CSE 403

Software Engineering

Winter 2023

Build systems

This week

- Build systems
  - What is a build system?
  - Best practices
  - Gradle live demo
- Testing and Continuous Integration (CI)
- Code review

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 of these tasks should be handled manually?

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 of these tasks should be handled manually?

NONE!

How to automate these tasks?

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

Orchestrate tasks with a build system!

What is a build system (build tool)?

A tool for automating software engineering **tasks**:

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

Build systems: tasks

Tasks are code!

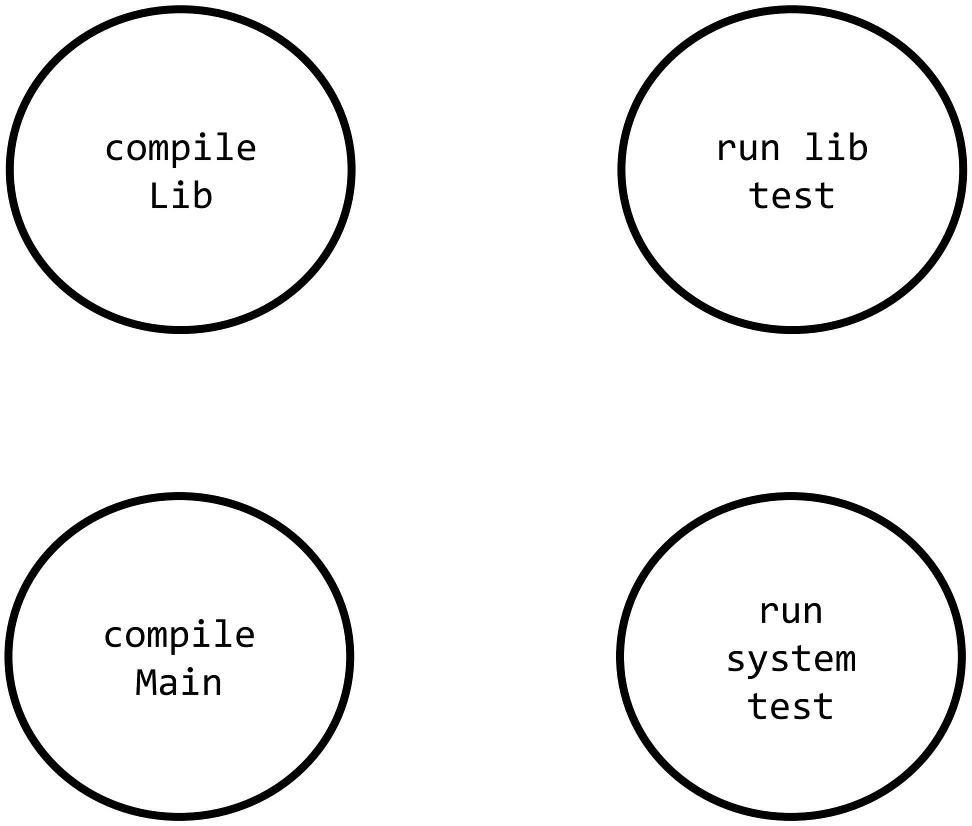
- Should be checked into version control
- Should be code-reviewed
- Should be tested

Build systems: dependencies between tasks

Example code and corresponding tests:

