CONTINUOUS INTEGRATION

G. Molines

2020-2021

Université Nice Sophia Antipolis

POLYTECH NICE-SOPHIA

INTRODUCTION

Goals

- What is hard?
- → integration
- Integrate early!

BUILD – ONE MACHINE

Building on one machine

Stages:

- Prepare files
- Compile source
- Compile tests
- Copy resources
- Package
- Invoke tests
- Check test results
- Generate test report

Example: in java:

Stages:

- Prepare files
- Compile source
- Compile tests
- Copy resources
- Package
- Invoke tests
- Check test results
- Generate test report

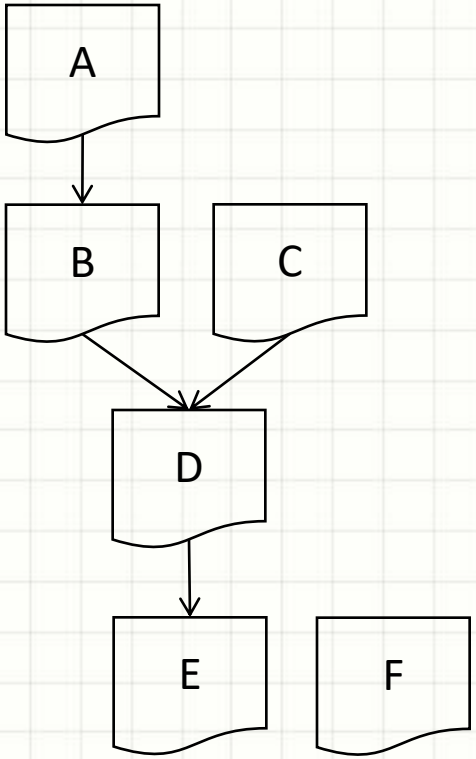
Tools:

- .
- javac
- javac
- cp
- jar
- Junit
- check errorlevel
- junitreport

Orchestration

- Shell script
  - Not portable
  - Redo everything, all the time
- Ant script
  - Portable
  - Can skip target
  - Becomes cumbersome on large projects
- Maven
  - Portable
  - Dependency management
  - Stage enforced

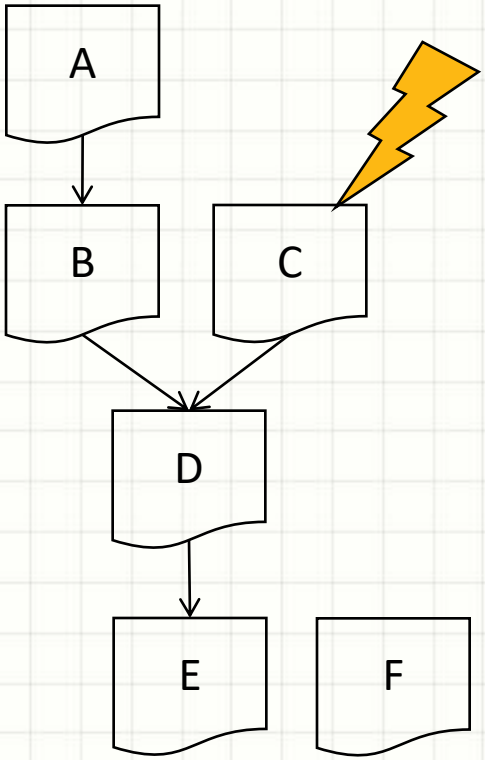
Module dependency graph



Orchestration:

- Build A
- Build B
- Build C
- Build D
- Buid E
- Build F

Code change



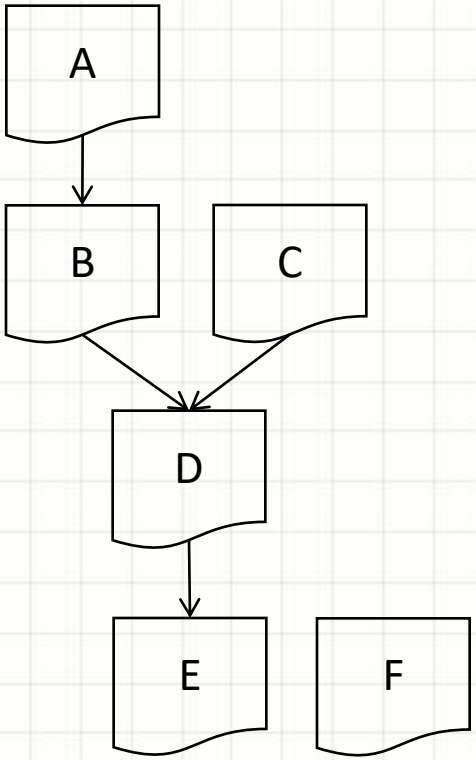
Orchestration:

