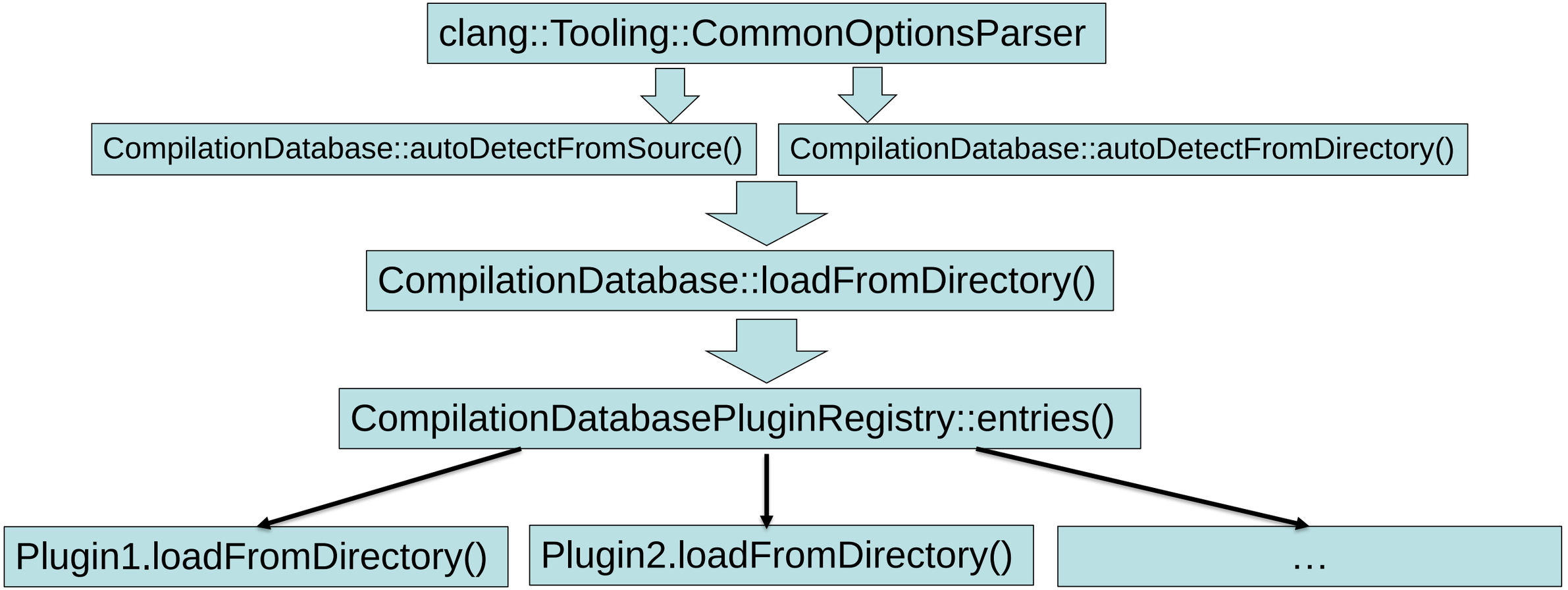
Compilation databases:
how to help a clang-based tool to understand your compile commands

Aleksandr Platonov

Compilation database

- Detect project files
- Specific build options for each file
  - Include paths
  - Definitions
  - Compilation flags
  - …

Compilation databases loading



Compilation database plugins

- FixedCompilationDatabasePlugin
  - Plain-text flags file
  - loadFromDirectory(Dir) => loadFromFile(Directory + “compile_flags.txt”)
- JSONCompilationDatabasePlugin
  - JSON formatted compilation database
  - loadFromDirectory(Dir) => loadFromFile(Dir + “compile_commands.json”)

Fixed compilation database: format

- The same flags for every translation unit
- One argument per line

compile_flags.txt

```
-DTEST=1
-I
../include
…
```

- Paths are relative to compile_flags.txt directory

JSON compilation database: format

- Command format

compile_commands.json

```
[
  { "directory": "/home/user/test/build",
    "command": "/usr/bin/g++ -DTEST=1 -o file.o -c ../file.cpp",
    "file": “/home/user/test/file.cpp },
 ...
]
```

- Arguments format

compile_commands.json

```
[
  { "directory": "/home/user/test/build",
    “arguments": ["/usr/bin/g++", "-DTEST=1", "-o", "file.o", "-c", "../file.cpp“],
    "file": “/home/user/test/file.cpp" },
 ...
]
```

JSON compilation database: how to create (1/2)

