Build Systems

CSE 403, Spring 2018

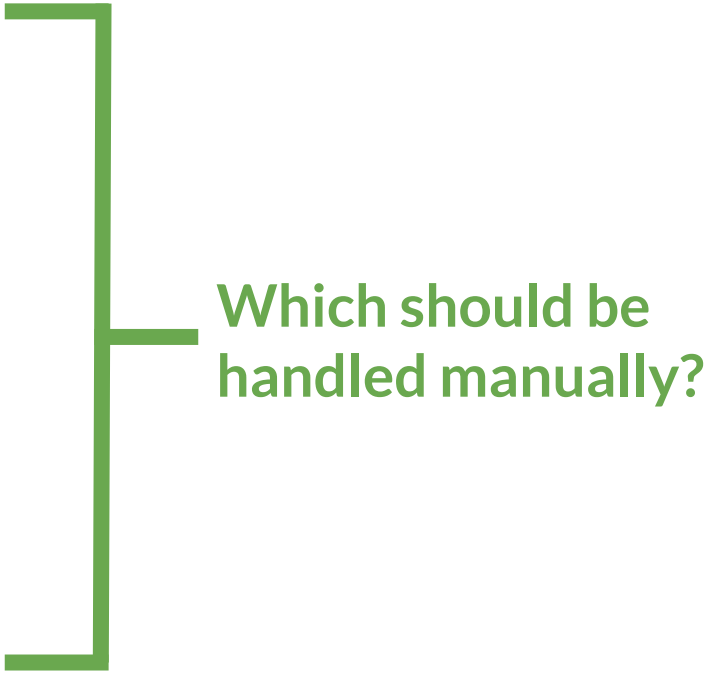
What does a developer do?

What does a developer do?

- Get the source code
- Install dependencies
- Compile the code
- Run static analysis
- Generate documentation
- Run tests
- Create artifacts for customers
- Ship!

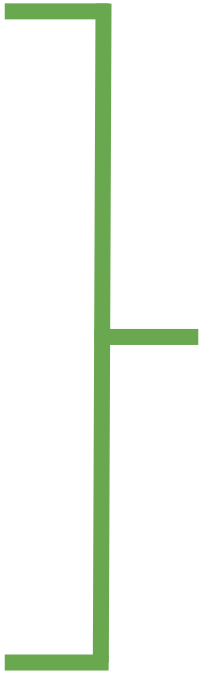
What does a developer do?

- Get the source code
- Install dependencies
- Compile the code
- Run static analysis
- Generate documentation
- Run tests
- Create artifacts for customers
- Ship!



What does a developer do?

- Get the source code
- Install dependencies
- Compile the code
- Run static analysis
- Generate documentation
- Run tests
- Create artifacts for customers
- Ship!



Which should be handled manually?

NONE!

What to do instead?

What to do instead?

Automate with a build system!

What is a build system?

What is a build system?

- A tool for automating software engineering tasks

What is a build system?

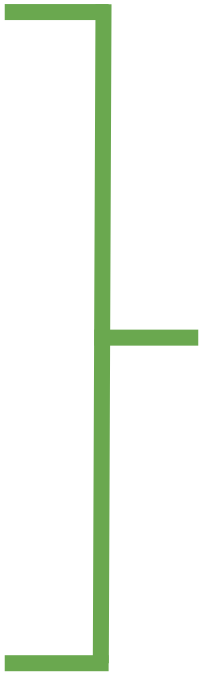
- A tool for automating software engineering tasks
  - Getting the source code
  - Installing dependencies
  - Compiling the code
  - Running static analysis
  - Generating documentation
  - Running tests
  - Creating artifacts for customers
  - Shipping!

What is a build system?

- A tool for automating software engineering tasks
  - Getting the source code
  - Installing dependencies
  - Compiling the code
  - Running static analysis
  - Generating documentation
  - Running tests
  - Creating artifacts for customers
  - Shipping!

What is a build system?

- A tool for automating software engineering **tasks**
  - Getting the source code
  - Installing dependencies
  - Compiling the code
  - Running static analysis
  - Generating documentation
  - Running tests
  - Creating artifacts for customers
  - Shipping!



A good build system handles all these

Tasks

- A task is something that the build system can do

Tasks

- A task is something that the build system can do
  - Getting the source code
  - Installing dependencies
  - Compiling the code
  - Running static analysis
  - Generating documentation
  - Running tests
  - Creating artifacts for customers
  - Shipping!

Tasks

- A task is something that the build system can do
  - Getting the source code
  - Installing dependencies
  - Compiling the code
  - Running static analysis
  - Generating documentation
  - Running tests
  - Creating artifacts for customers
  - Shipping!

All tasks!

Tasks

Tasks are code!

Tasks

Tasks are code!

