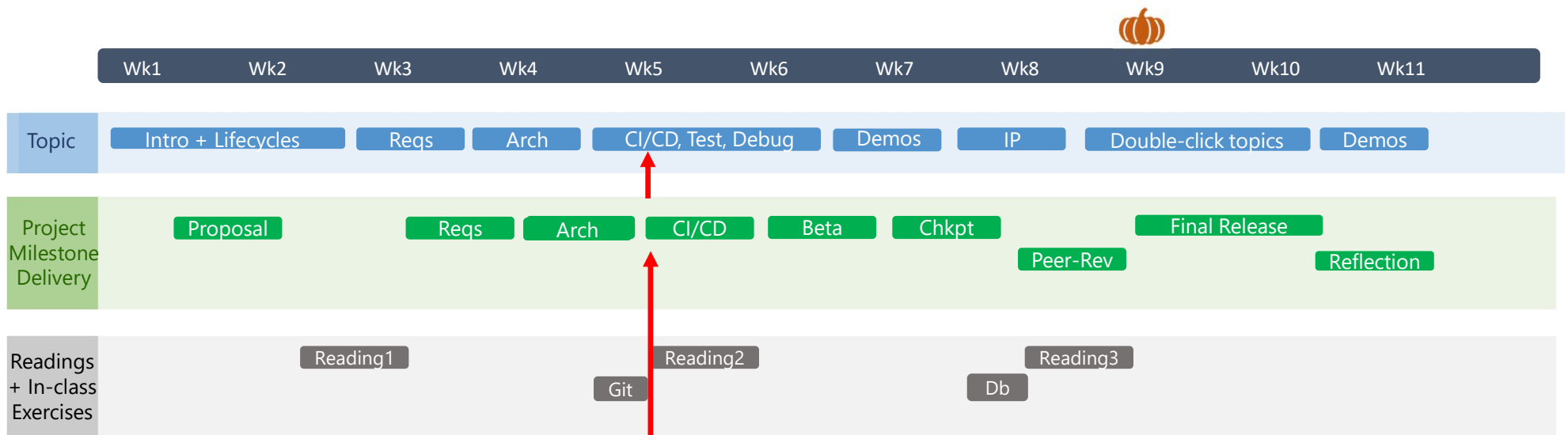


# CSE 403 Software Engineering

Build systems & Continuous Integration and  
Deployment

Autumn 2023

# We are moving through the SDLC components



# 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

Assignment 3:  
Git, Testing, and  
Continuous  
Integration  
+  
Reading Reflection 2

**Due 10/31**

# 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?

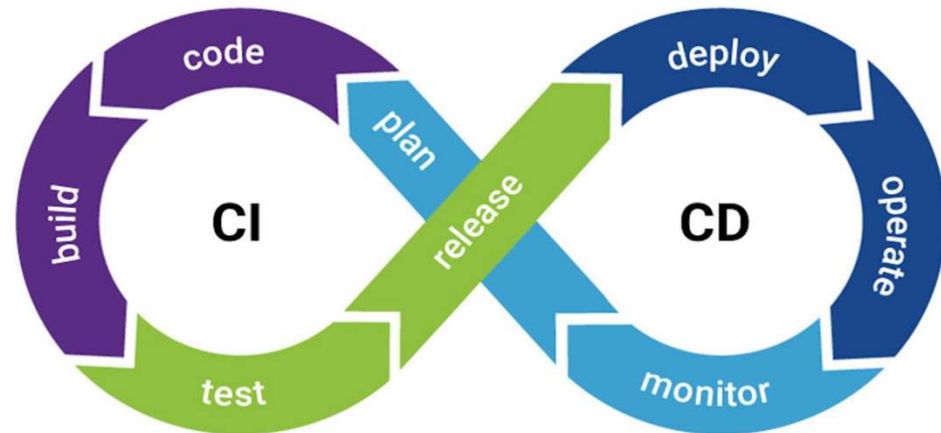
- Get the code
- Install dependencies
- Compile the code
- Run static analysis
- Generate documentation
- Run tests
- Create artifacts for customers
- Ship
- Operate, monitor, repeat

Which of these tasks should be handled manually?

**NONE!**

# Instead, orchestrate with a tool

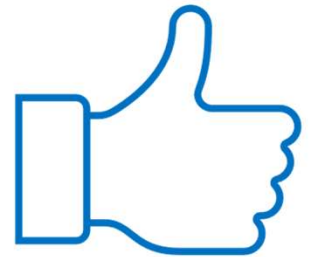
- **Build system**: a tool for automating compilation and related tasks
  - Is a component of a **continuous integration/deployment system** as today we automate more than just the build step of producing shippable software
- ✓ Get the source code
  - ✓ Install dependencies
  - ✓ Run static analysis
  - ✓ Compile the code
  - ✓ Generate documentation
  - ✓ Run tests
  - ✓ Create artifacts for customers
  - ✓ Ship!
  - ✓ **Operate, Monitor, Repeat**



# 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 to pick up the right dependencies
3. Only runs the compiles needed due to dependency changes

## 2. **Efficiency and reliability**

1. Automates the build process so that new and old team members, even working in different dev environments, can move quickly from development to shipping code
2. Eliminates the chance of missing steps due to tribal knowledge and/or simply errors

