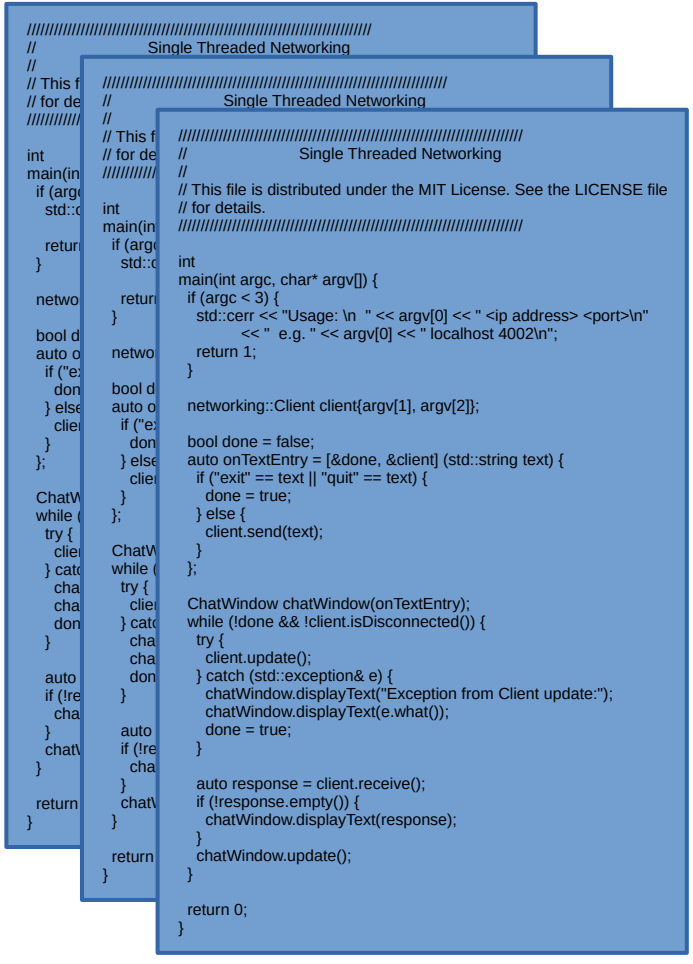
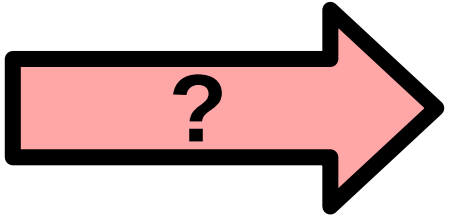
CMPT 373
Software Development Methods

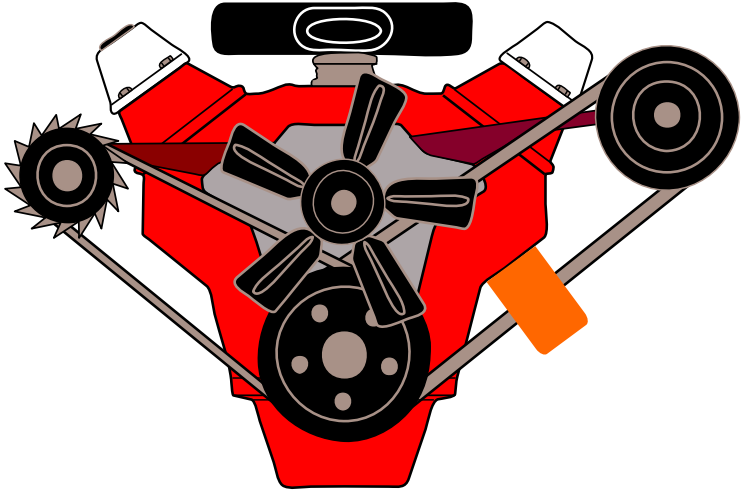
Building Software

Nick Sumner
wsumner@sfu.ca

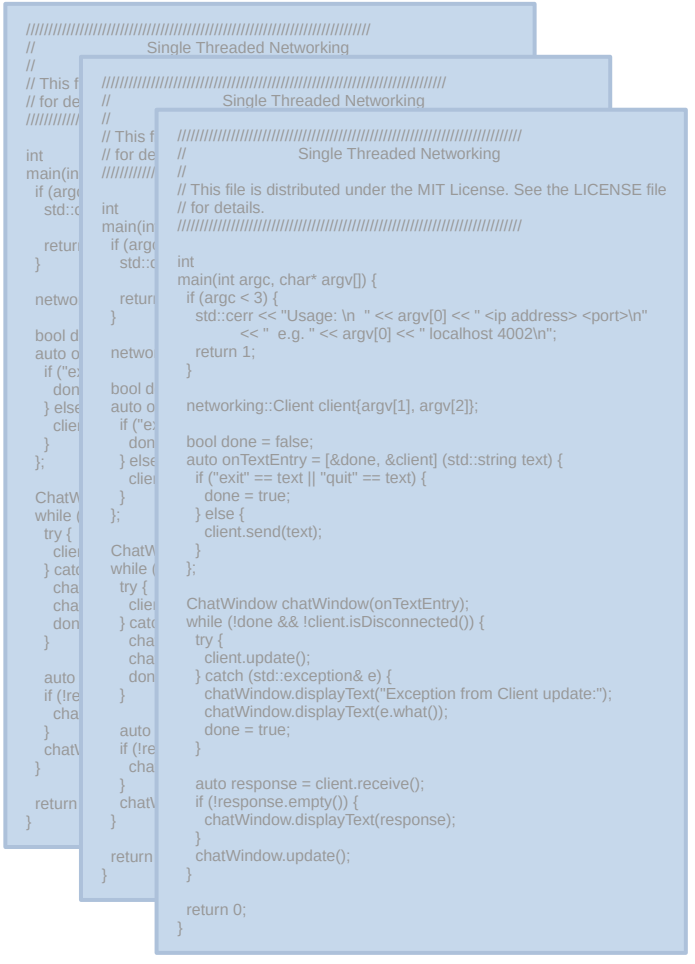
What does it take to build software?

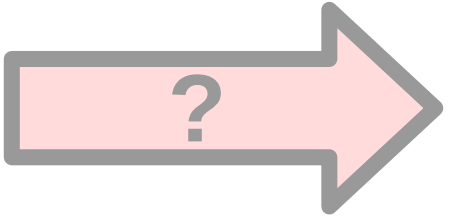


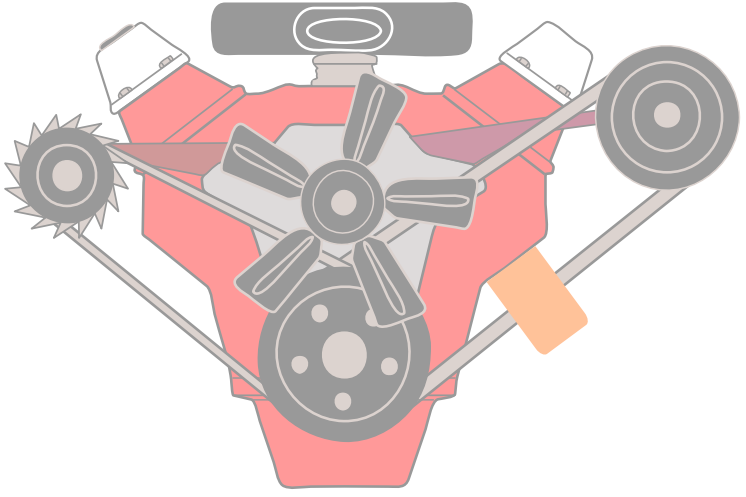




What does it take to build software?

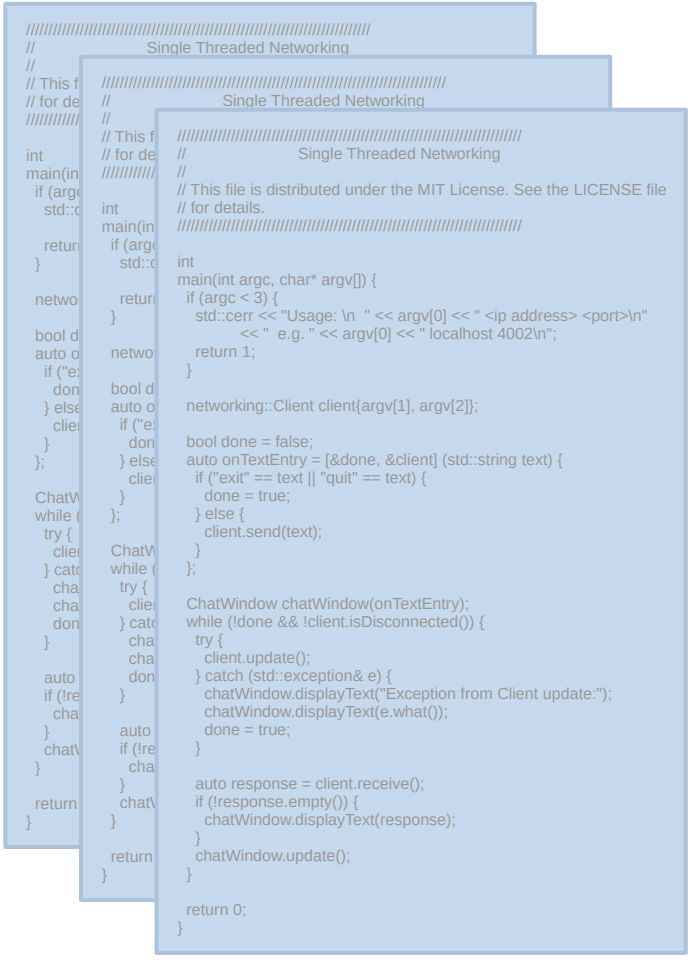


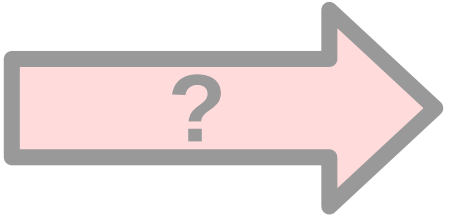


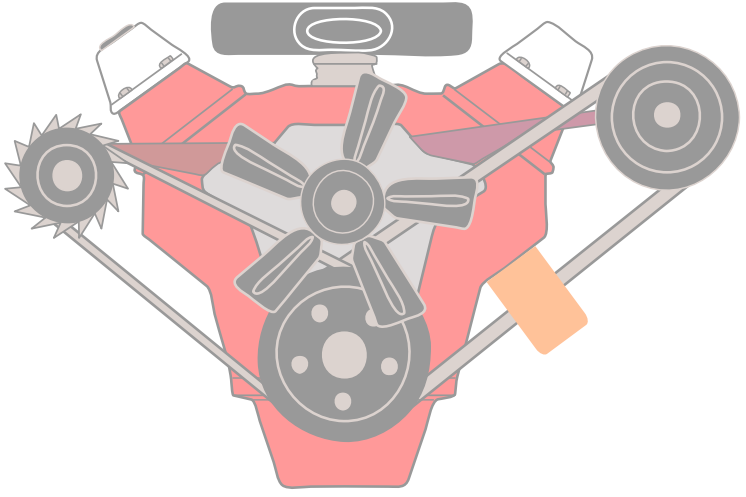


```
gcc source1.c source2.c -o myprogram
```

What does it take to build software?



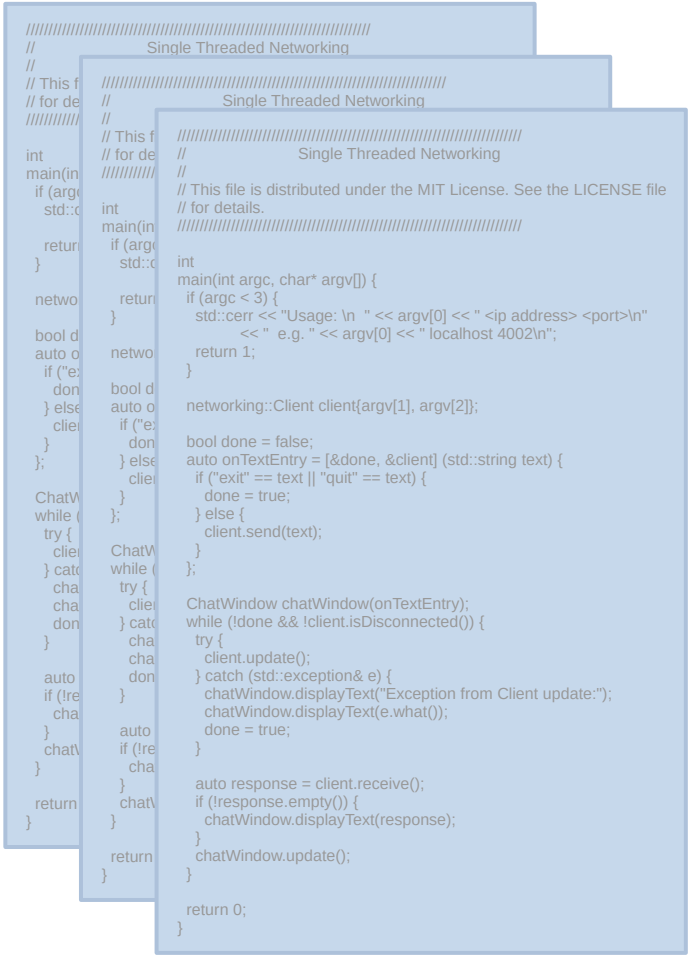


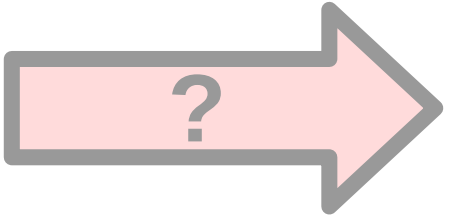


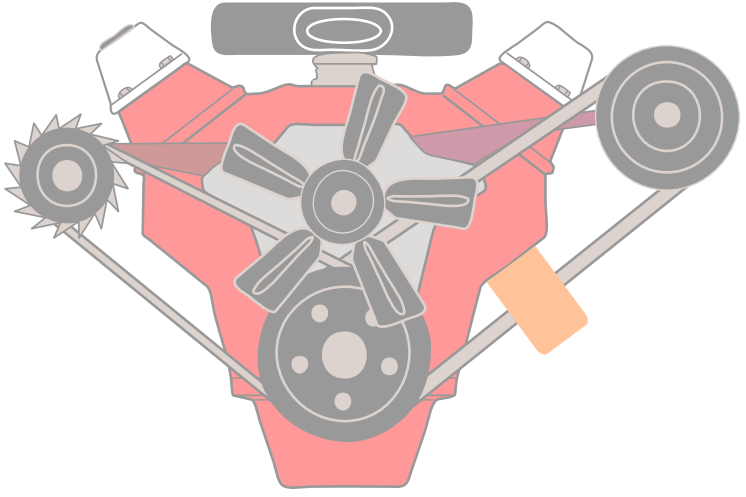
```
gcc source1.c source2.c -o myprogram
```

CTRL + SHIFT + B or maybe F5?

What does it take to build software?



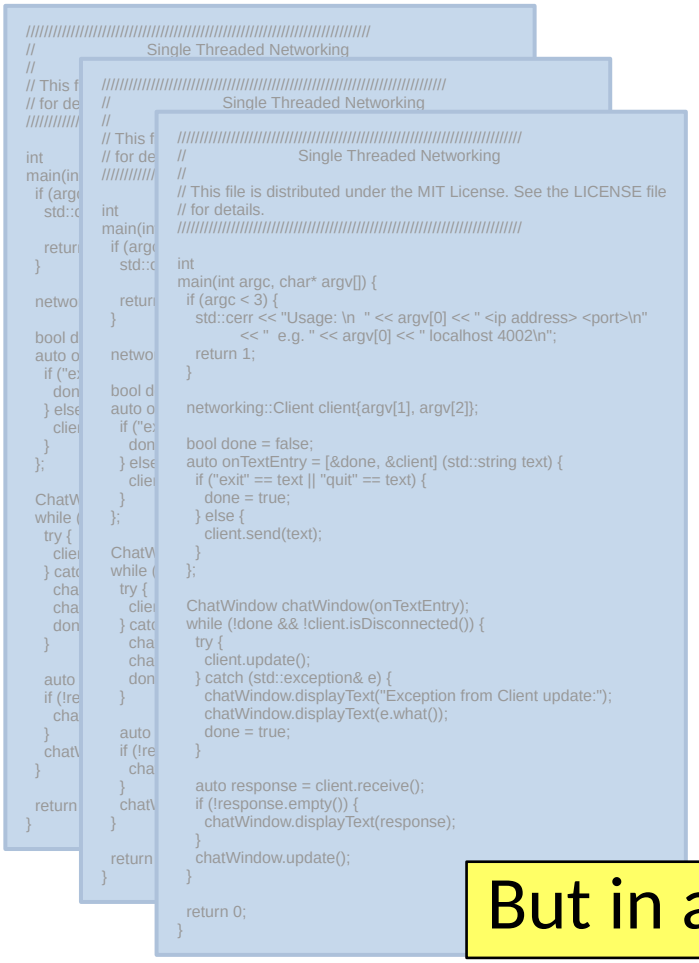


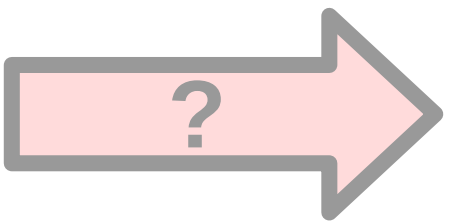


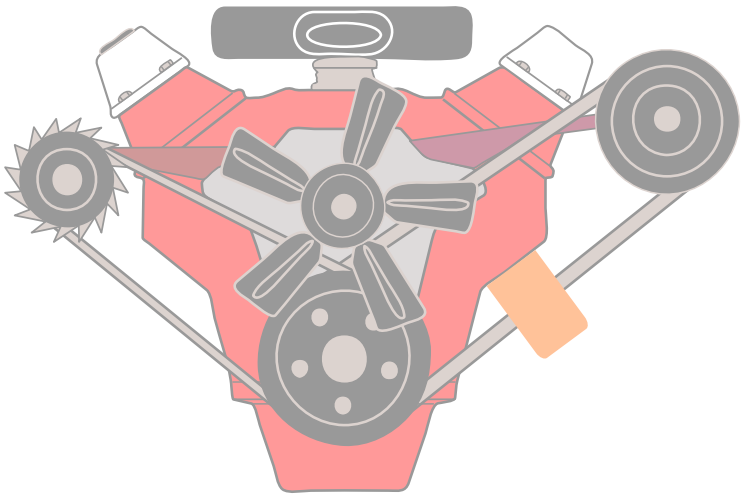
```
gcc source1.c source2.c -o myprogram
```

CTRL + SHIFT + B or maybe F5?

What does it take to build software?







```
gcc source1.c source2.c -o myprogram
```

CTRL + SHIFT + B or maybe F5?

But in a real project it can *involve* & *do* a lot more!

What does it take to build software?

- Build Engineering?
- Release Engineering?
- Build Configuration?
- Build Automation?
- Dependency Management?
- Continuous Integration?

A lot more.

What does it take to build software?

- Build Engineering?
- Release Engineering?
- Build Configuration?
- Build Automation?
- Dependency Management?
- Continuous Integration?

A lot more.

Just getting something to compile reproducibly can be nontrivial

What does it mean to build software?

- Building software includes (at least):

What does it mean to build software?

- Building software includes (at least):
  - version control integration

What does it mean to build software?

- Building software includes (at least):
  - version control integration
  - identifying dependencies & their versions

What does it mean to build software?

- Building software includes (at least):
  - version control integration
  - identifying dependencies & their versions
  - configuring build commands for different build modes & environments

What does it mean to build software?

- Building software includes (at least):
  - version control integration
  - identifying dependencies & their versions
  - configuring build commands for different build modes & environments
  - writing instructions for how to configure & build

What does it mean to build software?

- Building software includes (at least):
  - version control integration
  - identifying dependencies & their versions
  - configuring build commands for different build modes & environments
  - writing instructions for how to configure & build
  - test configuration & execution (performance & correctness)

What does it mean to build software?

- Building software includes (at least):
  - version control integration
  - identifying dependencies & their versions
  - configuring build commands for different build modes & environments
  - writing instructions for how to configure & build
  - test configuration & execution (performance & correctness)
  - automated code quality checking

What does it mean to build software?

- Building software includes (at least):
  - version control integration
  - identifying dependencies & their versions
  - configuring build commands for different build modes & environments
  - writing instructions for how to configure & build
  - test configuration & execution (performance & correctness)
  - automated code quality checking
  - scalable compilation & linking (caching, parallelism, scheduling, …)

What does it mean to build software?

- Building software includes (at least):
  - version control integration
  - identifying dependencies & their versions
  - configuring build commands for different build modes & environments
  - writing instructions for how to configure & build
  - test configuration & execution (performance & correctness)
  - automated code quality checking
  - scalable compilation & linking (caching, parallelism, scheduling, …)
  - possibly even deployment

What does it mean to build software?

- Building software includes (at least):
  - version control integration
  - identifying dependencies & their versions
  - configuring build commands for different build modes & environments
  - writing instructions for how to configure & build
  - test configuration & execution (performance & correctness)
  - automated code quality checking
  - scalable compilation & linking (caching, parallelism, scheduling, …)
  - possibly even deployment
- It is the foundation of getting anything done.

What does it mean to build software?

- How many of you know how to ***build*** software?

What does it mean to build software?