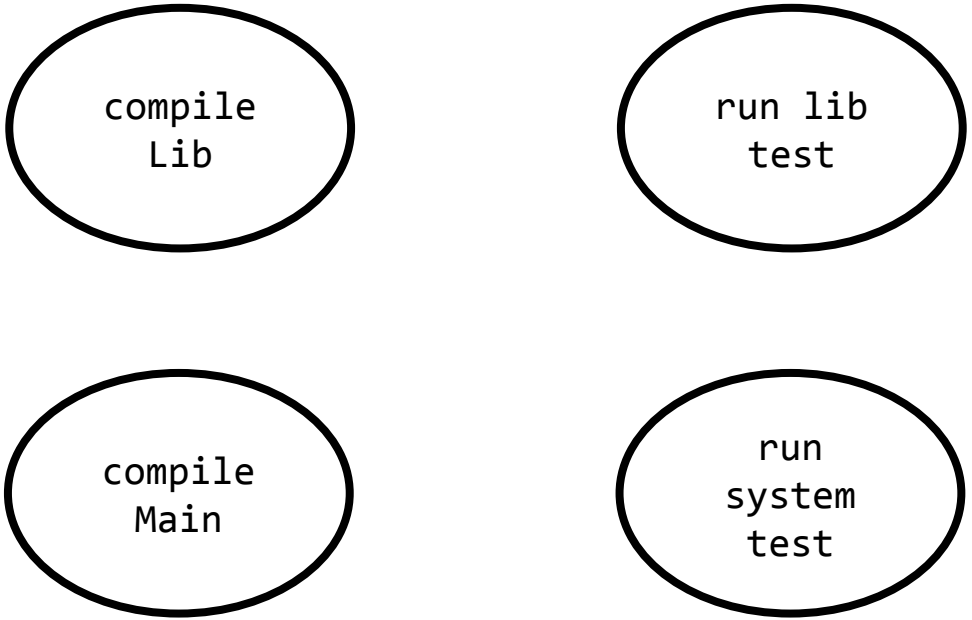
- Should be checked into version control
- Should be code-reviewed
- Should be tested

Dependencies between tasks

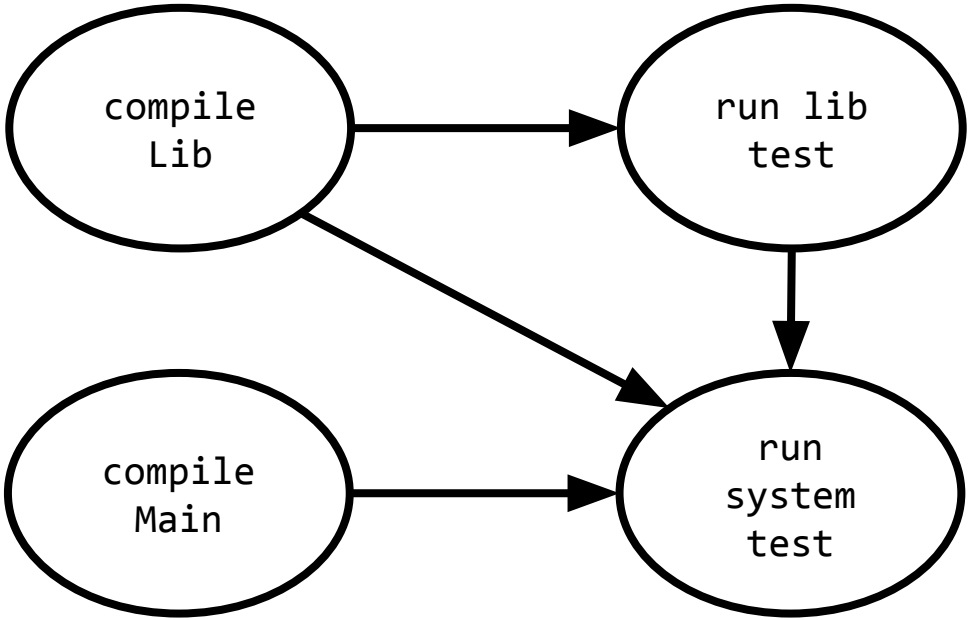
```
> ls src/

Lib.java   LibTest.java   Main.java    SystemTest.java
```

Dependencies between tasks



Dependencies between tasks



Dependencies between tasks

- A large project may have thousands of tasks

Dependencies between tasks

- A large project may have thousands of tasks
  - What order to run in?
  - How to speed up?

Dependencies between tasks

- A large project may have thousands of tasks
  - **What order to run in?**
  - How to speed up?

Determining task ordering

- Dependencies between tasks form a directed acyclic graph

Determining task ordering

- Dependencies between tasks form a directed acyclic graph

Topological sort!