# Let's focus on dependency management

---

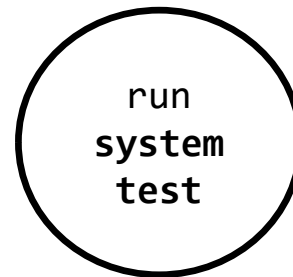
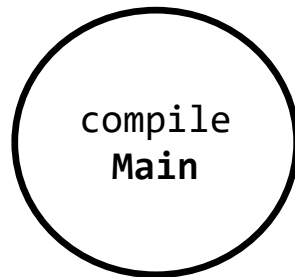
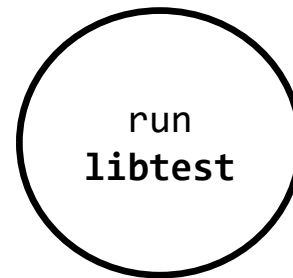
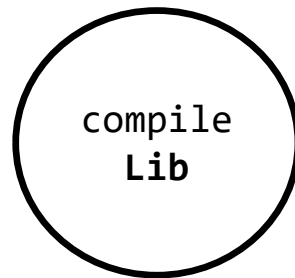
Simple example:

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

# Build systems: dependencies between tasks

---

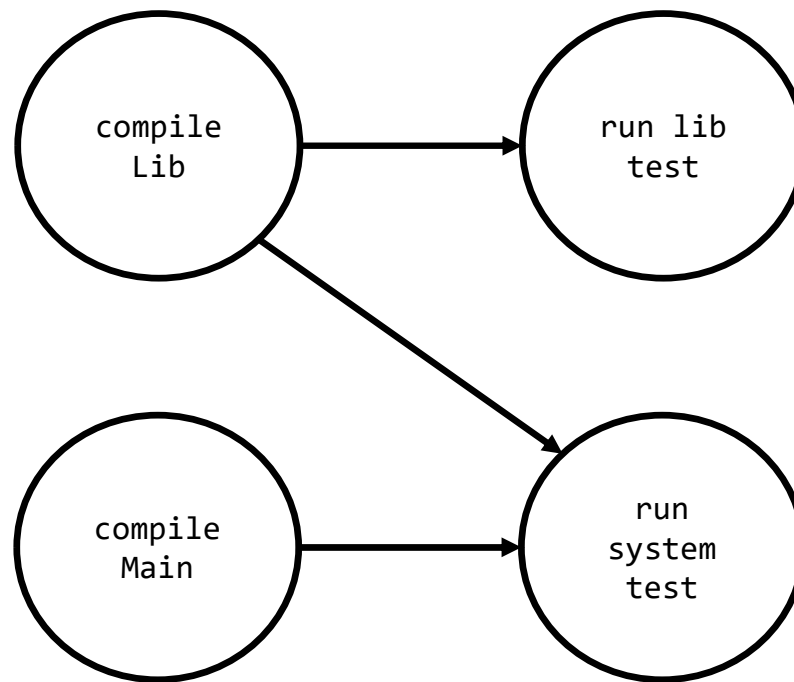
```
% ls src/  
Lib.java  
LibTest.java  
Main.java  
SystemTest.java
```



What are the dependencies between these tasks?  
And why do I care?

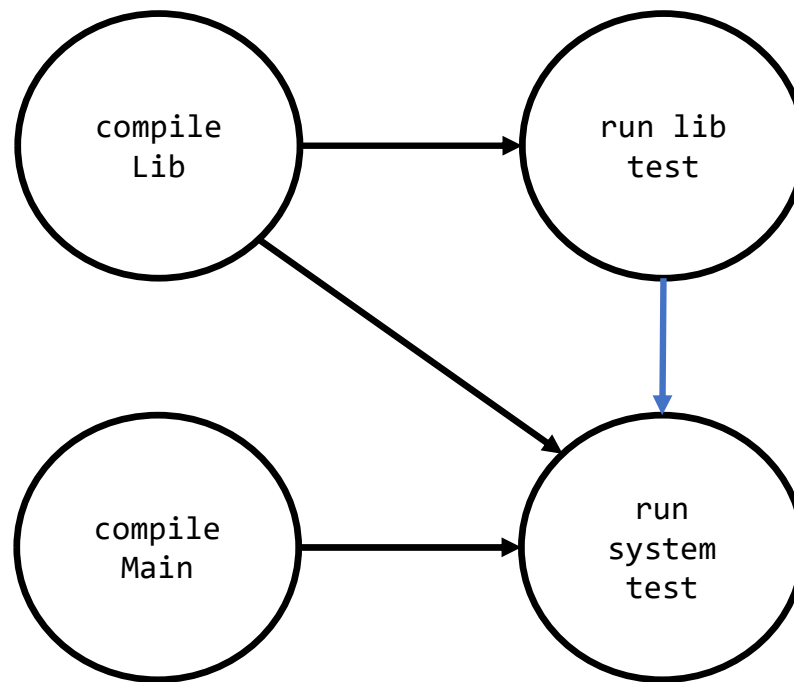
# Build systems: dependencies between tasks

