CS354R

DR SARAH ABRAHAM

DEVOPS AND QUALITY ASSURANCE

WHAT IS DEVOPS?

- Development Operations
- Backend facilitation of development
  - Handles local and remote hardware
  - Maintains build infrastructure and pipeline
  - Monitors end user activity
- Hybrid sysadmin/developer
- Very, very important functionality for any reasonably-sized operation
  - Skimping on your backend setup (or having an undocumented setup) will waste MANY developer hours
  - This is called “technical debt”

BUILD SYSTEMS

- Creates software binaries from source code
- Reduces programmer time and effort to create executables
- Allows for easier build targeting across multiple platforms
- Building includes:
  - Compiling
  - Linking
  - Packaging
  - And ideally testing!
- What are some built systems you've used?

MAKE UTILITY

- Determines what to compile/recompile and issues commands
- Makefile provides information that the `make` utility requires
  - Relationships between program files
  - Commands for updating files
- Typical `make` use is:
  1. Compile source files to generate object files
  2. Create executable from object files

MAKEFILE

- Defines how to build files for desired targets
  - CC defines compiler
  - CFLAGS defines compiler flags
  - INCLUDES defines additional header paths
  - LFLAGS defines libraries to link to project
  - Target executable is first target entry in the file

MAKEFILE EXAMPLE

```
# the compiler: gcc for C program, define as g++ for C++

CC = gcc

# compiler flags:

#  -g    adds debugging information to the executable file

#  -Wall turns on most, but not all, compiler warnings

CFLAGS  = -g -Wall

# the build target executable:

TARGET = myprog

all: $(TARGET)

$(TARGET): $(TARGET).c

   $(CC) $(CFLAGS) -o $(TARGET) $(TARGET).c

clean:

   $(RM) $(TARGET)
```

(https://www.cs.swarthmore.edu/~newhall/unixhelp/howto_makefiles.html)

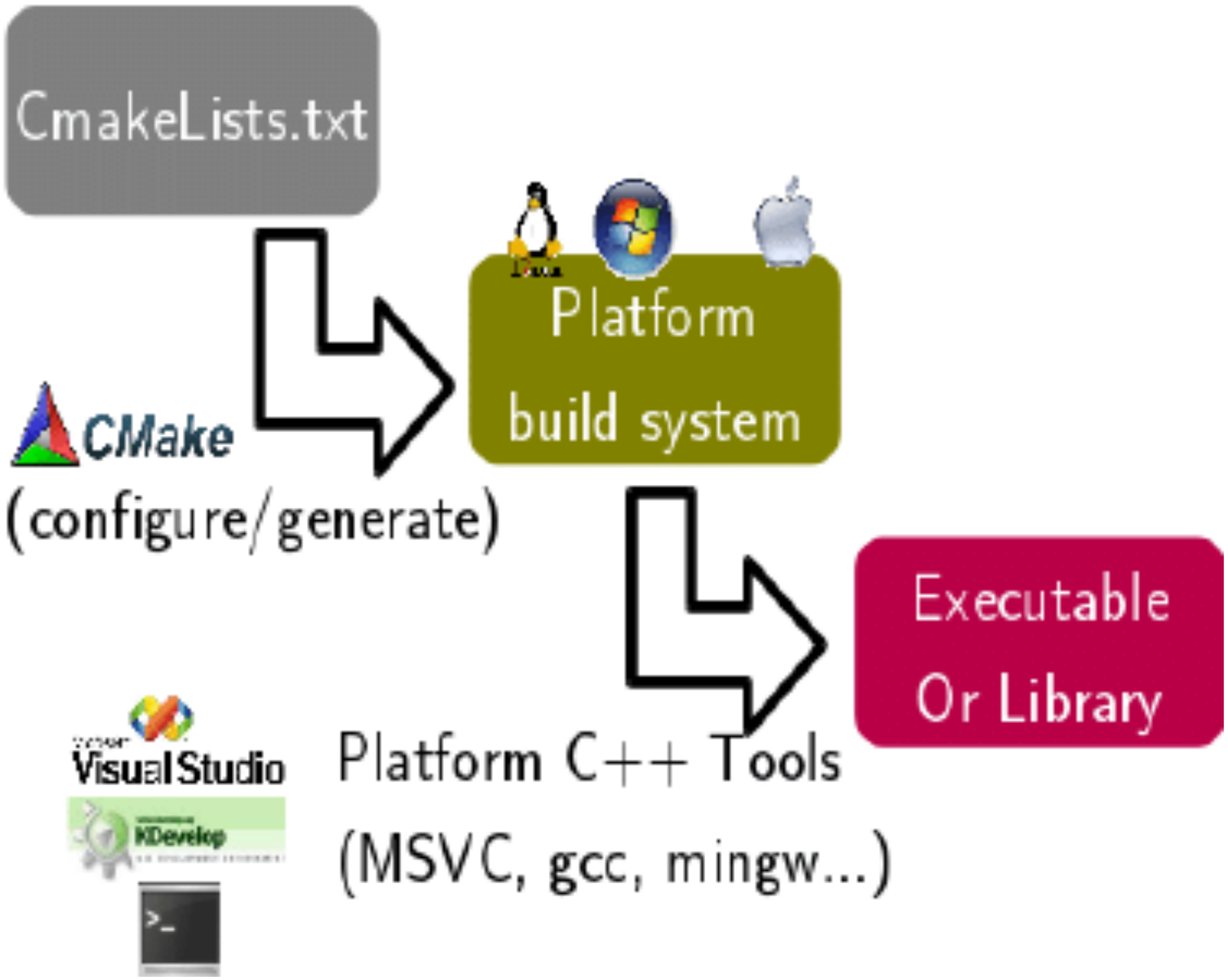
WHAT'S THE PROBLEM WITH MAKE?