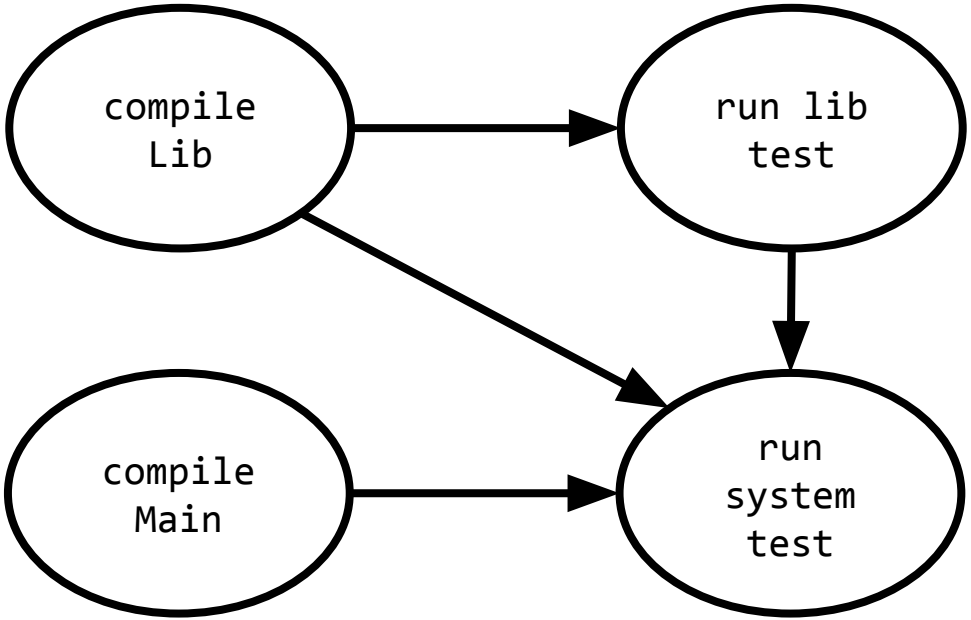
Topological sort

- Any ordering on the nodes such that all dependencies are satisfied

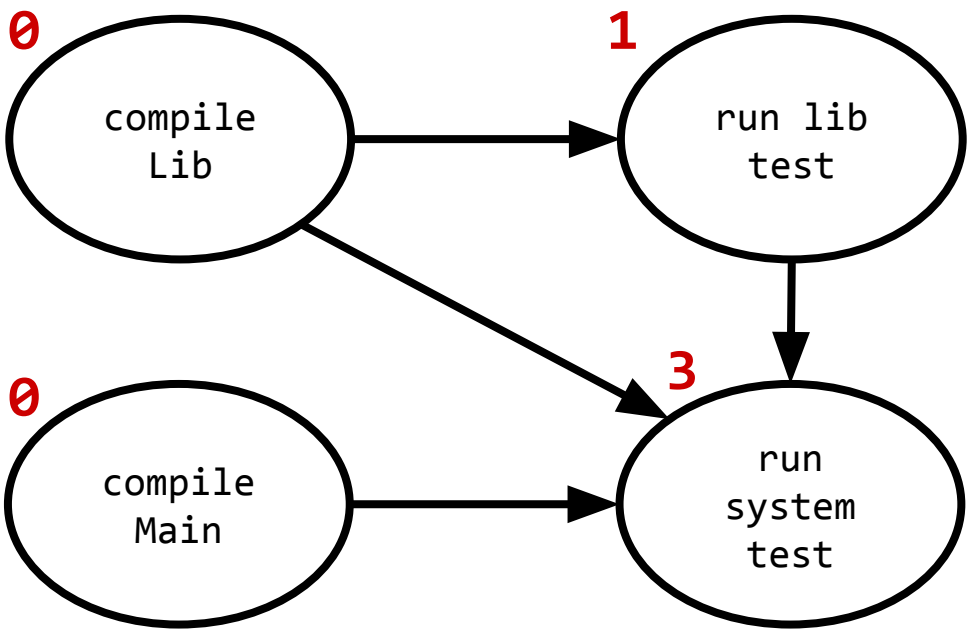
Topological sort

- Any ordering on the nodes such that all dependencies are satisfied
- Implement by computing *indegree* (number of incoming edges) for each node