- How many of you know how to ***build*** software?
- You should at least ask yourself:
  - What tools do you use?

What does it mean to build software?

- How many of you know how to ***build*** software?
- You should at least ask yourself:
  - What tools do you use?

Grunt? Gulp? Cmake? Bazel? Gradle? Maven? Scons?
Incredibuild? CloudBuild?
Travis? Jenkins? CircleCI?
Junit? Cucumber? Pytest? Gtest?
Coverity? Clang Static Analyzer?
OpenTelemetry? Prometheus? Jaeger?
…

What does it mean to build software?

- How many of you know how to ***build*** software?
- You should at least ask yourself:
  - What tools do you use?
  - What workflow?
  - What are the painful points?
  - What are the risks?
  - What benefits do you get?
  - Why haven't you made them less painful?

What does it mean to build software?

- How many of you know how to ***build*** software?
- You should at least ask yourself:
  - What tools do you use?
  - What workflow?
  - What are the painful points?
  - What are the risks?
  - What benefits do you get?
  - Why haven't you made them less painful?

What does it mean to build software?

- How many of you know how to ***build*** software?
- You should at least ask yourself:
  - What tools do you use?
  - What workflow?
  - What are the painful points?
  - What are the risks?
  - What benefits do you get?
  - Why haven't you made them less painful?

What does it mean to build software?

- How many of you know how to ***build*** software?
- You should at least ask yourself:
  - What tools do you use?
  - What workflow?
  - What are the painful points?
  - What are the risks?
  - What benefits do you get?
  - Why haven't you made them less painful?

What does it mean to build software?

- How many of you know how to ***build*** software?
- You should at least ask yourself:
  - What tools do you use?
  - What workflow?
  - What are the painful points?
  - What are the risks?
  - What benefits do you get?
  - Why haven't you made them less painful?

Modeling a Build

- To build software, we must consider:

Modeling a Build

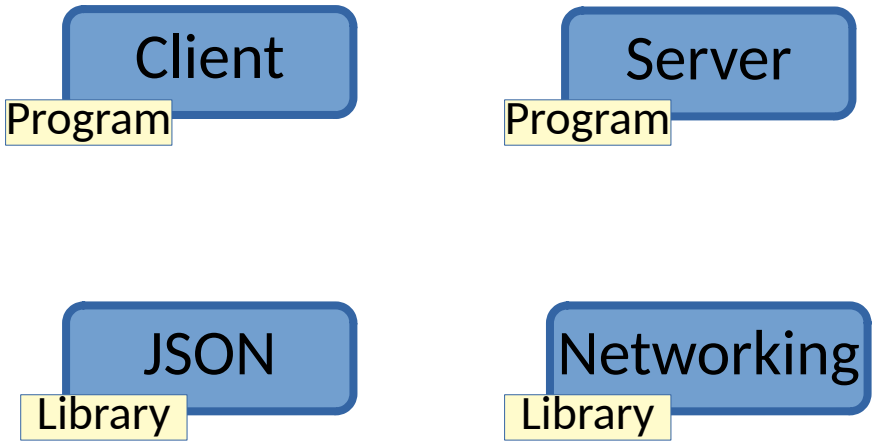
- To build software, we must consider:
  - **Components & Objectives**

Modeling a Build

- To build software, we must consider:
  - Components & Objectives

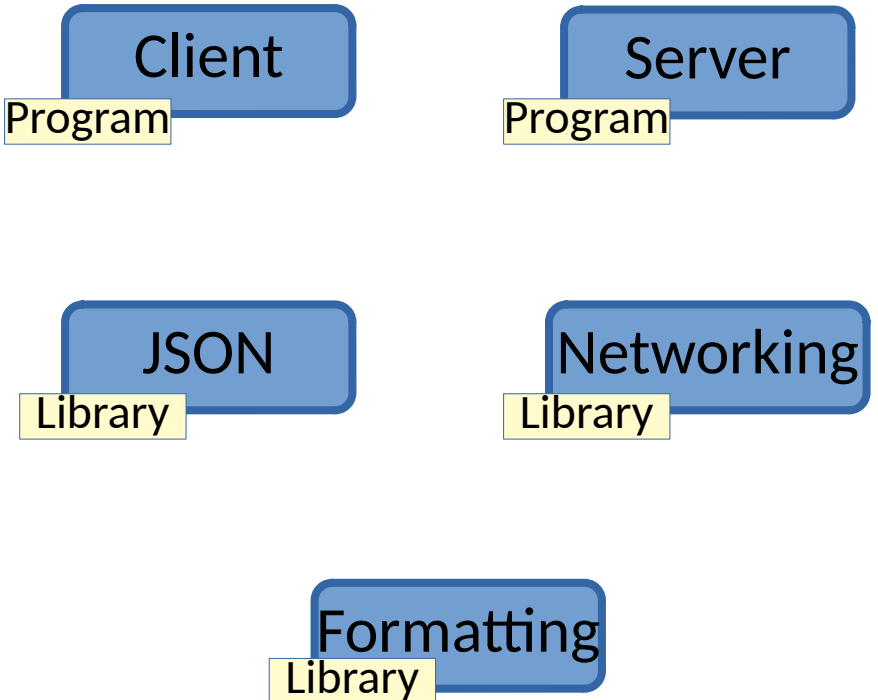
Modeling a Build

- To build software, we must consider:
  - Components & Objectives



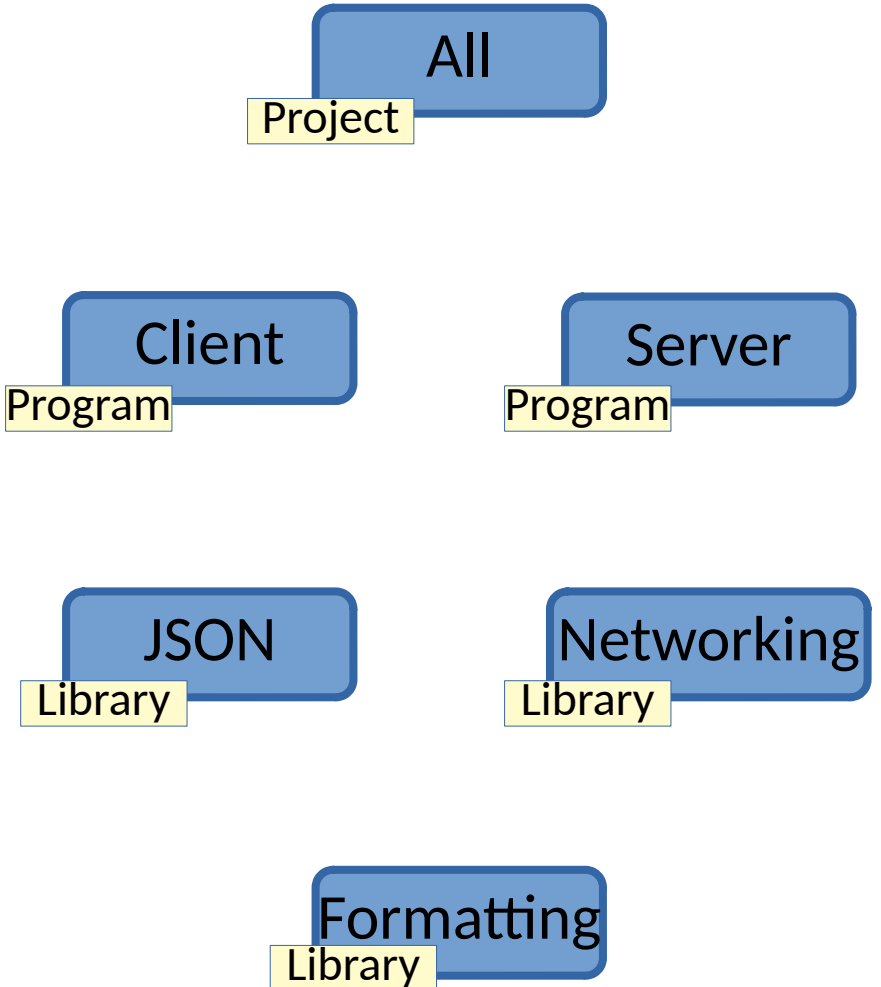
Modeling a Build

- To build software, we must consider:
  - Components & Objectives



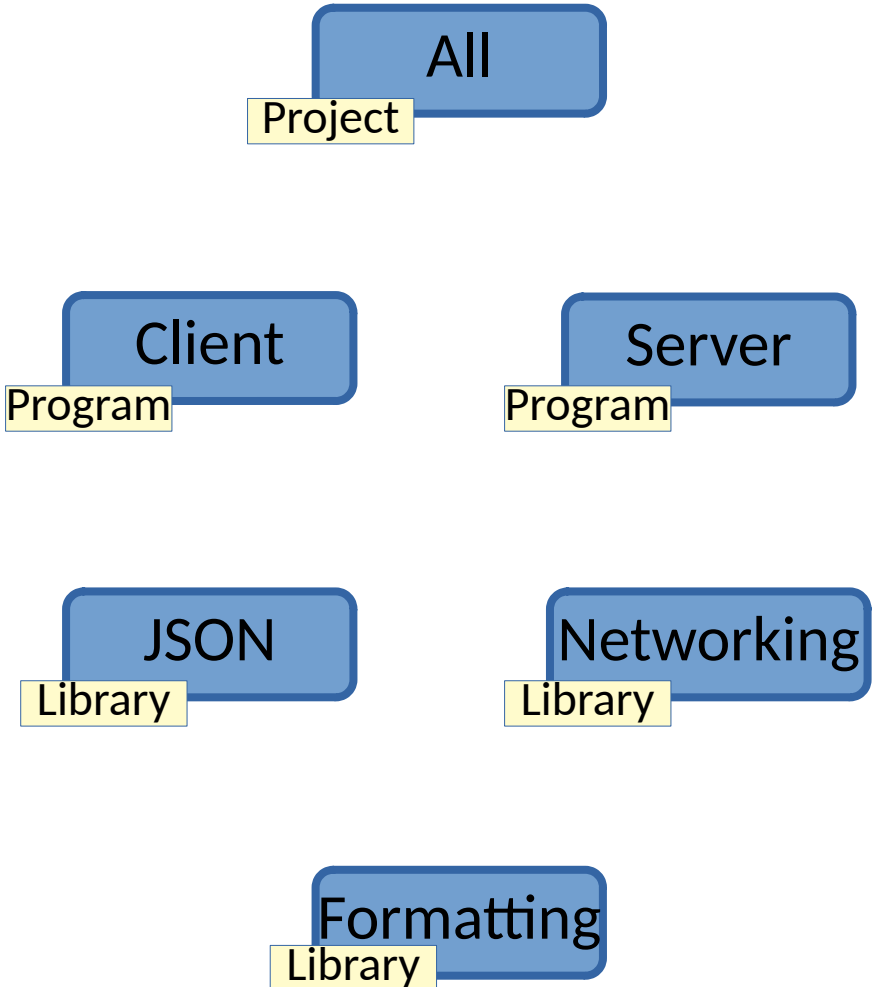
Modeling a Build

- To build software, we must consider:
  - Components & Objectives



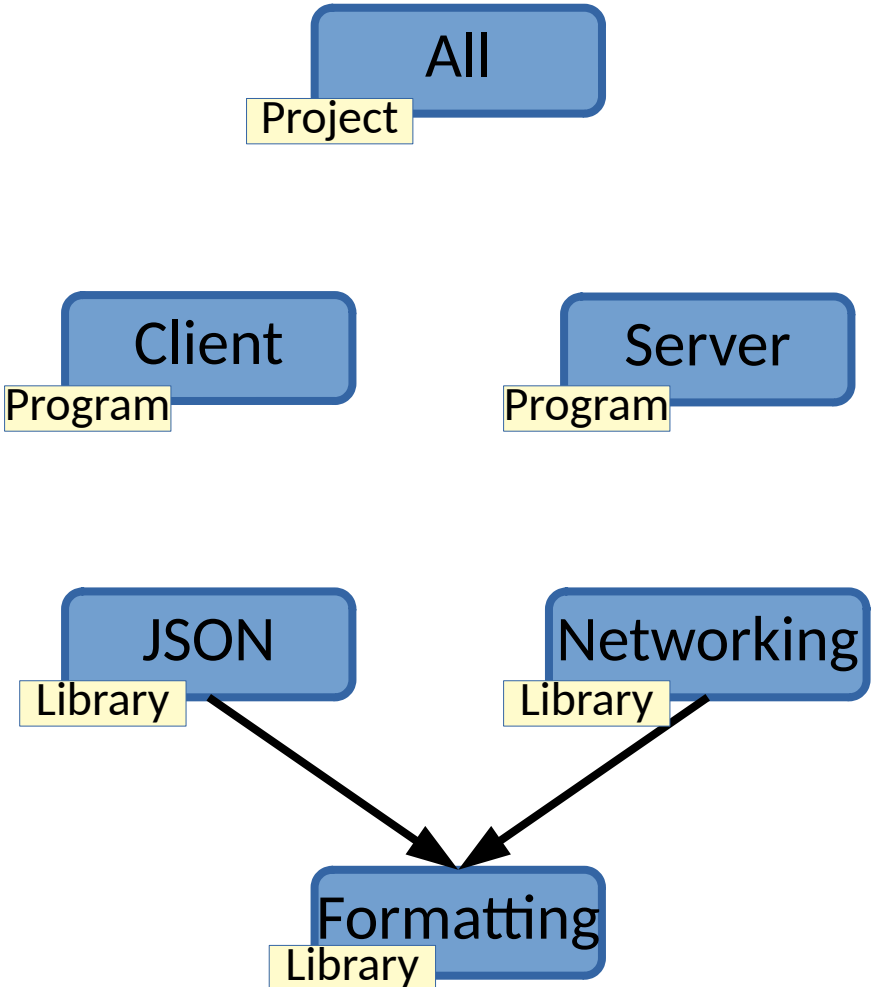
Modeling a Build

- To build software, we must consider:
  - Components & Objectives
  - **Dependencies between them**



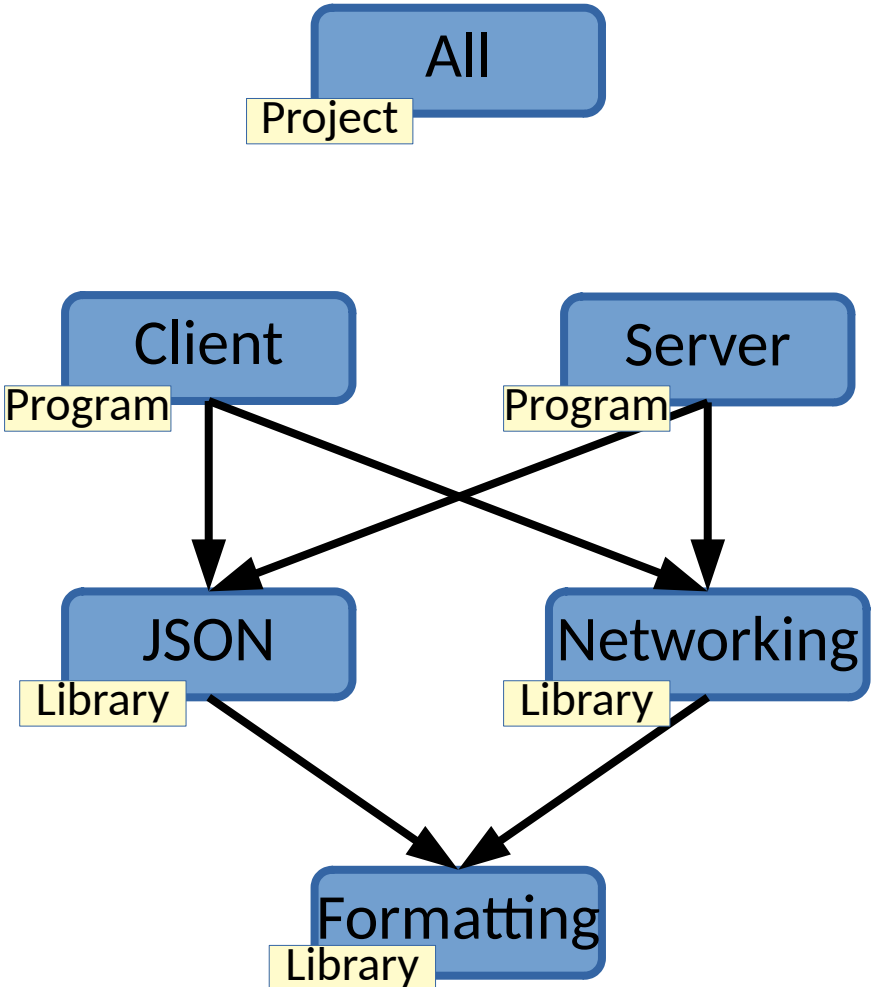
Modeling a Build

- To build software, we must consider:
  - Components & Objectives
  - Dependencies between them



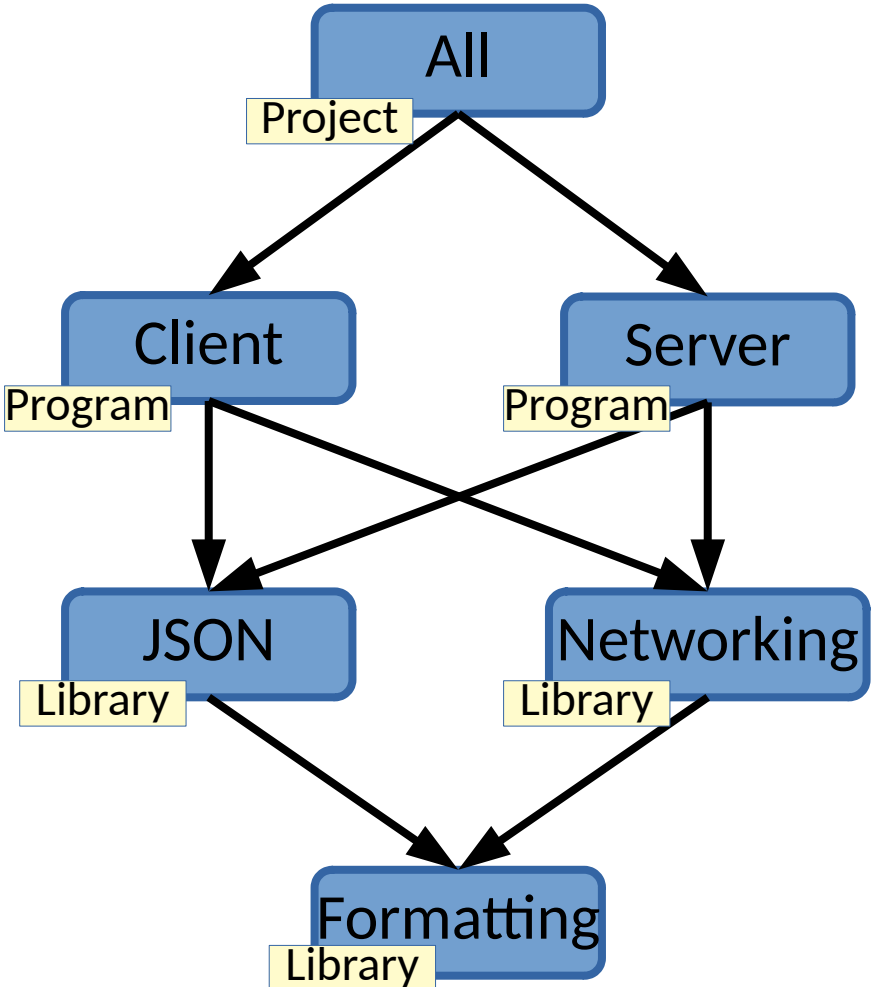
Modeling a Build

- To build software, we must consider:
  - Components & Objectives
  - Dependencies between them



Modeling a Build

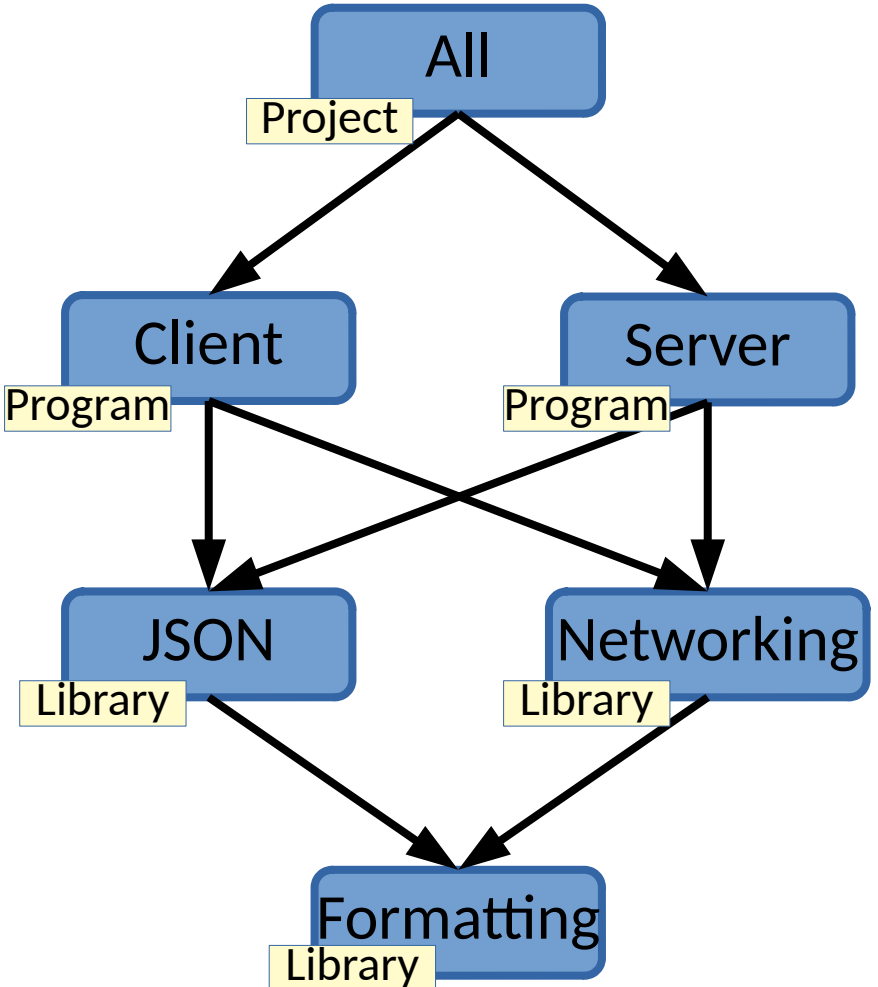
- To build software, we must consider:
  - Components & Objectives
  - Dependencies between them