---



# Build systems: dependencies between tasks

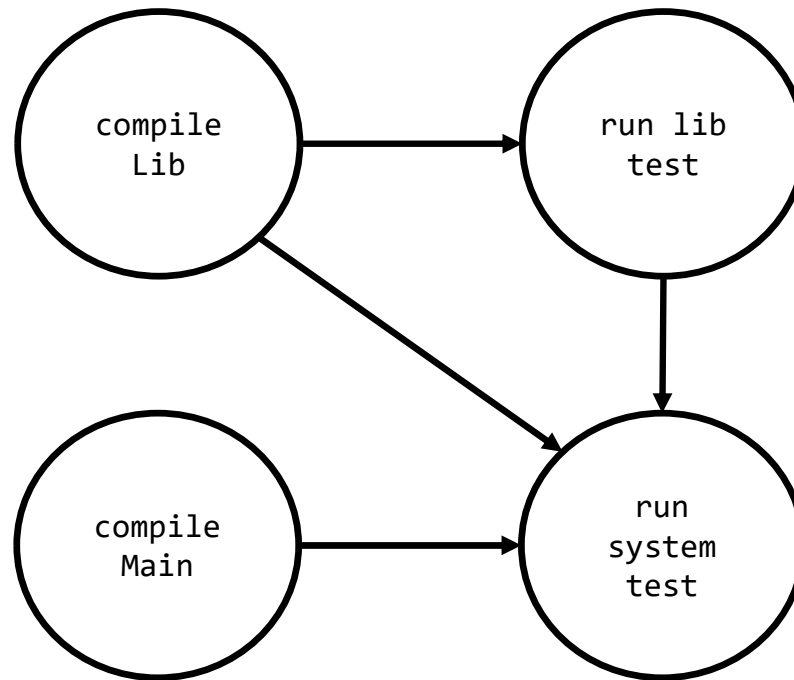
---



# Build systems: dependencies between tasks

---

In what order should we run these tasks?



# Build systems can determine task order

---

## **Large projects have thousands of tasks**

- Dependencies between tasks form a directed acyclic graph
- Build tools use a **topological sort** to create an order to compile
  - 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
  - See Appendix for example

## **External code (libraries) also can be complex**

- Build systems can manage these dependencies as well!

# Let's focus on efficiency and reliability

---

Actually, I think we understand these 😊

So, let's focus on the opportunity for **static analysis** BEFORE the compile step

Examples:

- Credential scan
- Date scan
- Sensitive data scan

What might be others?

Is this worthwhile?



# Build systems: opportunity for static analysis

github.com/Yelp/detect-secrets

☰ README.md

detect-secrets-ci failing pypi package 1.4.0 homebrew 1.4.0 PRs welcome

Donate Charity

## 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 build?

github.com/bearer/bearer

☰ README.md



# bearer

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.

# Here's an example build system 'input'

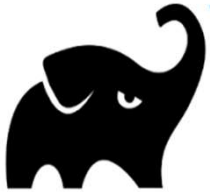
---

Basic-Stats  
"ant"  
**build.xml**

(from Monday's in-class  
exercise)

Simple-C  
"make"  
**Makefile**

# Assignment: evaluate and select a build system



Many  
other  
options!

Over to  
you to  
research



<b>JAVA+</b>		
	gradle	Open-source successor to <b>ant</b> and <b>maven</b>
	bazel	Open-source version of Google's internal build tool (blaze)
<b>PYTHON</b>		
	hatch	Implements standards from the Python standard (uses TOML files, has PIP integration)
	poetry	Packaging and dependence manager
	tox	Automate and standardize testing
<b>JAVASCRIPT</b>		
	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] production
- **Goal:** always have a deployment-ready build that has passed through a standardized testing process



