CSE 403 Software Engineering

Build systems &
Continuous Integration and Deployment

Today’s outline

- **Build systems**
- **Continuous integration and deployment systems**

  - What are these
  - How do they relate
  - Best practices
  - Ideas to explore for your projects

What does a developer do?

The code is written … now what?

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

What does a developer do?

The code is written … now what?

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

Which of these tasks should be handled manually?

What does a developer do?

The code is written … now what?

- Get the source code
- Install dependencies
- Compile the code
- Run static analysis
- Generate documentation
- Run tests
- Create artifacts for customers
- Ship
- Operate, monitor, repeat

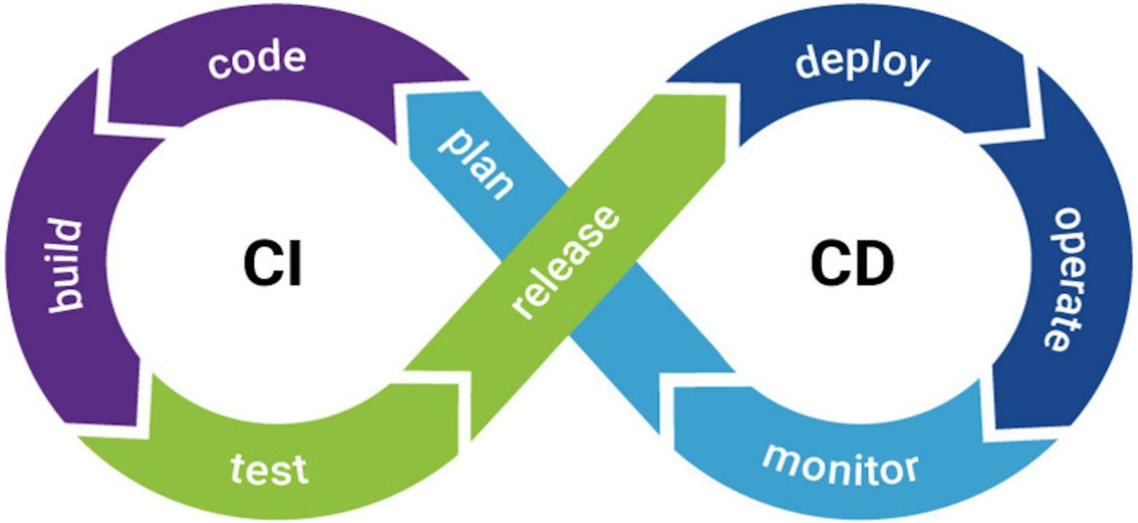
Which of these tasks should be handled manually?

NONE!

Instead, orchestrate with a tool

- **Build system**: a tool for automating compilation and other tasks
- Is a component of a **continuous integration/deployment system**

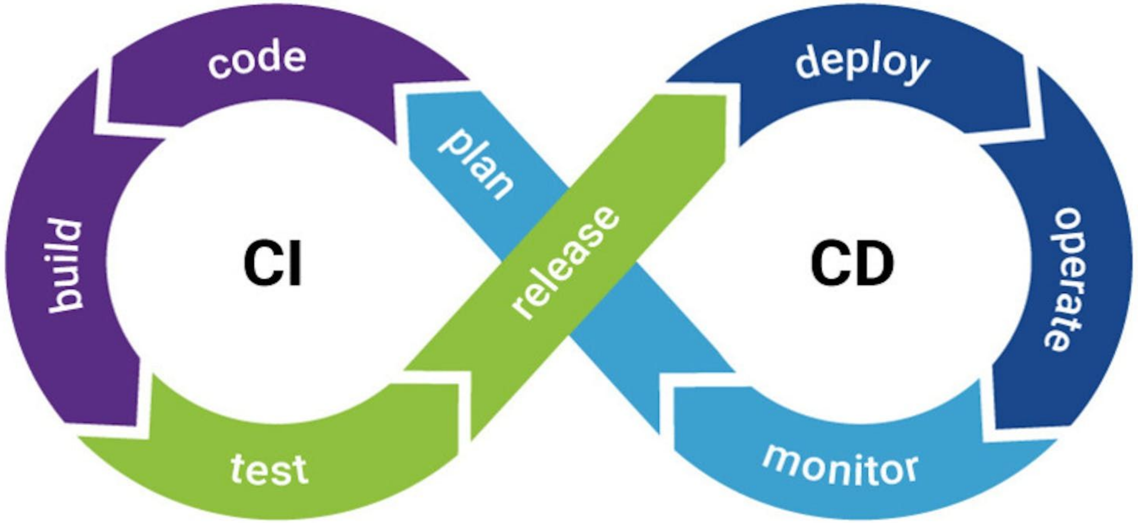
✔Get the source code
✔Install dependencies
✔Run static analysis
✔Compile the code
✔Generate documentation
✔Run tests
✔Create artifacts for customers
✔Ship!
✔Operate, Monitor, Repeat



Instead, orchestrate with a tool

- **Build system**: a tool for automating compilation and other **tasks**
- Is a component of a **continuous integration/deployment system**

✔Get the source code
✔Install dependencies
✔Run static analysis
✔Compile the code
✔Generate documentation
✔Run tests
✔Create artifacts for customers
✔Ship!
✔Operate, Monitor, Repeat



All tasks!

Build systems: tasks

Tasks are code!

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

Adding to our SE best practices list

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

So how can a **build** system help us?

1. **Dependency management**
    1. Identifies dependencies between files (including externals)
    2. Runs the compiles in the right order
    3. Only runs the compiles needed due to dependency changes

2. **Efficiency and reliability**
    1. Automates the build process, for any team member in any environment
    2. Formalizes the build process (no tribal knowledge)
    3. Eliminates the chance of errors
    4. Speeds up the process

Roles of a build system

A build system:

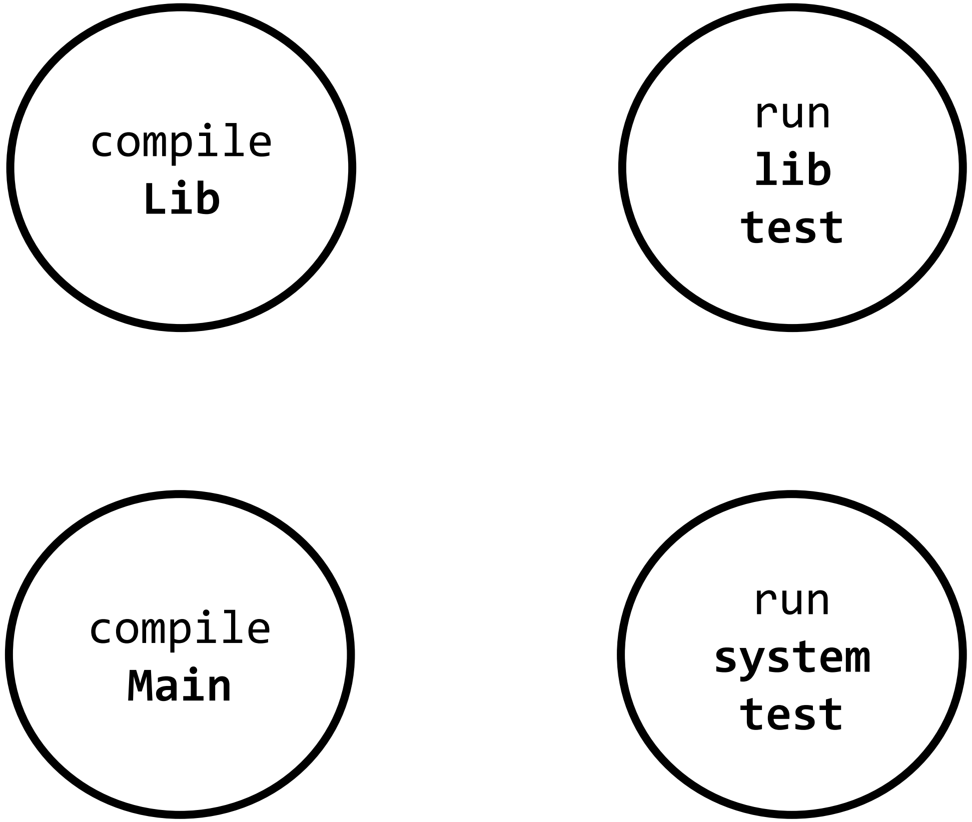
- defines **tasks** (and external resources, such as libraries)
- defines **dependencies** among tasks (a graph)
- **executes** the tasks