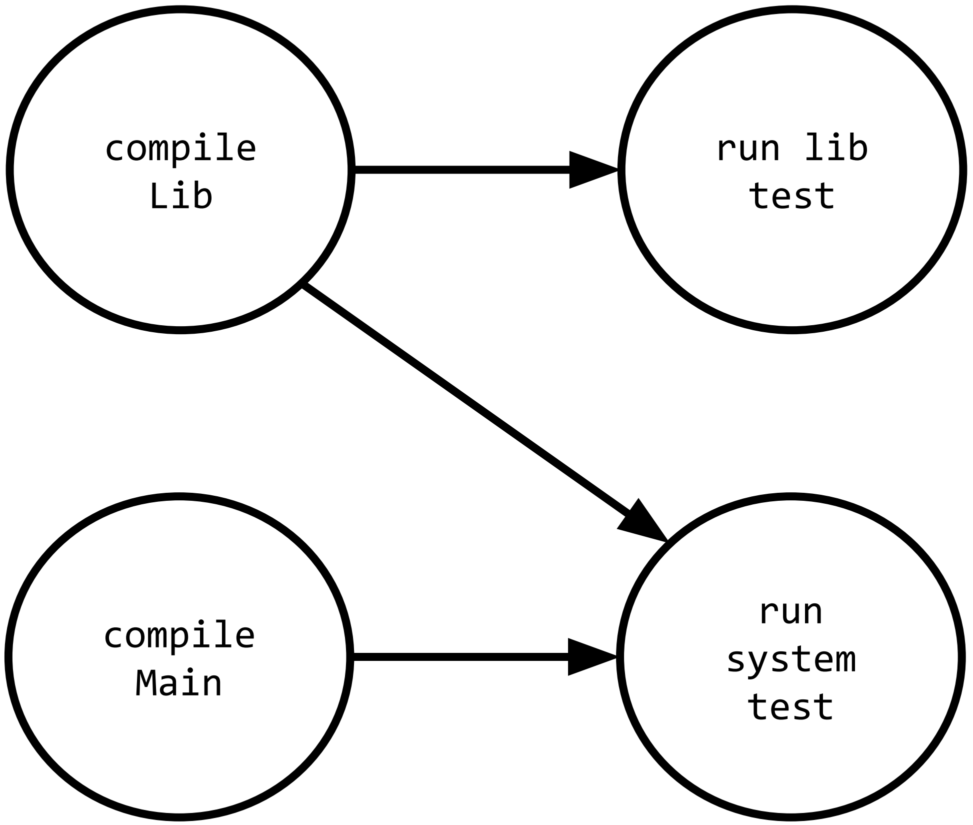
```
> ls src/

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

Build systems: dependencies between tasks

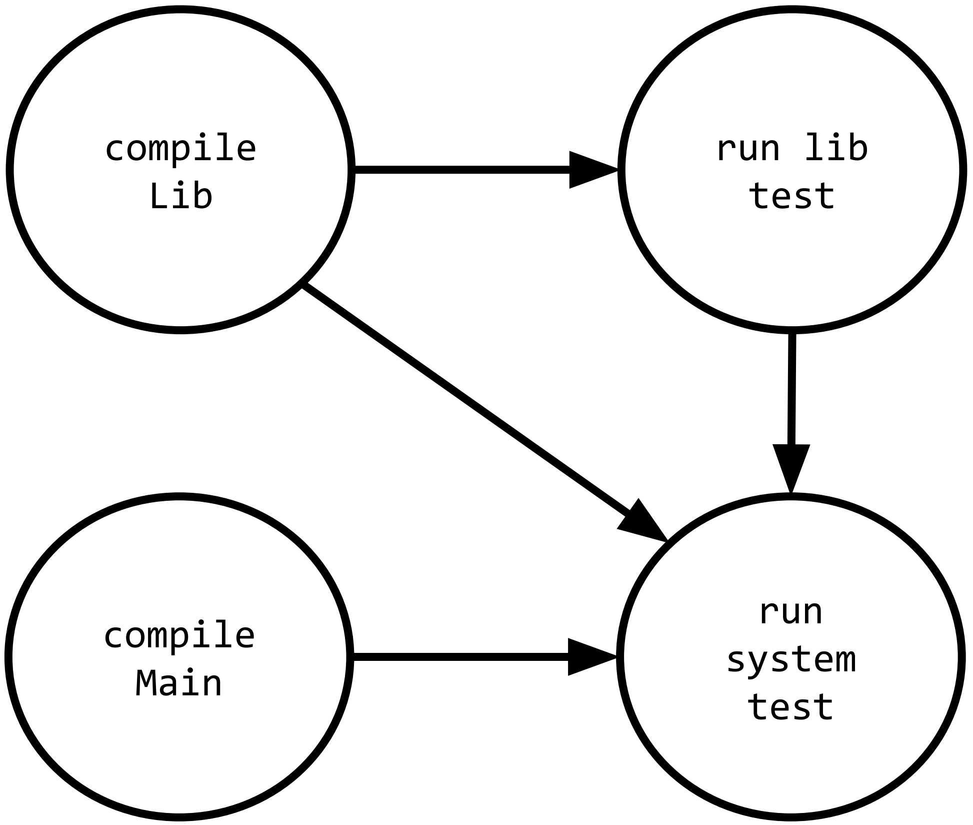


What are the dependencies between these tasks?

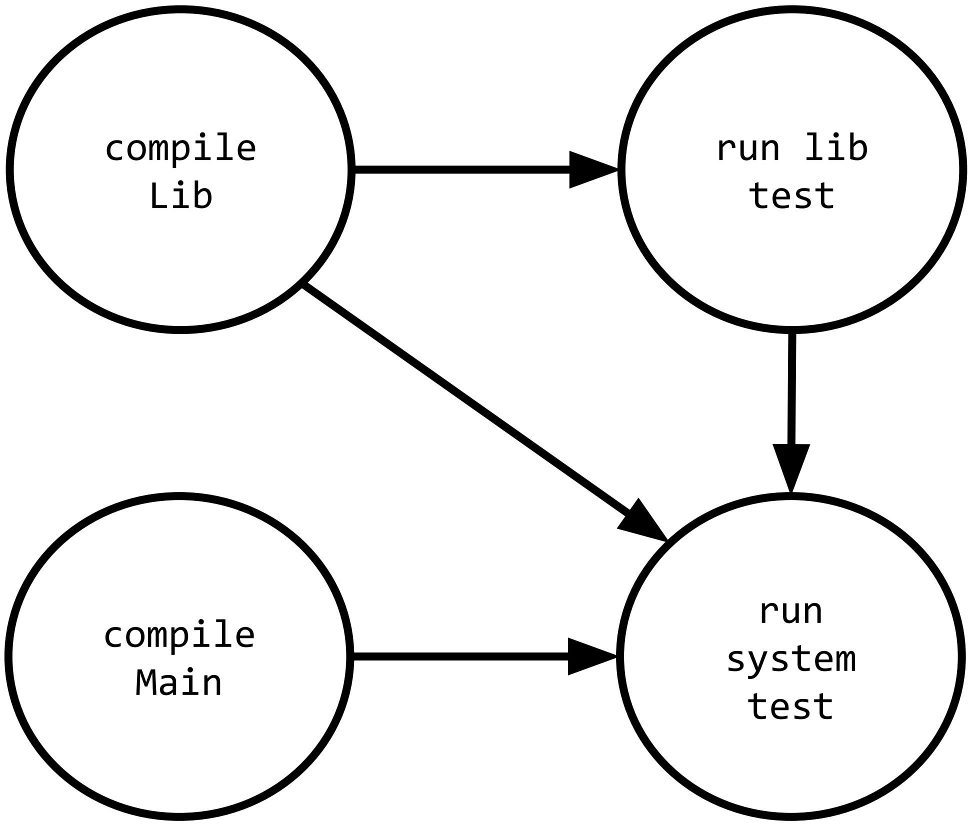
Build systems: dependencies between tasks



Build systems: dependencies between tasks



Build systems: dependencies between tasks