- As project complexity increases, complexity of reading and writing Makefiles also increases
  - More library dependencies
  - Dynamically linked libraries
  - Numerous compiler flags
  - Multiple targets
- Difficult to cross-compile to non-Linux platforms

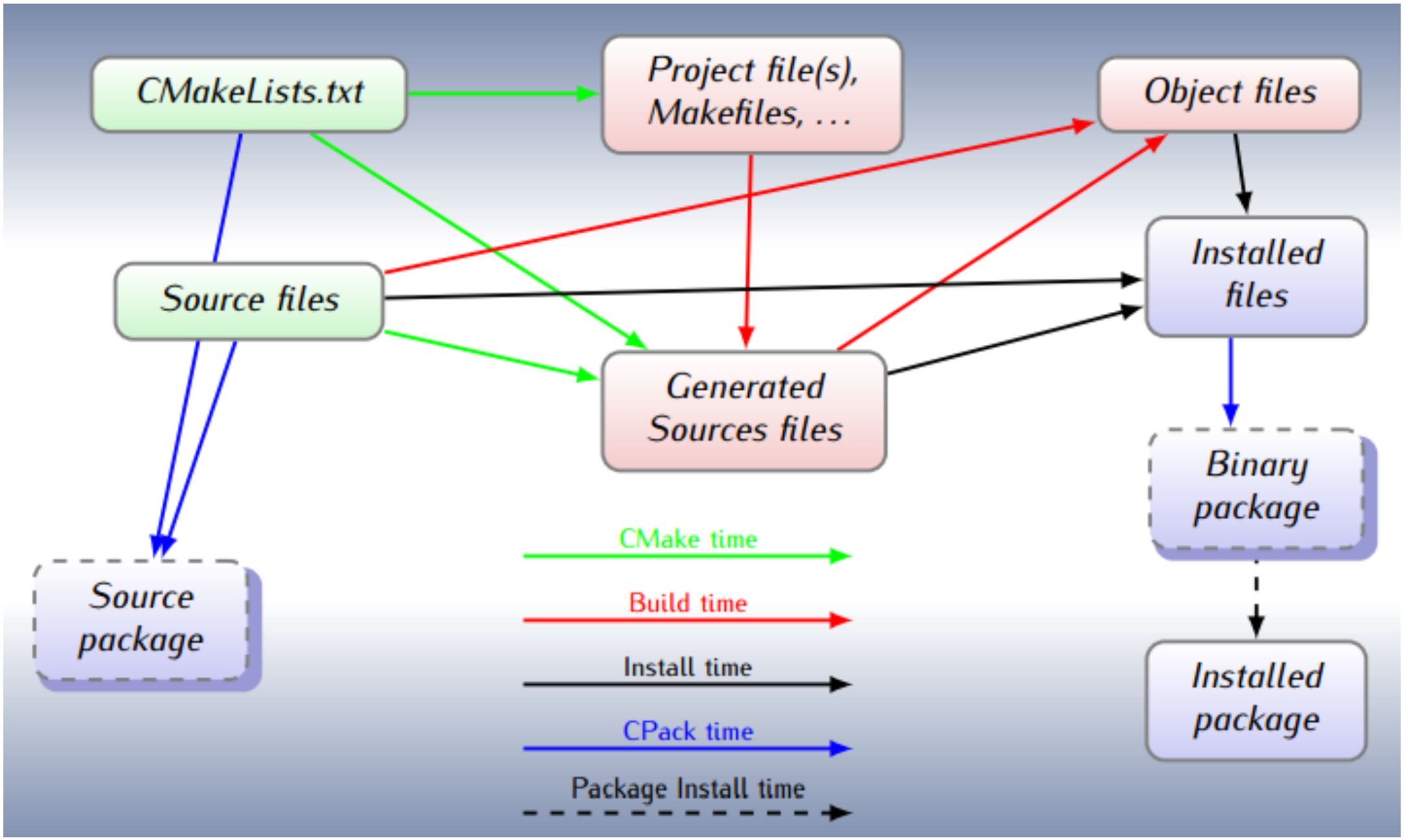
CMAKE OVERVIEW

- Cross-platform, compiler-independent build system
- Commonly used and can build for Xcode and Visual Studio
- Works well for complex source directories and cross-linked libraries
- Build process:
  - CMakeLists.txt contains commands for configuring Makefile
  - `make` then builds project