# Just like build, there are many CI tool options



**Jenkins**



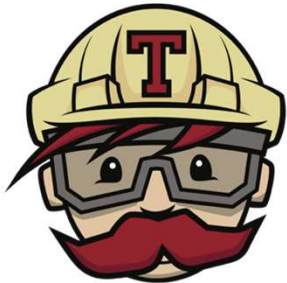
GitHub Actions



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

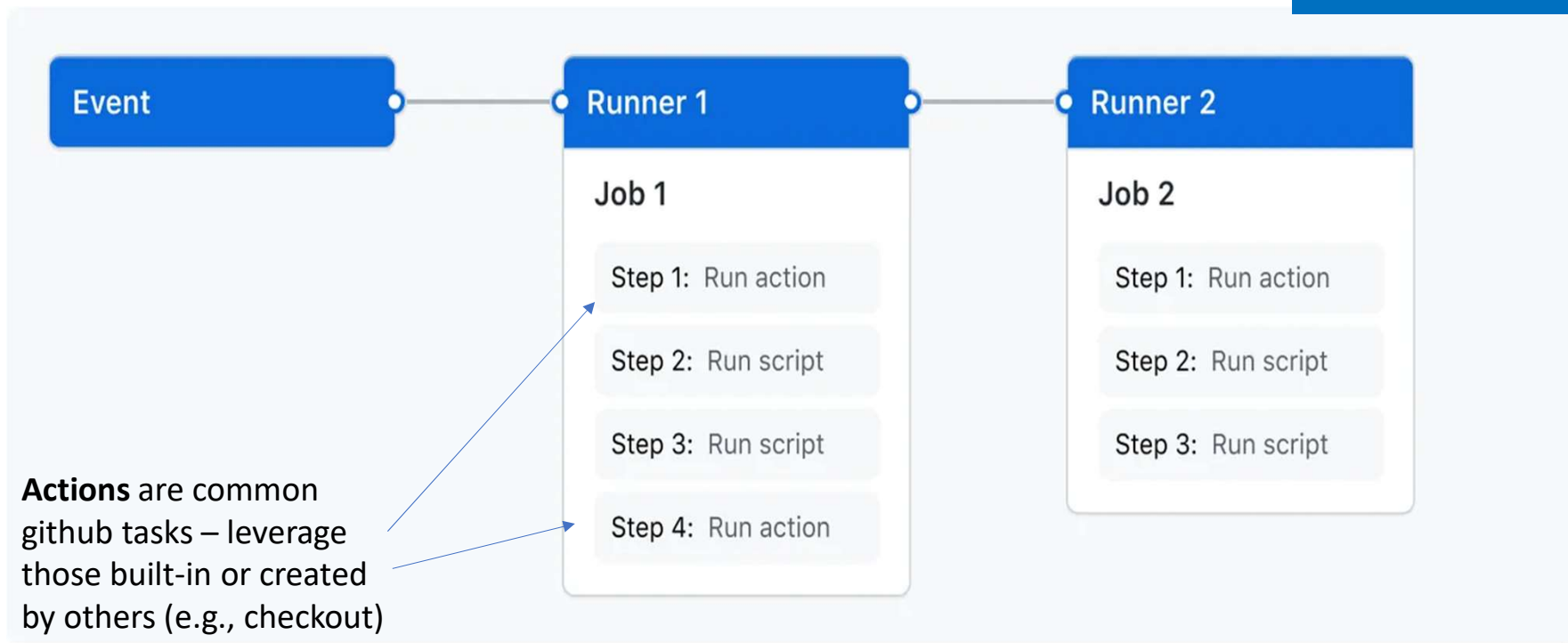


Using GitHub CI terminology but concepts span other CI systems

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

# CI basics (w/ GitHub CI)

What SW architecture is this using?



<https://docs.github.com/en/actions/learn-github-actions/understanding-github-actions>



# Let's try writing our own simple workflow

---

Follow along at:

<https://github.com/alv880/UW-CSE403-Au23-Projects>

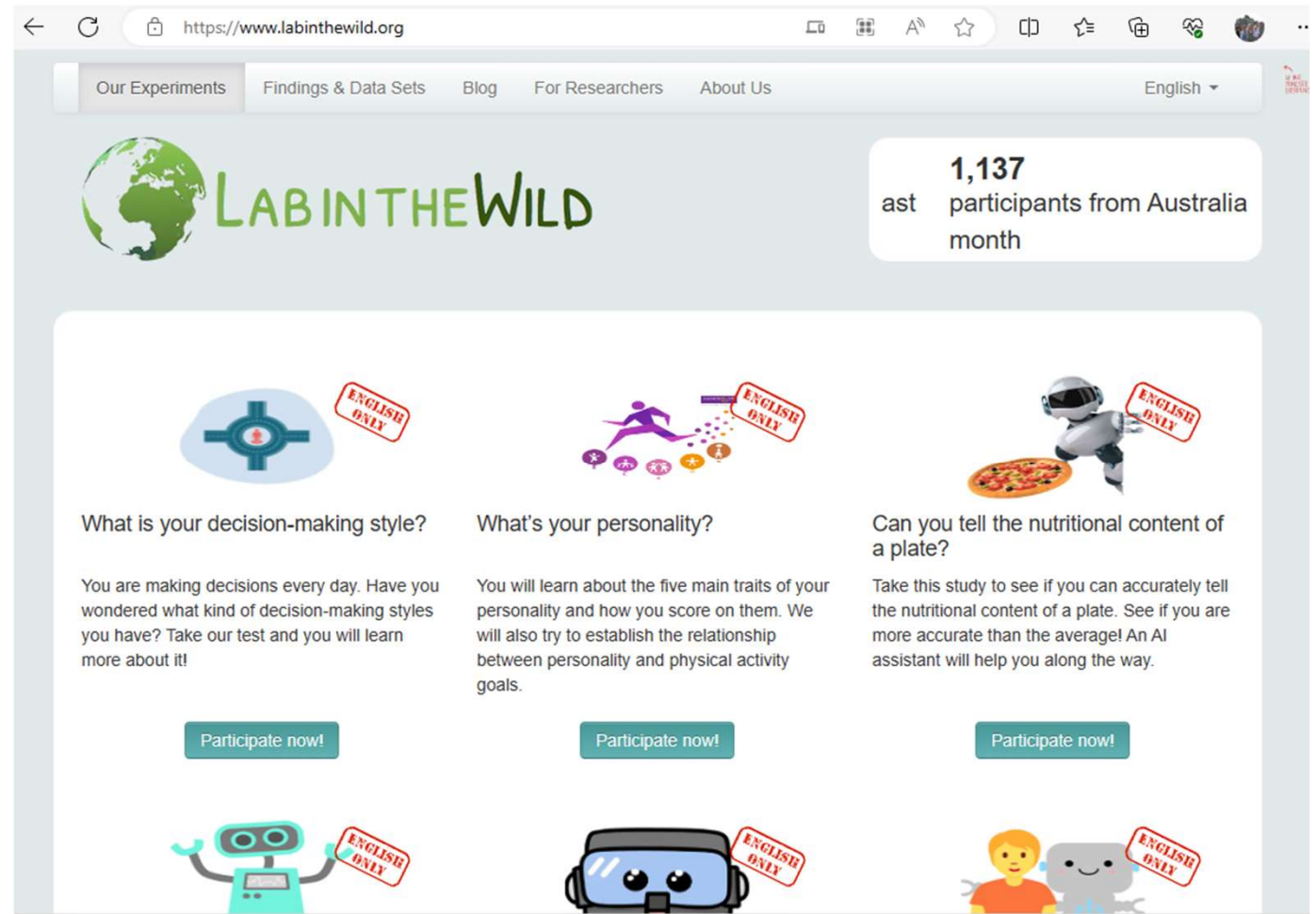
Nice light starter tutorial – Automation Step by Step:

<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 too provided this example



The screenshot shows the website <https://www.labinthewild.org>. The navigation bar includes "Our Experiments", "Findings & Data Sets", "Blog", "For Researchers", and "About Us". The language is set to "English".