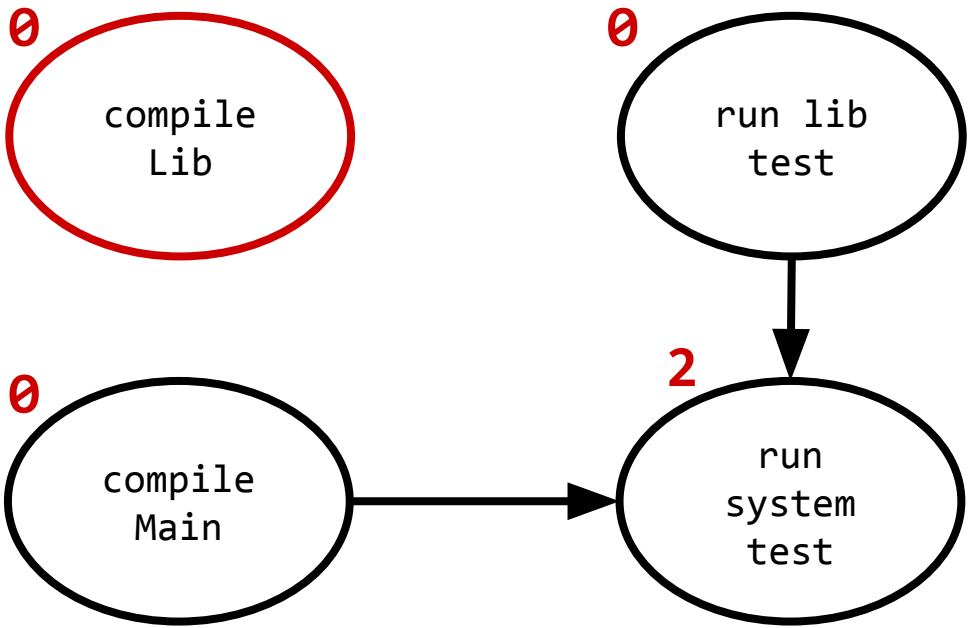
Topological sort



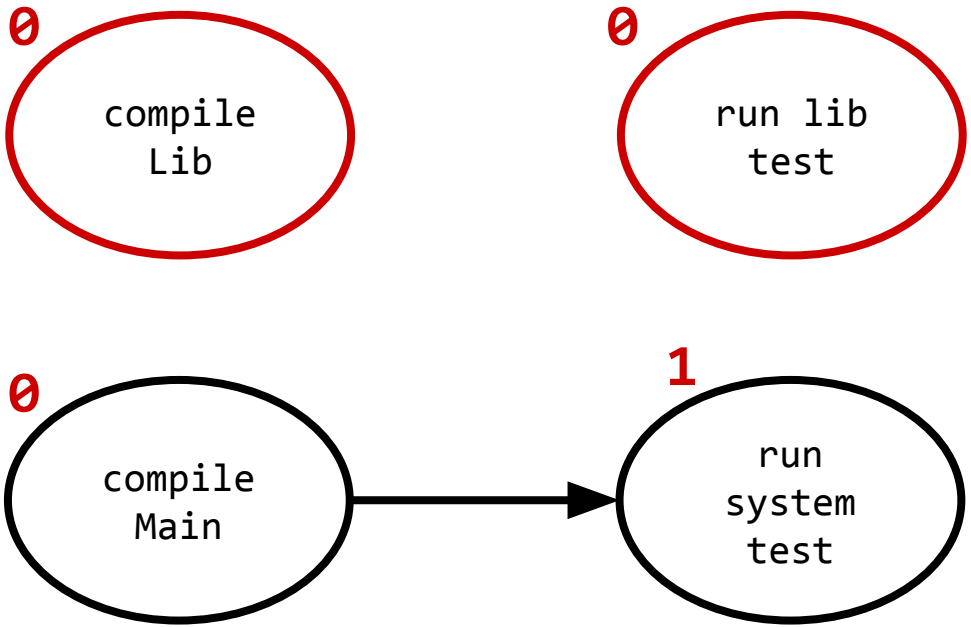
Topological sort



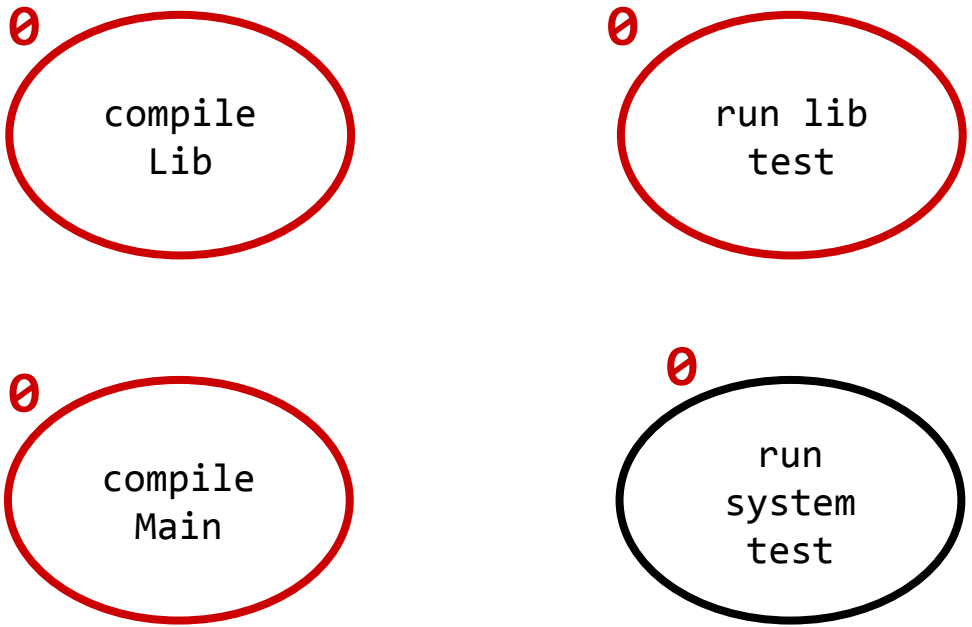
Topological sort



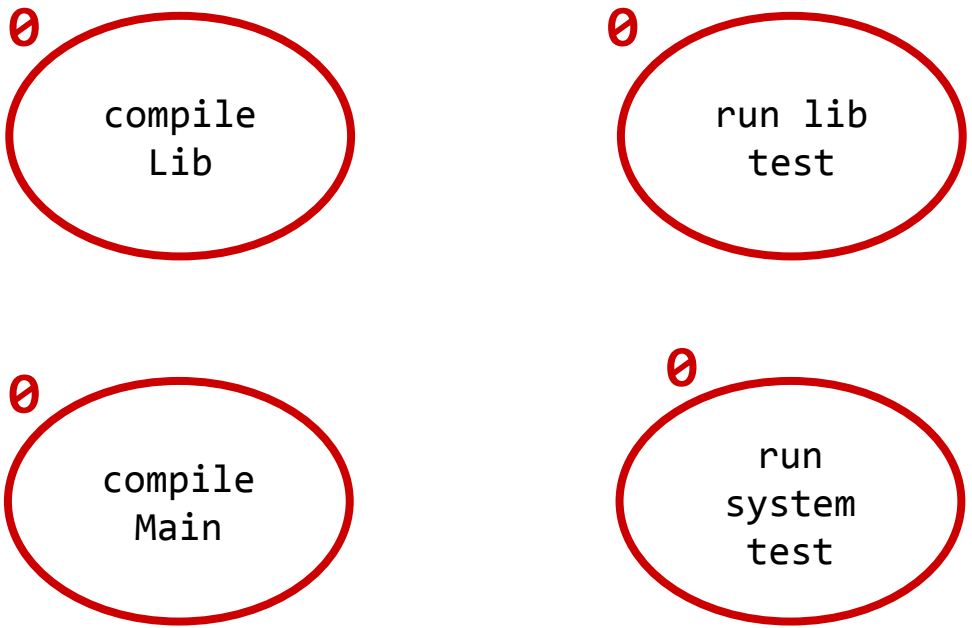
Topological sort



Topological sort



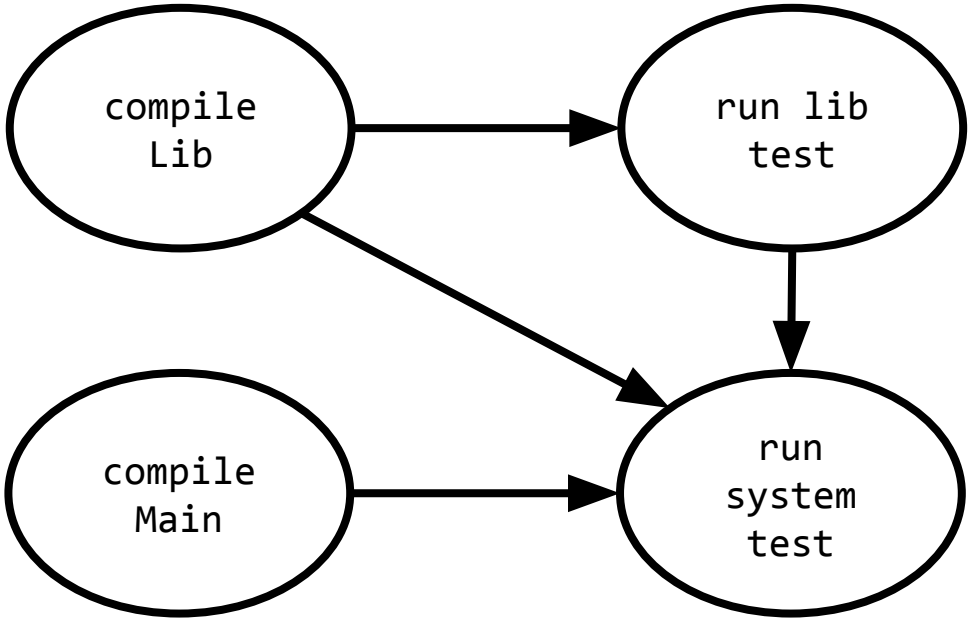
Topological sort



Topological sort