Modeling a Build

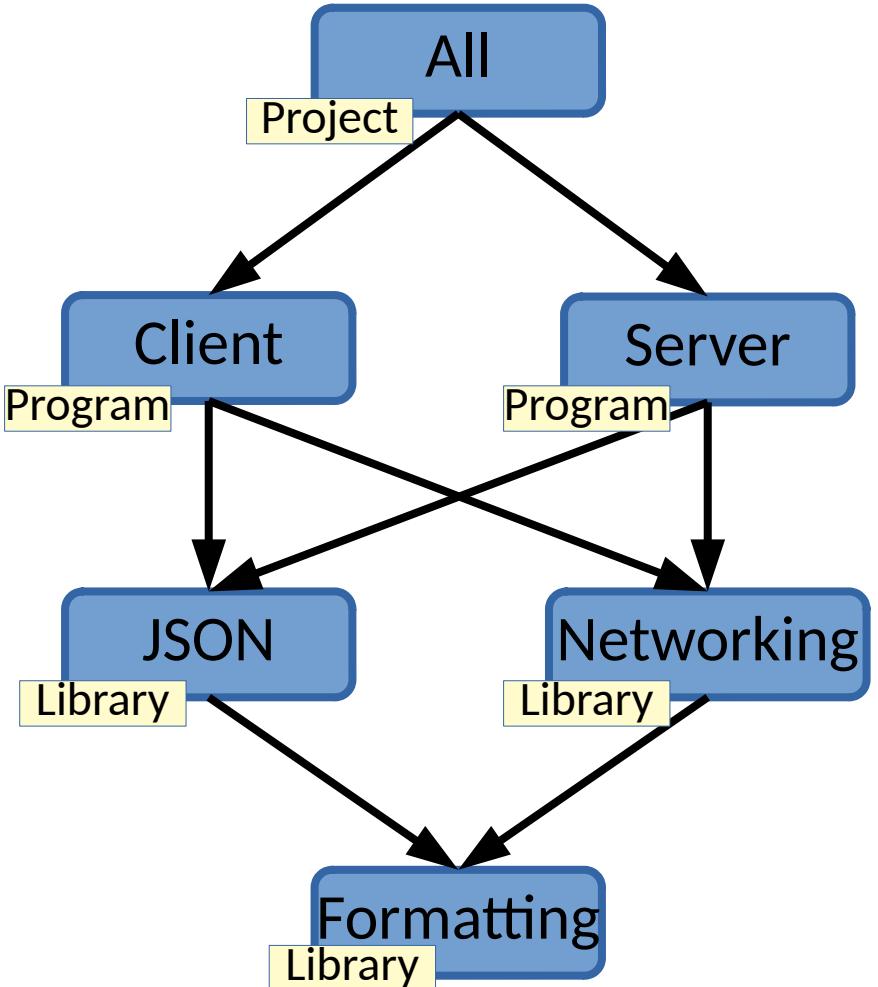
- To build software, we must consider:
  - Components & Objectives
  - Dependencies between them

This defines the ***dependency graph*** of a project.



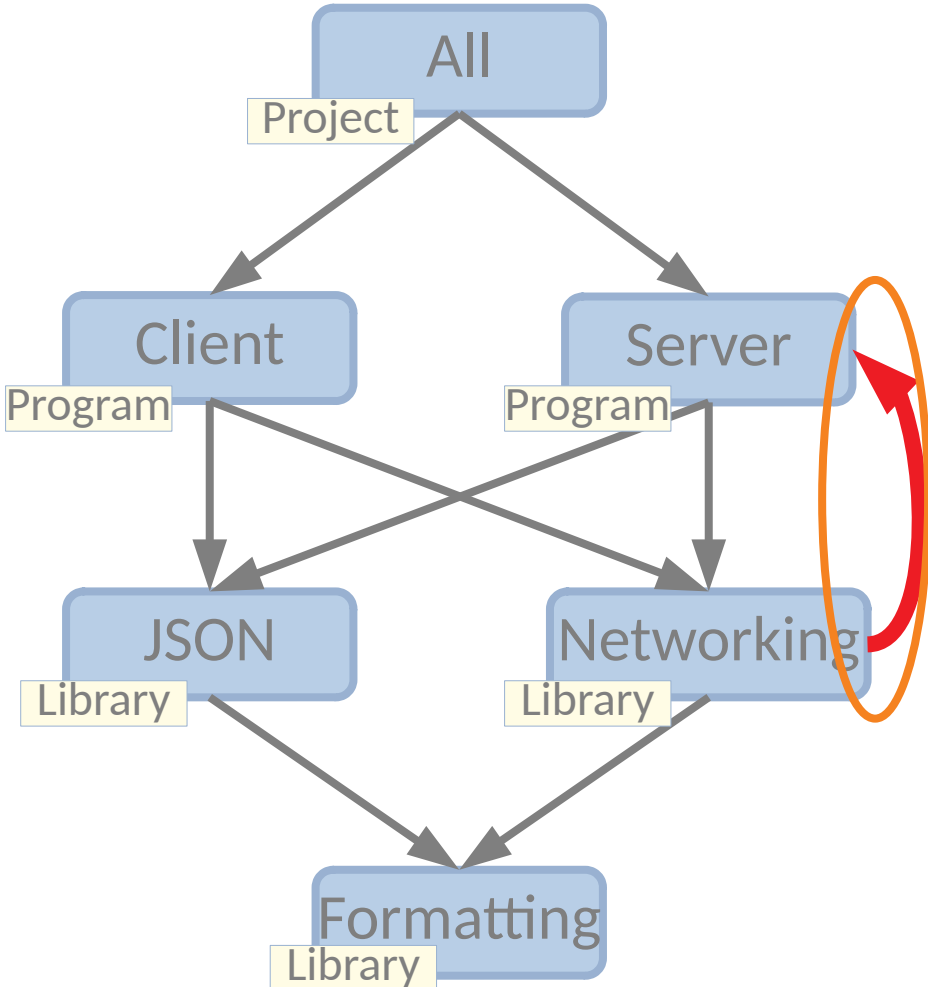
Modeling a Build

- To build software, we must consider:
  - Components & Objectives
  - Dependencies between them
- The dependency graph can already help to analyze our project!



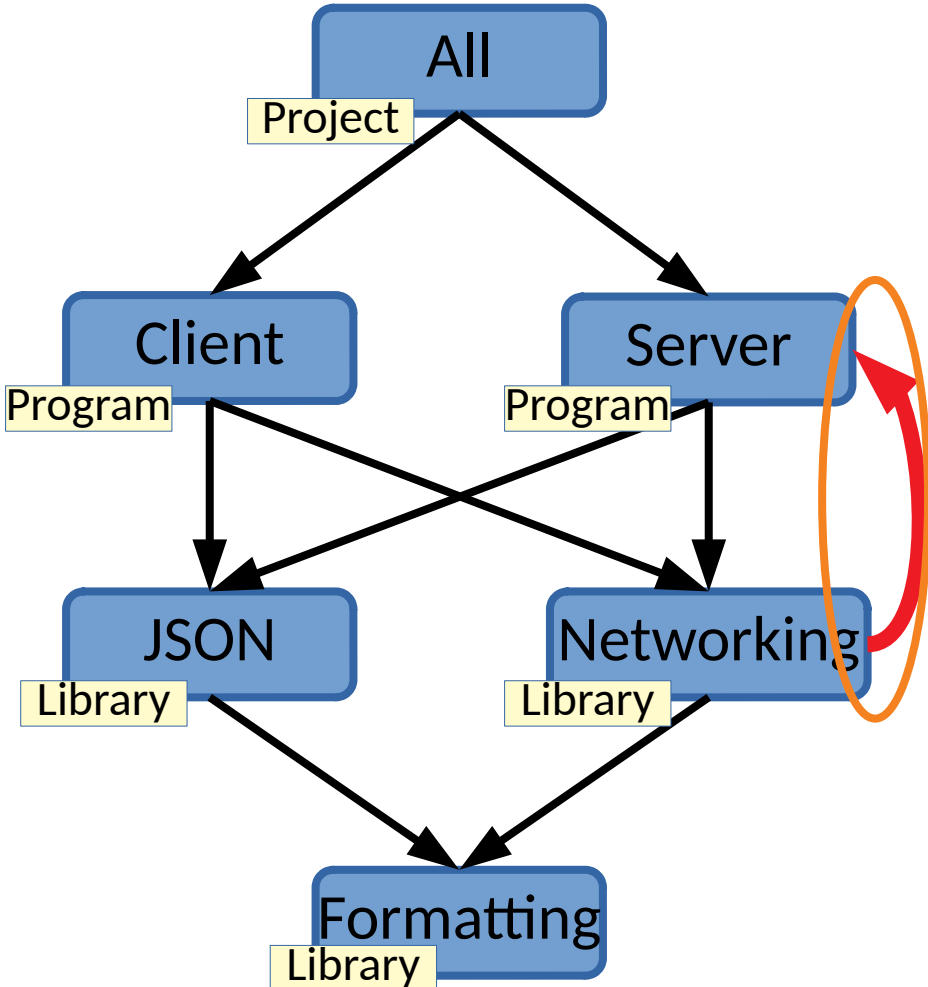
Modeling a Build

- To build software, we must consider:
  - Components & Objectives
  - Dependencies between them
- The dependency graph can already help to analyze our project!



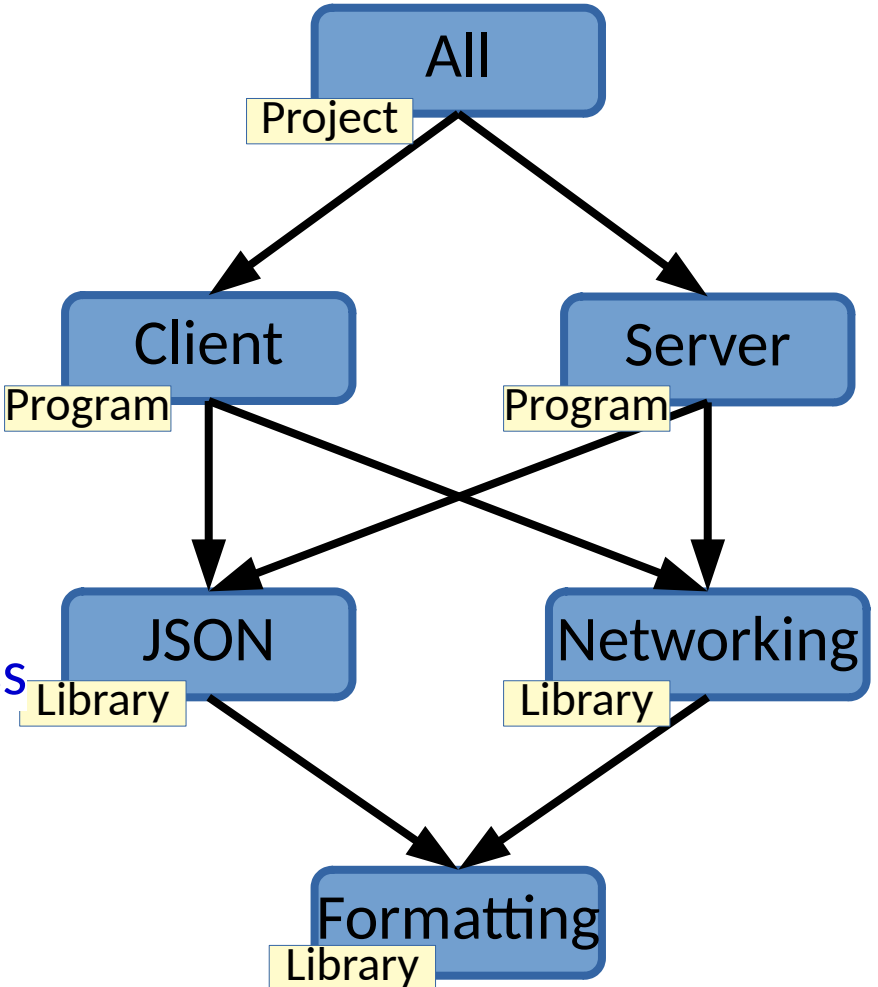
Modeling a Build

- To build software, we must consider:
  - Components & Objectives
  - Dependencies between them
- The dependency graph can already help to analyze our project!
  - A good dependency graph is a DAG



Modeling a Build

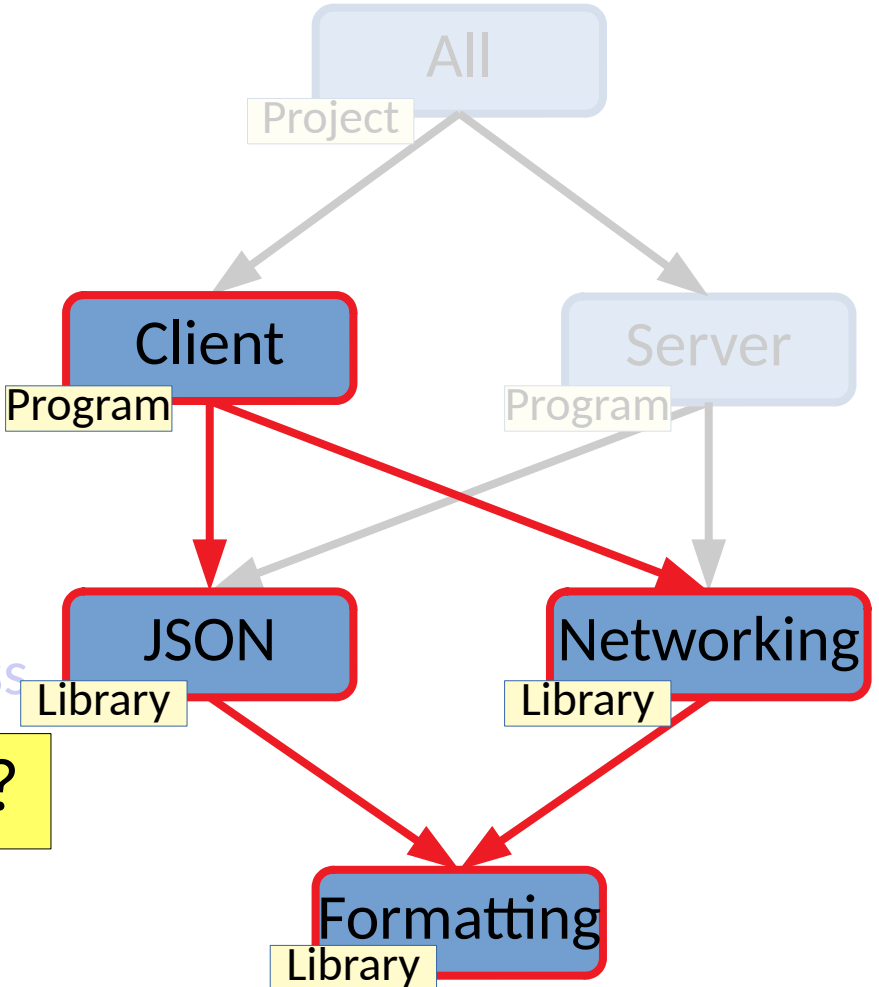
- To build software, we must consider:
  - Components & Objectives
  - Dependencies between them
- The dependency graph can already help to analyze our project!
  - A good dependency graph is a DAG
- Modern build management uses the dependency DAG to drive build process



Modeling a Build

- To build software, we must consider:
  - Components & Objectives
  - Dependencies between them
- The dependency graph can already help to analyze our project!
  - Actual dependency graph is a DAG
- Modern build management uses the dependency DAG to drive build process

We can consider C++, but it applies in general (even for many dynamic languages)



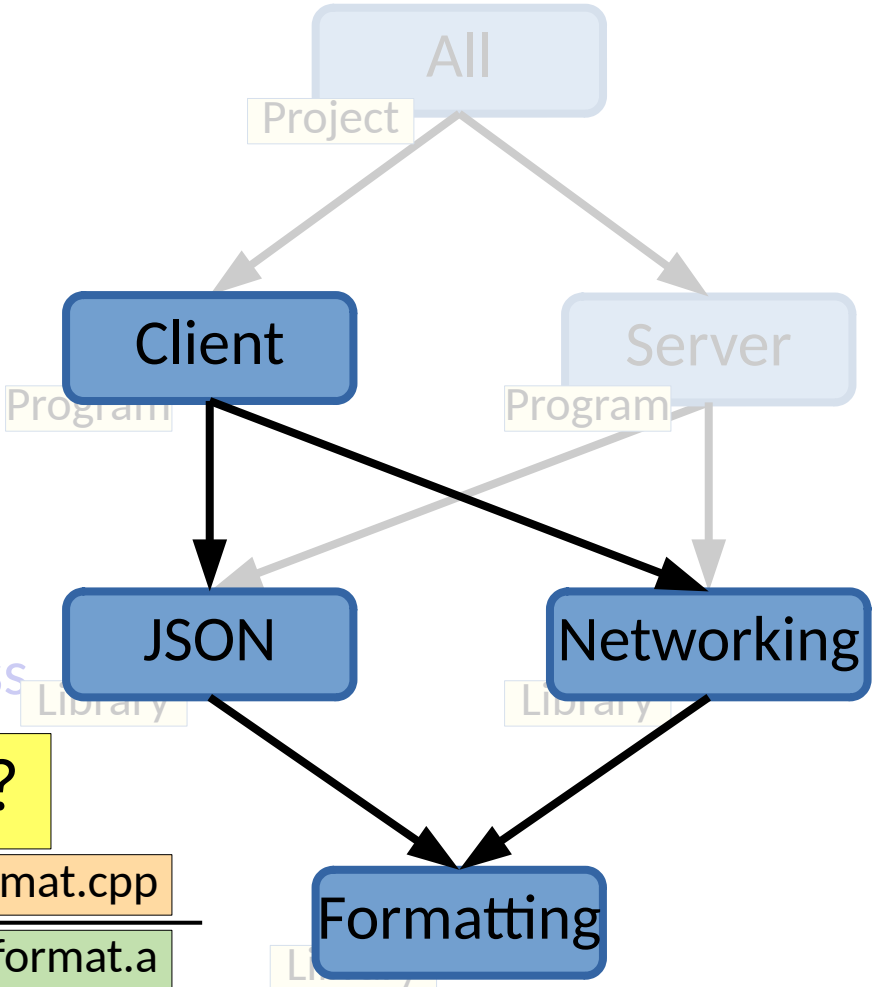
What must the build system perform?

used / required

generated

Modeling a Build

- To build software, we must consider:
  - Components & Objectives
  - Dependencies between them
- The dependency graph can already help to analyze our project!
  - A good dependency graph is a DAG
- Modern build management uses the dependency DAG to drive build process



What must the build system perform?

used / required

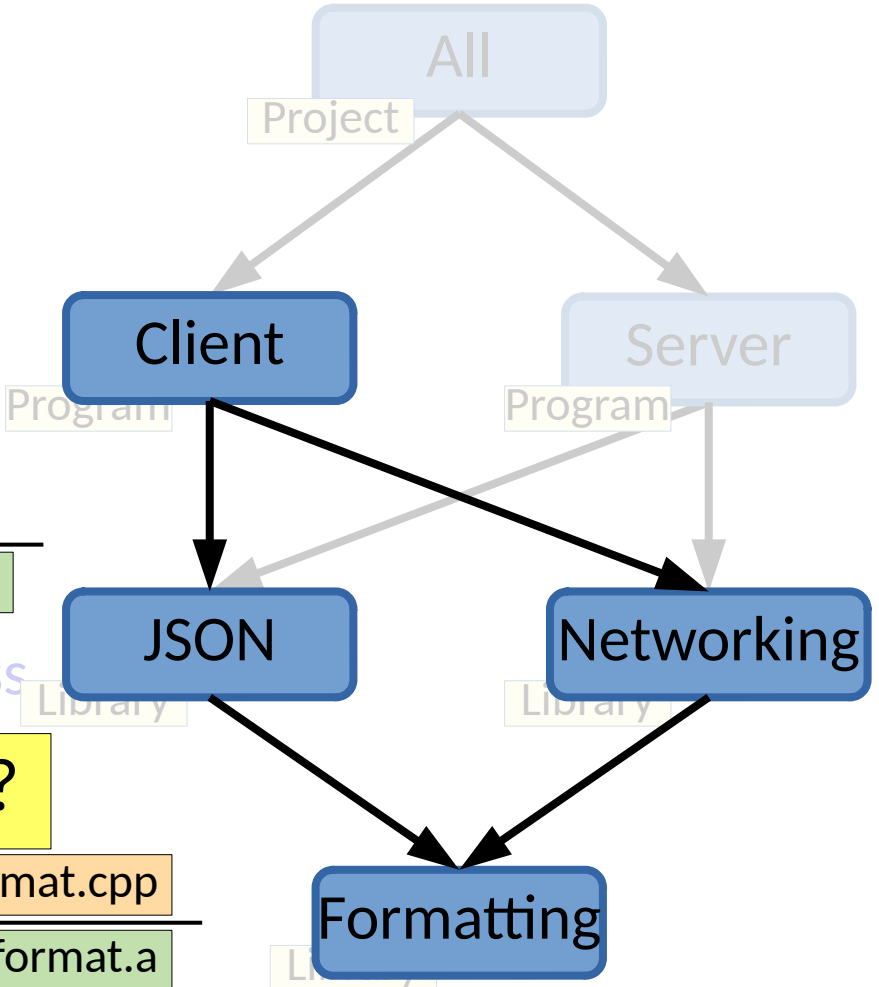
generated

format.cpp

libformat.a

Modeling a Build

- To build software, we must consider:
  - Components & Objectives
  - Dependencies between them
- The dependency graph can already help to analyze our project!
  - A good dependency graph is a DAG
- Modern build management uses the dependency DAG to drive build process



libjson.a

What must the build system perform?