In what order should we run these tasks?

Build systems: determining task order

Large projects have thousands of tasks

- Dependencies between tasks form a directed acyclic graph.

Build systems: determining task order

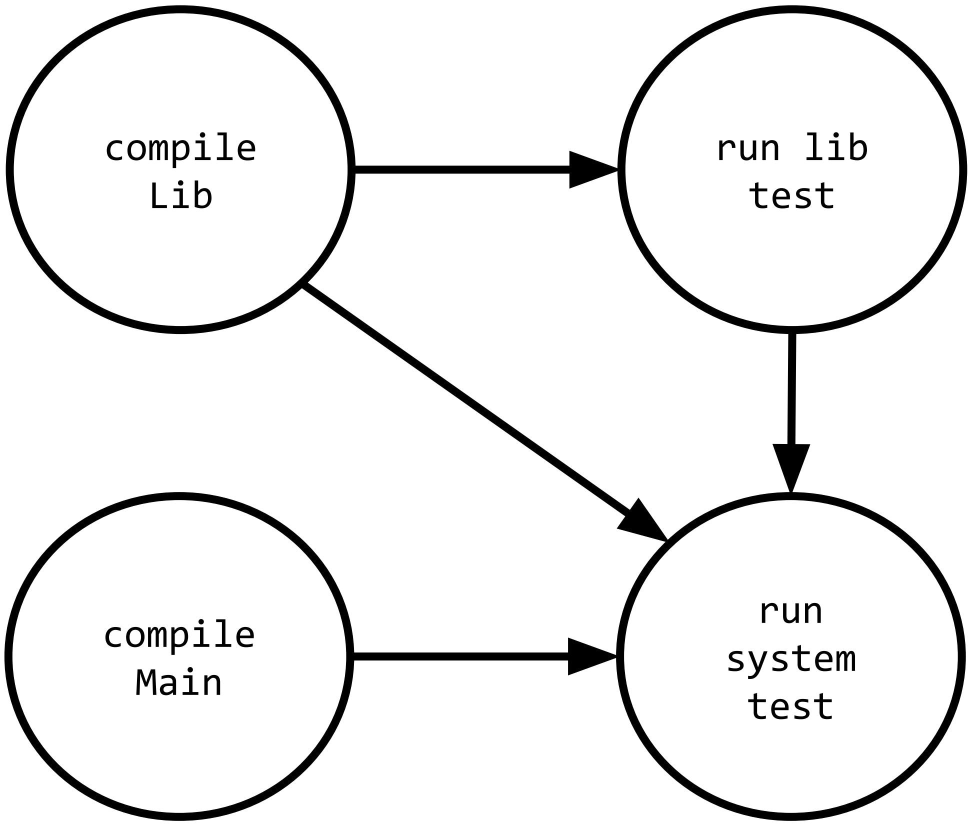
Large projects have thousands of tasks

- Dependencies between tasks form a directed acyclic graph.

Topological sort