The main header features the "LAB IN THE WILD" logo with a globe icon. A statistics box indicates "1,137 participants from Australia month".

Three experiment cards are displayed:

- What is your decision-making style?**: Includes an icon of a crosshair with a red dot and a red "ENGLISH ONLY" stamp. Description: "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!". Button: "Participate now!".
- What's your personality?**: Includes an icon of a person running with stars and a red "ENGLISH ONLY" stamp. Description: "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.". Button: "Participate now!".
- Can you tell the nutritional content of a plate?**: Includes an icon of a robot holding a pizza and a red "ENGLISH ONLY" stamp. Description: "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.". Button: "Participate now!".

At the bottom, there are three more icons with "ENGLISH ONLY" stamps: a robot, a car, and a person with a robot.

# Example: CI with Github actions

The screenshot shows the GitHub Actions interface for a repository named 'labinthewild / LITW-API'. The 'Actions' tab is selected, showing a workflow run titled 'CI Tests run only on push for now. PL + Push was duplicating runs. #15'. The run is in a 'Success' state and was triggered by a push from user 'nigini' 1 minute ago. The workflow file 'ci-test.yml' is shown, with a matrix job 'test' that has completed successfully. The interface includes a sidebar with navigation options like 'Code', 'Issues', 'Pull requests', and 'Actions', and a main content area with a summary and run details.

Repository: labinthewild / LITW-API (Private)

Workflow: CI - UnitTesting

Run Title: 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

Actor	Branch	Job	Status	Total duration	Artifacts
nigini pushed	0eaf405	ci_tests	Success	1m 26s	-

ci-test.yml

on: push

Matrix: test

- 1 job completed

Show all jobs

Unit tests are triggered on every push of new code

# Example: CI with Github actions

```
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 -m pip install hatch
      - name: Pre-fly setup
        run: cp $GITHUB...GITHUB_ENV
      - name: Test with hatch
        run: |
          hatch run test:test
```

← Workflow name

← Trigger

← Linux OS environment

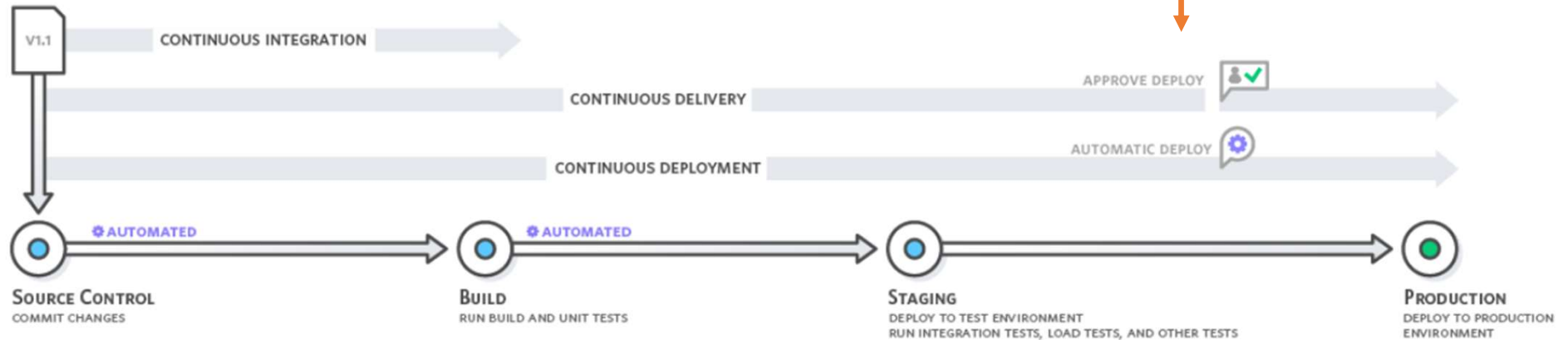
← Code reuse with established “actions”

← One command to run test suite

123

# Continuous delivery/deployment basics

Why would you not always automatically deploy?