used / required

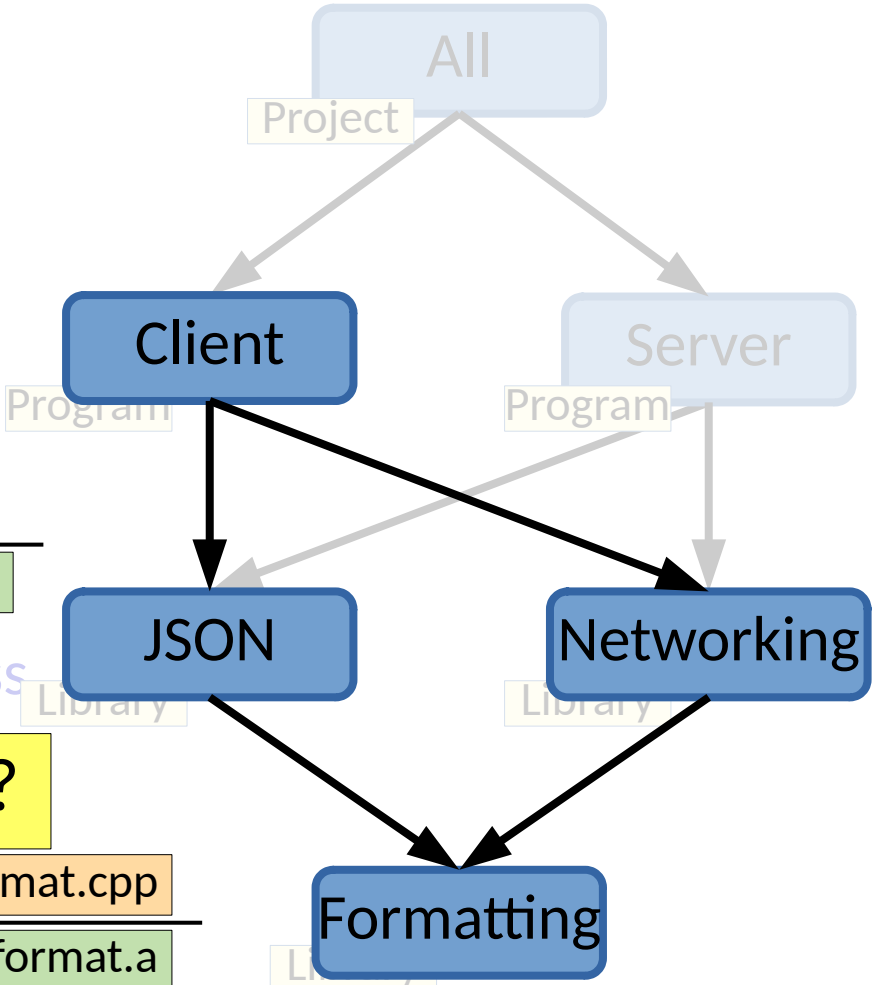
generated

format.cpp

libformat.a

Modeling a Build

- To build software, we must consider:
  - Components & Objectives
  - Dependencies between them
- The dependency graph can already help to analyze our project!
  - A good dependency graph is a DAG
- Modern build management uses the dependency DAG to drive build process



json.cpp

libjson.a

format.cpp

libformat.a

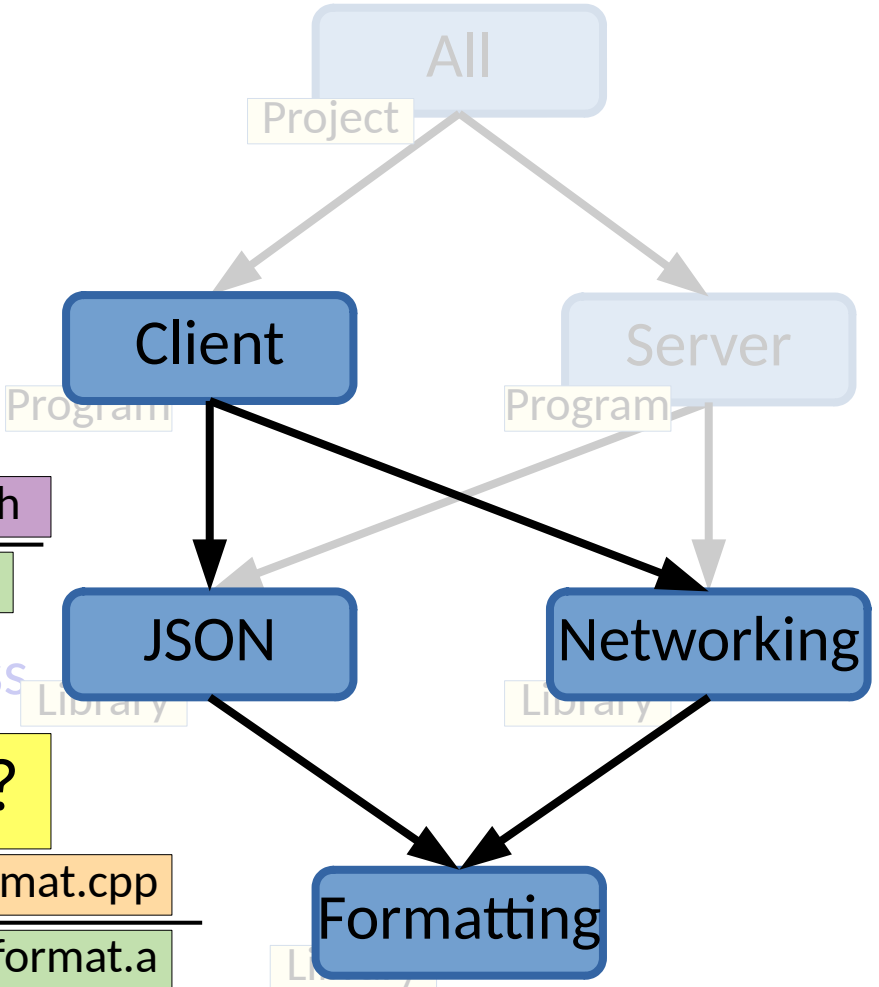
What must the build system perform?

used / required

generated

Modeling a Build

- To build software, we must consider:
  - Components & Objectives
  - Dependencies between them
- The dependency graph can already help to analyze our project!
  - A good dependenc
- Modern build management uses the dependency DAG to drive build process



json.cpp <…>/format.h

libjson.a

format.cpp

libformat.a

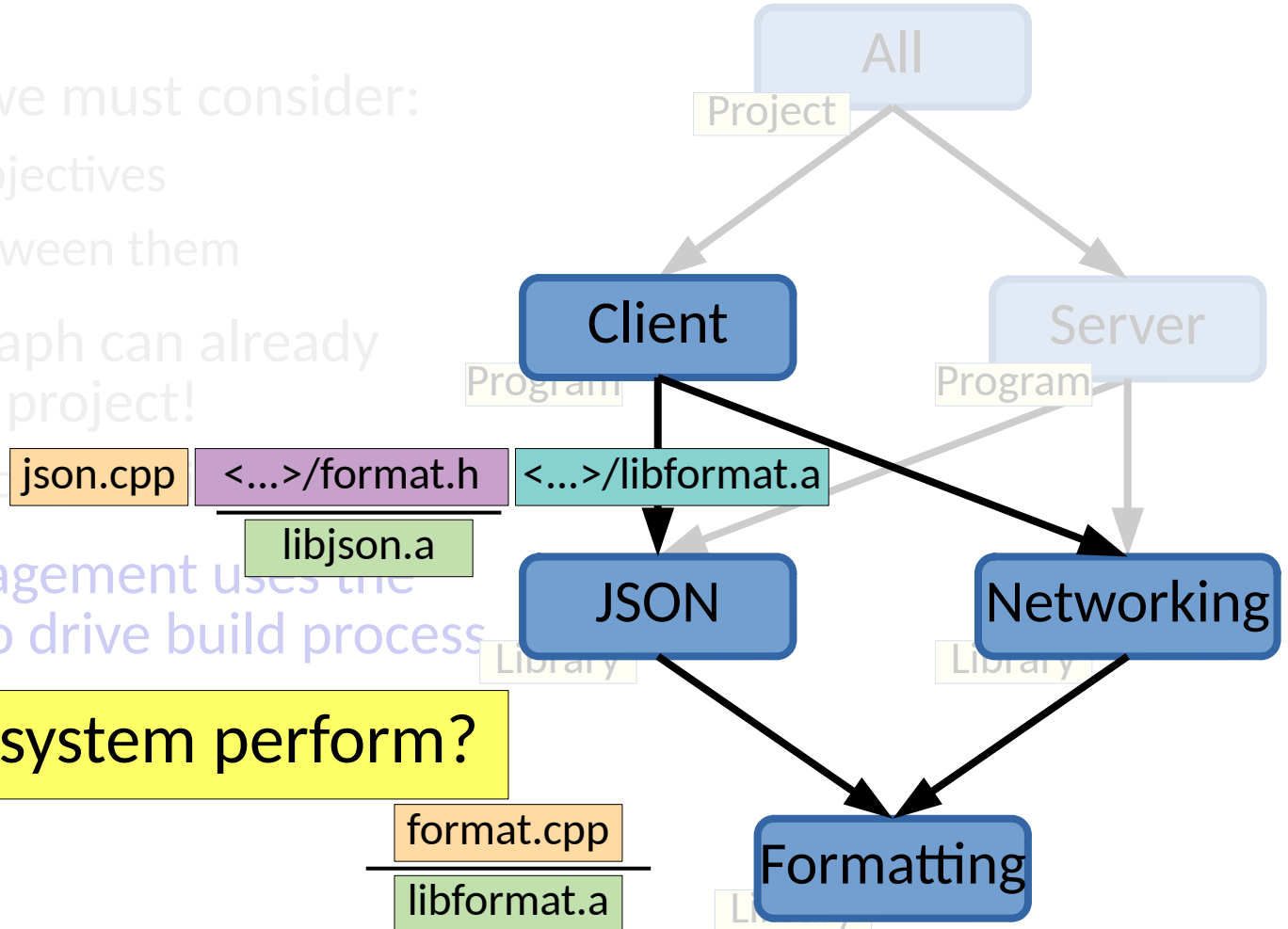
What must the build system perform?

used / required

generated

Modeling a Build

- To build software, we must consider:
  - Components & Objectives
  - Dependencies between them
- The dependency graph can already help to analyze our project!
  - A good dependenc
- Modern build management uses the dependency DAG to drive build process



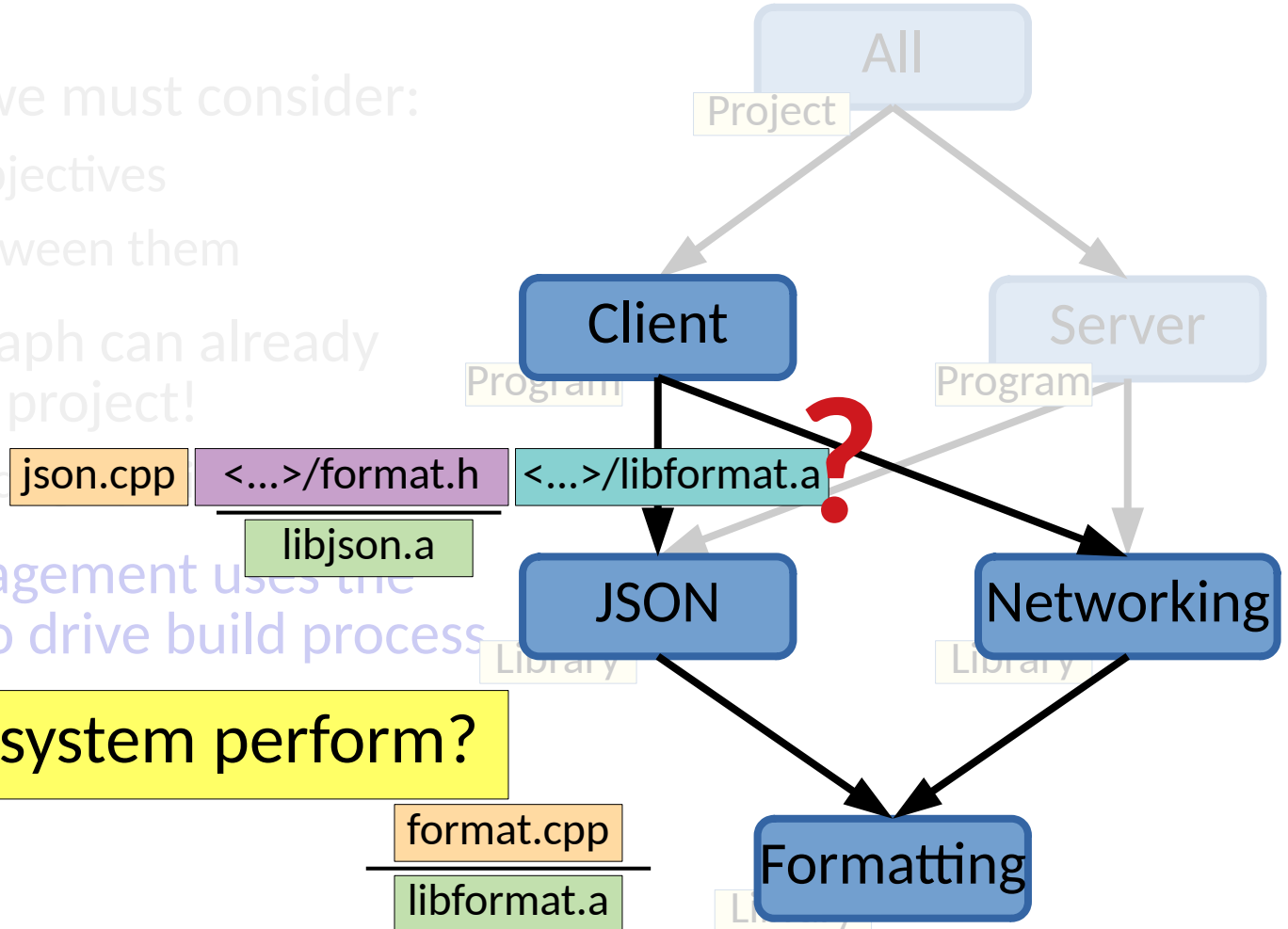
What must the build system perform?

used / required

generated

Modeling a Build

- To build software, we must consider:
  - Components & Objectives
  - Dependencies between them
- The dependency graph can already help to analyze our project!
  - A good dependenc
- Modern build management uses the dependency DAG to drive build process



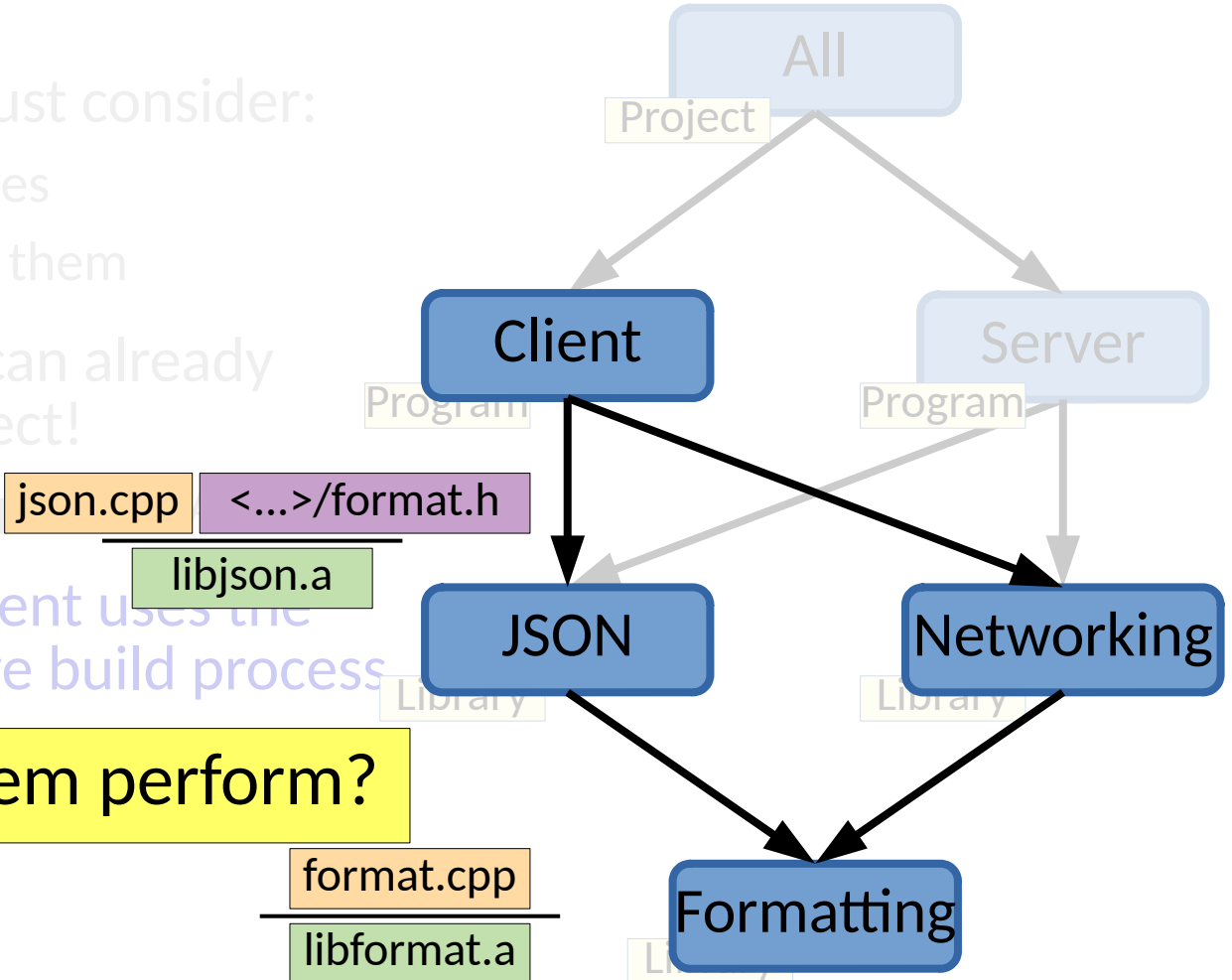
What must the build system perform?

used / required

generated

Modeling a Build

- To build software, we must consider:
  - Components & Objectives
  - Dependencies between them
- The dependency graph can already help to analyze our project!
  - A good dependency gra
- Modern build management uses the dependency DAG to drive build process