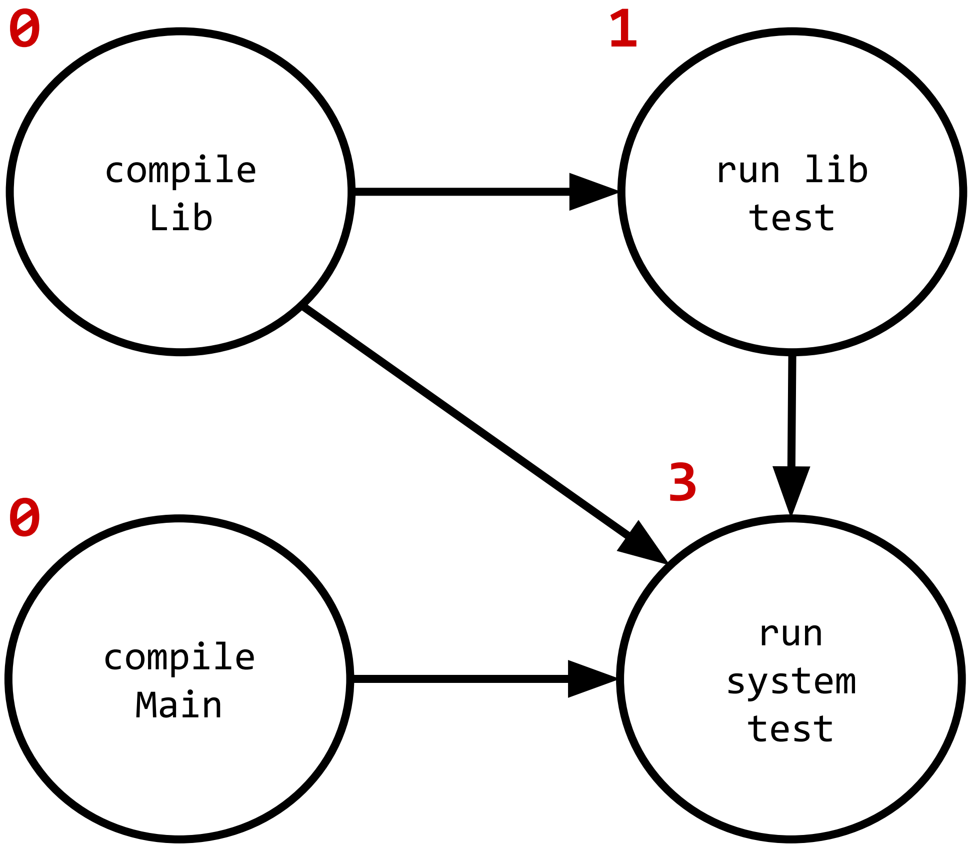
- Order nodes such that all dependencies are satisfied
- **Implemented by computing indegree**
  (number of incoming edges) for each node

Build systems: topological sort

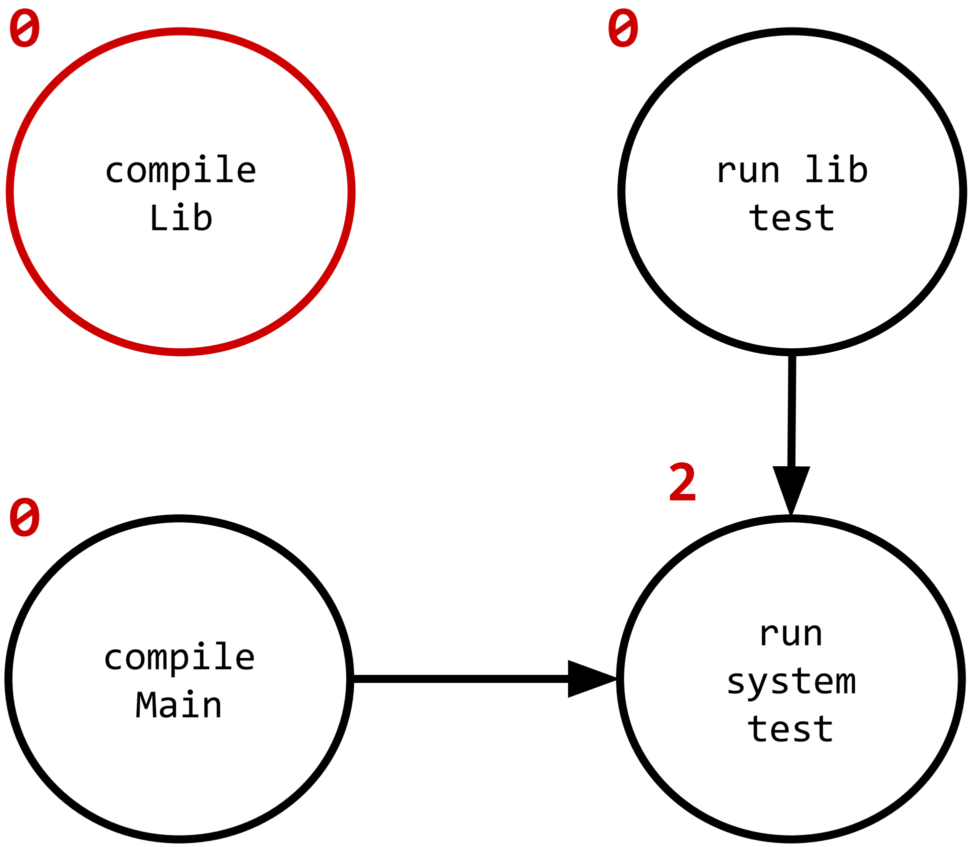


What’s the indegree of each node?

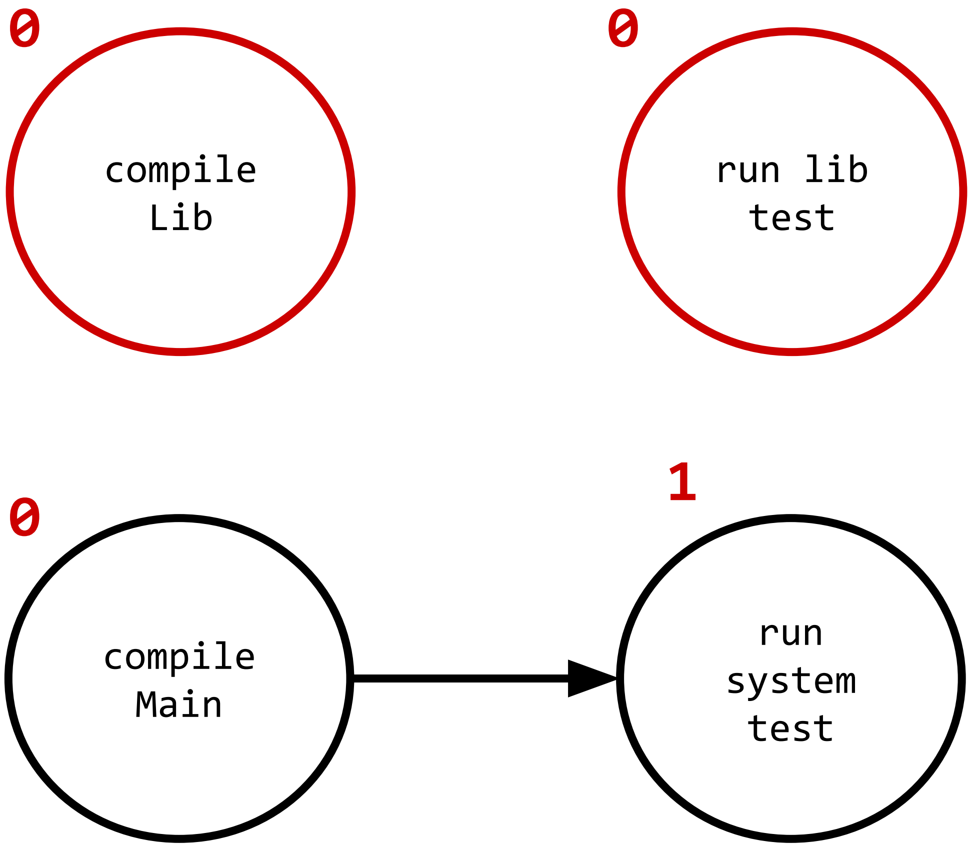
Build systems: topological sort



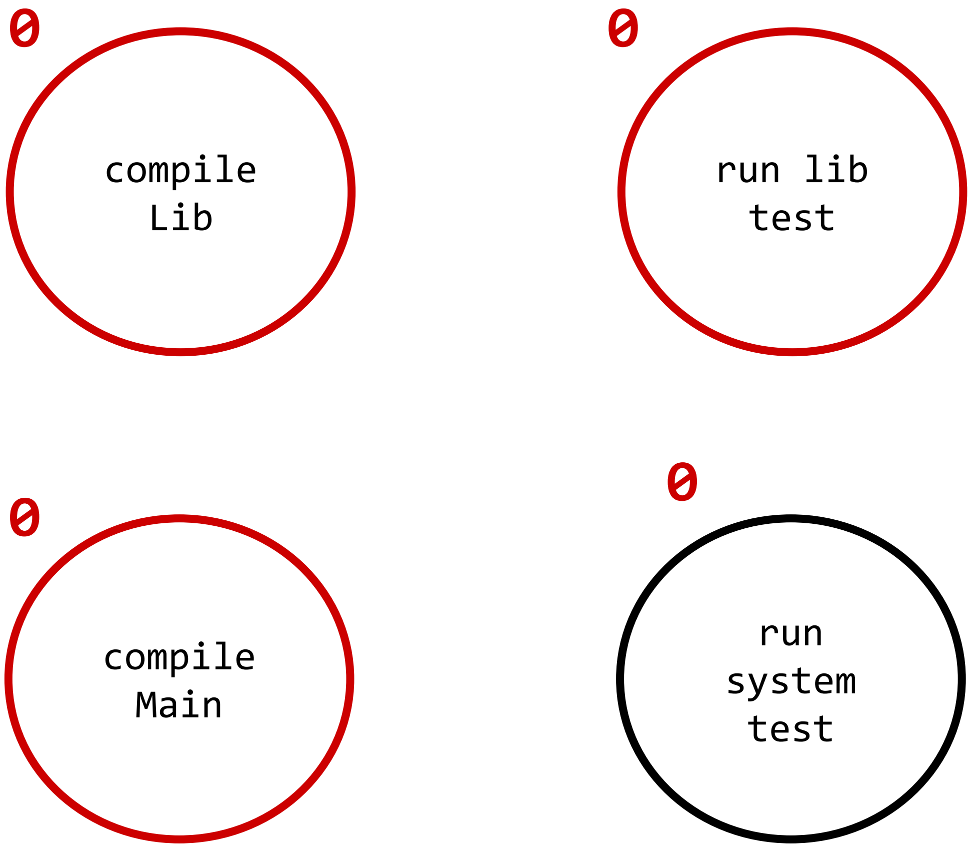