- Build A
- Build B
- Build C
- Build D
- Buid E
- Build F

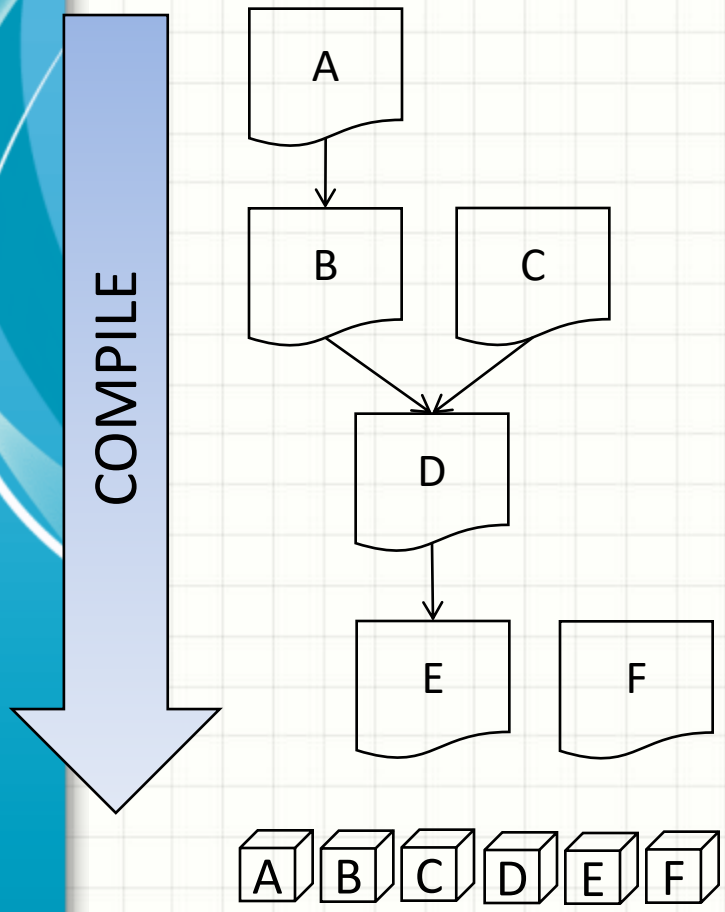
- How to be smarter?
  - Don’t rebuild A & B
  - → Make, ant
- Still, that’s a lot to build…
- We need more horsepower
- How about a build server?