Valid sorts:

1. compile Lib, run lib test, compile Main, run system test

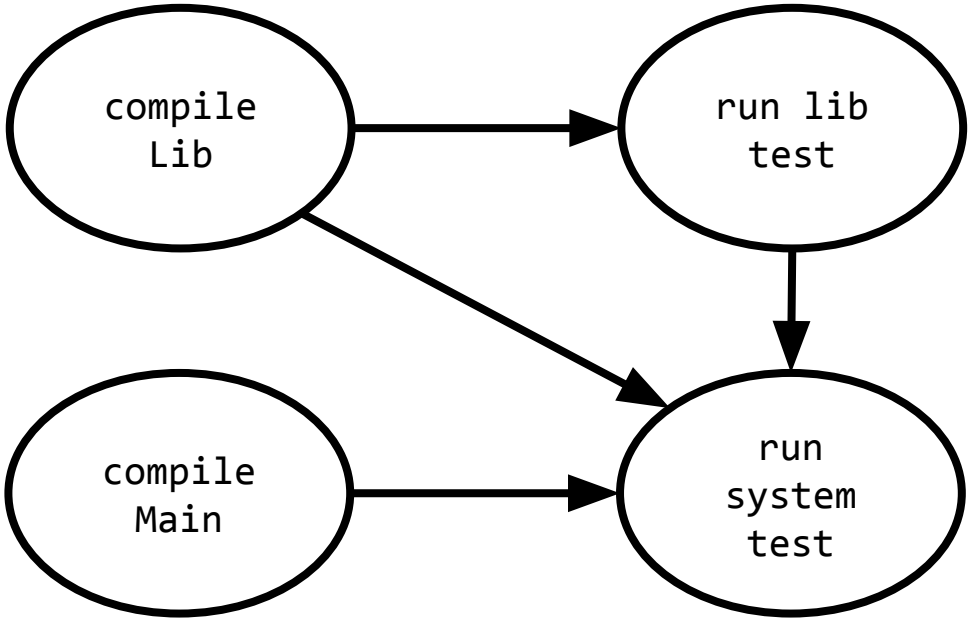


Topological sort

Valid sorts:

1. compile Lib, run lib test, compile Main, run system test

2. compile Main, compile Lib, run lib test, run system test



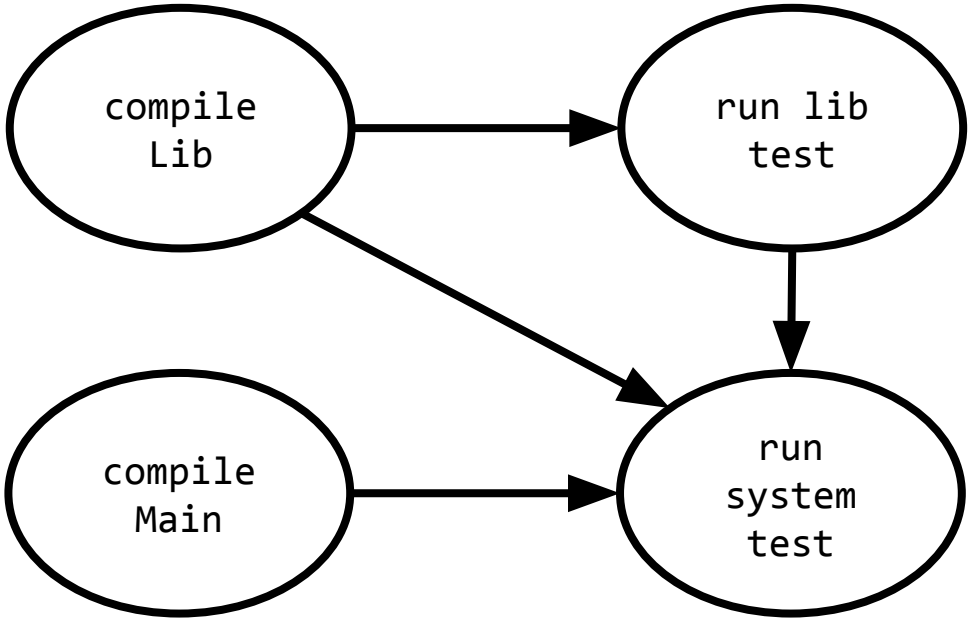
Topological sort

Valid sorts:

