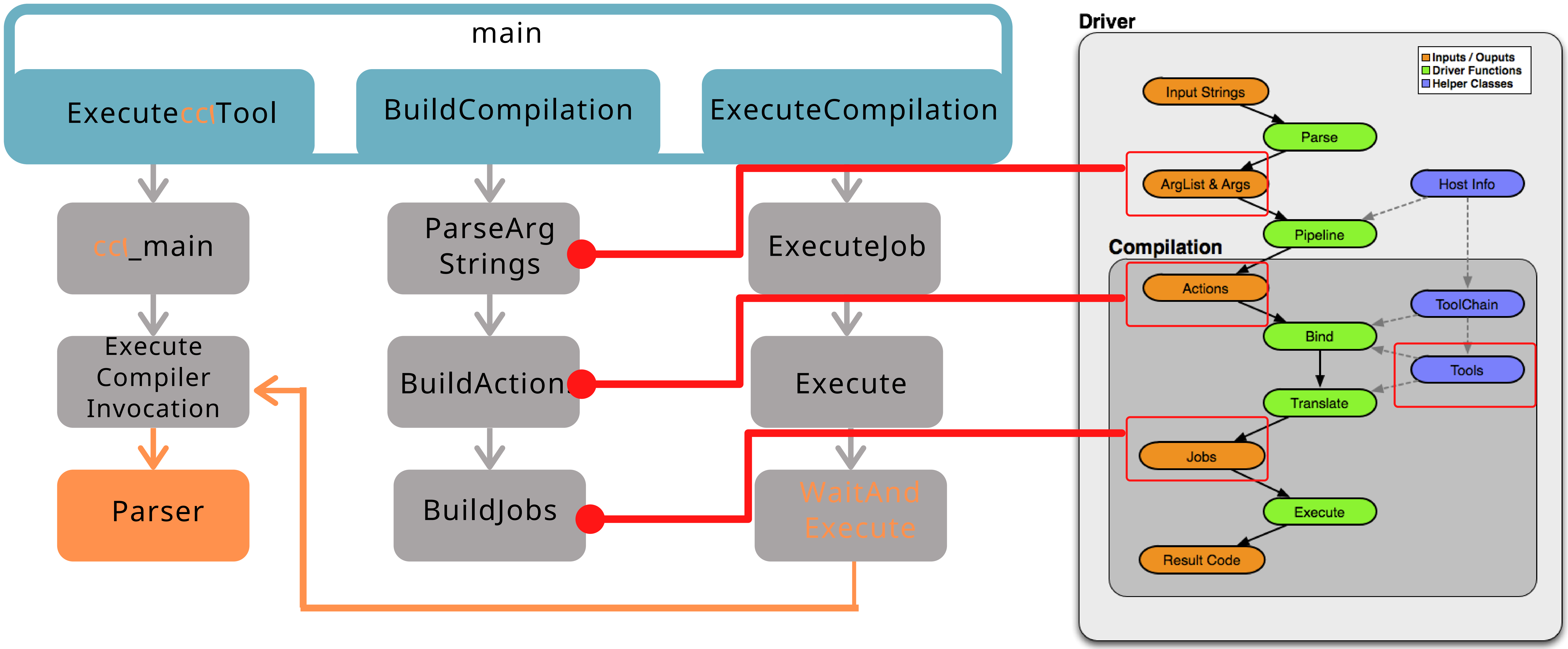
```
> clang -### factorial.c
clang version 10.0.0
Target: x86_64-unknown-linux-gnu
Thread model: posix
InstalledDir: /data/llvm/build/bin
"/data/llvm/build/bin/clang-10" "-cc1" "-triple" "x86_64-unknown-linux-gnu" "-emit-obj"
                                "-mrelax-all" "-disable-free" "-main-file-name" "factorial.c"
                                "-mrelocation-model" "static" "-mthread-model" "posix"
                                "-mframe-pointer=all" "-fmath-errno"
                                "-internal-isystem" "/data/llvm/build/lib/clang/10.0.0/include"
                                ...
                                "-x" "c" "factorial.c"
"/usr/bin/ld" "-z" "relro" "--hash-style=gnu" "--eh-frame-hdr" "-m" "elf_x86_64"
              "-dynamic-linker" "/lib64/ld-linux-x86-64.so.2" "-o" "a.out"
              ...
```