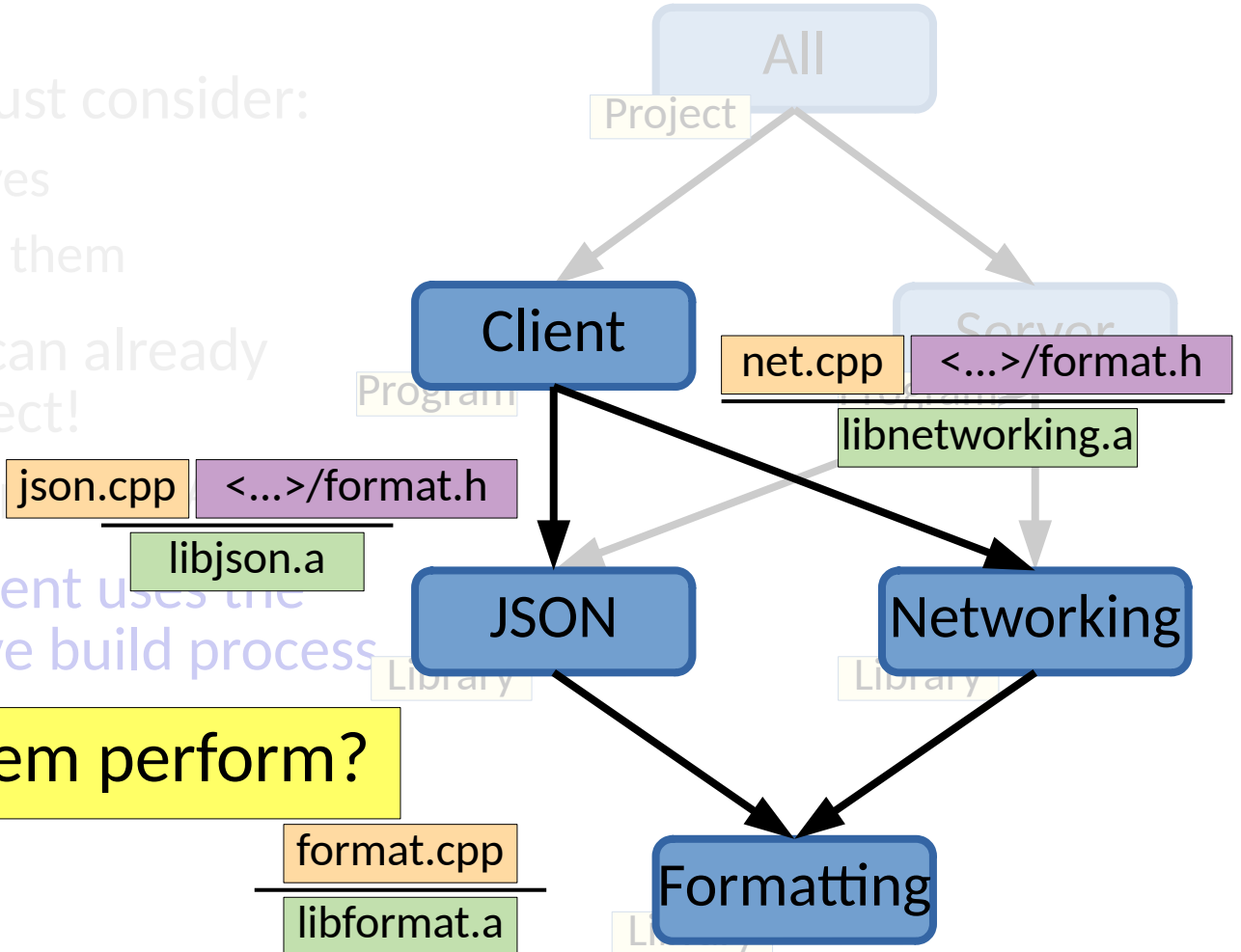
What must the build system perform?

used / required

generated

Modeling a Build

- To build software, we must consider:
  - Components & Objectives
  - Dependencies between them
- The dependency graph can already help to analyze our project!
  - A good dependency gra
- Modern build management uses the dependency DAG to drive build process



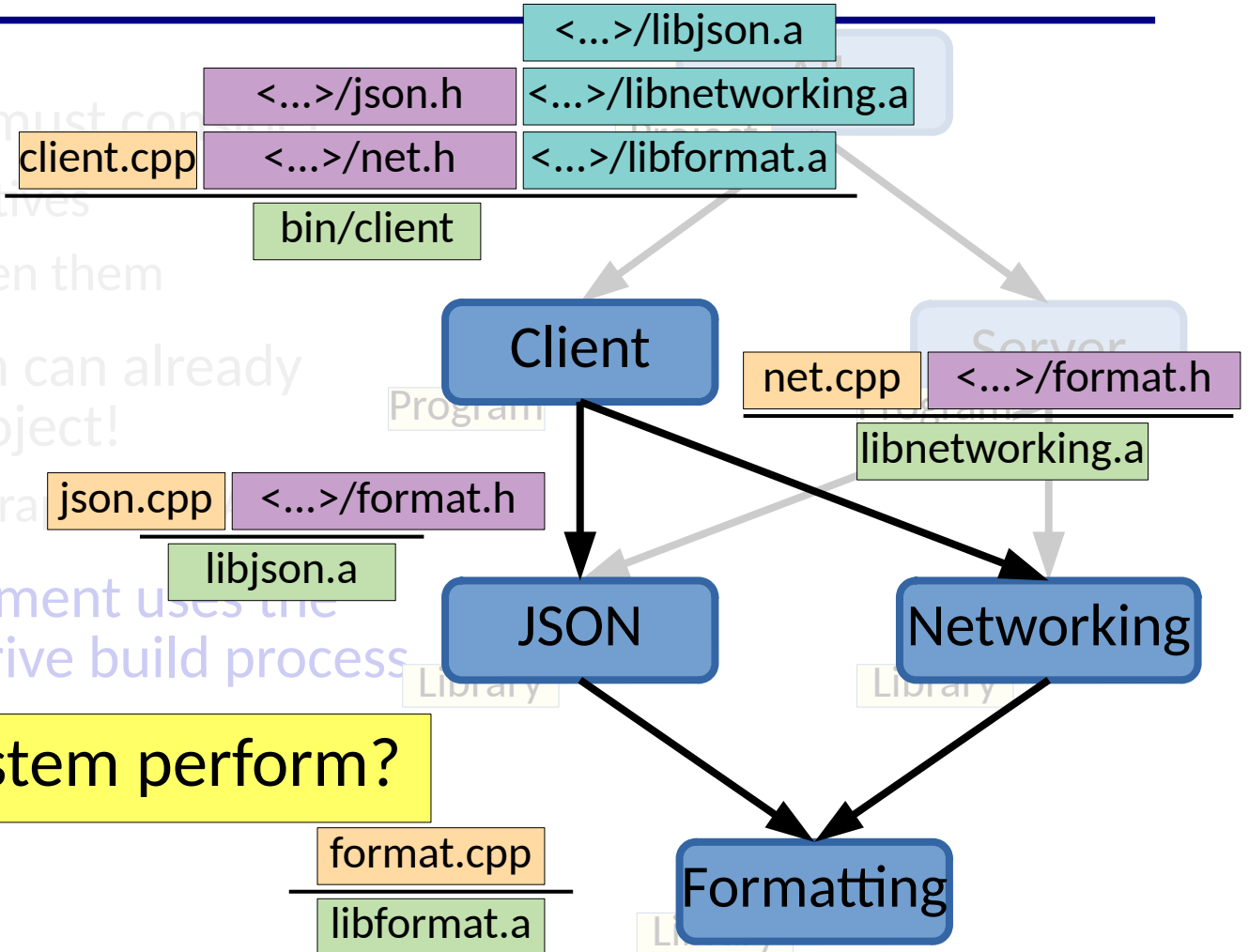
What must the build system perform?

used / required

generated

Modeling a Build

- To build software, we must compute:
  - Components & Objectives
  - Dependencies between them
- The dependency graph can already help to analyze our project!
  - A good dependency gra
- Modern build management uses the dependency DAG to drive build process



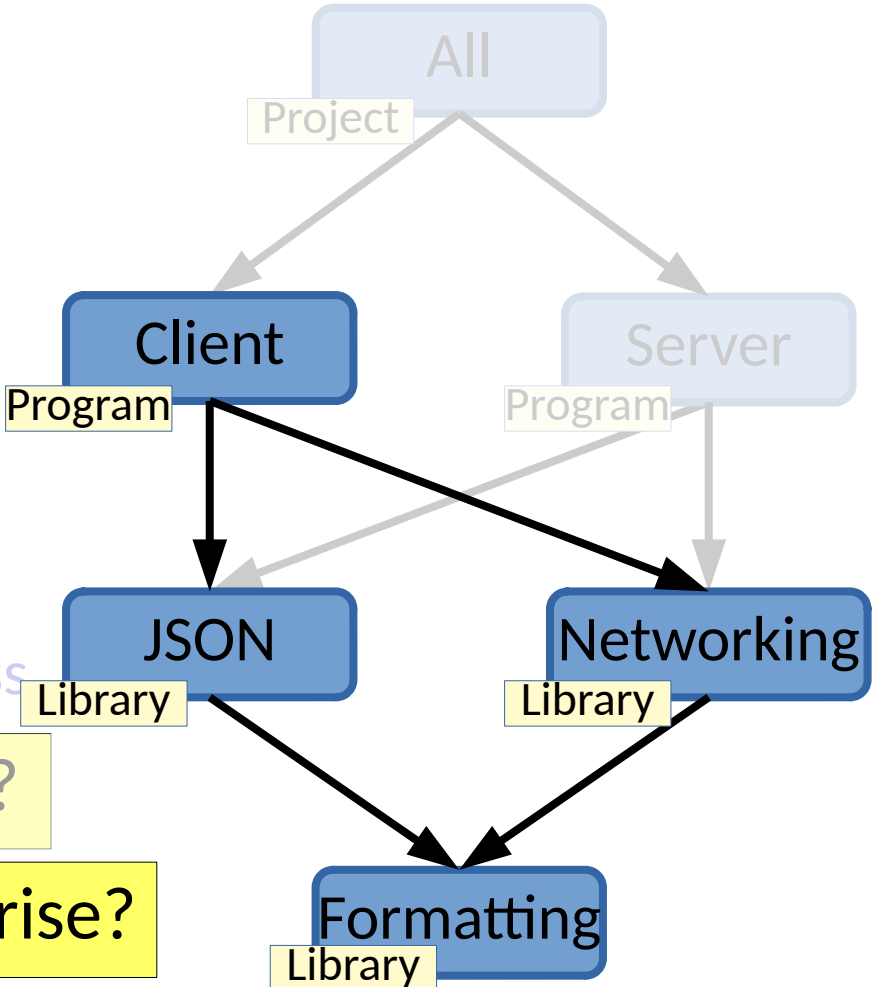
What must the build system perform?

used / required

generated

Modeling a Build

- To build software, we must consider:
  - Components & Objectives
  - Dependencies between them
- The dependency graph can already help to analyze our project!
  - A good dependency graph is a DAG
- Modern build management uses the dependency DAG to drive build process

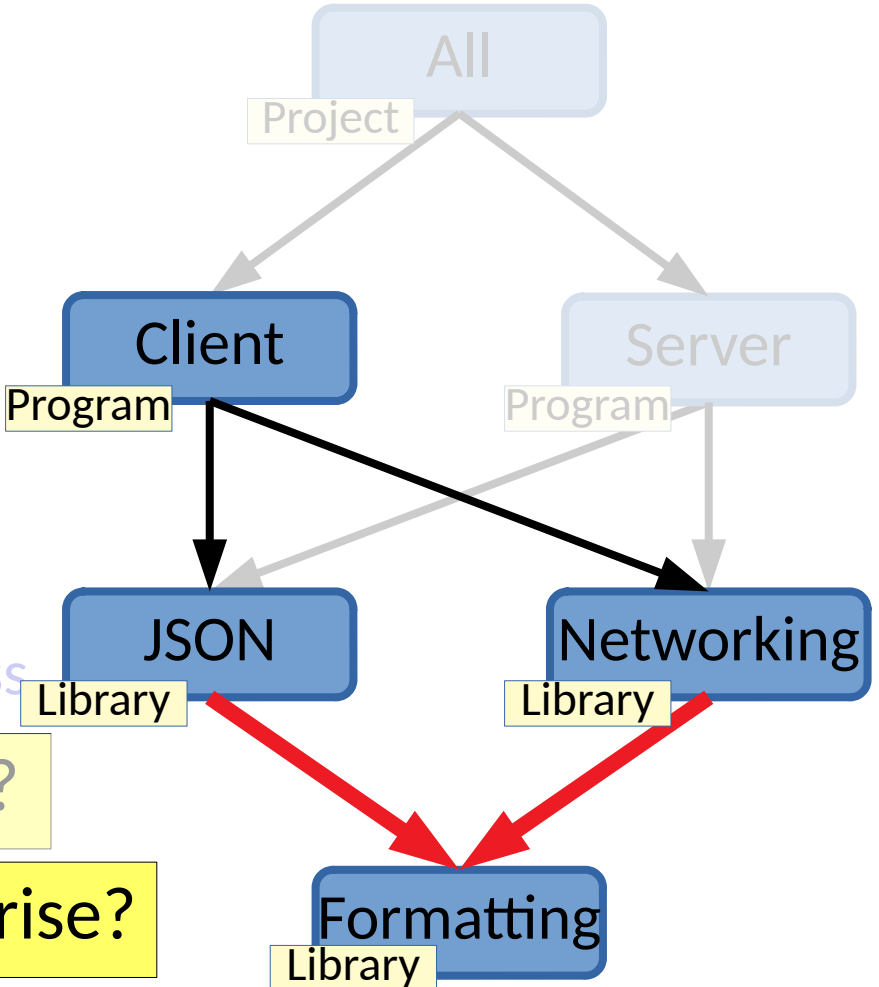


What must the build system perform?

Can you think of problems that may arise?

Modeling a Build

- To build software, we must consider:
  - Components & Objectives
  - Dependencies between them
- The dependency graph can already help to analyze our project!
  - A good dependency graph is a DAG
- Modern build management uses the dependency DAG to drive build process

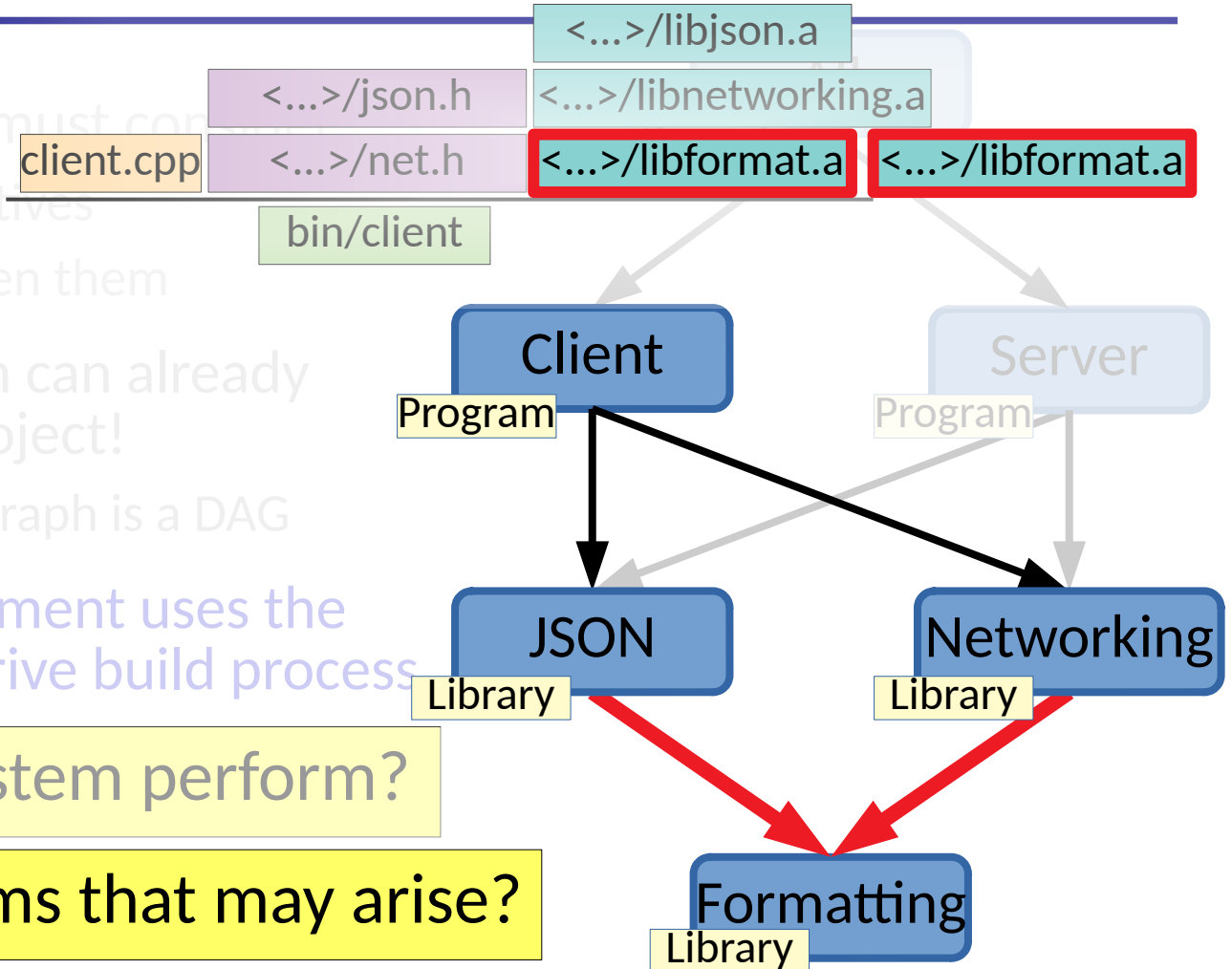


What must the build system perform?

Can you think of problems that may arise?

Modeling a Build

- To build software, we must consider
  - Components & Objectives
  - Dependencies between them
- The dependency graph can already help to analyze our project!
  - A good dependency graph is a DAG
- Modern build management uses the dependency DAG to drive build process



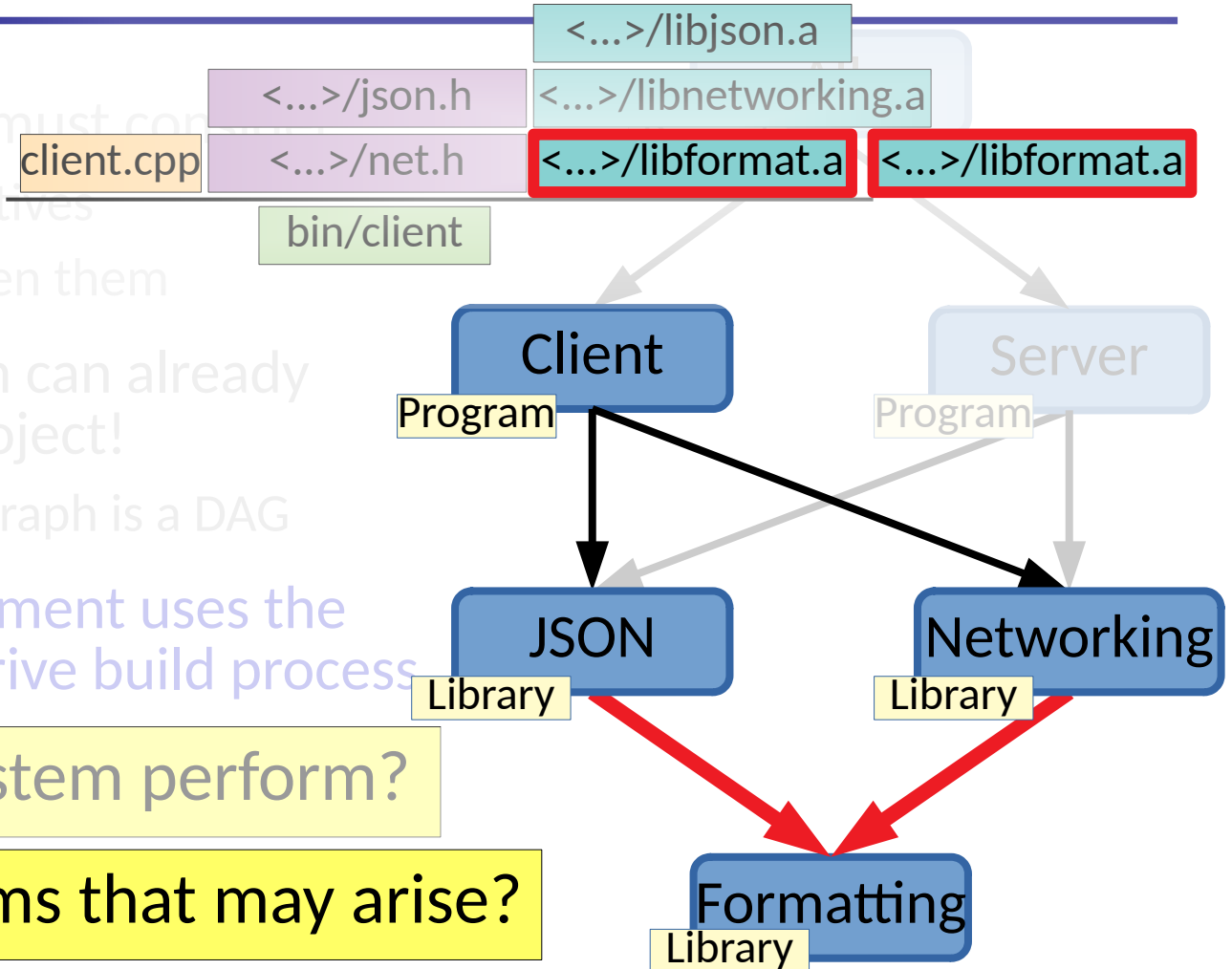
What must the build system perform?

Can you think of problems that may arise?