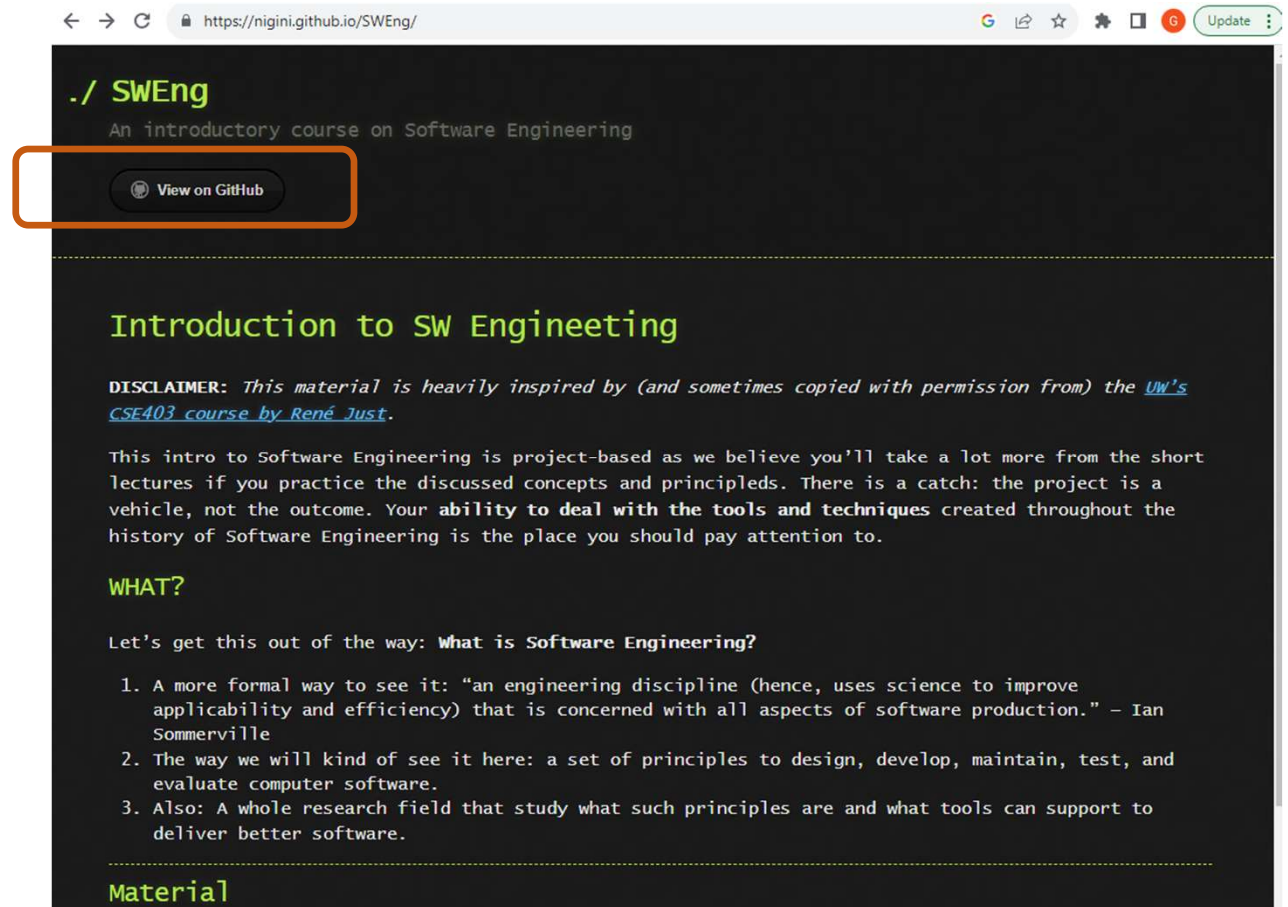


Staging before Production is very typical of industry practices

# Example: CD with GitHub Pages

Spring '23 class hosted their 403 class website on GitHub pages

Used CD so that updates triggered publishing the website update



# Example: CD configuration

The screenshot shows the GitHub repository settings for 'nigini / SWEng' (Public). The repository name is highlighted with an orange box. The navigation bar includes links for Code, Issues (4), Pull requests (9), Actions, Projects, Wiki, Security, Insights, and Settings (highlighted with an orange box). The left sidebar lists various settings categories: General, Access (Collaborators, Moderation options), Code and automation (Branches, Tags, Rules (Beta), Actions), Webhooks, Environments, Codespaces, and Pages (highlighted with an orange box). The main content area is titled 'GitHub Pages' and includes a description, a live site URL (<https://nigini.github.io/SWEng/>), and deployment settings. Under 'Build and deployment', the 'Source' is set to 'Deploy from a branch'. The 'Branch' is set to 'main' and the directory is set to '/ (root)'. A 'Save' button is visible. A link to 'Learn more' is provided for the current branch. A note at the bottom suggests adding a Jekyll theme.

# Example: CD configuration

The screenshot shows the GitHub Actions interface for a workflow named 'pages build and deployment #52'. The repository is 'nigini / SWEng' (Public). The 'Actions' tab is selected in the navigation bar. The workflow summary shows it was triggered via dynamic 2 days ago, with a status of 'Success', a total duration of 52s, and 1 artifact. The workflow steps are: 'build' (24s), 'report-build-status' (2s), and 'deploy' (7s). The 'deploy' step includes a link to the deployment page: <https://nigini.github.io/SWEng/>.

nigini / SWEng Public

<> Code Issues 4 Pull requests 9 **Actions** Projects Wiki Security Insights Settings

pages build and deployment #52

Summary

Jobs

- build
- report-build-status
- deploy

Run details

Usage

Triggered via	Status	Total duration	Artifacts
dynamic 2 days ago	Success	52s	1

pages-build-deployment  
on: dynamic

build 24s

report-build-status 2s

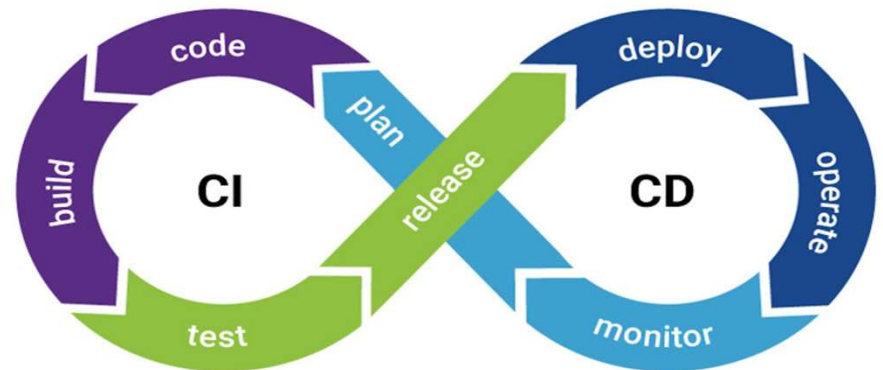
deploy 7s  
<https://nigini.github.io/SWEng/>



# Build, CI - Remember these best practices

---

- 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!