Outline

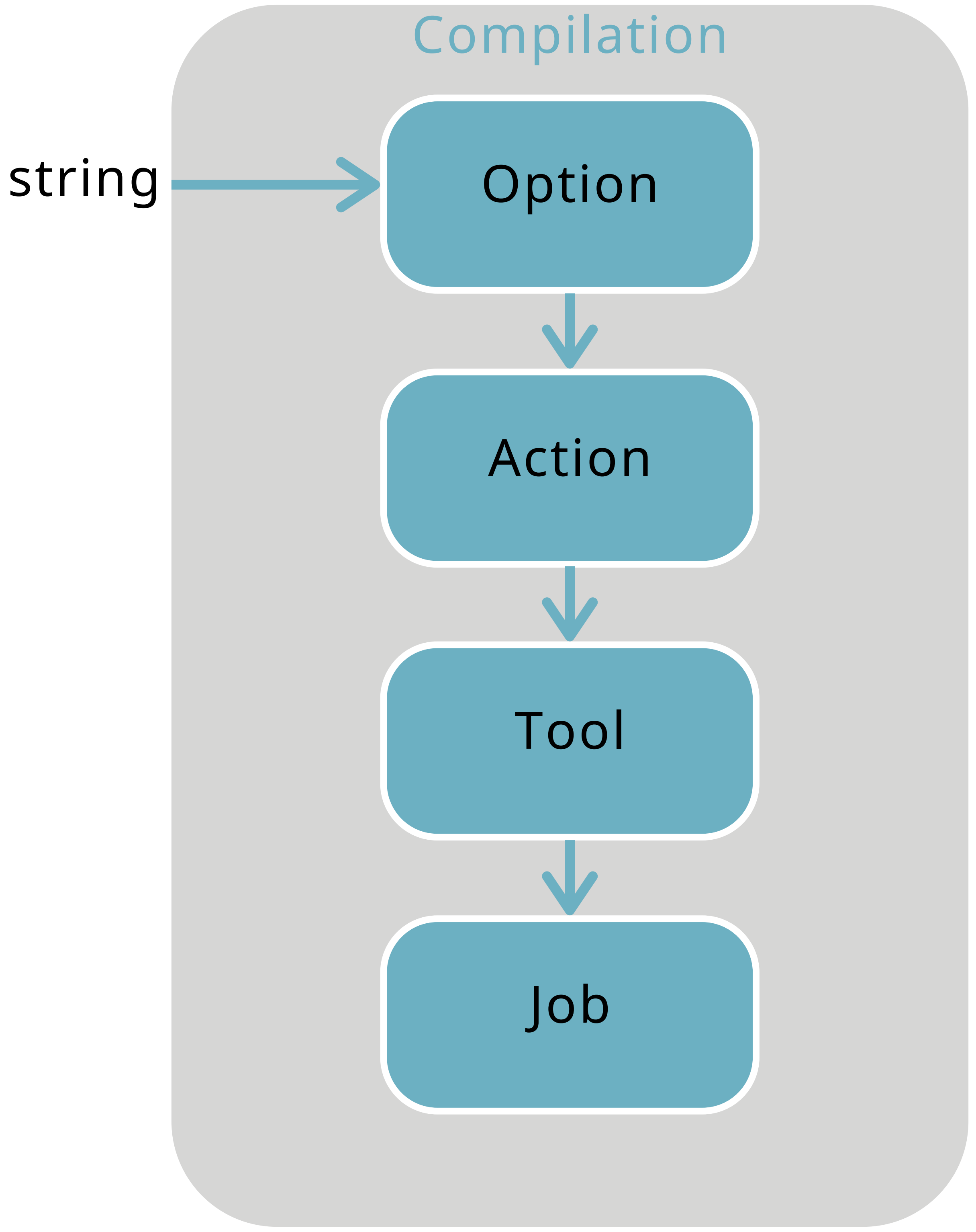
2. About Clang

- Clang is a compiler driver.
  - Driving all phases of a compiler invocation, e.g. preprocessing, compiling, linking.
- Clang is a C language family frontend.
  - Compiling C-like code to LLVM IR. Known as cc1.
- What's the relationship between driver and cc1 ?