Simple example code for dependency mgmt

```
% ls src/
   Lib.java
   LibTest.java
   Main.java
   SystemTest.java
```

Build systems: dependencies between tasks

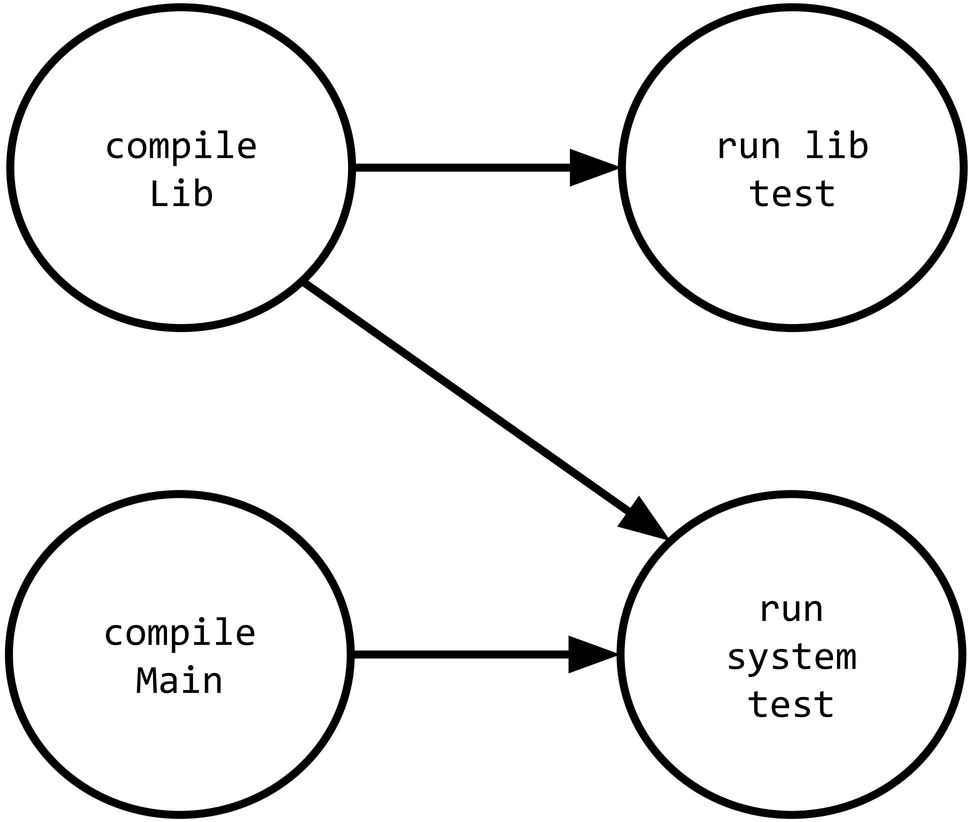
```
% ls src/
    Lib.java
    LibTest.java
    Main.java
    SystemTest.java
```



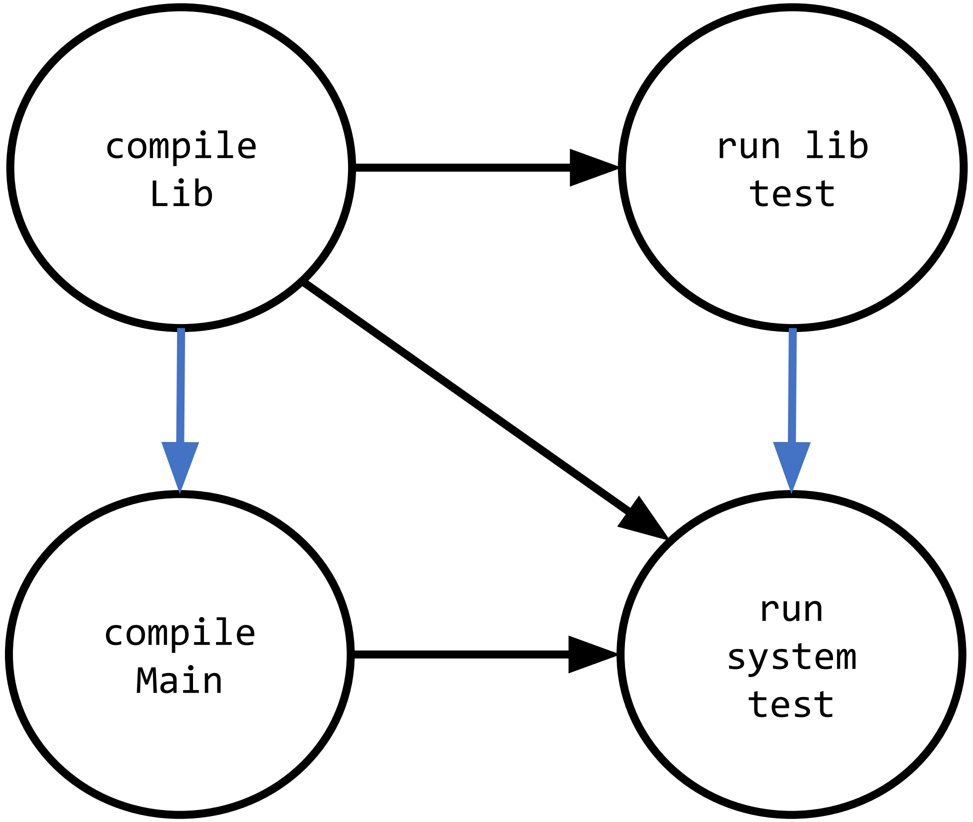
What are the dependencies between these tasks?

And why do I care?

Build systems: dependencies between tasks

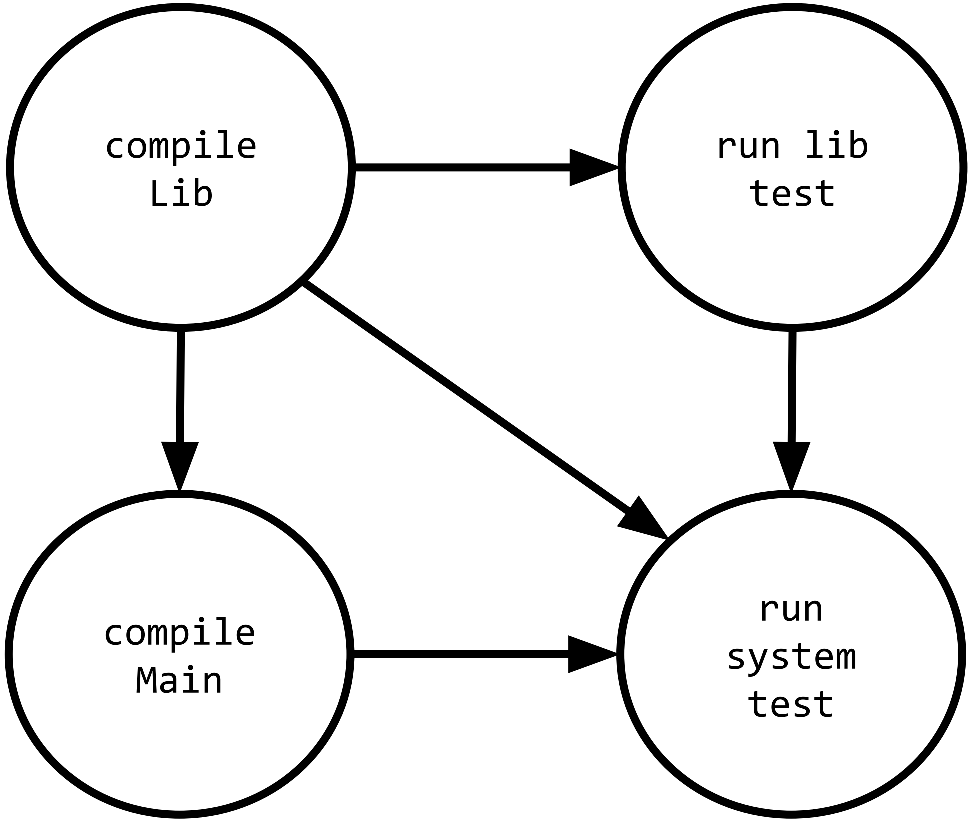


Build systems: dependencies between tasks



Build systems: dependencies between tasks

In what order should we run these tasks?