CMAKE OVERVIEW



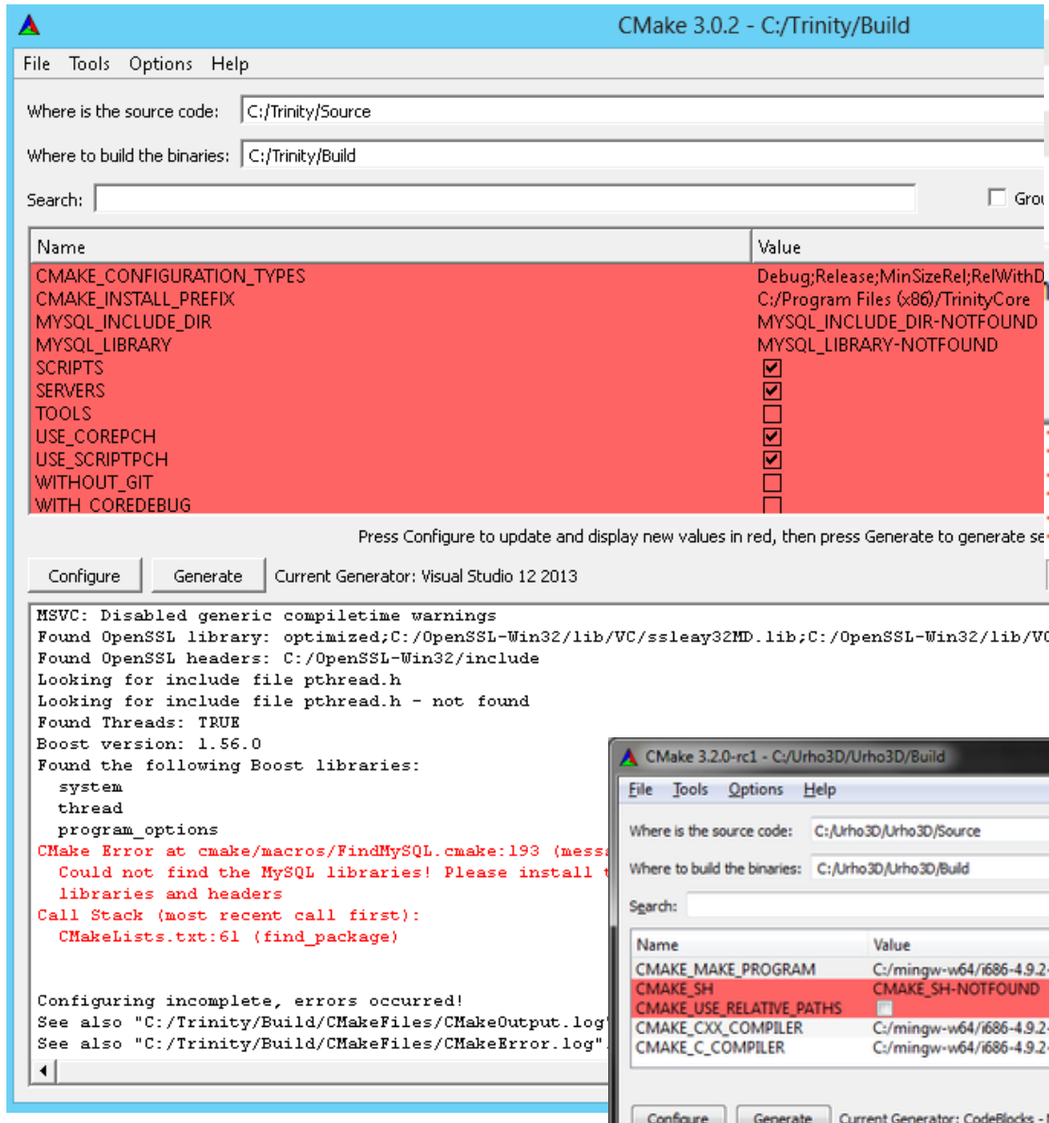
CMAKE BUILD PROCESS

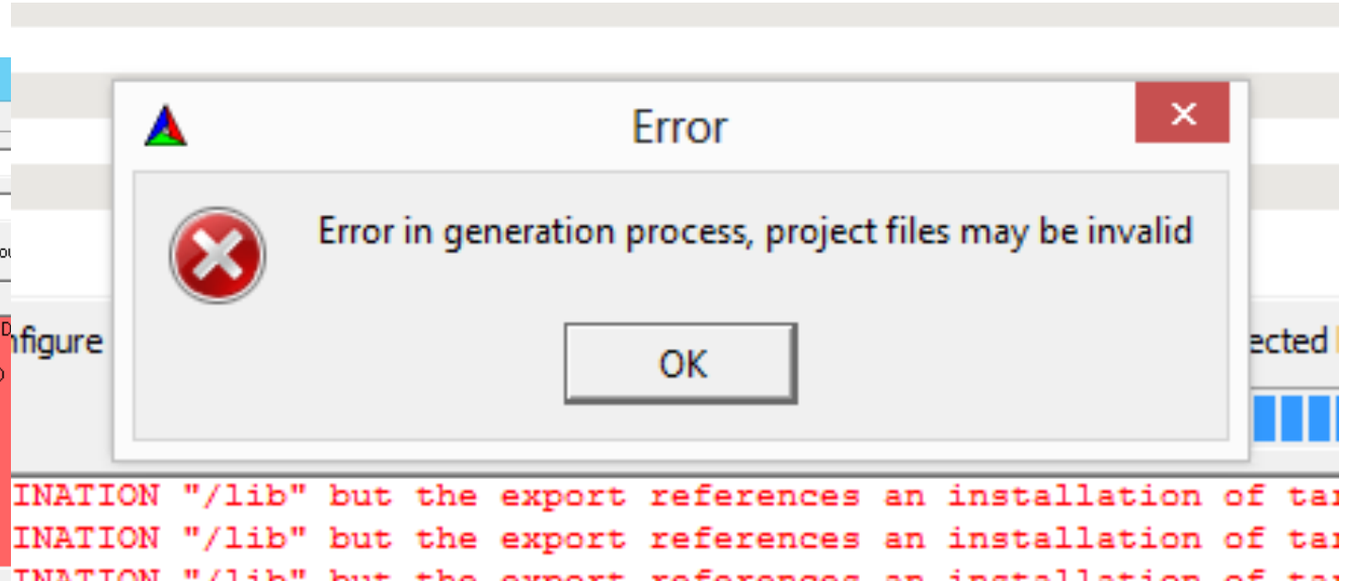


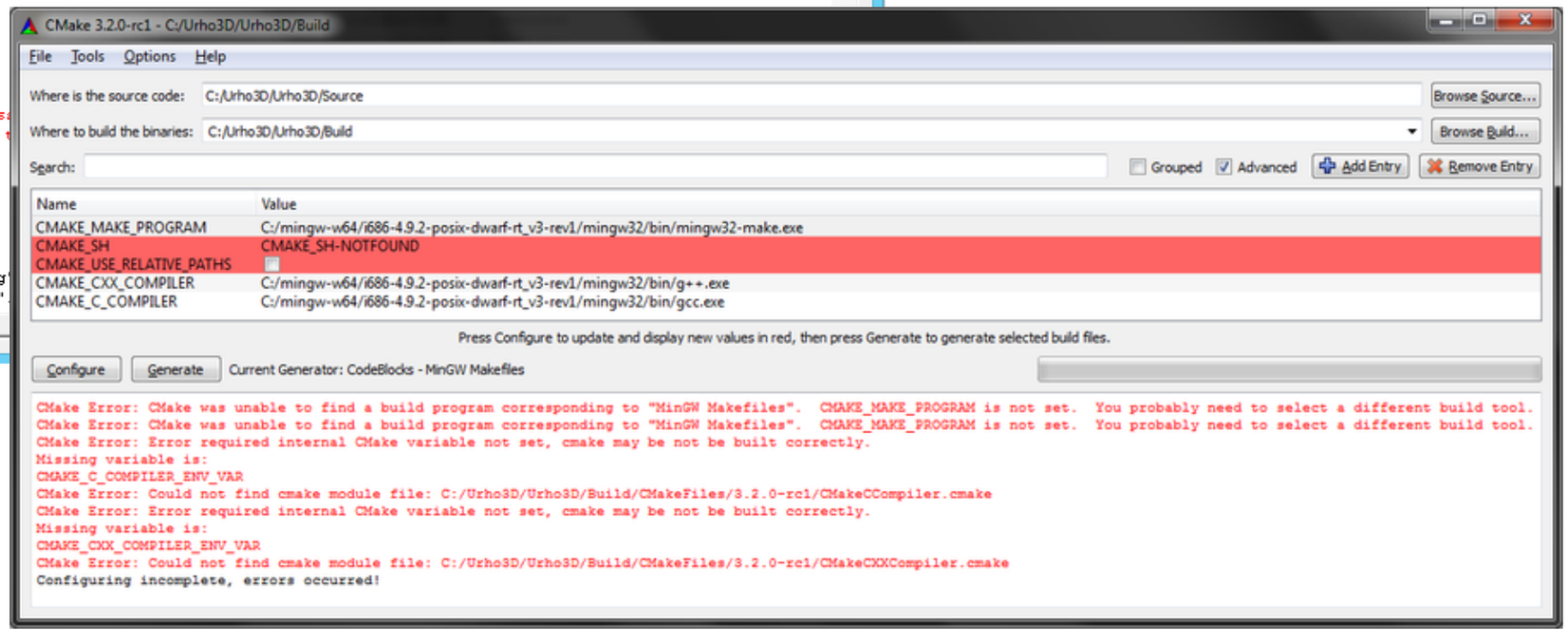
SOME CMAKE COMMANDS

- `project(project_name)` sets project name
- `add_executable(executable_name, project_file)` builds an executable from given project_file.cpp
- `include_directories(include_dir)` adds header directory to build environment
- `add_library(lib_name, lib_source)` adds library to project
- ...and it goes on from there!

HOW TO DEBUG CMAKE?







DEBUGGING BUILD SYSTEMS

- Errors often related to library paths/dependencies
  - Not necessarily any nice tools for debugging
  - Must rely on good old-fashioned sleuthing
    - Liberal use of `dpkg` (or equivalent) checks, etc
- Questions to consider:
  - What is the library's expected version?
  - Where is the library located?
- A note about using StackOverflow
  - Collective knowledge of Internet, so usually someone has encountered something similar before you
  - Read carefully to avoid going down ratholes!

SCONS OVERVIEW

- Open source software construction tool that supports cross-platform development and many compilers
- Built on Python
- Can be distributed with a software product
- Does not need to generate files
- Can be slower on large projects
  - Not as fully implemented as CMake but it’s what Godot uses!