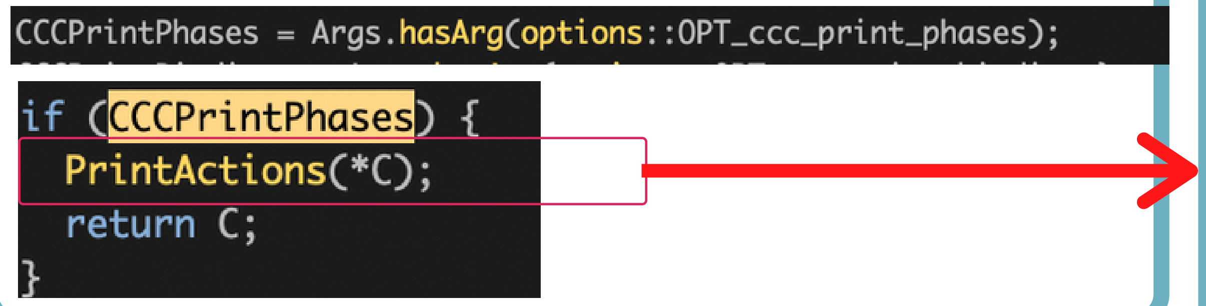
2.1 Clang As Driver



2.1 Clang As Driver



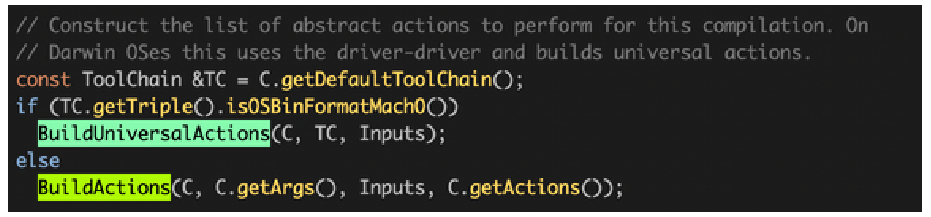
Options.td use tableGen to define sepcfic Compile options.
By: options::OPT_XXX
Eg: def ccc_print_phases



```
CCCPrintPhases = Args.hasArg(options::OPT_ccc_print_phases);
```

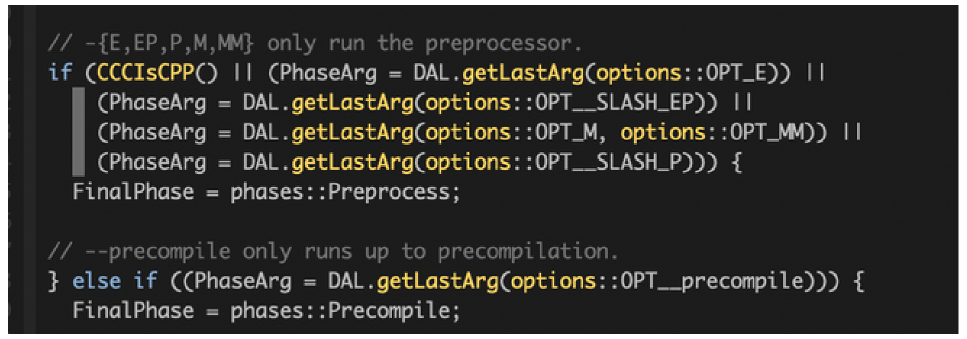
```
if (CCCPrintPhases) {
  PrintActions(*C);
  return C;
}
```

Driver.cpp creates corresponding Actions according to options.