Build systems determine task order

Large projects have thousands of tasks

- Dependencies between tasks form a directed acyclic graph
- Use a topological sort to create an order for tasks
  - See Appendix for example

External code (libraries) also can be complex

- List all dependencies for reproducibility
  - A *hermetic build* is "insensitive to the libraries and other software installed on the build machine"[1]
- Build systems can manage external dependencies as well!
- And/or use a dependency manager

[1] https://landing.google.com/sre/sre-book/chapters/release-engineering/

Dependency manager

Unix: apt, yum
Java: Maven Central
JavaScript: NPM
Python: PIP
Ruby: RubyGems

Roles of a build system

A build system:

- defines **tasks**
- defines **dependencies** among tasks (a graph)
- **executes** the tasks

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)
   }
}
```

kind of rule

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)
   }
}
```

code!
(usually, following
conventions is enough)

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"]),
)
```

kind of rule

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)

How to speed up a build

- Incrementalize - only rebuild what you have to
    - Compute hash codes for inputs to each task
        - Watch out: there are more inputs than you think
    - Before executing a task, check input hashes
    - If they have not changed since the last time the task was executed, skip it!
- Execute many tasks in parallel
- Cache artifacts (in the cloud)

Static analysis

Can run before or after the compile step

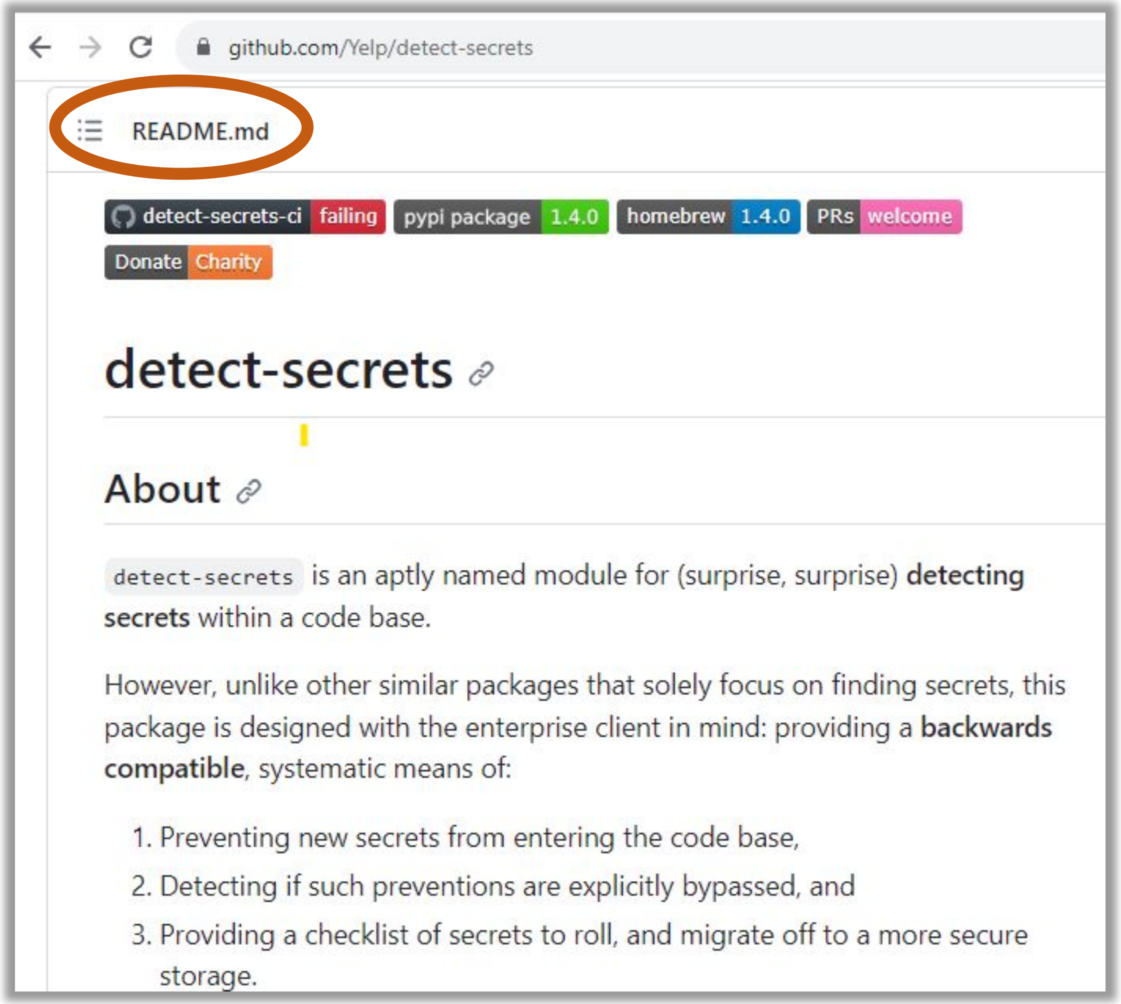
Examples:

- Credential scan
- Date scan
- Sensitive data scan

What might be others?

Is this worthwhile?

Build systems: opportunity for static analysis



detect-secrets

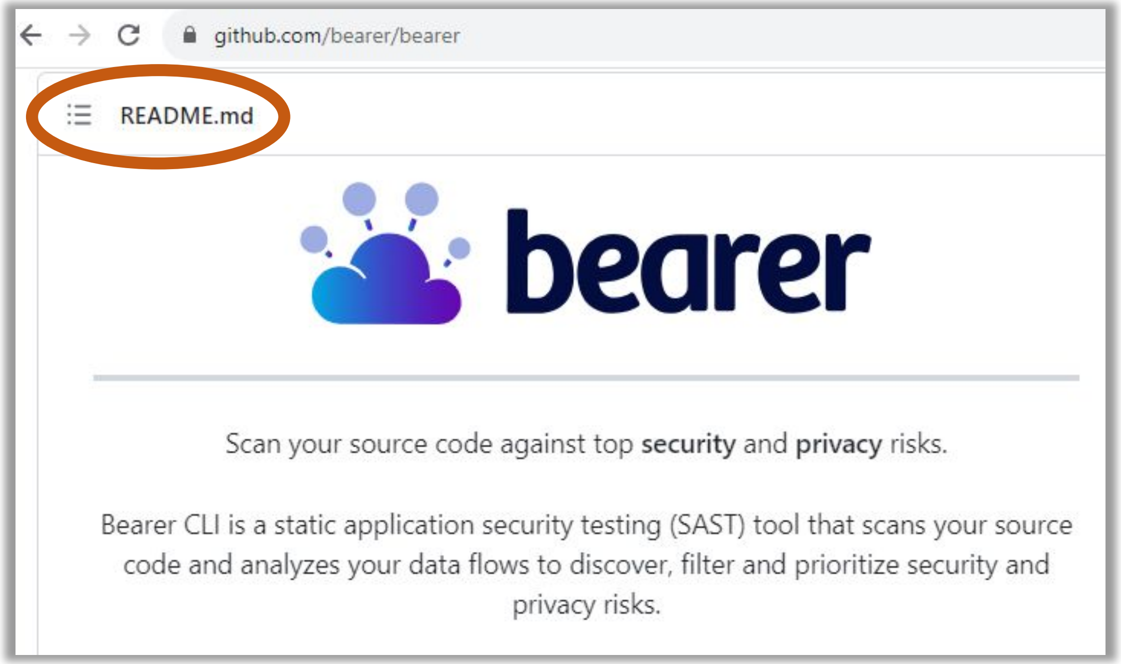
About

`detect-secrets` is an aptly named module for (surprise, surprise) **detecting secrets** within a code base.

However, unlike other similar packages that solely focus on finding secrets, this package is designed with the enterprise client in mind: providing a **backwards compatible**, systematic means of:

1. Preventing new secrets from entering the code base,
2. Detecting if such preventions are explicitly bypassed, and