Build systems: topological sort



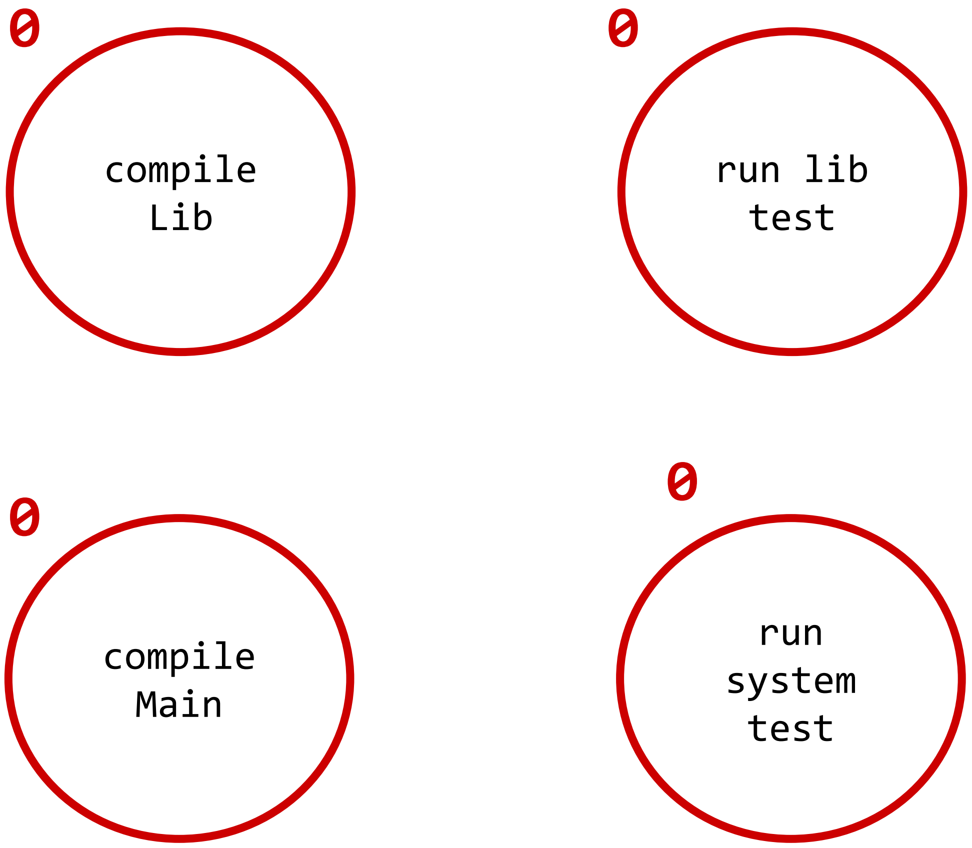
Build systems: topological sort



Build systems: topological sort



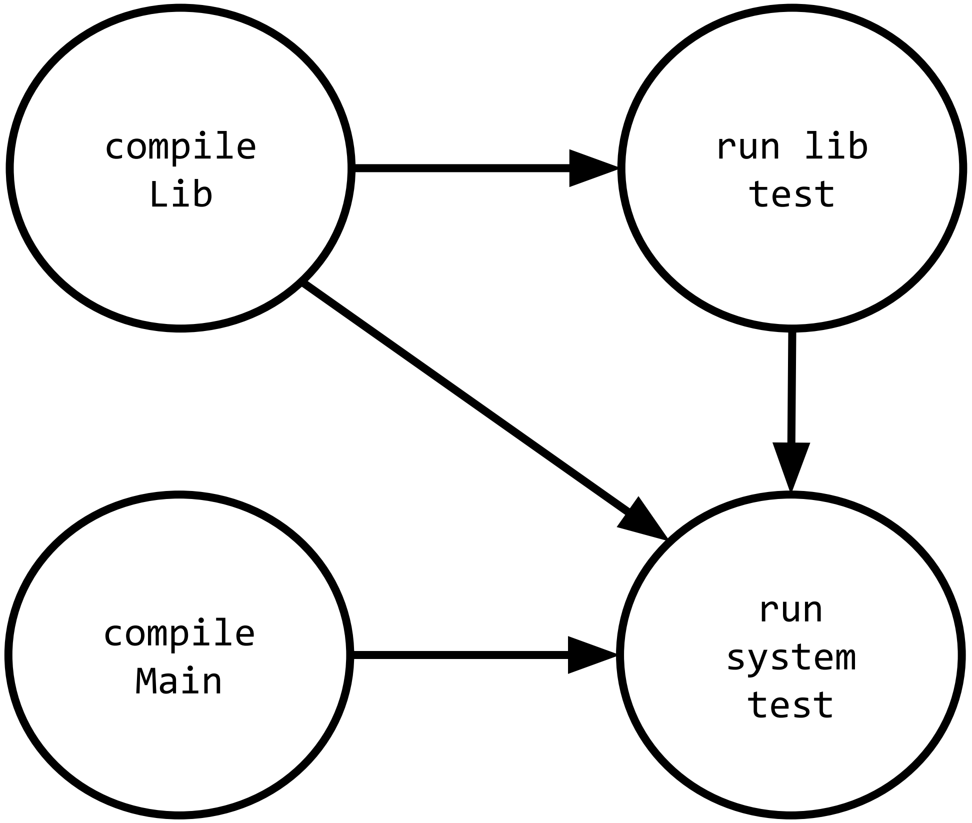
Build systems: topological sort



Build systems: topological sort

Valid sorts:

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

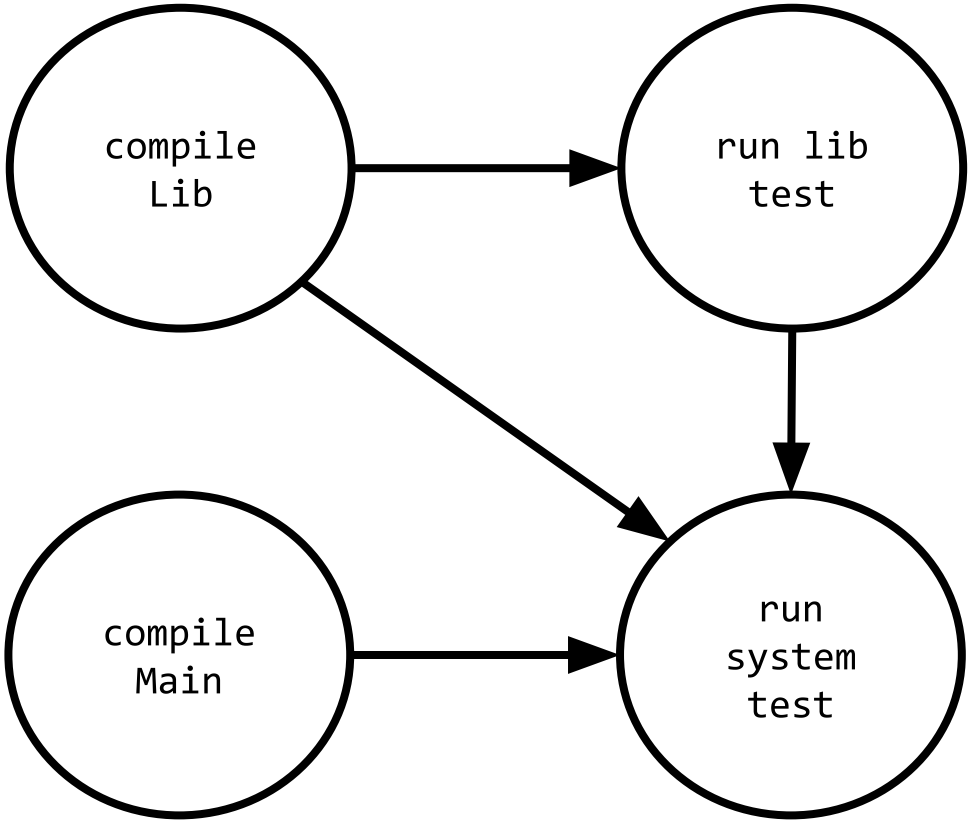


Build systems: topological sort