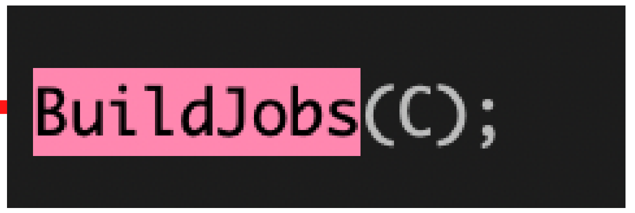
```
// Construct the list of abstract actions to perform for this compilation. On
// Darwin OSes this uses the driver-driver and builds universal actions.
const ToolChain &TC = C.getDefaultToolChain();
if (TC.getTriple().isOSBinFormatMachO())
  BuildUniversalActions(C, TC, Inputs);
else
  BuildActions(C, C.getArgs(), Inputs, C.getActions());
```

Driver::getFinalPhasesActions has abstraction of preprocess, compile, linking. Execute all by default



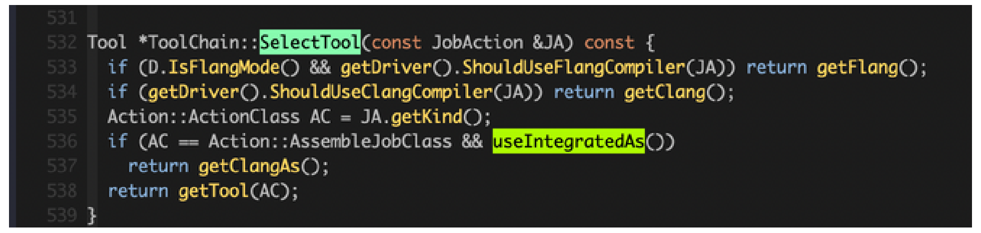
```
// -{E,EP,P,M,MM} only run the preprocessor.
if (CCCIsCPP() || (PhaseArg = DAL.getLastArg(options::OPT_E)) ||
    (PhaseArg = DAL.getLastArg(options::OPT__SLASH_EP)) ||
    (PhaseArg = DAL.getLastArg(options::OPT_M, options::OPT_MM)) ||
    (PhaseArg = DAL.getLastArg(options::OPT__SLASH_P))) {
  FinalPhase = phases::Preprocess;

// --precompile only runs up to precompilation.
} else if ((PhaseArg = DAL.getLastArg(options::OPT__precompile))) {
  FinalPhase = phases::Precompile;
```