3. Providing a checklist of secrets to roll, and migrate off to a more secure storage.

Could these types of static analysis tools be run earlier than CI?



Scan your source code against top **security** and **privacy** risks.

Bearer CLI is a static application security testing (SAST) tool that scans your source code and analyzes your data flows to discover, filter and prioritize security and privacy risks.

There are a *lot* of build systems

make

ant
maven
gradle
rake
SCons
sbt

blaze
buck

A build system:

- defines **tasks**
- defines **dependencies** among tasks (a graph)
- **executes** the tasks

Build system code may run at graph construction time or at task execution time

Assignment: evaluate and select a build system

Many other options!

Over to you to research

Java+		
	gradle	Open-source successor to **ant** and **maven**
	bazel	Open-source version of Google's internal build tool (blaze)
Python		
	hatch	Implements standards from the Python standard (uses TOML files, has PIP integration)
	poetry	Packaging and dependence manager
	tox	Automate and standardize testing
JavaScript		
	npm	Standard package/task manager for Node, "Largest software registry in the world."
	webpack	Module bundler for modern JavaScript applications
	gulp	Tries to improve dependency and packing

Today’s outline

- Build systems
- **Continuous integration and deployment systems** ← We are here
  - What are these and
  - How do they relate
  - Best practices
  - Ideas to explore for your projects

CI/CD: What's the difference?

Continuous Integration (CI)

- Devs regularly integrate code into a shared repository
- System builds/tests automatically with each update
- Complements local developer workflows (e.g., may run diff tests)
- **Goal:** to find/address bugs quicker, improve quality, reduce time to get to working code

Continuous Deployment (CD) [Continuous **Delivery**]

- Builds on top of CI
- Automatically pushes changes [to staging environment and then] to production
- **Goal:** always have a deployment-ready build that has passed through a standardized testing process

https://aws.amazon.com/devops/what-is-devops/

Just like build, there are many CI tool options

Jenkins

GitHub Actions

AWS CodePipeline

Azure Pipelines

Travis CI

GitLab

circleci

Bitbucket Pipelines