Valid sorts:

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

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



$$\begin{tikzcd}
\text{compile Lib} \arrow[r] \arrow[dr] & \text{run lib test} \arrow[d] \\
\text{compile Main} \arrow[r] & \text{run system test}
\end{tikzcd}$$

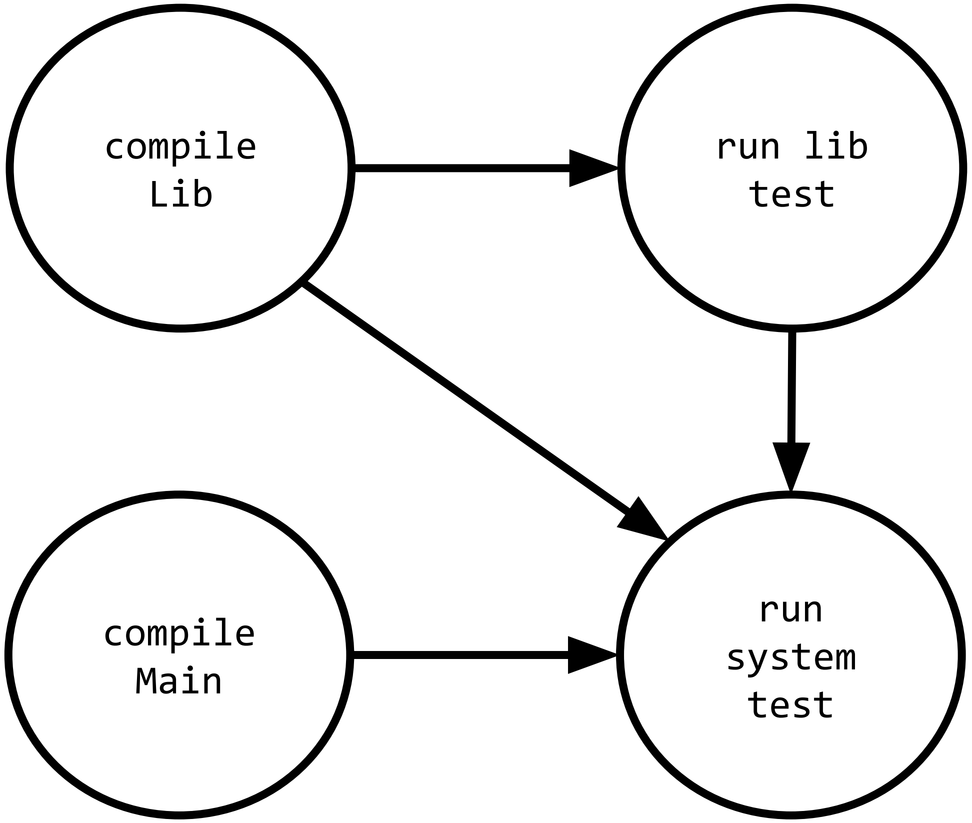
Build systems: topological sort

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



Which of these sorts is preferable?

Build systems: examples

gradle

Open-source successor to ant and maven

- Groovy/Kotlin DSL (vs. xml)
- Many defaults for (maven) conventions
- Can query Maven Central for dependency resolution

bazel

Open-source version of Google's internal build tool (blaze)

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)
    }
}
```