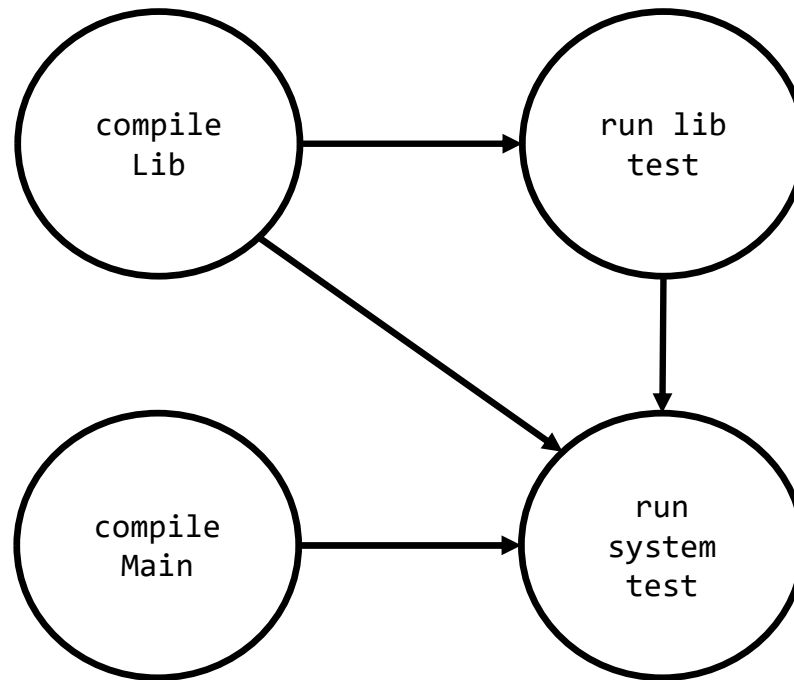


# Appendix - Topological sort example

---

## Build systems: topological sort

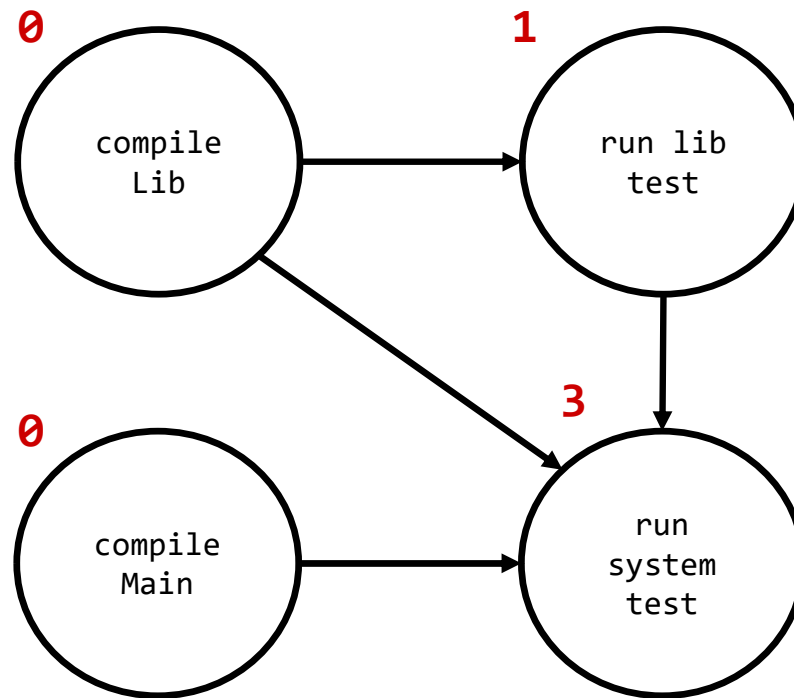
---



What's the indegree of each node?