Assignment: Research, evaluate and choose a CI system

Continuous **integration** basics

- A CI **workflow** is **triggered** when an **event** occurs in your [shared] repo
  - Example events
    - Push
    - Pull request
    - Issue creation

- A workflow contains **jobs** that run in a defined order
  - A job is like a shell-script and can have multiple steps
  - Jobs run in their own vm/container called a **runner**
  - Example jobs
    - Run static analysis
    - Build, test
    - Deploy to test, deploy to prod

Using GitHub CI terminology but concepts span other CI systems

https://docs.github.com/en/actions

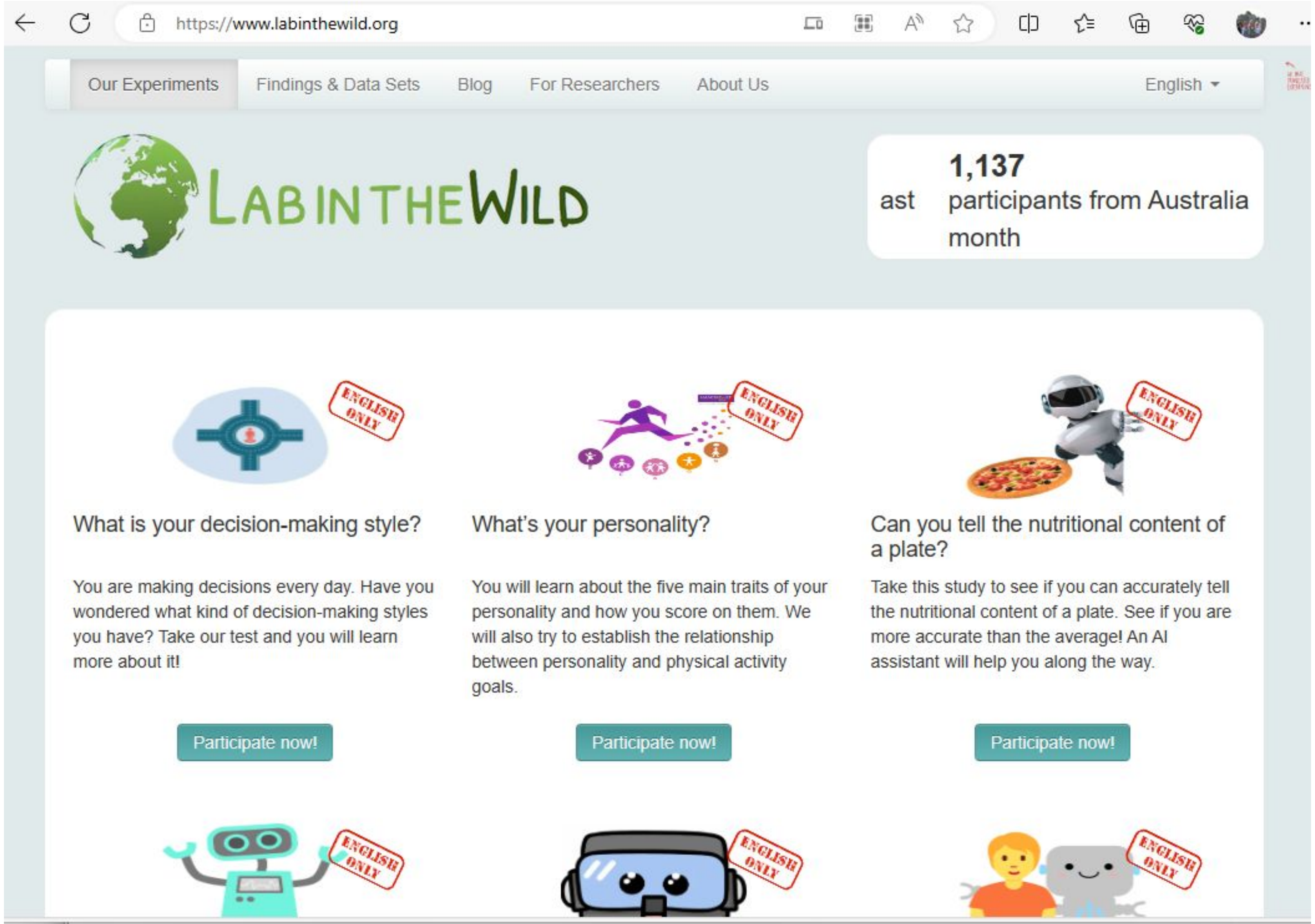
Nice light starter tutorial

Automation Step by Step:
https://www.youtube.com/watch?app=desktop&v=ylEy4eLdhFs

Example: CI at work at UW

Lab In The Wild is a research project drawing survey input from diverse community

Nigini Oliveira (researcher and 403 prof) provided this example



https://www.labinthewild.org

ast 1,137 participants from Australia month

What is your decision-making style?

You are making decisions every day. Have you wondered what kind of decision-making styles you have? Take our test and you will learn more about it!

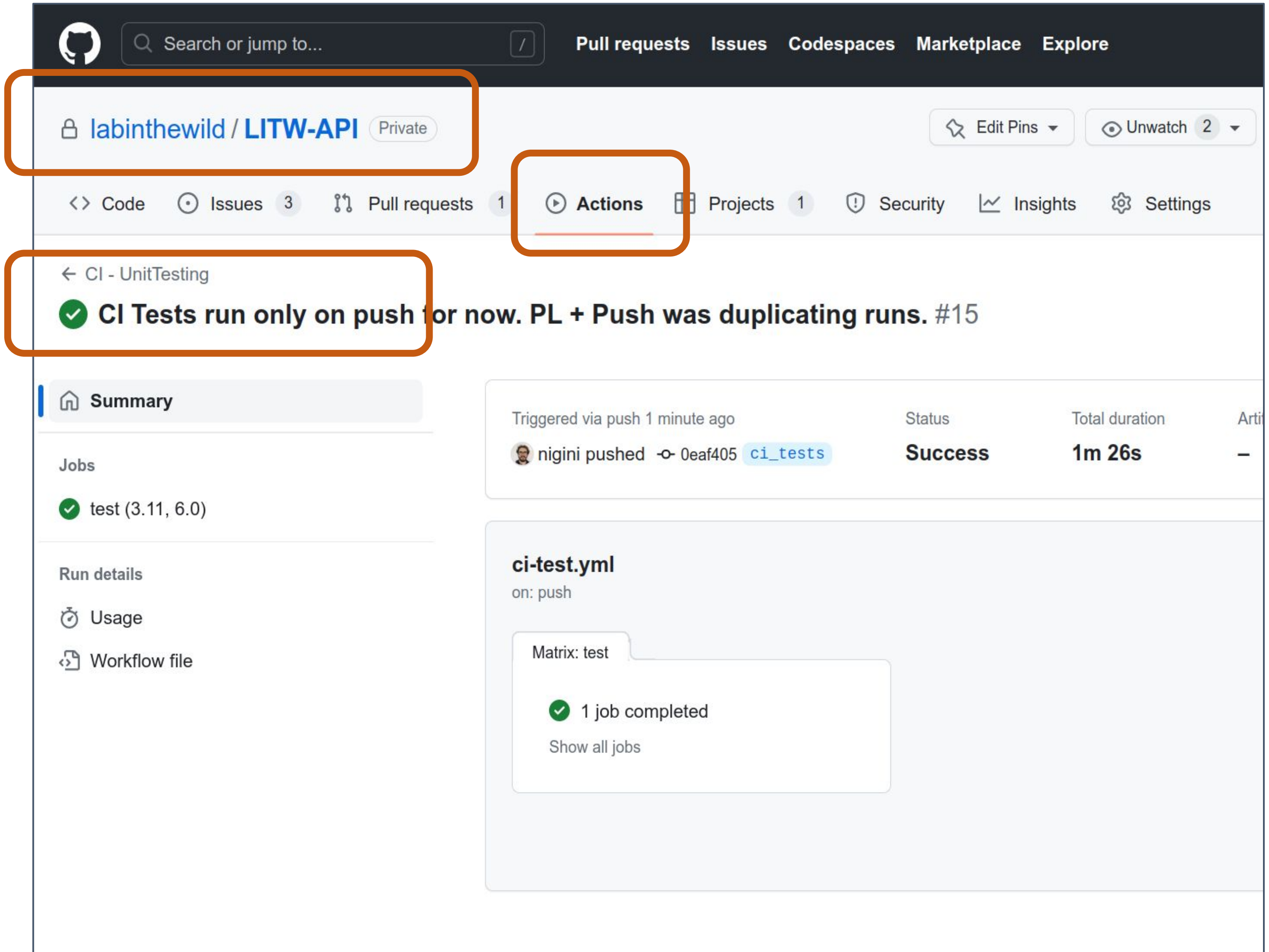
What's your personality?

You will learn about the five main traits of your personality and how you score on them. We will also try to establish the relationship between personality and physical activity goals.

Can you tell the nutritional content of a plate?

Take this study to see if you can accurately tell the nutritional content of a plate. See if you are more accurate than the average! An AI assistant will help you along the way.

Example: CI with Github actions



labinthewild / LITW-API Private

Code Issues 3 Pull requests 1 Actions Projects 1 Security Insights Settings

← CI - UnitTesting

✓ CI Tests run only on push for now. PL + Push was duplicating runs. #15

Summary

Jobs

✓ test (3.11, 6.0)

Run details

Usage

Workflow file

Triggered via push 1 minute ago

nigini pushed 0eaf405 ci_tests

Status

Success

Total duration

1m 26s

ci-test.yml

on: push

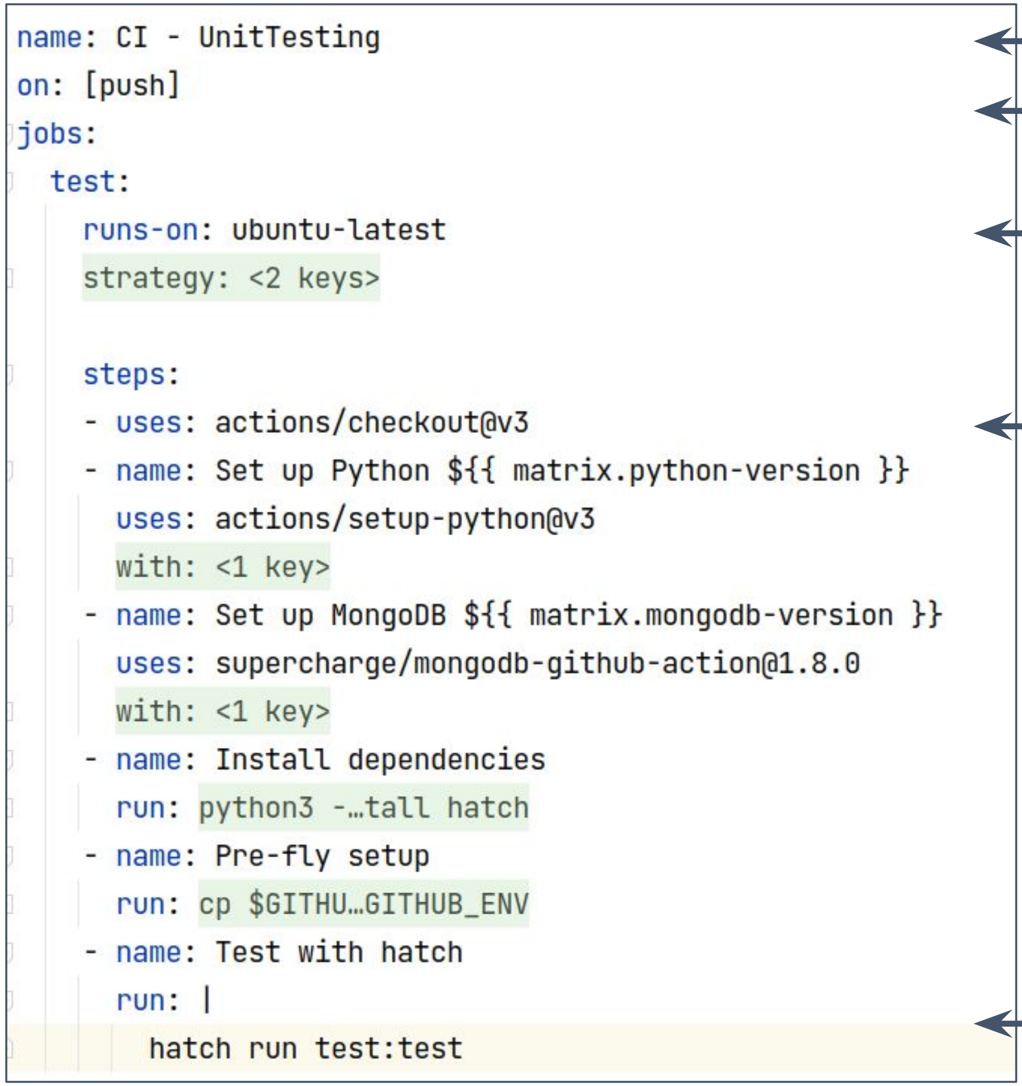
Matrix: test

✓ 1 job completed

Show all jobs

Example: CI with Github actions

Unit tests are triggered on every push of new code