1. compile Lib, run lib test, compile Main, run system test

2. compile Main, compile Lib, run lib test, run system test

3. compile Lib, compile Main, run lib test, run system test



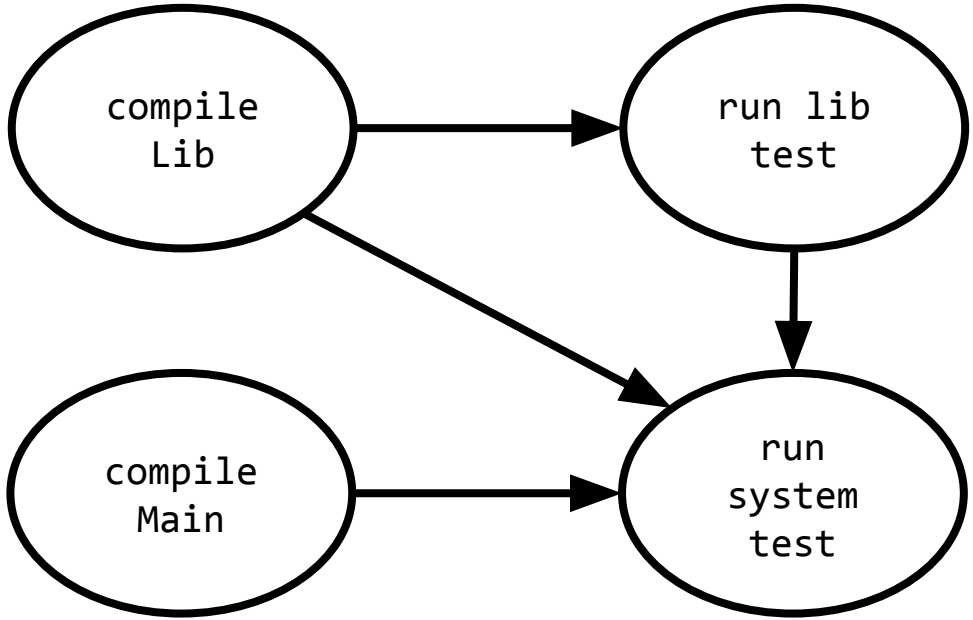
Topological sort

Valid sorts:

1. compile Lib, run lib test, compile Main, run system test

2. compile Main, compile Lib, run lib test, run system test

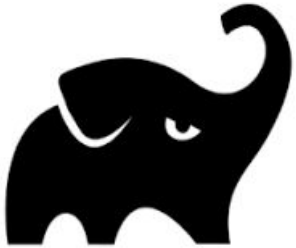
3. compile Lib, compile Main, run lib test, run system test



Why is this order silly?

Examples of modern build systems

`gradle`



https://gradle.org/

Apache's open-source successor to ant, maven

`bazel`

https://www.bazel.build/

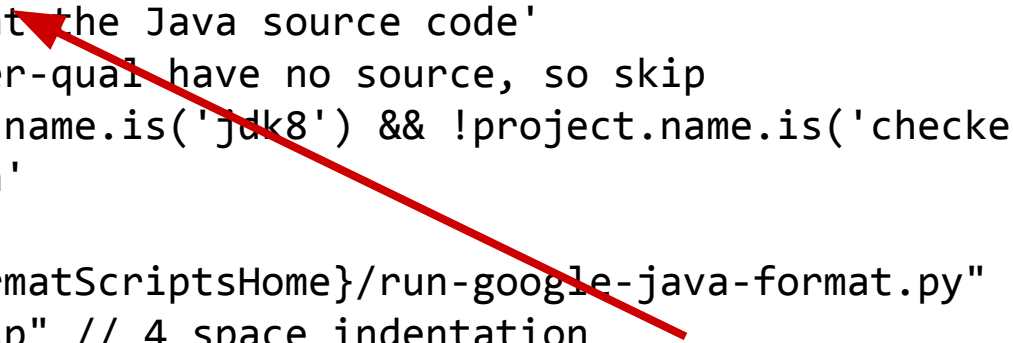
Google's internal build tool, open-sourced

Example task: `gradle`

```
task reformat(type: Exec, dependsOn: getCodeFormatScripts, group: 'Format') {
   description 'Format the Java source code'
   // jdk8 and checker-qual have no source, so skip
   onlyIf { !project.name.is('jdk8') && !project.name.is('checker-qual') }
   executable 'python'
   doFirst {
       args += "${formatScriptsHome}/run-google-java-format.py"
       args += "--aosp" // 4 space indentation
       args += getJavaFilesToFormat(project.name)
   }
}
```