Modeling a Build

- To build software, we must consider:
  - Components & Objectives
  - Dependencies between them
- The dependency graph can already help us analyze our project!
  - A good dependency graph is a DAG
- Modern build management uses the dependency DAG to drive build process

Conflicts

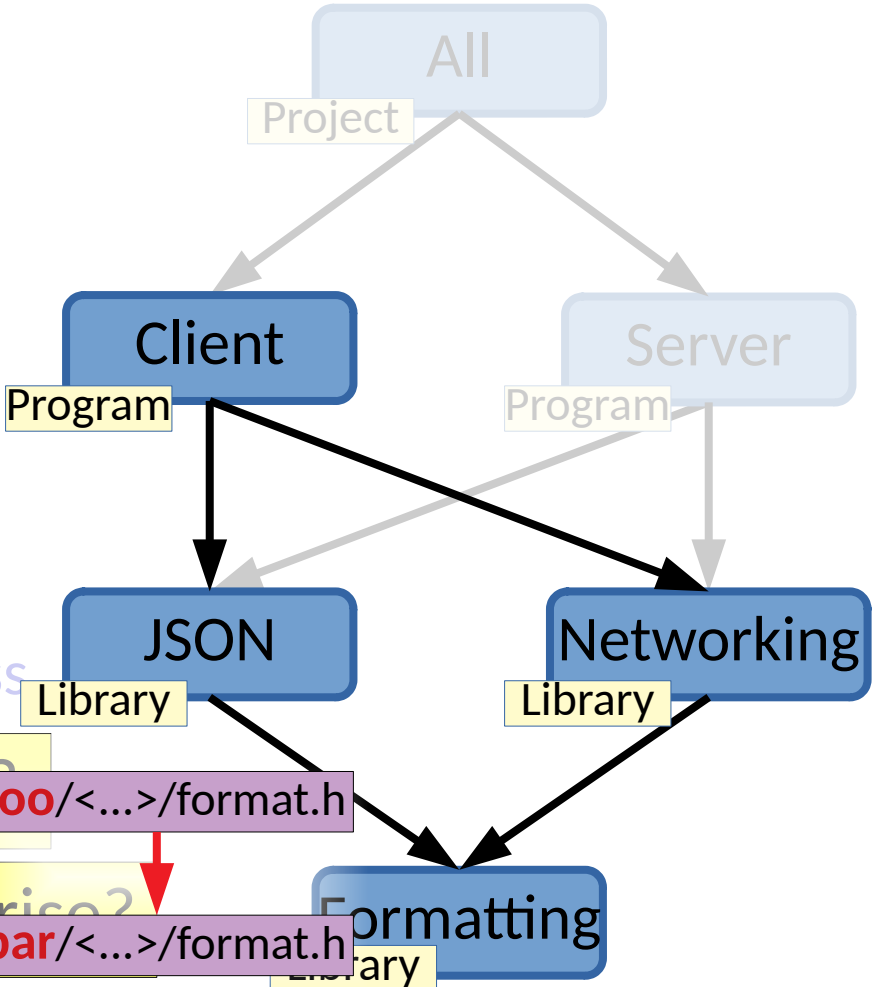


What must the build system perform?

Can you think of problems that may arise?

Modeling a Build

- To build software, we must consider:
  - Components & Objectives
  - Dependencies between them
- The dependency graph can already help to analyze our project!
  - A good dependency graph is a DAG
- Modern build management uses the dependency DAG to drive build process

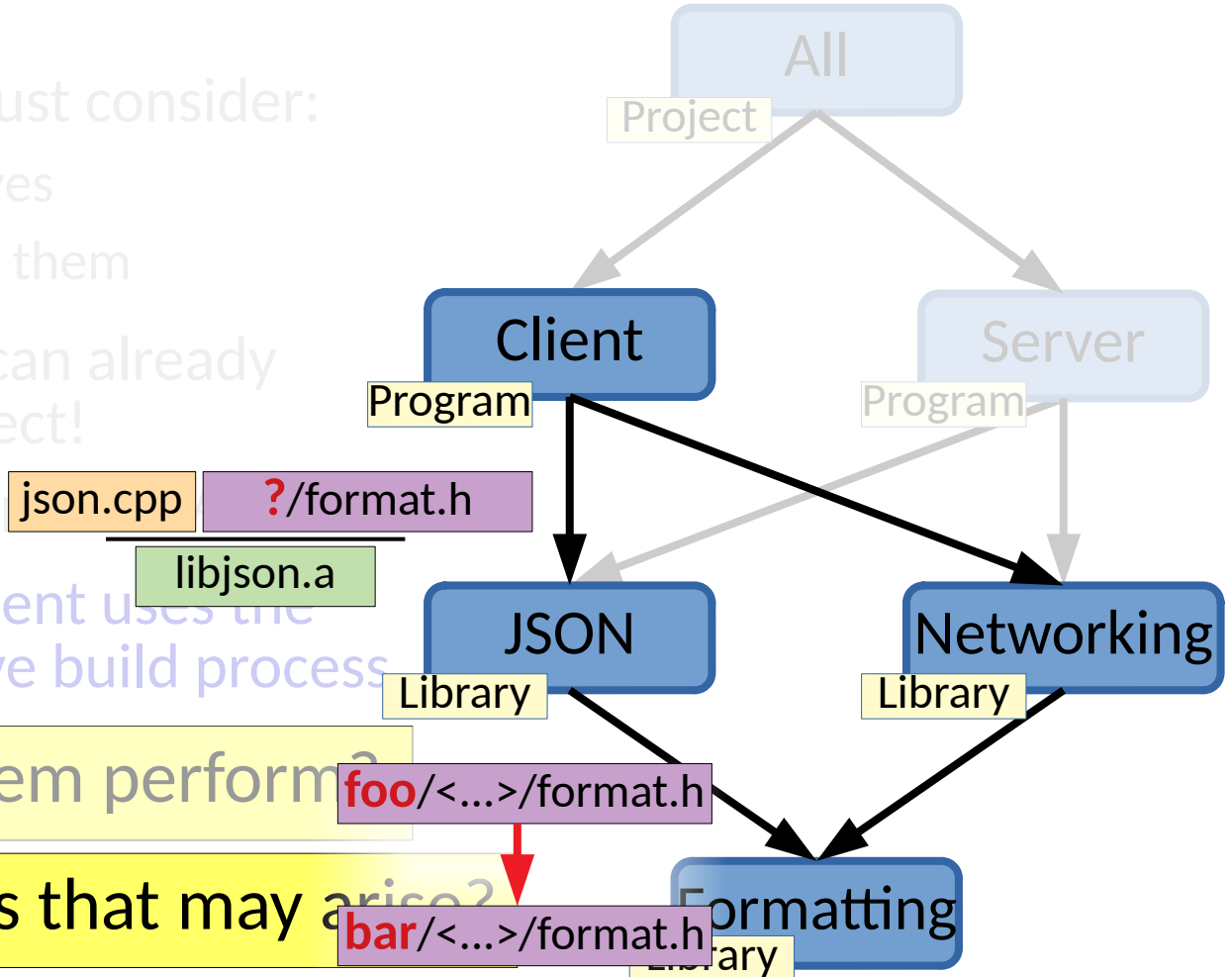


What must the build system perform?

Can you think of problems that may arise?

Modeling a Build

- To build software, we must consider:
  - Components & Objectives
  - Dependencies between them
- The dependency graph can already help to analyze our project!
  - A good dependency gra
- Modern build management uses the dependency DAG to drive build process



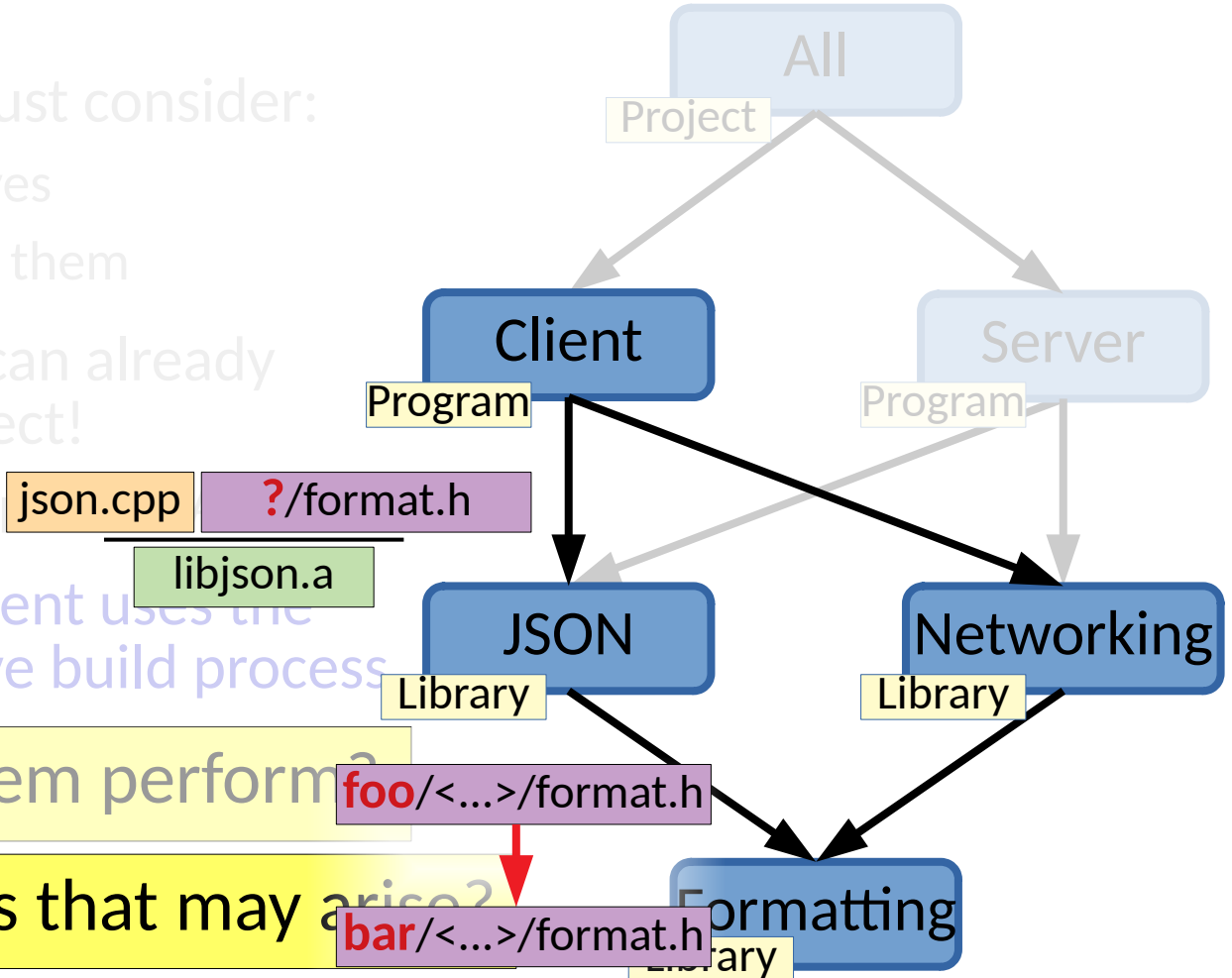
What must the build system perform?

Can you think of problems that may arise?

Modeling a Build

- To build software, we must consider:
  - Components & Objectives
  - Dependencies between them
- The dependency graph can already help decouple project!
  - A good dependency graph
- Modern build management uses the dependency DAG to drive build process

Decoupled Evolution

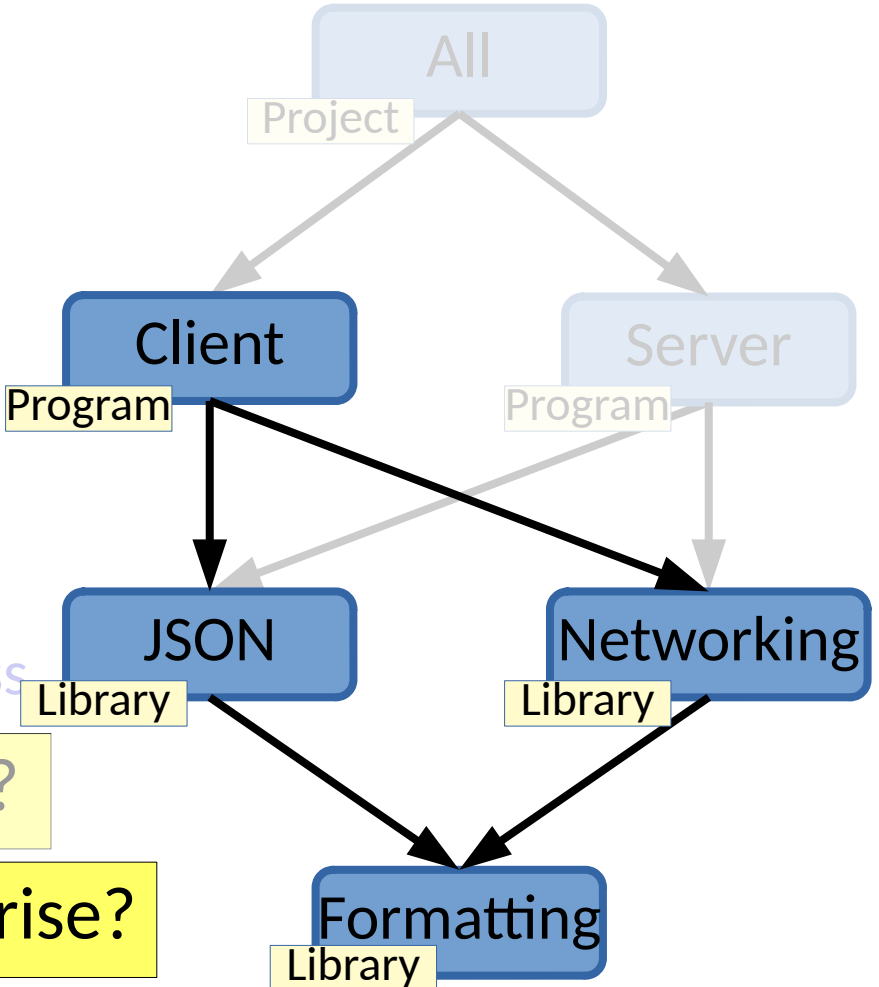


What must the build system perform?

Can you think of problems that may arise?

Modeling a Build

- To build software, we must consider:
  - Components & Objectives
  - Dependencies between them
- The dependency graph can already help to analyze our project!
  - A good dependency graph is a DAG
- Modern build management uses the dependency DAG to drive build process

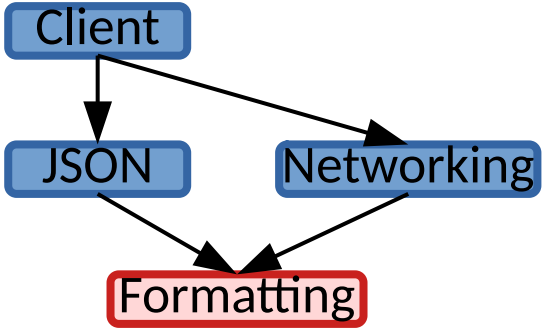


What must the build system perform?

Can you think of problems that may arise?

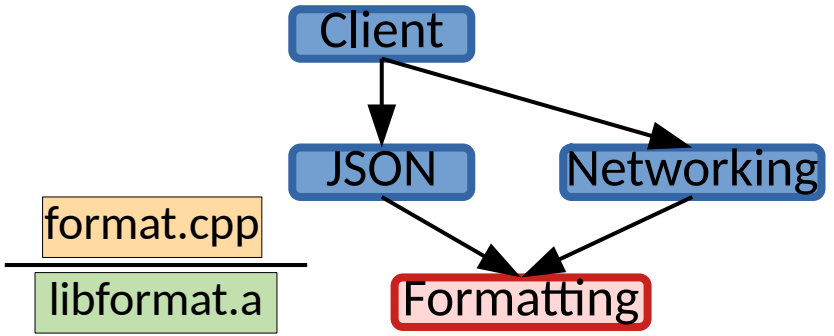
Modeling a Build

- For each component, build management requires
  - Direct build requirements



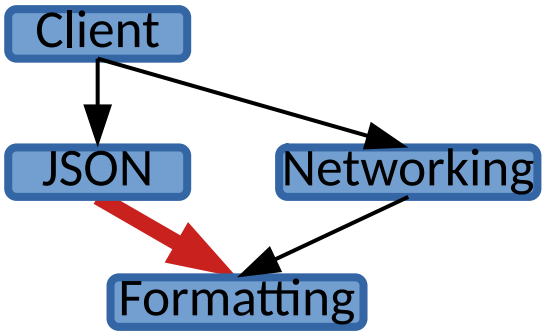
Modeling a Build

- For each component, build management requires
  - Direct build requirements



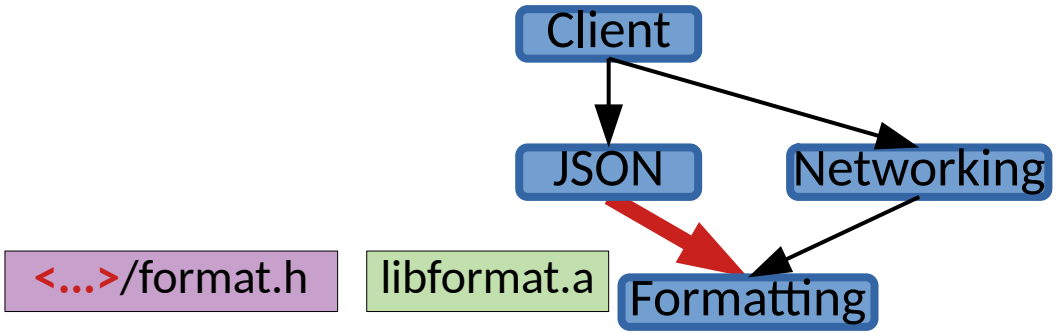
Modeling a Build

- For each component, build management requires
  - Direct build requirements
  - Transitive usage requirements



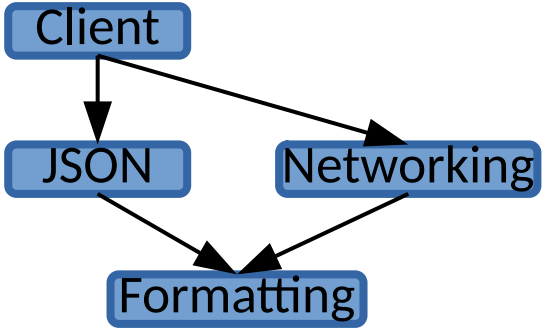
Modeling a Build

- For each component, build management requires
  - Direct build requirements
  - Transitive usage requirements



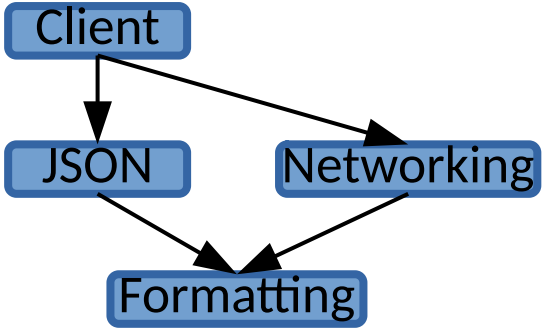
Modeling a Build

- For each component, build management requires
  - Direct build requirements
  - Transitive usage requirements
  - Note, these just separate *interfaces* from *implementation*



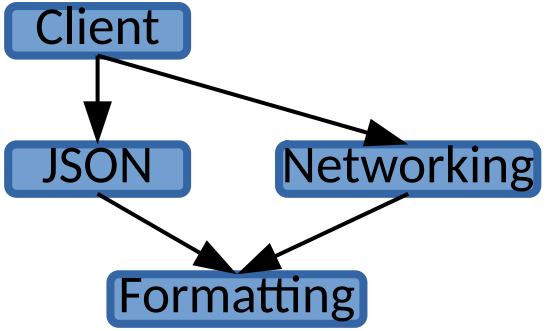
Modeling a Build

- For each component, build management requires
  - Direct build requirements
  - Transitive usage requirements
  - Note, these just separate *interfaces* from *implementation*
- The dependency graph allows us to use these to infer how to build a component correctly based on its dependencies



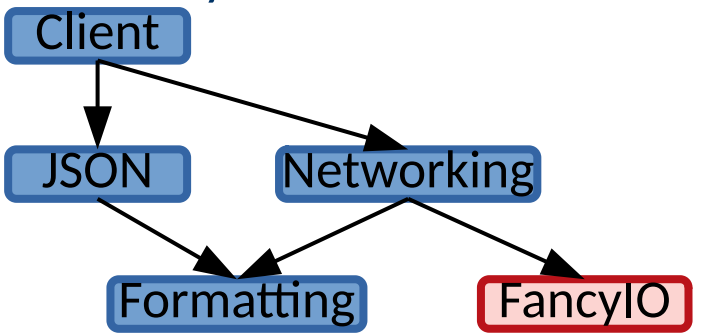
Modeling a Build

- For each component, build management requires
  - Direct build requirements
  - Transitive usage requirements
  - Note, these just separate *interfaces* from *implementation*
- The dependency graph allows us to use these to infer how to build a component correctly based on its dependencies
  - NOTE:
    Even if a dependency uses a new library, the build system should detect it.



Modeling a Build

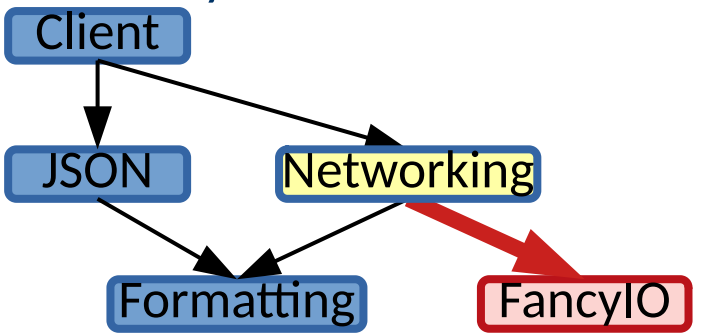
- For each component, build management requires
  - Direct build requirements
  - Transitive usage requirements
  - Note, these just separate *interfaces* from *implementation*
- The dependency graph allows us to use these to infer how to build a component correctly based on its dependencies
  - NOTE:
    Even if a dependency uses a new library, the build system should detect it.



Modeling a Build

- For each component, build management requires
  - Direct build requirements
  - Transitive usage requirements
  - Note, these just separate *interfaces* from *implementation*
- The dependency graph allows us to use these to infer how to build a component correctly based on its dependencies
  - NOTE:
    Even if a dependency uses a new library, the build system should detect it.