```
name: CI - UnitTesting
on: [push]
jobs:
  test:
    runs-on: ubuntu-latest
    strategy: <2 keys>

    steps:
    - uses: actions/checkout@v3
    - name: Set up Python ${{ matrix.python-version }}
      uses: actions/setup-python@v3
      with: <1 key>
    - name: Set up MongoDB ${{ matrix.mongodb-version }}
      uses: supercharge/mongodb-github-action@1.8.0
      with: <1 key>
    - name: Install dependencies
      run: python3 -…tall hatch
    - name: Pre-fly setup
      run: cp $GITHU…GITHUB_ENV
    - name: Test with hatch
      run: |
        hatch run test:test
```

Workflow name

Trigger

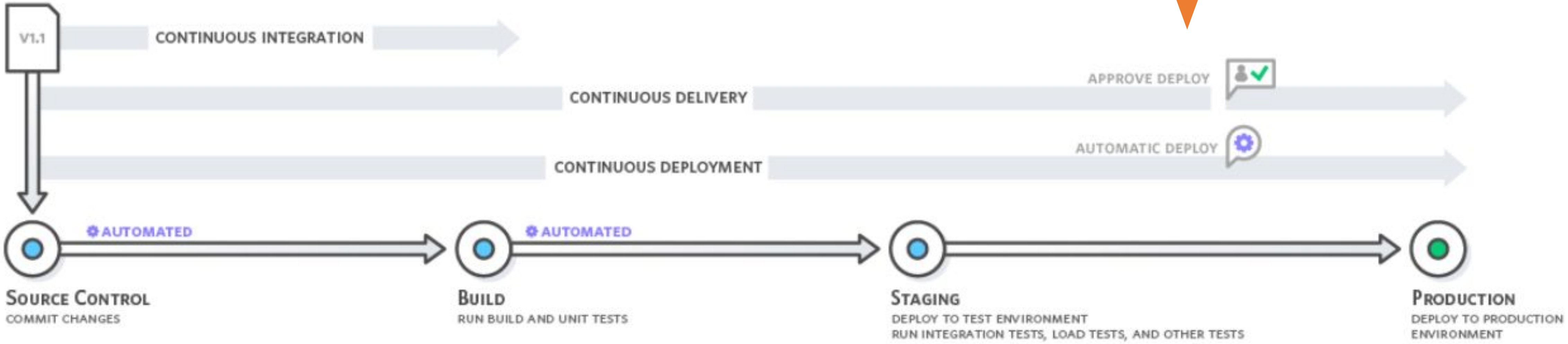
Linux OS environment

Code reuse with established “actions”

One command to run test suite

Continuous delivery/deployment basics