C is a Compilation instance.



```
BuildJobs(C);
```

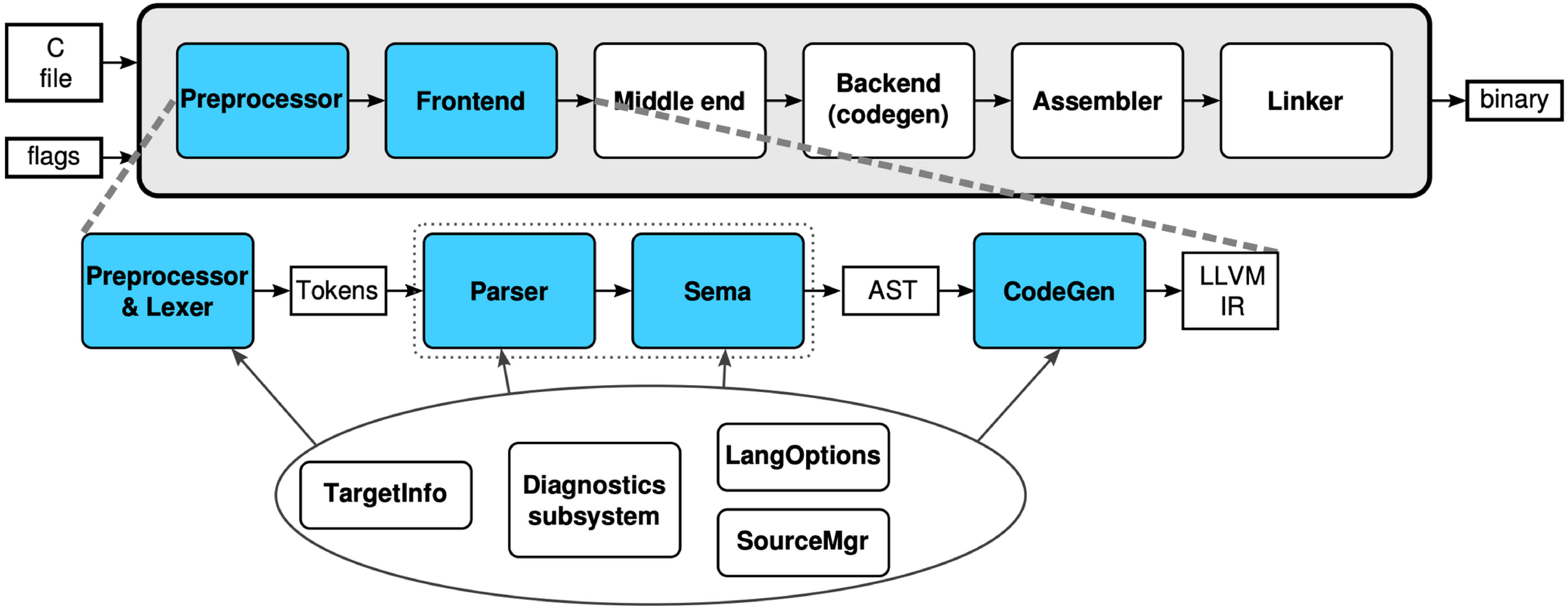
Driver.cpp construct Job instances according to Actions of Compliation
BuildJobs
BuildJobsForAction
selectTool



```
Tool *ToolChain::SelectTool(const JobAction &JA) const {
  if (D.IsFlangMode() && getDriver().ShouldUseFlangCompiler(JA)) return getFlang();
  if (getDriver().ShouldUseClangCompiler(JA)) return getClang();
  Action::ActionClass AC = JA.getKind();
  if (AC == Action::AssembleJobClass && useIntegratedAs())
    return getClangAs();
  return getTool(AC);
}
```

Tool meas obj-tool like as， ld etc.