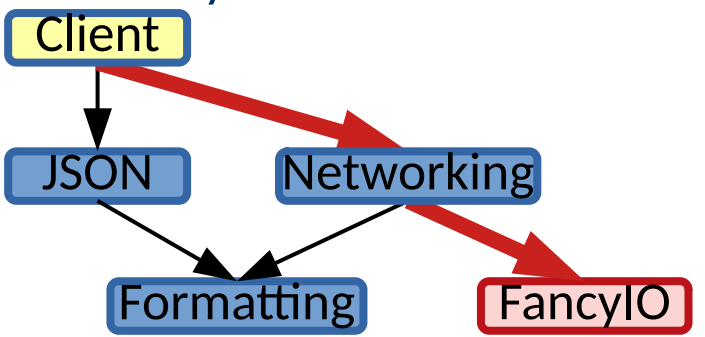
Include directories for Networking should change



Modeling a Build

- For each component, build management requires
  - Direct build requirements
  - Transitive usage requirements
  - Note, these just separate *interfaces* from *implementation*
- The dependency graph allows us to use these to infer how to build a component correctly based on its dependencies
  - NOTE:
    Even if a dependency uses a new library, the build system should detect it.

Include directories for Networking should change
Linked libraries for Client should change



Modeling a Build

- For each component, build management requires
  - Direct build requirements
  - Transitive usage requirements
  - Note, these just separate *interfaces* from *implementation*
- The dependency graph allows us to use these to infer how to build a component correctly based on its dependencies
  - NOTE:
    Even if a dependency uses a new library, the build system should detect it.
- Let’s dive into one specific system to see how this is done….

What will be be using?

- CMake
  - Cross-platform build management tool
  - Used by large projects like KDE, Wireshark, LLVM, …

What will be be using?

- CMake
  - Cross-platform build management tool
  - Used by large projects like KDE, Wireshark, LLVM, …
- What does it do?
  - Given a specification & configuration of your project, CMake creates the build commands for you
  - Analogous to autoconf (but easier to use)

What will be be using?

- CMake
  - Cross-platform build management tool
  - Used by large projects like KDE, Wireshark, LLVM, ...
- What does it do?
  - Given a specification & configuration of your project, CMake creates the build commands for you
  - Analogous to autoconf (but easier to use)

You describe the dependency graph.
It figures out how to build the software.

What will be be using?

- CMake
  - Cross-platform build management tool
  - Used by large projects like KDE, Wireshark, LLVM, …
- What does it do?
  - Given a specification & configuration of your project, CMake creates the build commands for you
  - Analogous to autoconf (but easier to use)

[DEMO]

What does this add?

- Why not just write makefiles manually?

What does this add?

- Why not just write makefiles manually?
  - May need different makefiles for different…

What does this add?

- Why not just write makefiles manually?
  - May need different makefiles for different
    - Operating Systems
    - Compilers
    - Libraries
    - Build Modes
    - …

What does this add?

- Why not just write makefiles manually?
  - May need different makefiles for different
    - Operating Systems
    - Compilers
    - Libraries
    - Build Modes
    - …
  - May need different source files for different “”

What does this add?

- Why not just write makefiles manually?
  - May need different makefiles for different
    - Operating Systems
    - Compilers
    - Libraries
    - Build Modes
    - …
  - May need different source files for different “”
  - Specification can clearly capture
    - Libraries, versions, & even how to download them automatically
    - Semantics of compilation & how to use in analysis tools

What does this add?

- Scalability
  - Replace “make” with analogous scalable tools (“ninja”)

What does this add?

- Scalability
  - Replace “make” with analogous scalable tools (“ninja”)
- Easier tool integration
  - CMake can export compilation rules for other tools

What does this add?

- Scalability
  - Replace “make” with analogous scalable tools (“ninja”)
- Easier tool integration
  - CMake can export compilation rules for other tools

[DEMO]

Preliminary: Out of source builds

- A common bad habit is “in source” building

Preliminary: Out of source builds

- A common bad habit is “in source” building
  - Why is this bad?

Preliminary: Out of source builds

- A common bad habit is “in source” building
  - Why is this bad?
  - May need multiple builds at once: debug, release, …
  - Pollutes version control
  - Makes clean builds complicated

Preliminary: Out of source builds

- A common bad habit is “in source” building
  - Why is this bad?
  - May need multiple builds at once: debug, release, …
  - Pollutes version control
  - Makes clean builds complicated
- Use “out of source” builds instead

Using CMake

- CMakeLists.txt
  - A script in every directory of your project that controls how to build “things” in that directory

Using CMake

- CMakeLists.txt
  - A script in every directory of your project that controls how to build “things” in that directory
- Simple syntax
  - Case insensitive commands

```
command( argument1 argument2 argument3 ...)
```

  - Let's revisit demo 1!

Targets & Commands

- CMake allows you to specify targets
  - Executables, libraries, “objects”

```
add_executable(helloworld)
add_library(hellohelper STATIC)
```

Targets & Commands

- CMake allows you to specify targets
  - Executables, libraries, “objects”

```
add_executable(helloworld)
add_library(hellohelper STATIC)
```

- And commands that can describe how to build those targets
  - Automatic for executable & library
  - `add_custom_command` can build others
    - Documentation
    - Media

Specifying Requirements

- Recall build requirements & usage requirements.

Specifying Requirements

- Recall build requirements & usage requirements.
- `target_*` commands allow you to specify the requirements of a target

Specifying Requirements

- Recall build requirements & usage requirements.
- `target_*` commands allow you to specify the requirements of a target

```
target_sources(hellohelper
   PRIVATE helloworld.cpp
)
```

Specifying Requirements

- Recall build requirements & usage requirements.
- **`target_*`** commands allow you to specify the requirements of a target

```
target_sources(hellohelper
  PRIVATE helloworld.cpp
)
target_include_directories(hellohelper
  INTERFACE ${CMAKE_CURRENT_SOURCE_DIR}/include/
)
```

Specifying Requirements

- Recall build requirements & usage requirements.
- **`target_*`** commands allow you to specify the requirements of a target

```
target_sources(hellohelper
  PRIVATE helloworld.cpp
)
target_include_directories(hellohelper
  INTERFACE ${CMAKE_CURRENT_SOURCE_DIR}/include/
)
target_link_libraries(helloworld
  PRIVATE hellohelper
)
```

Specifying Requirements

- Recall build requirements & usage requirements.
- **`target_*`** commands allow you to specify the requirements of a target

```
target_sources(hellohelper
  PRIVATE helloworld.cpp
)
target_include_directories(hellohelper
  INTERFACE ${CMAKE_CURRENT_SOURCE_DIR}/include/
)
target_link_libraries(helloworld
  PRIVATE hellohelper
)
```

Using Libraries

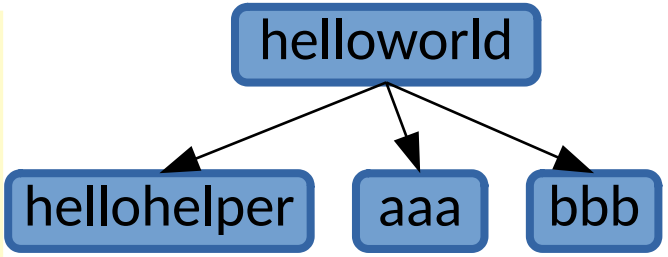
- You can simply specify the libraries that a target directly uses

```
target_link_libraries(helloworld
  PRIVATE hellohelper aaa bbb
)
target_link_libraries(hellohelper
  INTERFACE fancyformatting ccc
)
```

Using Libraries

- You can simply specify the libraries that a target directly uses

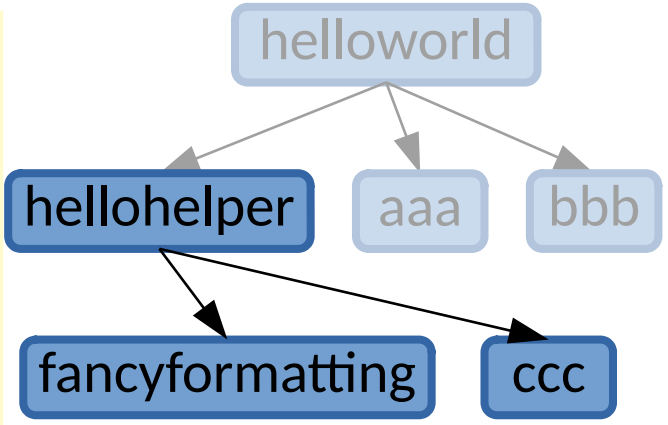
```
target_link_libraries(helloworld
   PRIVATE hellohelper aaa bbb
)
target_link_libraries(hellohelper
   INTERFACE fancyformatting ccc
)
```



Using Libraries

- You can simply specify the libraries that a target directly uses

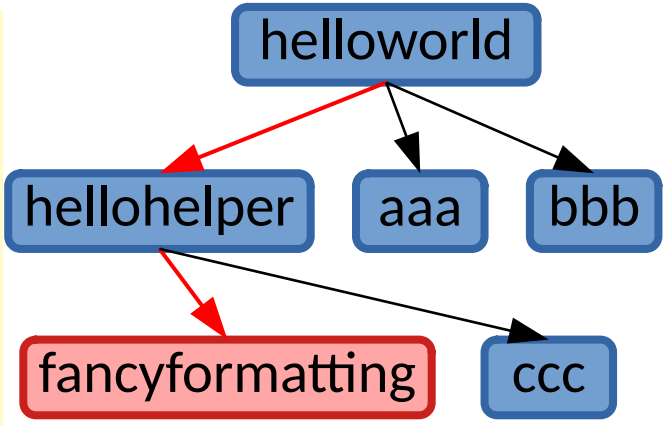
```
target_link_libraries(helloworld
   PRIVATE hellohelper aaa bbb
)
target_link_libraries(hellohelper
   INTERFACE fancyformatting ccc
)
```



Using Libraries

- You can simply specify the libraries that a target directly uses

```
target_link_libraries(helloworld
   PRIVATE hellohelper aaa bbb
)
target_link_libraries(hellohelper
   INTERFACE fancyformatting ccc
)
```

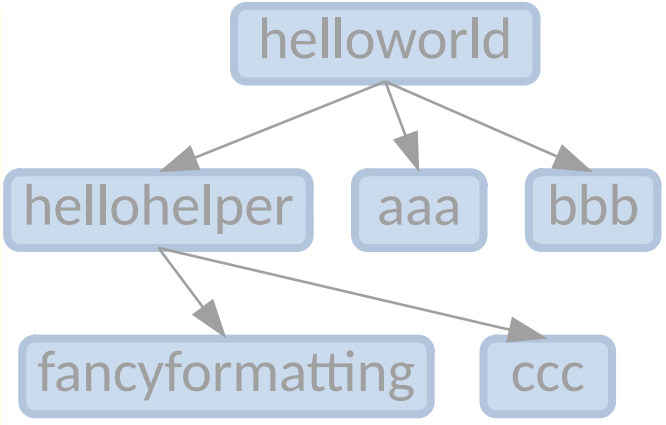


- Transitive interface dependencies of libraries will be linked in as required

Using Libraries

- You can simply specify the libraries that a target directly uses

```
target_link_libraries(helloworld
  PRIVATE hellohelper aaa bbb
)
target_link_libraries(hellohelper
  INTERFACE fancyformatting ccc
)
```

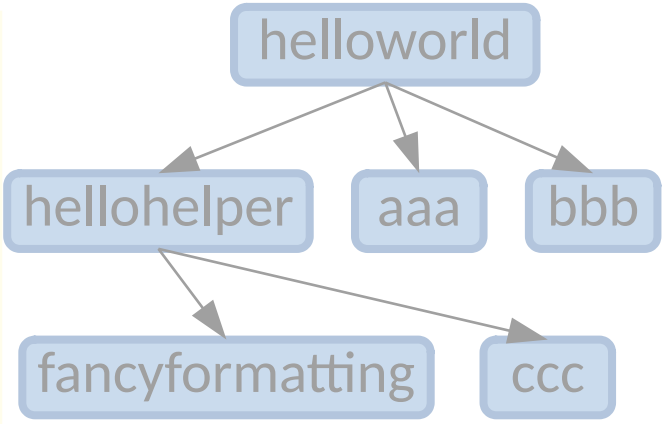


- Transitive interface dependencies of libraries will be linked in as required
- Include directories, etc. from libraries will also be *inferred*

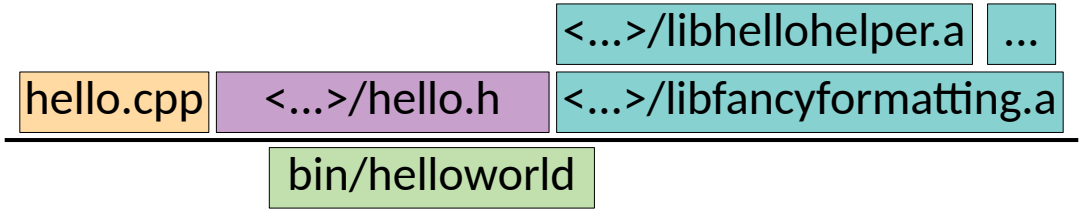
Using Libraries

- You can simply specify the libraries that a target directly uses

```
target_link_libraries(helloworld
  PRIVATE hellohelper aaa bbb
)
target_link_libraries(hellohelper
  INTERFACE fancyformatting ccc
)
```



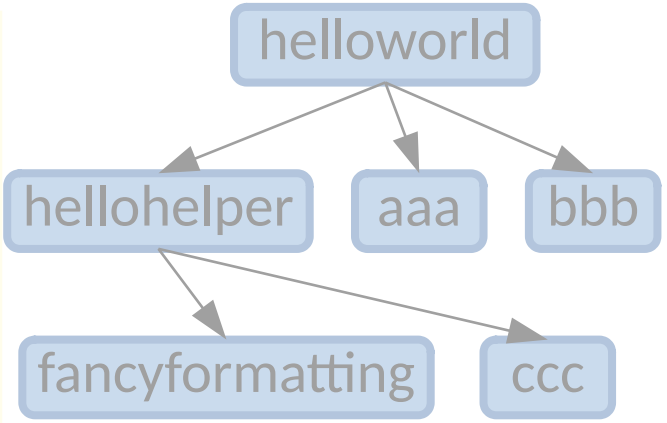
- Transitive interface dependencies of libraries will be linked in as required
- Include directories, etc. from libraries will also be *inferred*



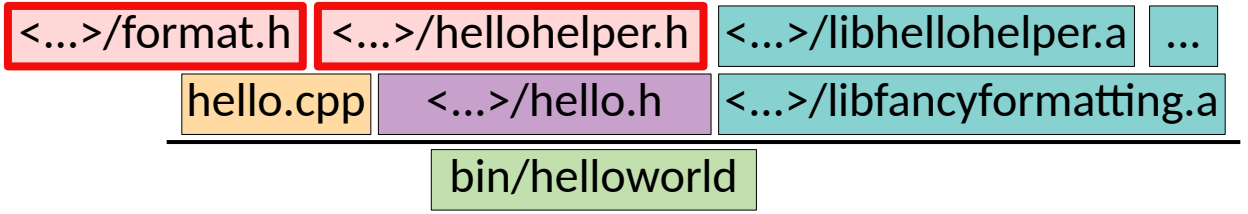
Using Libraries

- You can simply specify the libraries that a target directly uses

```
target_link_libraries(helloworld
  PRIVATE hellohelper aaa bbb
)
target_link_libraries(hellohelper
  INTERFACE fancyformatting ccc
)
```



- Transitive interface dependencies of libraries will be linked in as required
- Include directories, etc. from libraries will also be *inferred*



Using Libraries

- Note: This means using creating & defining new libraries is easy!

Using Libraries

- Note: This means using creating & defining new libraries is easy!
- How might this affect program structure and design?

Using Libraries

- Note: This means using creating & defining new libraries is easy!
- How might this affect program structure and design?

Consider how this relates to SOA and microservices as well!

Using Libraries

- Note: This means using creating & defining new libraries is easy!
- How might this affect program structure and design?
- How might it help us begin to handle complexity?

Using Libraries

- Note: This means using creating & defining new libraries is easy!
- How might this affect program structure and design?
- How might it help us begin to handle complexity?

CMake has several other mundane build system facilities…

General project management

- Specifying project properties
  - Define a project to access variables that control that project

```
project(projectname)
```

General project management

- Specifying project properties
  - Define a project to access variables that control that project

    `project(projectname)`
- Print information out during the build process

  ```
  message("Built with flags: ${CMAKE_CXX_FLAGS}")
  ```

General project management

- Specifying project properties
  - Define a project to access variables that control that project

    `project(projectname)`
- Print information out during the build process

  `message("Built with flags: ${CMAKE_CXX_FLAGS}")`
- Controlling where things are built

```
set(CMAKE_RUNTIME_OUTPUT_DIRECTORY
    "${PROJECT_BINARY_DIR}/bin")
set(CMAKE_LIBRARY_OUTPUT_DIRECTORY
    "${PROJECT_BINARY_DIR}/lib")
```

General project management

- Finding a resource that you need to use

```
find_package(externalproject)
find_library(library)
```

General project management

- Finding a resource that you need to use

```
find_package(externalproject)
find_library(library)
```

- Installation

```
install(TARGETS target1 target2 ...
  DESTINATION /tmp/
)
```

Control structures

- IF

```
if(condition)
elsif(condition2)
else()
endif()
```

Control structures

- IF

```
if(condition)
elsif(condition2)
else()
endif()
```

Why might you want conditionals in a build process?

Control structures

- IF

```
if(condition)
elsif(condition2)
else()
endif()
```

- Looping

```
foreach(loop_var arg1 arg2 ...)
  command(${loop_var})
endforeach(loop_var)
while(condition)...
```

Control structures

- IF

```
if(condition)
elsif(condition2)
else()
endif()
```

- Looping

```
foreach(loop_var arg1 arg2 ...)
  command(${loop_var})
endforeach(loop_var)
while(condition)...
```

- Functions

```
function(function_name arg1 arg2 ...)
  command(${arg1})
endFunction(function_name)
```

Pulling Remote Dependencies

- Managing dependencies (e.g. with CPM)

```
include(cmake/CPM.cmake)
```

Pulling Remote Dependencies

- Managing dependencies (e.g. with CPM)

```
include(cmake/CPM.cmake)

CPMAddPackage(
  NAME something
  GIT_REPOSITORY https://github.com/nsumner/something.git
  GIT_TAG v0.0.1
)
```

Pulling Remote Dependencies

- Managing dependencies (e.g. with CPM)

```
include(cmake/CPM.cmake)

CPMAddPackage(
  NAME something
  GIT_REPOSITORY https://github.com/nsumner/something.git
  GIT_TAG v0.0.1
)

target_link_libraries(demo
  PRIVATE something
)
```

Pulling Remote Dependencies

- Managing dependencies (e.g. with CPM)

```
include(cmake/CPM.cmake)

CPMAddPackage(
  NAME something
  GIT_REPOSITORY https://github.com/nsumner/something.git
  GIT_TAG v0.0.1
)

target_link_libraries(demo
  PRIVATE something
)
```

What are the tradeoffs of this?