Why would you not always automatically deploy?

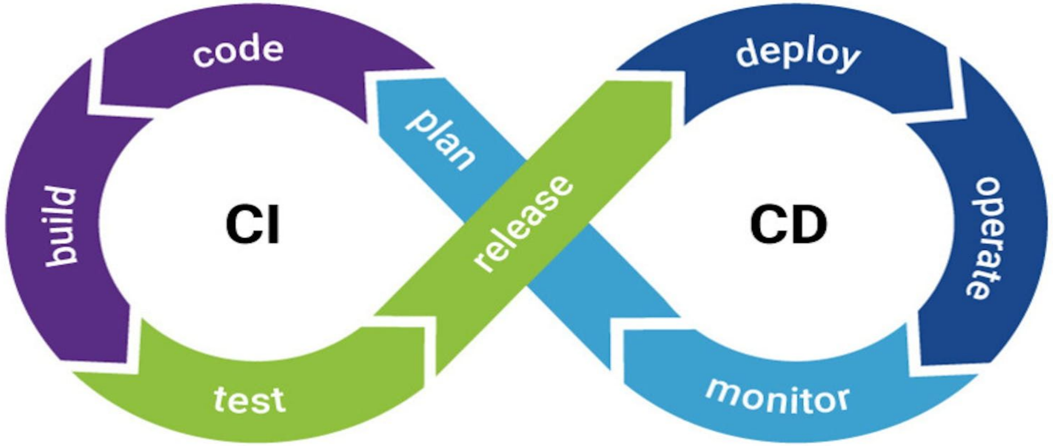


Staging before Production is very typical of industry practices

What is Continuous Delivery? – **Amazon Web Services**

Build & CI – Remember these best practices

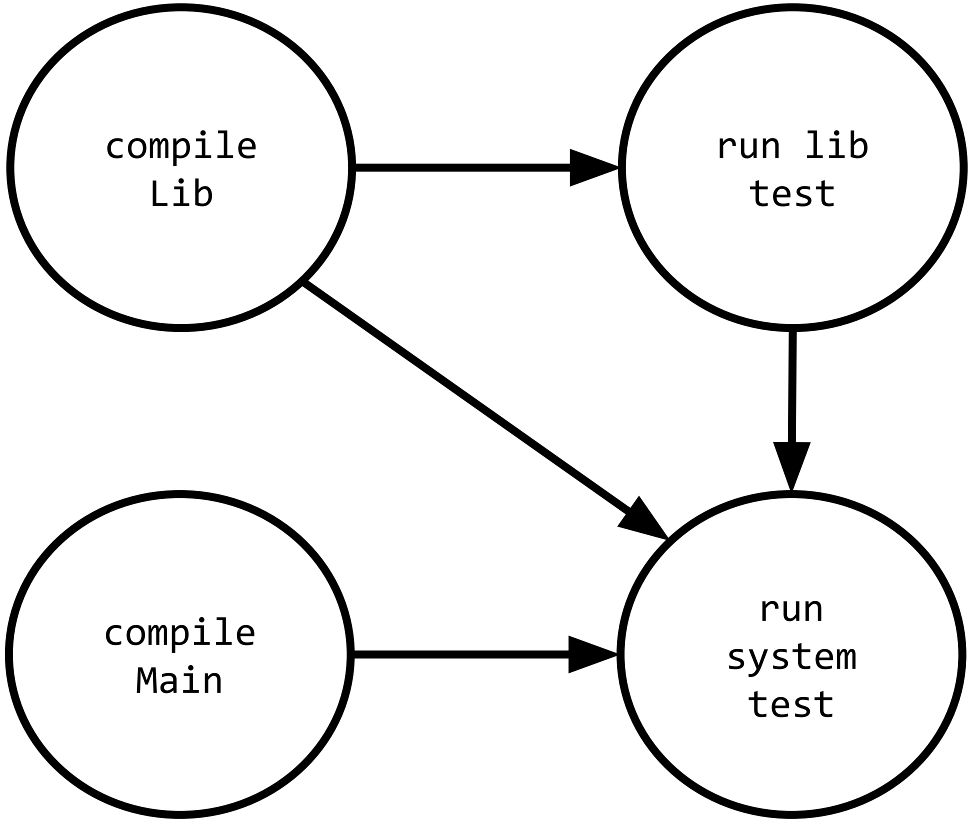
- Automate everything!
- Always use a build tool (one-step build)
- Use CI to build and test your code on every commit
- Don't depend on anything that's not in the build file (hermetic)
- Don't break the build!



Appendix – Topological sort example

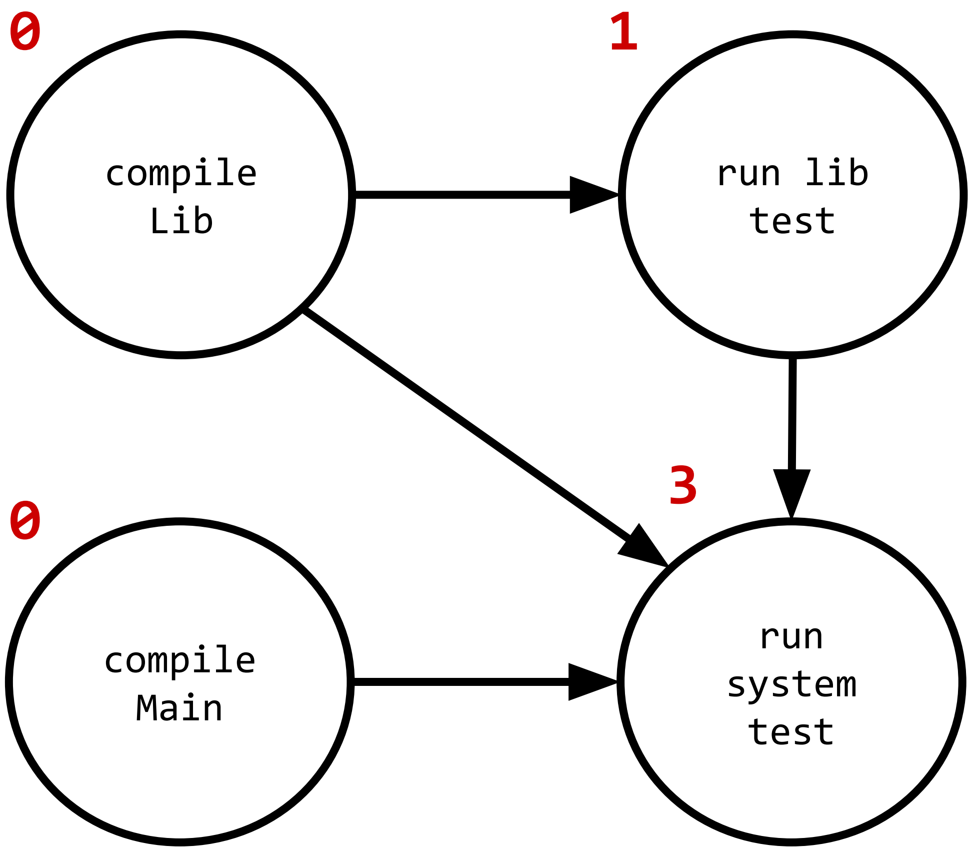
- Build tools use a topological sort to create an order to compiles
    - Order nodes such that all dependencies are satisfied
    - Implemented by computing indegree (number of incoming edges) for each node