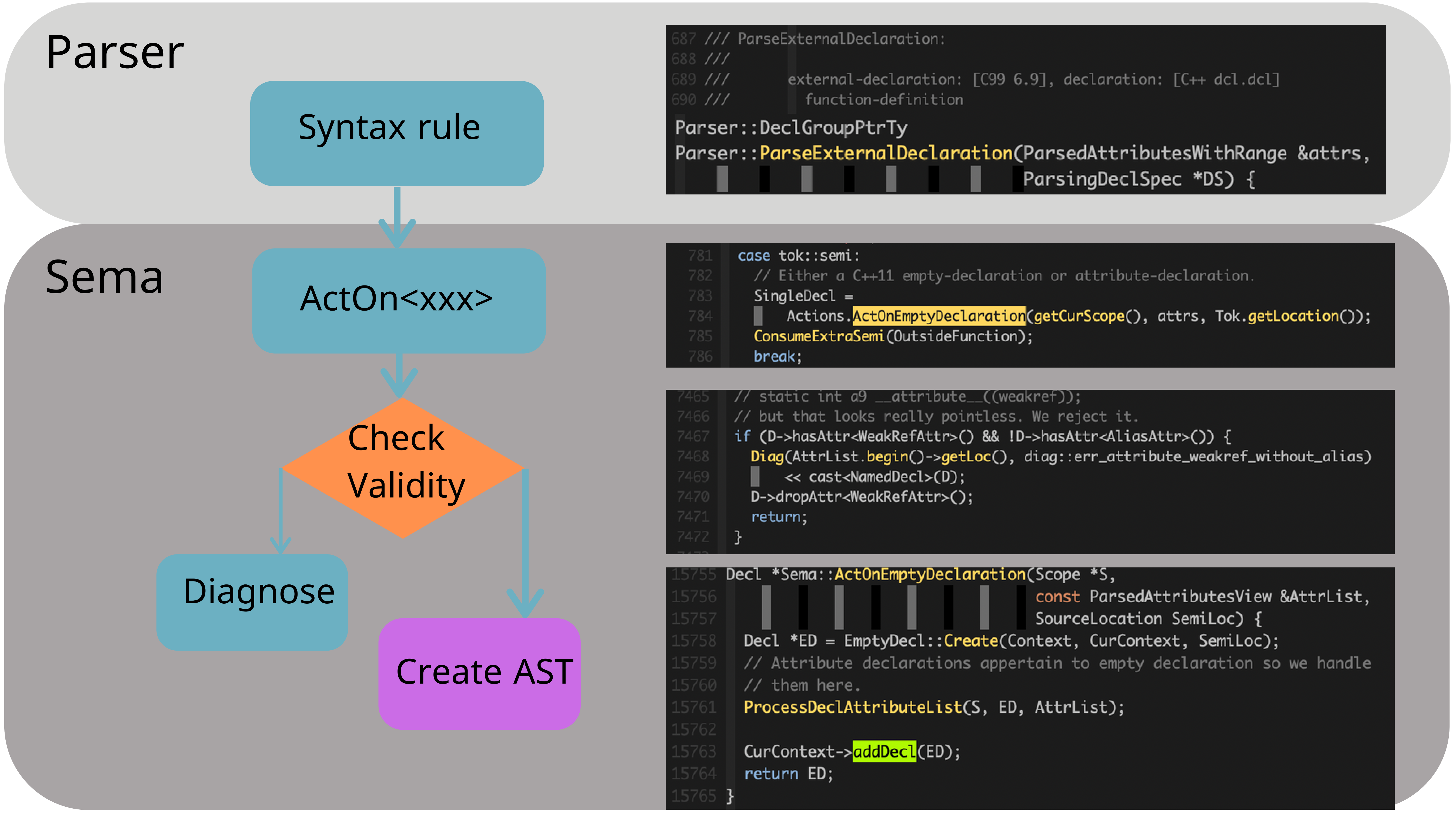
2.2 Clang As Frontend



2.2.1 Diagnostics subsystem

- Better reading
  - Severity: note, warning, or error
  - Source location: xxx.cpp:1:1
  - Message: “unknown type name ’int’; did you mean ’int’?”
- Defined in Diagnostic*Kinds.td TableGen files
- Emitted through helper function Diag

2.2.1 Diagnostics subsystem



```
/// ParseExternalDeclaration:
///
///       external-declaration: [C99 6.9], declaration: [C++ dcl.dcl]
///         function-definition
Parser::DeclGroupPtrTy
Parser::ParseExternalDeclaration(ParsedAttributesWithRange &attrs,
                                 ParsingDeclSpec *DS) {
```