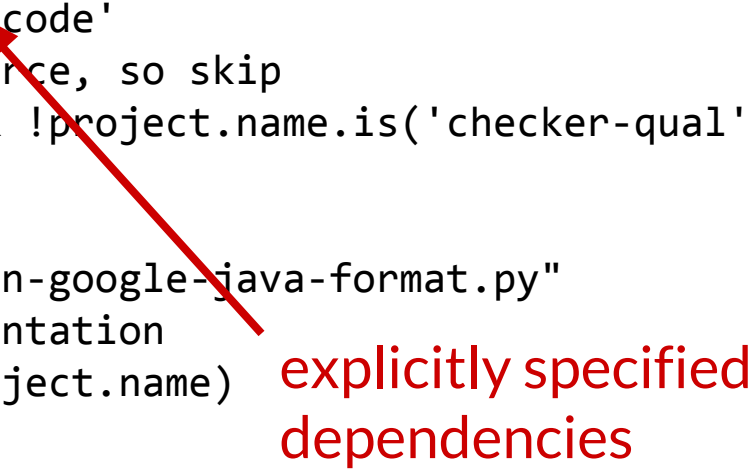
Example task: `gradle`

```
task reformat(type: Exec, dependsOn: getCodeFormatScripts, group: 'Format') {
   description 'Format the Java source code'
   // jdk8 and checker-qual have no source, so skip
   onlyIf { !project.name.is('jdk8') && !project.name.is('checker-qual') }
   executable 'python'
   doFirst {
       args += "${formatScriptsHome}/run-google-java-format.py"
       args += "--aosp" // 4 space indentation
       args += getJavaFilesToFormat(project.name)
   }
}
```



Example task: `gradle`

```
task reformat(type: Exec, dependsOn: getCodeFormatScripts, group: 'Format') {
   description 'Format the Java source code'
   // jdk8 and checker-qual have no source, so skip
   onlyIf { !project.name.is('jdk8') && !project.name.is('checker-qual') }
   executable 'python'
   doFirst {
       args += "${formatScriptsHome}/run-google-java-format.py"
       args += "--aosp" // 4 space indentation
       args += getJavaFilesToFormat(project.name)
   }
}
```



Example task: `gradle`

```
task reformat(type: Exec, dependsOn: getCodeFormatScripts, group: 'Format') {
   description 'Format the Java source code'
   // jdk8 and checker-qual have no source, so skip
   onlyIf { !project.name.is('jdk8') && !project.name.is('checker-qual') }
   executable 'python'
   doFirst {
       args += "${formatScriptsHome}/run-google-java-format.py"
       args += "--aosp" // 4 space indentation
       args += getJavaFilesToFormat(project.name)
   }
}
```

Example task: `bazel`

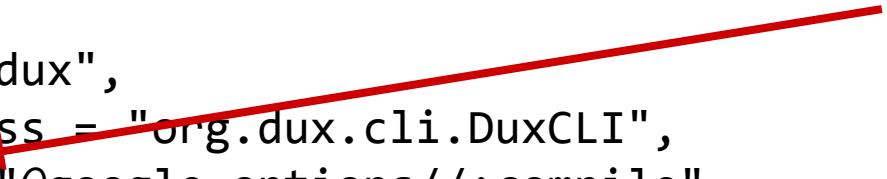
```
java_binary(
    name = "dux",
    main_class = "org.dux.cli.DuxCLI",
    deps = ["@google_options//:compile",
            "@checker_qual//:compile",
            "@google_cloud_storage//:compile",
            "@slf4j//:compile",
            "@logback_classic//:compile"],
    srcs = glob(["src/org/dux/cli/*.java",
                 "src/org/dux/backingstore/*.java"),
)
```

Example task: `bazel`

```
java_binary(
    name = "dux",
    main_class = "org.dux.cli.DuxCLI",
    deps = ["@google_options//:compile",
            "@checker_qual//:compile",
            "@google_cloud_storage//:compile",
            "@slf4j//:compile",
            "@logback_classic//:compile"],
    srcs = glob(["src/org/dux/cli/*.java",
                 "src/org/dux/backingstore/*.java"),
)
```

Example task: `bazel`

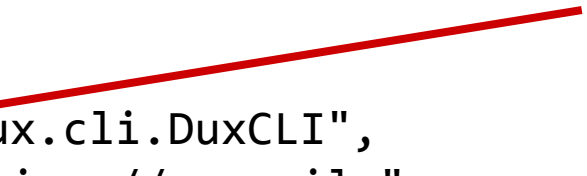
```
java_binary(
    name = "dux",
    main_class = "org.dux.cli.DuxCLI",
    deps = ["@google_options//:compile",
            "@checker_qual//:compile",
            "@google_cloud_storage//:compile",
            "@slf4j//:compile",
            "@logback_classic//:compile"],
    srcs = glob(["src/org/dux/cli/*.java",
                 "src/org/dux/backingstore/*.java"),
)
```



explicitly specified dependencies

Example task: `bazel`

```
java_binary(
    name = "dux",
    main_class = "org.dux.cli.DuxCLI",
    deps = ["@google_options//:compile",
            "@checker_qual//:compile",
            "@google_cloud_storage//:compile",
            "@slf4j//:compile",
            "@logback_classic//:compile"],
    srcs = glob(["src/org/dux/cli/*.java",
                 "src/org/dux/backingstore/*.java"),
)
```



explicitly specified dependencies (also bazel tasks)

External and internal dependencies

- A list of tasks (internal) or libraries (external)

External and internal dependencies

- A list of tasks (internal) or libraries (external)

```
deps = ["@google_options//:compile",
        "@checker_qual//:compile",
        "@google_cloud_storage//:compile",
        "@slf4j//:compile",
        "@logback_classic//:compile"],
```

```
dependencies {
    compile group:
        'org.hibernate',
        name: 'hibernate-core',
        version : '3.6.7.Final'
    testCompile group:
        'junit',
        name: 'junit',
        version : '4.+'
}
```

https://docs.gradle.org/current/userguide/artifact dependencies tutorial.html

Why list dependencies?