SCONS BUILD PROCESS

- Uses Python scripts as configuration files
  - SConstruct
- Creates complete dependency graph of all files
- Traverses graph to build target files using the SCons Build Engine

SCONS API

- Environment object stores the build configuration
  - `env = DefaultEnvironment()`
- Can customize build information in a per-platform way
  - `CCFLAGS` are options passed to compiler
  - `LINKFLAGS` are options passed to the linker
  - `CPPPATH` lists directories that have necessary includes
  - `LIBPATH` lists directories that have necessary libraries

GODOT AND GDEXTENSION

- What did you do to complete the GDExtension tutorial?

1. Used Godot game engine to create a project
2. Used GDExtension to dynamically connect external C++ code to Godot (i.e. built a Godot plugin)
3. Incorporated this C++ functionality into gameplay and editor (i.e. used the plugin)

CREATING DYNAMIC GODOT PLUGINS USING GDEXTENSION

1. Compile Godot from source or download pre-compiled binary
   - Builds and linked all necessary libraries from **core** and other modules
2. Generate GDExtension C++ bindings
   - api.json contains all metadata for Godot functions and properties
   - Building godot-cpp creates **static library** to link into custom plugin
3. Register plugin functionality with Godot's `ClassDB`
   - `ClassDB` accesses metadata for all classes available to Godot
4. Build plugin as a **dynamic library** to link into Godot projects without recompiling engine

GDEXTENSION PLUGIN COMPILATION

- Plugins must be compiled to work with the associated Godot project
  - Because of setup, there is no “hot reload”
    - Must close and open Godot editor to see changes
- `gdproject.gdextension` connects GDExtension dynamic libraries and any additional dynamic libraries used in C++ files
  - Must point to a properly placed plugin (i.e. the dynamic library)

STATIC VERSUS DYNAMIC LIBRARIES?

- Static libraries are connected to a program at compile time with the object code built into executable
  - Faster at runtime
  - Fewer compatibility issues
  - Must recompile program if library code is modified
  - Larger executable file size
- Dynamic libraries can be shared by programs and are connected at program runtime
  - Faster compile time
  - More possibilities for breaking
  - Smaller memory footprint at runtime
  - Smaller executable file size