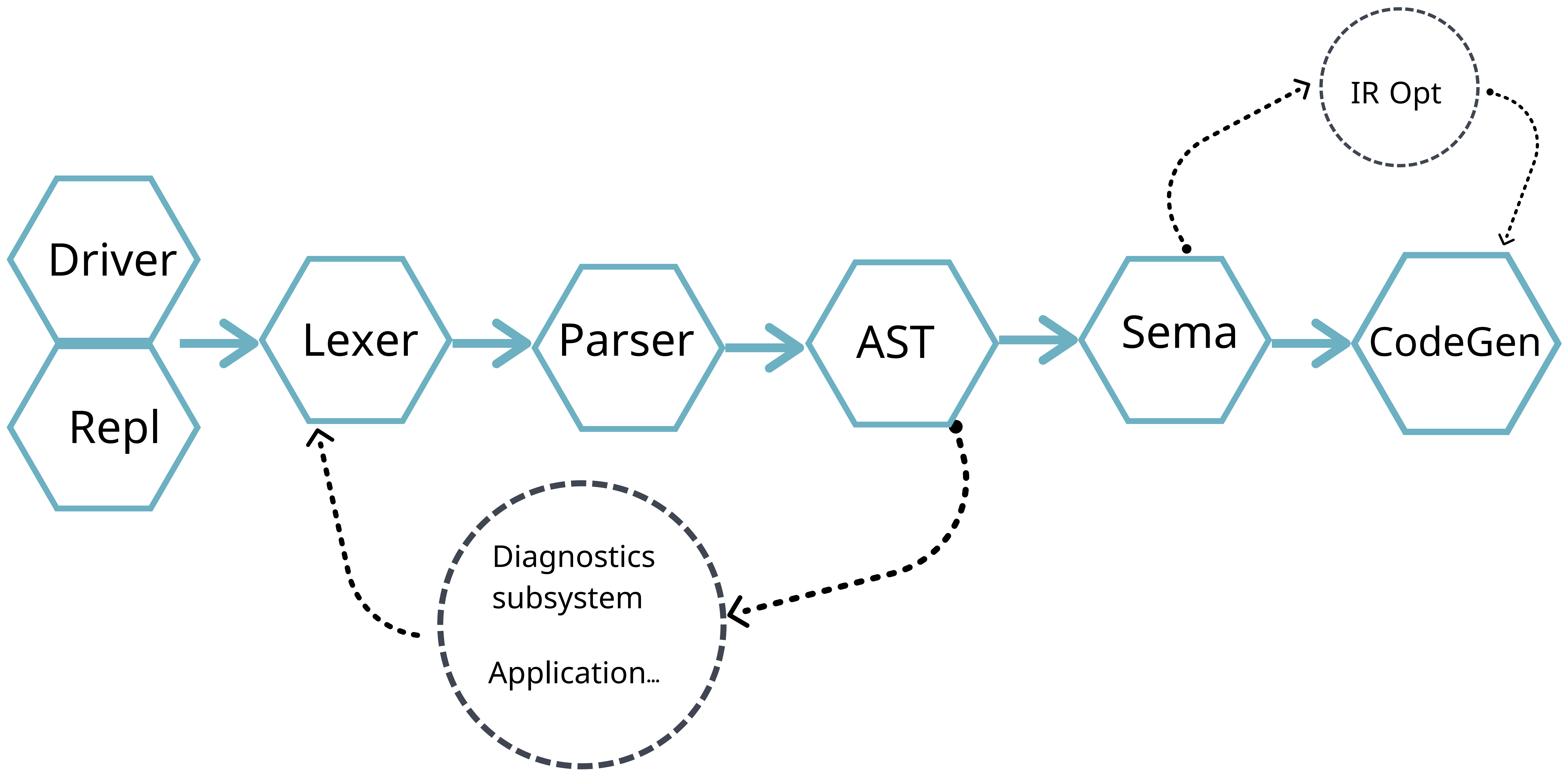
```
  case tok::semi:
    // Either a C++11 empty-declaration or attribute-declaration.
    SingleDecl =
        Actions.ActOnEmptyDeclaration(getCurScope(), attrs, Tok.getLocation());
    ConsumeExtraSemi(OutsideFunction);
    break;
```

```
// static int a9 __attribute__((weakref));
// but that looks really pointless. We reject it.
if (D->hasAttr<WeakRefAttr>() && !D->hasAttr<AliasAttr>()) {
  Diag(AttrList.begin()->getLoc(), diag::err_attribute_weakref_without_alias)
      << cast<NamedDecl>(D);
  D->dropAttr<WeakRefAttr>();
  return;
}
```

```
Decl *Sema::ActOnEmptyDeclaration(Scope *S,
                                  const ParsedAttributesView &AttrList,
                                  SourceLocation SemiLoc) {
  Decl *ED = EmptyDecl::Create(Context, CurContext, SemiLoc);
  // Attribute declarations appertain to empty declaration so we handle
  // them here.
  ProcessDeclAttributeList(S, ED, AttrList);

  CurContext->addDecl(ED);
  return ED;
}
```

Outline

3.Design of Deeplang FrontEnd



Thanks

2020-12-20