Analyzing Project Structure

- CMake can dump out the dependence graph in graphviz format

```
cmake –graphviz=deps.gv <path to project>
dot -Tpng deps.gv -o deps.svg
```

Analyzing Project Structure

- CMake can dump out the dependence graph in graphviz format

```
cmake –graphviz=deps.gv <path to project>
dot -Tpng deps.gv -o deps.svg
```

CMake has extensive documentation,
and you can find additional CMake specific information online

More Advanced Build Issues

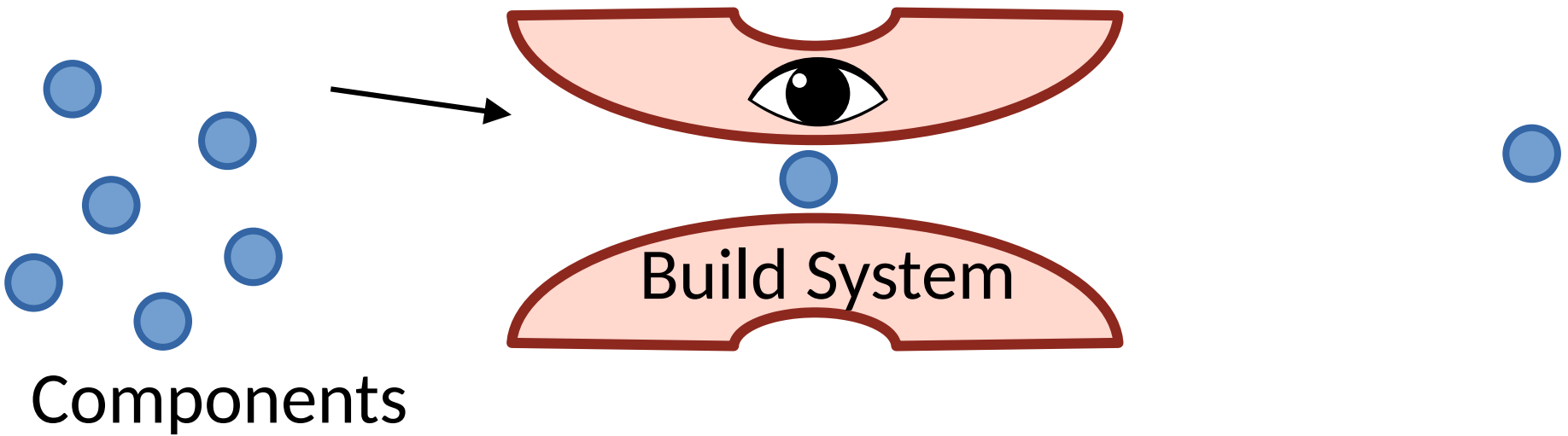
- Build systems are a foundation of workflow and DevOps

More Advanced Build Issues

- Build systems are a foundation of workflow and DevOps
  - They provide a “choke point” for controlling development

More Advanced Build Issues

- Build systems are a foundation of workflow and DevOps
  - They provide a “choke point” for controlling development



More Advanced Build Issues

- Build systems are a foundation of workflow and DevOps
  - They provide a “choke point” for controlling development
  - They can be a expensive & bottleneck when slow

More Advanced Build Issues

- Build systems are a foundation of workflow and DevOps
  - They provide a “choke point” for controlling development
  - They can be a expensive & bottleneck when slow
- Adding control

More Advanced Build Issues

- Build systems are a foundation of workflow and DevOps
  - They provide a “choke point” for controlling development
  - They can be a expensive & bottleneck when slow
- Adding control
  - Automated testing
  - Code analysis & metrics
  - Polyglot management
  - Deployment & rollout

More Advanced Build Issues

- Build systems are a foundation of workflow and DevOps
  - They provide a “choke point” for controlling development
  - They can be a expensive & bottleneck when slow
- Adding control
  - Automated testing
  - Code analysis & metrics
  - Polyglot management
  - Deployment & rollout

More Advanced Build Issues

- Build systems are a foundation of workflow and DevOps
  - They provide a “choke point” for controlling development
  - They can be a expensive & bottleneck when slow
- Adding control
  - Automated testing
  - Code analysis & metrics
  - Polyglot management
  - Deployment & rollout

More Advanced Build Issues

- Build systems are a foundation of workflow and DevOps
  - They provide a “choke point” for controlling development
  - They can be a expensive & bottleneck when slow
- Adding control
  - Automated testing
  - Code analysis & metrics
  - Polyglot management
  - Deployment & rollout

More Advanced Build Issues

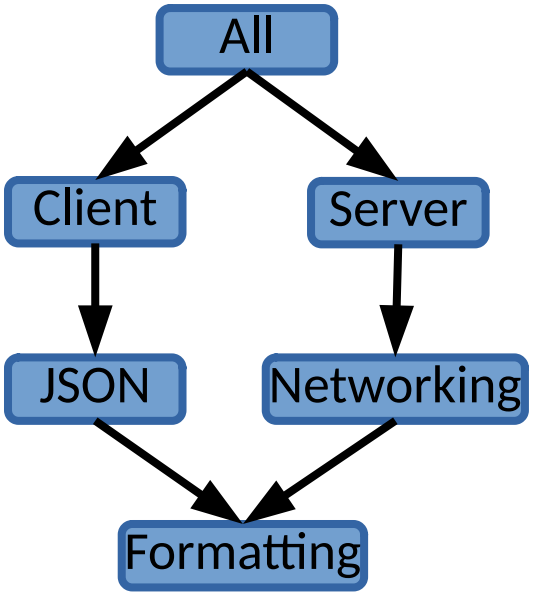
- Build systems are a foundation of workflow and DevOps
  - They provide a “choke point” for controlling development
  - They can be a expensive & bottleneck when slow
- Adding control
  - Automated testing
  - Code analysis & metrics
  - Polyglot management
  - Deployment & rollout
- Improving performance

More Advanced Build Issues

- Build systems are a foundation of workflow and DevOps
  - They provide a “choke point” for controlling development
  - They can be a expensive & bottleneck when slow
- Adding control
  - Automated testing
  - Code analysis & metrics
  - Polyglot management
  - Deployment & rollout
- Improving performance
  - Parallel & distributed

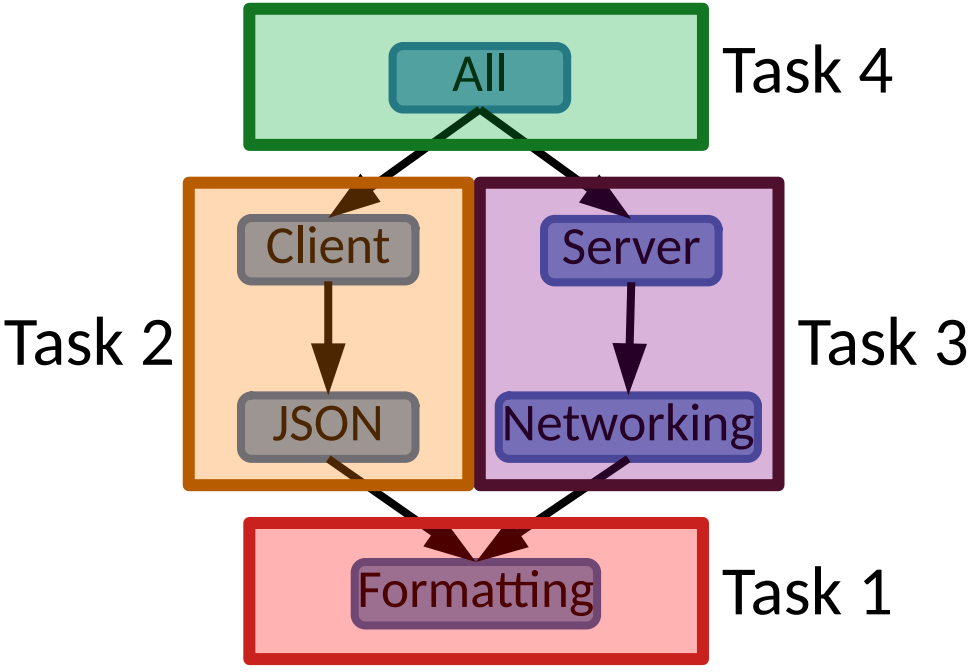
More Advanced Build Issues

- Build systems are a foundation of workflow and DevOps
  - They provide a “choke point” for controlling development
  - They can be a expensive & bottleneck when slow
- Adding control
  - Automated testing
  - Code analysis & metrics
  - Polyglot management
  - Deployment & rollout
- Improving performance
  - Parallel & distributed



More Advanced Build Issues

- Build systems are a foundation of workflow and DevOps
  - They provide a “choke point” for controlling development
  - They can be a expensive & bottleneck when slow
- Adding control
  - Automated testing
  - Code analysis & metrics
  - Polyglot management
  - Deployment & rollout
- Improving performance
  - Parallel & distributed

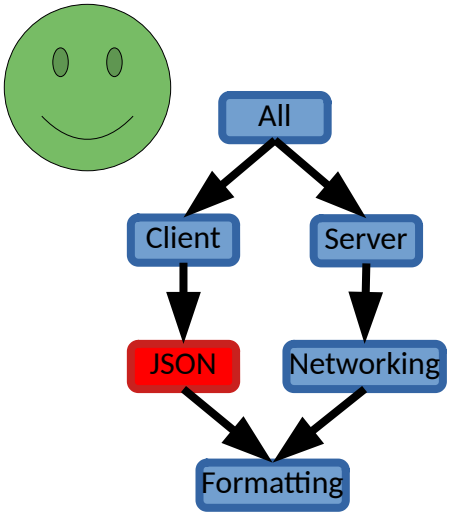


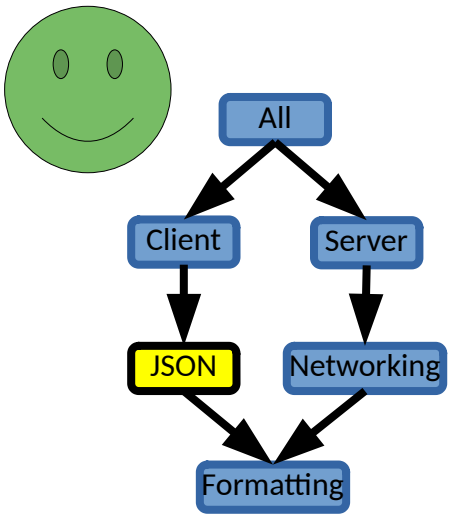
More Advanced Build Issues

- Build systems are a foundation of workflow and DevOps
  - They provide a “choke point” for controlling development
  - They can be a expensive & bottleneck when slow
- Adding control
  - Automated testing
  - Code analysis & metrics
  - Polyglot management
  - Deployment & rollout
- Improving performance
  - Parallel & distributed
  - Caching

More Advanced Build Issues

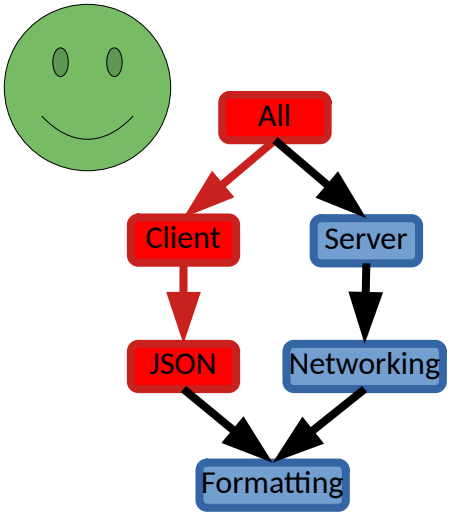
- Build systems are a foundation of workflow and DevOps
  - They provide a “choke point” for controlling development
  - They can be a expensive & bottleneck when slow
- Adding control
  - Automated testing
  - Code analysis & metrics
  - Polyglot management
  - Deployment & rollout
- Improving performance
  - Parallel & distributed
  - Caching

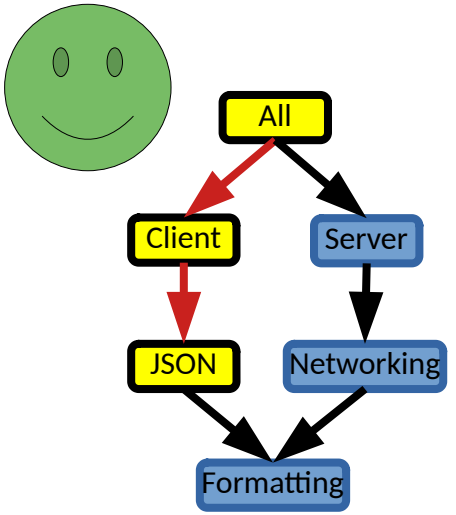




More Advanced Build Issues

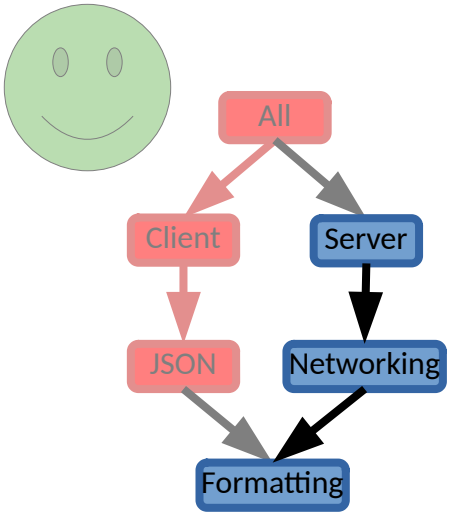
- Build systems are a foundation of workflow and DevOps
  - They provide a “choke point” for controlling development
  - They can be a expensive & bottleneck when slow
- Adding control
  - Automated testing
  - Code analysis & metrics
  - Polyglot management
  - Deployment & rollout
- Improving performance
  - Parallel & distributed
  - Caching

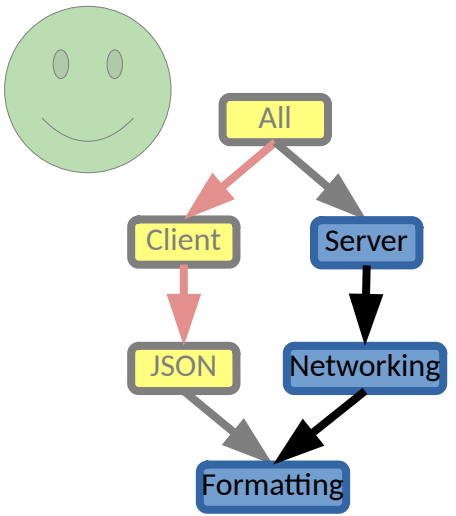




More Advanced Build Issues

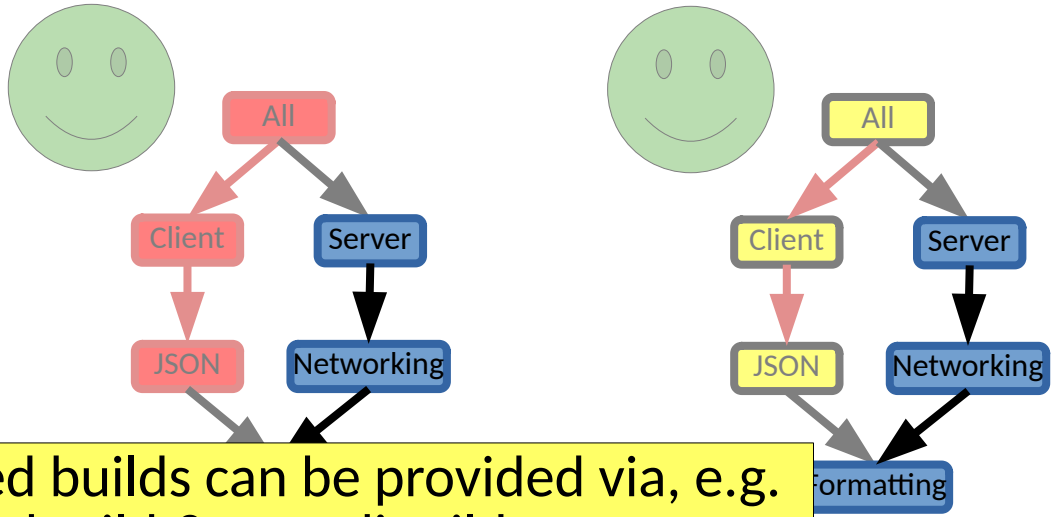
- Build systems are a foundation of workflow and DevOps
  - They provide a “choke point” for controlling development
  - They can be a expensive & bottleneck when slow
- Adding control
  - Automated testing
  - Code analysis & metrics
  - Polyglot management
  - Deployment & rollout
- Improving performance
  - Parallel & distributed
  - Caching





More Advanced Build Issues

- Build systems are a foundation of workflow and DevOps
  - They provide a “choke point” for controlling development
  - They can be a expensive & bottleneck when slow
- Adding control
  - Automated testing
  - Code analysis & metrics
  - Polyglot management
  - Deployment & rollout
- Improving performance
  - Paral
  - Cachi



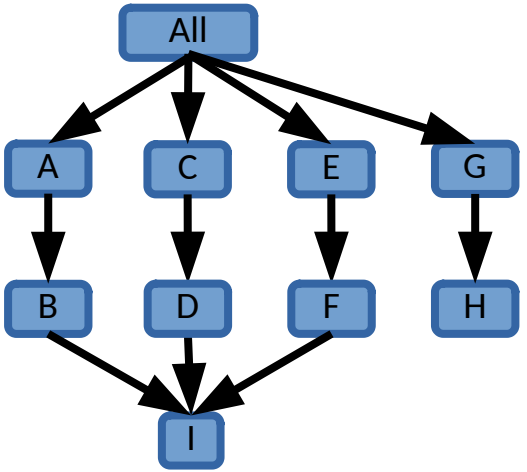
Distributed & cached builds can be provided via, e.g.
MS CloudBuild & IncrediBuild
Larger companies like Google have their own.

More Advanced Build Issues

- Build systems are a foundation of workflow and DevOps
  - They provide a “choke point” for controlling development
  - They can be a expensive & bottleneck when slow
- Adding control
  - Automated testing
  - Code analysis & metrics
  - Polyglot management
  - Deployment & rollout
- Improving performance
  - Parallel & distributed
  - Caching
  - Unification

More Advanced Build Issues

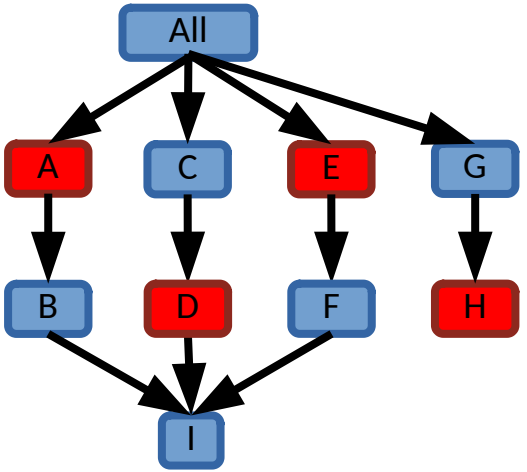
- Build systems are a foundation of workflow and DevOps
  - They provide a “choke point” for controlling development
  - They can be a expensive & bottleneck when slow
- Adding control
  - Automated testing
  - Code analysis & metrics
  - Polyglot management
  - Deployment & rollout
- Improving performance
  - Parallel & distributed
  - Caching
  - Unification



More Advanced Build Issues

- Build systems are a foundation of workflow and DevOps
  - They provide a “choke point” for controlling development
  - They can be a expensive & bottleneck when slow
- Adding control
  - Automated testing
  - Code analysis & metrics
  - Polyglot management
  - Deployment & rollout
- Improving performance
  - Parallel & distributed
  - Caching
  - Unification

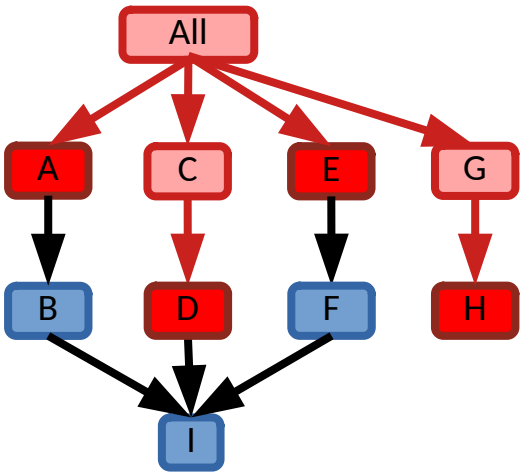
Suppose different component change frequently.
(high velocity, API churn, …)



More Advanced Build Issues

- Build systems are a foundation of workflow and DevOps
  - They provide a “choke point” for controlling development
  - They can be a expensive & bottleneck when slow
- Adding control
  - Automated testing
  - Code analysis & metrics
  - Polyglot management
  - Deployment & rollout
- Improving performance
  - Parallel & distributed
  - Caching
  - Unification

Suppose different component change frequently.
(high velocity, API churn, …)

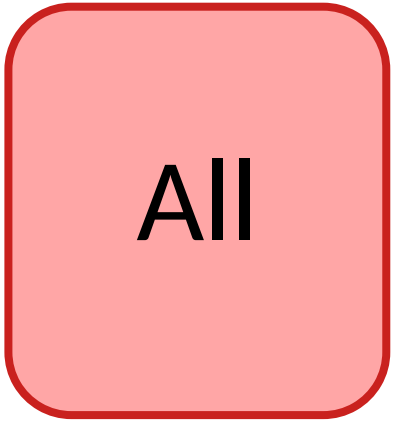


↓value of incremental
↑cost of parsing + dependencies

More Advanced Build Issues

- Build systems are a foundation of workflow and DevOps
  - They provide a “choke point” for controlling development
  - They can be a expensive & bottleneck when slow
- Adding control
  - Automated testing
  - Code analysis & metrics
  - Polyglot management
  - Deployment & rollout
- Improving performance
  - Parallel & distributed
  - Caching
  - Unification

Suppose different component change frequently.
(high velocity, API churn, …)



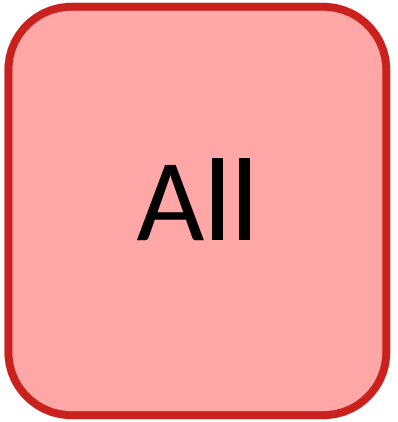
↓value of incremental
↑cost of parsing + dependencies

Just inline everything
into one file!

More Advanced Build Issues

- Build systems are a foundation of workflow and DevOps
  - They provide a “choke point” for controlling development
  - They can be a expensive & bottleneck when slow
- Adding control
  - Automated testing
  - Code analysis & metrics
  - Polyglot management
  - Deployment & rollout
- Improving performance
  - Parallel & distributed
  - Caching
  - Unification

Suppose different component change frequently.
(high velocity, API churn, …)



↓value of incremental
↑cost of parsing + dependencies

Just inline everything into one file!

“Unity builds” can be popular in game dev.

In Summary

- A modern build system leverages the dependency graph of a project

In Summary

- A modern build system leverages the dependency graph of a project
- Dependency graphs enable

In Summary

- A modern build system leverages the dependency graph of a project
- Dependency graphs enable
    1) *inference* of build and usage requirements

In Summary

- A modern build system leverages the dependency graph of a project
- Dependency graphs enable
  1) *inference* of build and usage requirements
  2) *compositional* reasoning about modules and build management

In Summary

- A modern build system leverages the dependency graph of a project
- Dependency graphs enable
  1) inference of build and usage requirements
  2) compositional reasoning about modules and build management
- One dominant system for C and C++ is Cmake

In Summary

- A modern build system leverages the dependency graph of a project
- Dependency graphs enable
  1) inference of build and usage requirements
  2) compositional reasoning about modules and build management
- One dominant system for C and C++ is Cmake
- You will get more personal experience with it over the semester if you have not already