- Reproducibility!

Why list dependencies?

- Reproducibility!

```
deps = ["@google_options//:compile",
        "@checker_qual//:compile",
        "@google_cloud_storage//:compile",
        "@slf4j//:compile",
        "@logback_classic//:compile"],
```

```
dependencies {
    compile group:
        'org.hibernate',
        name: 'hibernate-core',
        version: '3.6.7.Final'
    testCompile group:
        'junit',
        name: 'junit',
        version: '4.+'
}
```

https://docs.gradle.org/current/userguide/artifact_dependencies_tutorial.html

Dependencies between tasks

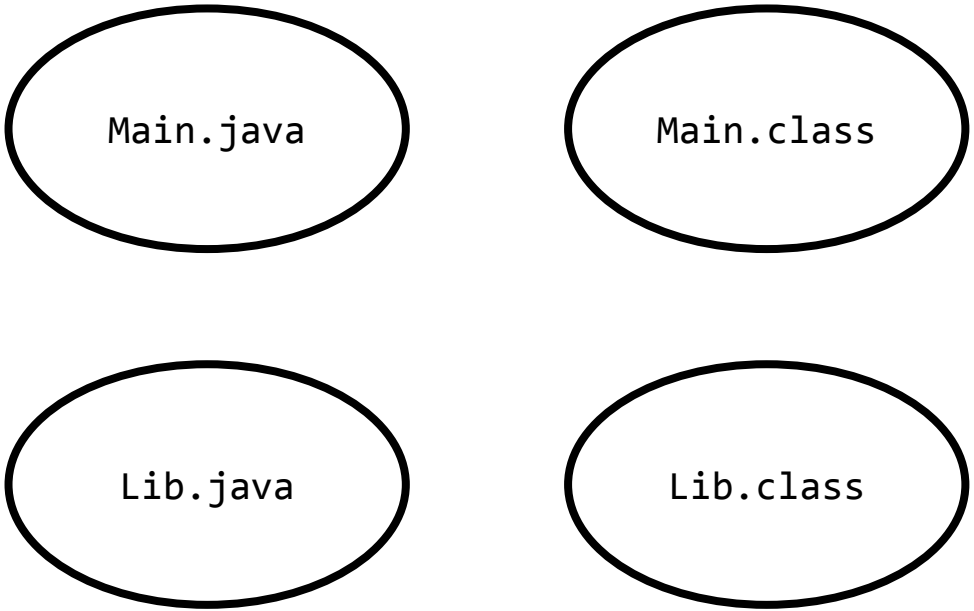
- A large project may have thousands of tasks
  - What order to run in?
  - **How to speed up?**

How to speed up builds?

How to speed up builds?

- Incrementalize - only rebuild what you have to

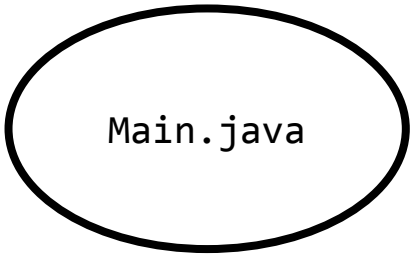
Incrementalization



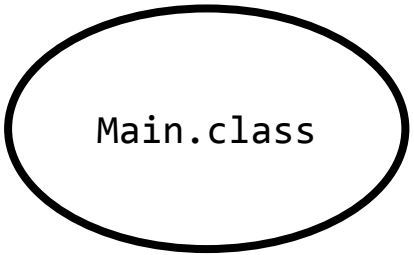
Incrementalization

1:31 PM

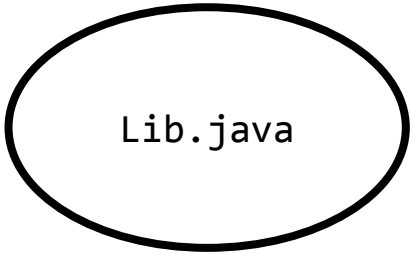
modified 10:45 AM



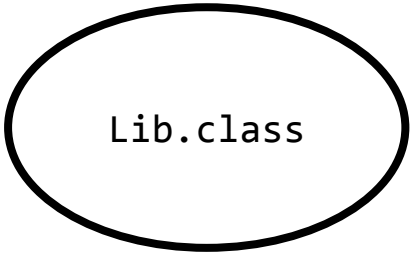
modified 11:06 AM



modified 1:30 PM



modified 11:06 AM



Incrementalization

- Compute hash codes for inputs to each task
- When about to execute a task, check input hashes - if they match the last time the task was executed, skip it!

How to speed up builds?

- Incrementalize - only rebuild what you have to
- Execute many tasks in parallel
- Cache artifacts in the cloud

Best practices

- Automate everything

Best practices

- Automate everything
- Always use a build tool

Best practices

- Automate everything
- Always use a build tool
- Have a build server that builds and tests your code on every commit (continuous integration)

Best practices

- Automate everything
- Always use a build tool
- Have a build server that builds and tests your code on every commit (continuous integration)
- Don’t break the build