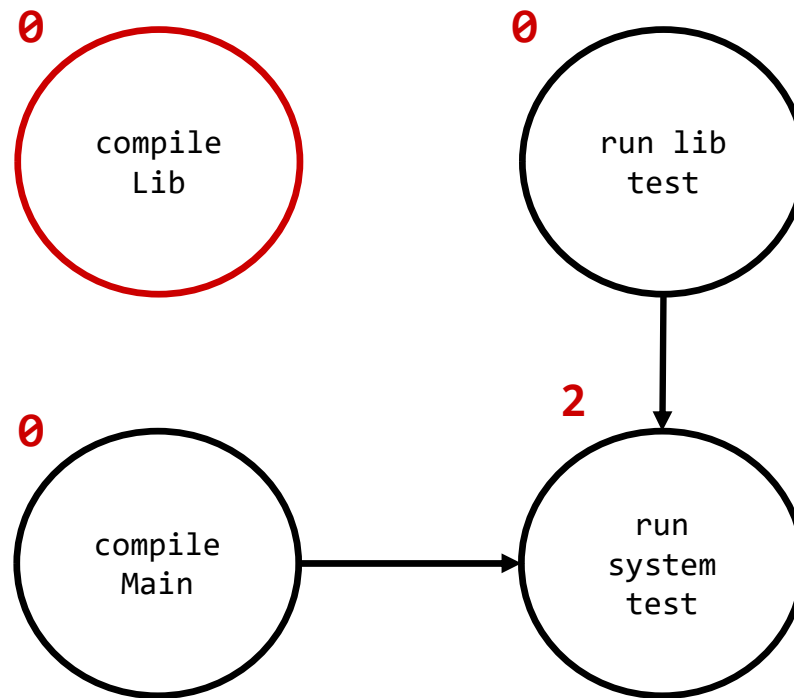
## Build systems: topological sort

---



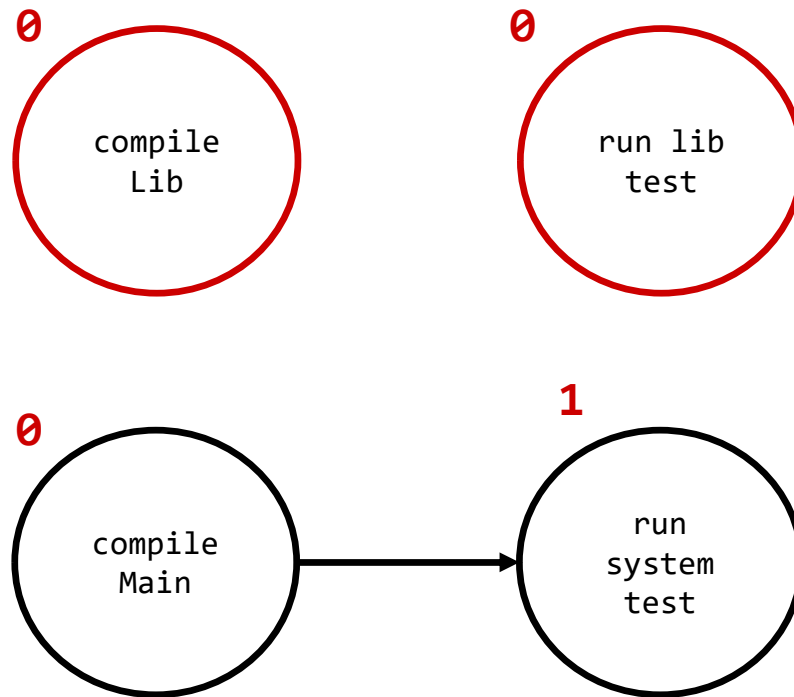
## Build systems: topological sort

---



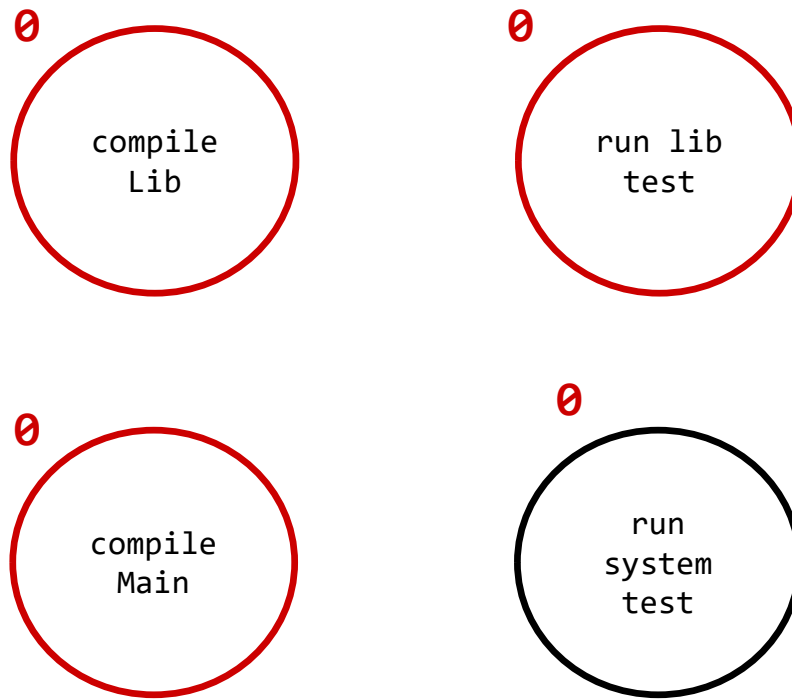
## Build systems: topological sort

---



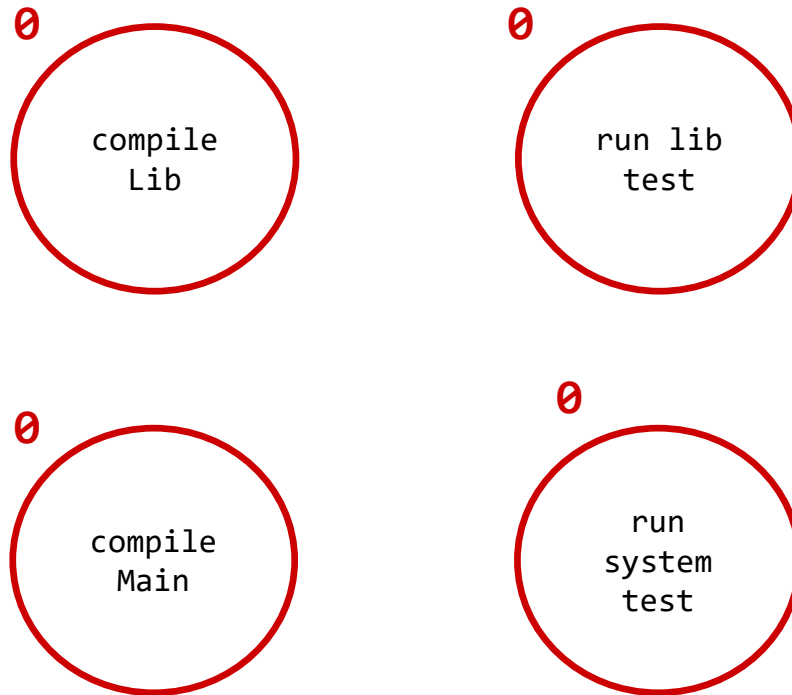
# Build systems: topological sort

---



## 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?