explicitly specified dependencies

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)
   }
}
```

actual source code (no xml)!

In many cases, following conventions and using built-in tasks is sufficient!

Best practices

- Automate everything (one-step build)!
- Always use a build tool.
- Use CI to build and test your code on every commit.
- Don’t depend on anything that’s not in the build file (hermetic)!
- Don’t break the build!

Live demo: Build systems

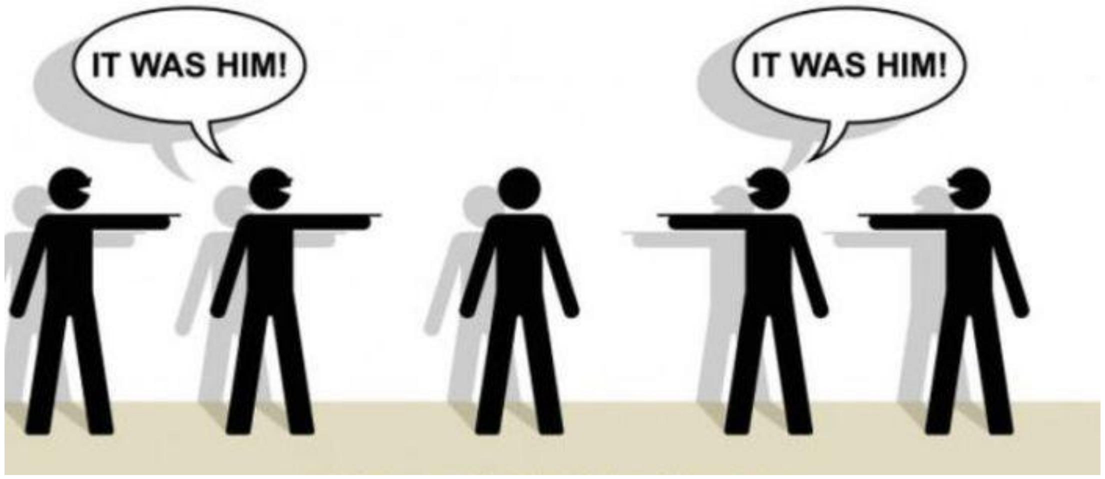
Set up:

1. Two clones of the basic-stats repo (cloned from Bitbucket).
2. Goal: migrate from Ant to Gradle.

Two scenarios:

1. Bad: Breaking the build on master
2. Good: New hermetic build on a branch

Live demo Part 1: Breaking the build



René breaking the build on master

build failure

Ben making a small change

Live demo Part 2: New hermetic build

- Development on a branch
- Hermetic build
- Backward compatibility
- Testing and code review