    - No dependencies go first and open door to the others

Build systems: topological sort

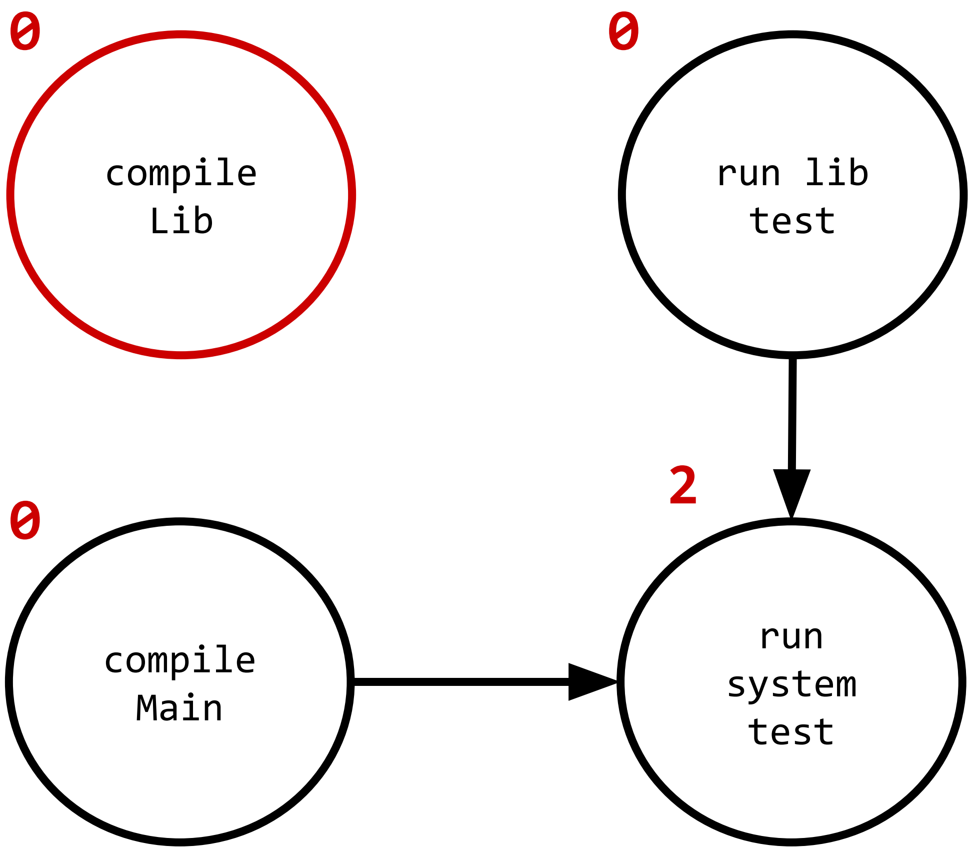


What’s the indegree of each node?

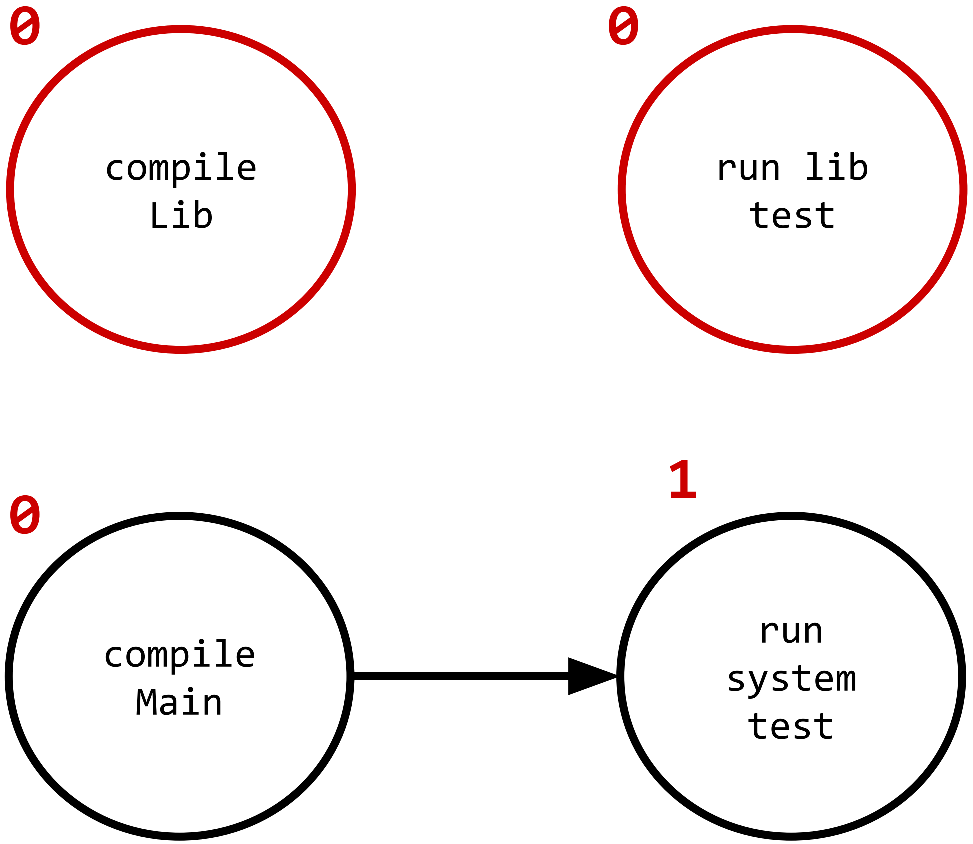
Build systems: topological sort



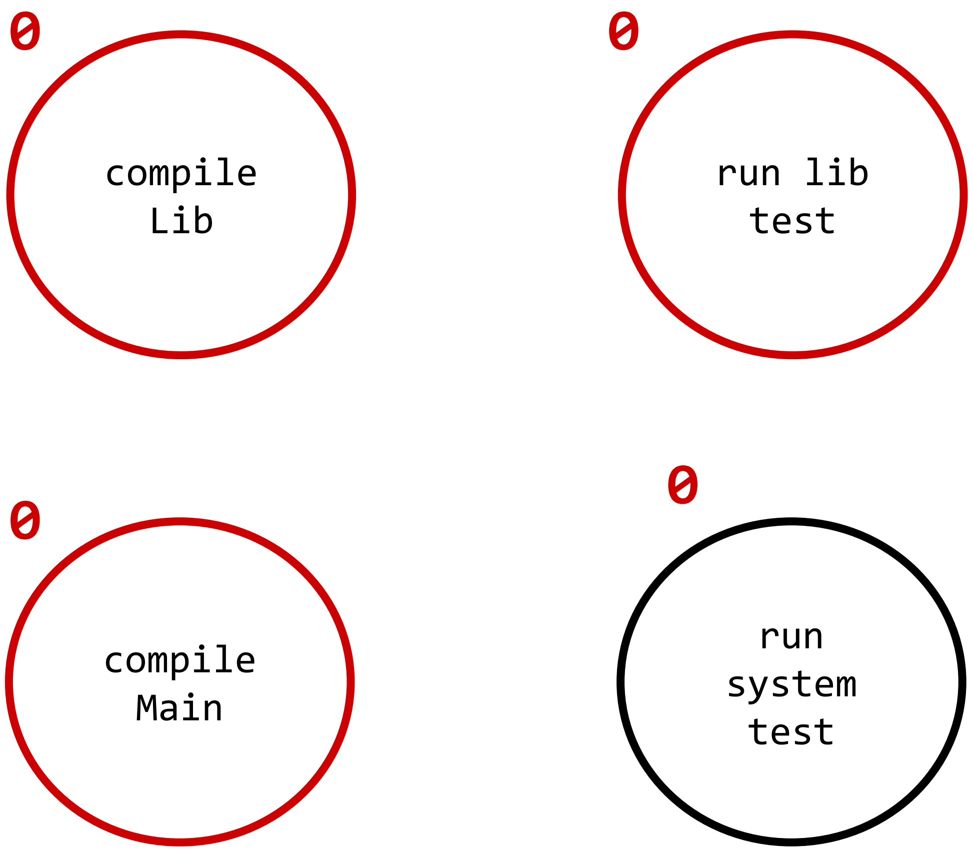
Build systems: topological sort



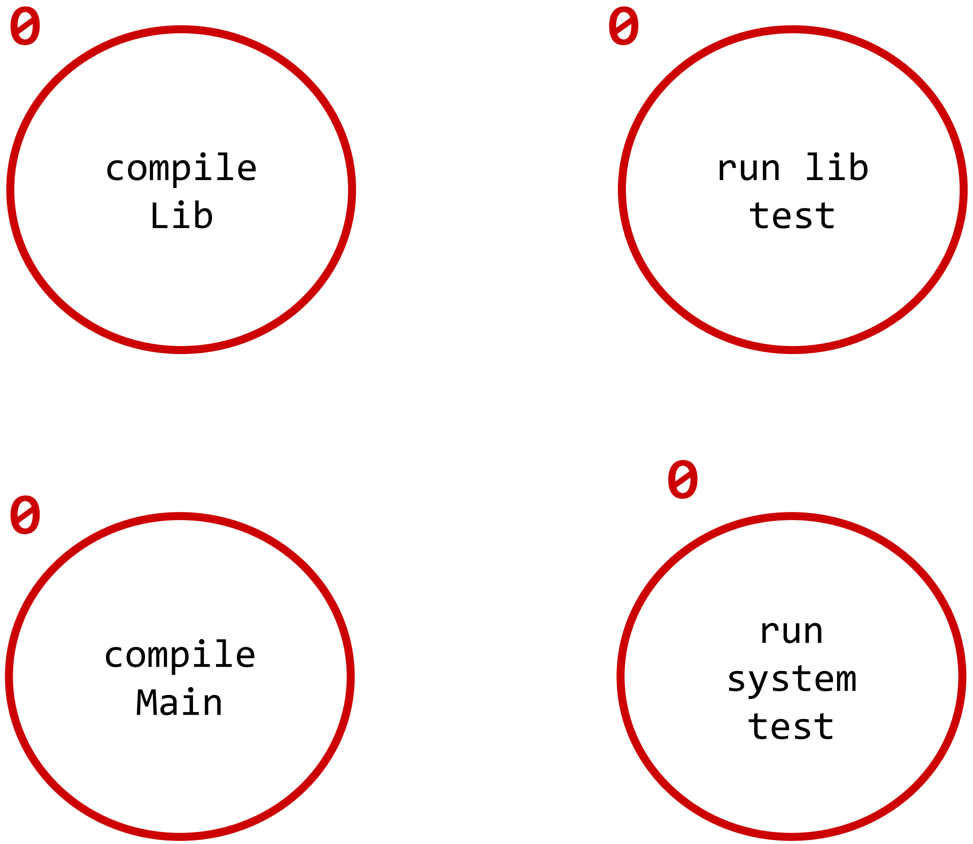
Build systems: topological sort



Build systems: topological sort



Build systems: topological sort



Build systems: topological sort

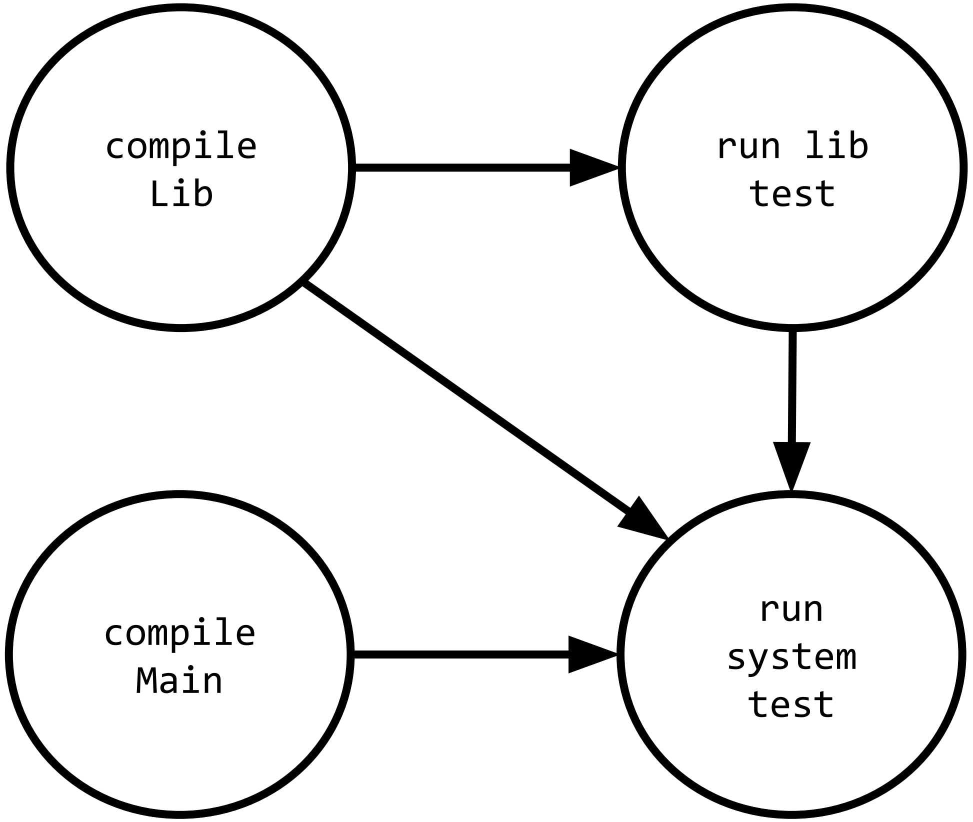
Valid sorts:

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

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

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

Which is preferable?


compile Lib
run lib test
compile Main
run system test