- CMake
  - CMAKE_EXPORT_COMPILE_COMMANDS

```
cmake -DCMAKE_EXPORT_COMPILE_COMMANDS=On …
```

  - Supported generators
    - Unix Makefiles (since 2.8.5)
    - Ninja (since 2.8.9)
- Clang (since 5.0)
  - -MJ

```
clang -MJ a.out.json …
```

- Ninja (since 1.2)
  - -t compdb

```
ninja -t compdb > compile_commands.json
```

JSON compilation database: how to create (2/2)

- Visual Studio
  - SourceTail extension https://github.com/CoatiSoftware/vs-sourcetrail
    - Visual Studio solution => JSON compilation database

- Intercept compiler calls
  - Bear https://github.com/rizsotto/Bear

```
bear -- build.sh
```

  - intercept-build https://github.com/rizsotto/scan-build

```
intercept-build build.sh
```

- Parse build logs
  - compiledb https://github.com/nickdiego/compiledb

```
compiledb --parse build-log.txt
```

Compilation database wrappers: motivation

- Some files are missed in the compilation database
- Custom toolchains
- Response files and shell expressions inside a compile command
- A compile command can be not 100% compatible with clang

Compilation database wrappers: arguments adjuster

- Modify command line arguments
  - Insert
  - Remove
  - Replace
- Used by a clang-based tool
  - Implicitly remove
    - Output-related arguments
    - Dependency file related arguments
  - --extra-args
    - Arguments to append
  - --extra-args-before
    - Arguments to prepend
  - Can be used for you own purposes

Compilation database wrappers: interpolating database

- Motivation
  - Files without an entry in the compilation database
    - Headers
    - Newly created files
  - Even a random command from the database is better than nothing
- Find the closest file
  - Filename without extension matches
  - Directory structure matches
- Borrow the compile command
  - Replace the filename
  - Remove output arguments
  - Adjust -x and -std flags

Compilation database wrappers: query driver (1/3)

- Motivation
  - Clang
    - ✓ performs toolchain specific searches for system headers
    - ✓ detects target triple using the compiler program name
  - Custom toolchain in a compile command
    - Unusual compiler name (e.g. my_compiler.sh)
    - Non-typical system include directories
    - The compiler knows his system includes and target

Compilation database wrappers: query driver (2/3)

- Extracts a compiler from the compile command
- Runs the compiler in verbose mode

```
…
Target: aarch64-linux-gnu
…
#include "..." search starts here:
#include <...> search starts here:
 /usr/lib/gcc-cross/aarch64-linux-gnu/7/../../../../aarch64-linux-gnu/include/c++/7
 /usr/lib/gcc-cross/aarch64-linux-gnu/7/../../../../aarch64-linux-gnu/include/c++/7/aarch64-linux-gnu
 /usr/lib/gcc-cross/aarch64-linux-gnu/7/../../../../aarch64-linux-gnu/include/c++/7/backward
 /usr/lib/gcc-cross/aarch64-linux-gnu/7/include
 /usr/lib/gcc-cross/aarch64-linux-gnu/7/include-fixed
 /usr/lib/gcc-cross/aarch64-linux-gnu/7/../../../../aarch64-linux-gnu/include
 /usr/include
End of search list.
…
```

- Extracts target and system includes
- Adds -isystem… and --target=… options

Compilation database wrappers: query driver (3/3)

- Implemented as a part of **clangd** tool
  - Disabled by default
  - Execute binaries only for explicitly specified drivers
  - --query-driver command line option
- Works with GCC-compatible toolchains only
- Can’t solve all problems of clang and GCC compatibility
  - Example: various macros depending on GCC version

My compilation database: motivation

- Build system without JSON database generation ability
  - Compiler calls interception?
    - Compiler and OS dependent
- JSON database is not efficient for huge projects
  - Size of compile_commands.json = several Gbs
- A command modification => reload the whole project

My compilation database: compilation database class

```
class MyCompilationDatabase : public CompilationDatabase  {
…
  std::vector<std::string> getAllFiles() const override { // optional
    …
  }

  std::vector<CompileCommand> getCompileCommands(StringRef FilePath) const override {
    …
  }

  std::vector<CompileCommand> getAllCompileCommands() const override {
    …
  }
…
};
```

My compilation database: compilation database plugin

```
class MyCompilationDatabasePlugin : public CompilationDatabasePlugin {
…
  std::unique_ptr<CompilationDatabase>
  loadFromDirectory(StringRef Directory, std::string &ErrorMessage) override {
     // Create MyCompilationDatabase object and return unique pointer to it
  }
…
};


…

static CompilationDatabasePluginRegistry::Add<MyCompilationDatabasePlugin>
X( “my-compilation-database” , “My compilation database” );
```

Thank you