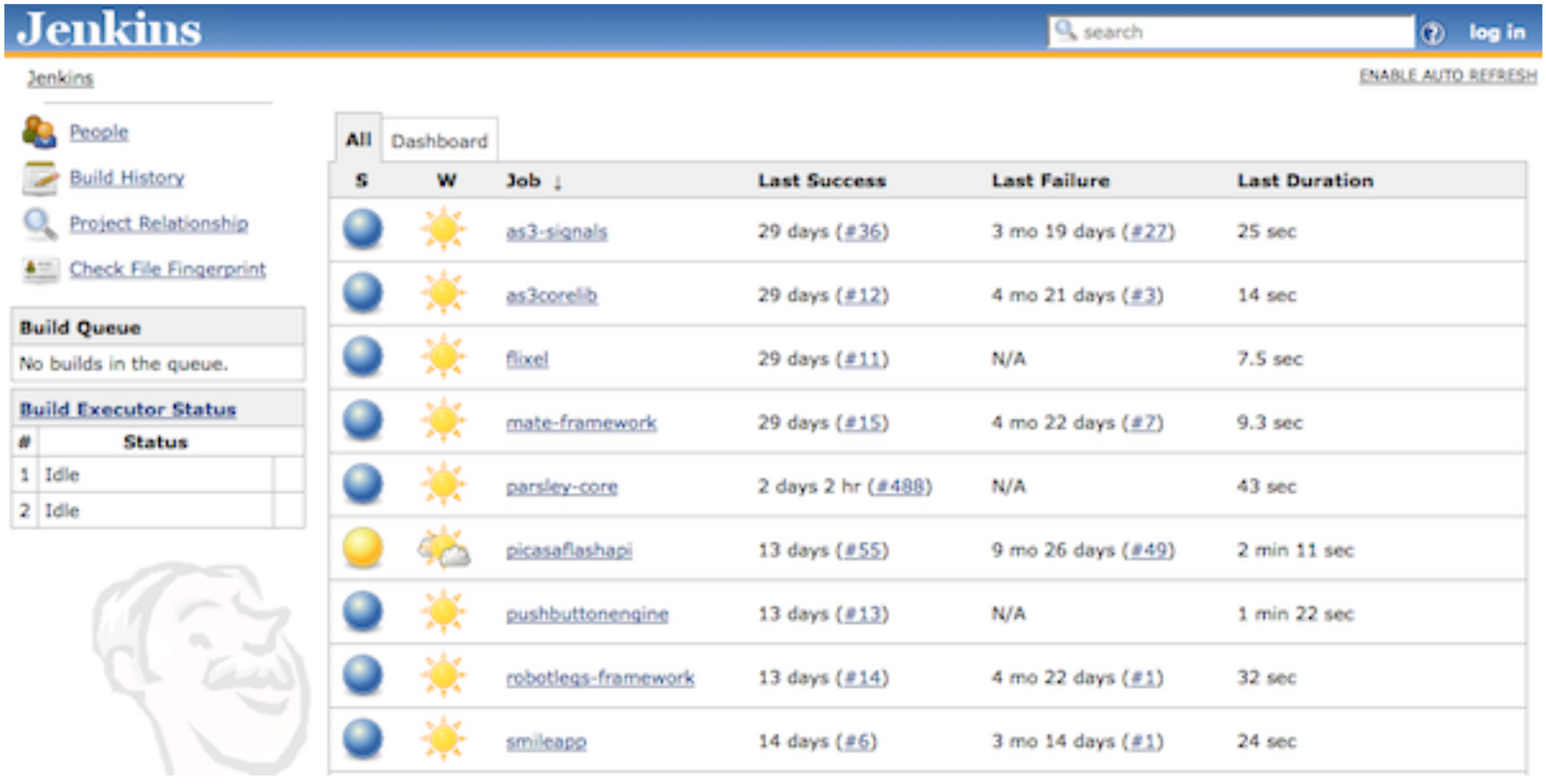
CONSIDER...

- Why does Godot structure GDExtension the way it does?

CONTINUOUS INTEGRATION

- Developer code is frequently committed to the shared repository
- Advantages:
  - Prevents late-stage problems
  - Keeps work pipeline flowing
- Requires:
  - Well-established work flow
  - Automatic build scheduling
  - Relatively fast builds
  - Unit tests to prevent erroneous code (in theory)
- What sort of branching schema work well for this?

CI SYSTEMS



Example: Jenkins

QUALITY ASSURANCE

- Quality Assurance (QA) assures product's quality is at acceptable, expected level for customers
- Feedback loop:
  - Design –> Develop –> Test
- Dedicated QA expedites process of tracking and correcting bugs and features
- Complementary role to designers and developers
- Game QA is generally a meat-grinder, but QA in other software industries can be senior-level programmers or designers

IDEAL BUG REPORTS

- Bug reports should have:
  - Descriptive title
  - Encountered behavior
  - Expected behavior
  - Steps to reproduce
  - Screenshots or video of bug
- Useful for asking about issues on Piazza/Discord, incidentally!

QA VERSUS USER TESTING

- QA is often internal to a project
  - Testers are trained and directed
  - At least some understanding of project's systems
    - Often looking to break things
- User testing validates design by taking product "into the wild"
  - Testers are likely part of product's target demographic
  - No understanding of project's systems required
  - Ideally interact with system in "expected" use-case

BUG TRIAGE

- Process of assessing bug severity and priority
- Bug severity determines how serious (i.e. game-breaking/profit-losing, etc) a bug is
- Bug priority determines how important it is to fix a bug
- Some examples:
  - What could be a high severity/high priority bug?
  - What could be a low severity/low priority bug?
  - What could be a high severity/low priority bug?
  - What could be a low severity/high priority bug?

REFERENCES

- Make
  - http://www.gnu.org/software/make/manual/make.html
- CMake
  - https://cmake.org/cmake-tutorial/
  - https://github.com/robbie-cao/note/blob/master/cmake.md
- SCons
  - https://github.com/SCons/scons/wiki/sconsvsotherbuildtools
- QA
  - http://qablog.practitest.com/2008/12/principles-of-good-bug-reporting/