Dedicated Build Server

Classical product build

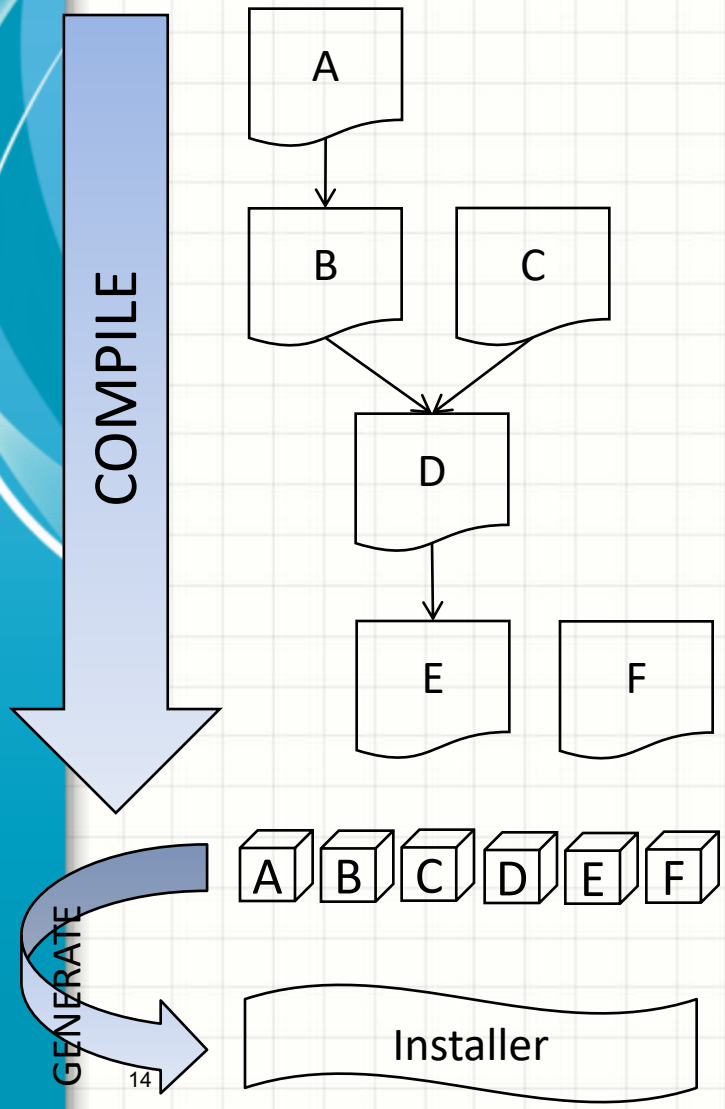


Classical product build

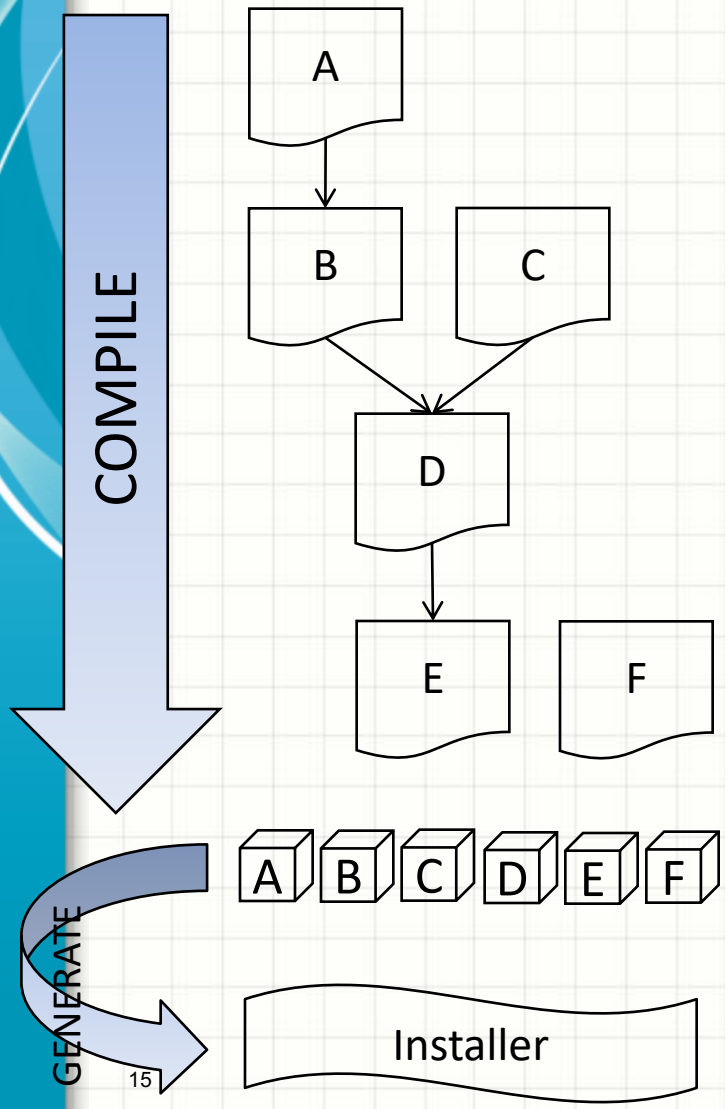


Note: compile really means compile + run unit-tests.
If a module's unit-tests fail, do we consider its « compilation » succesfull ?

Classical product build



Classical product build

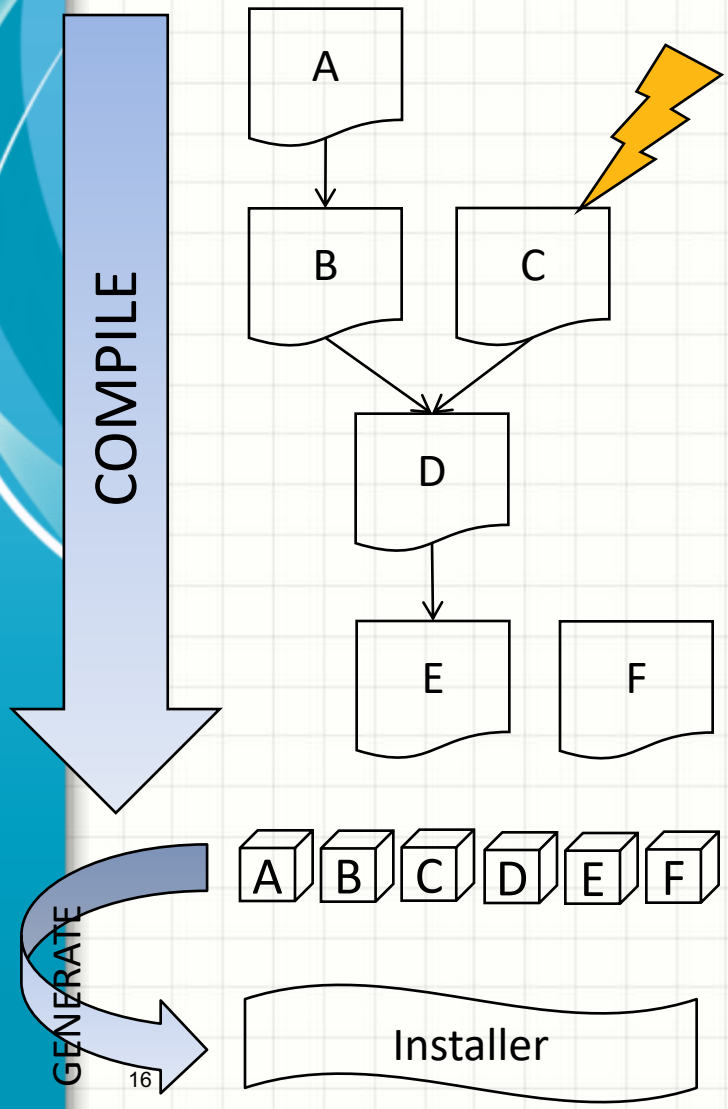


Automate

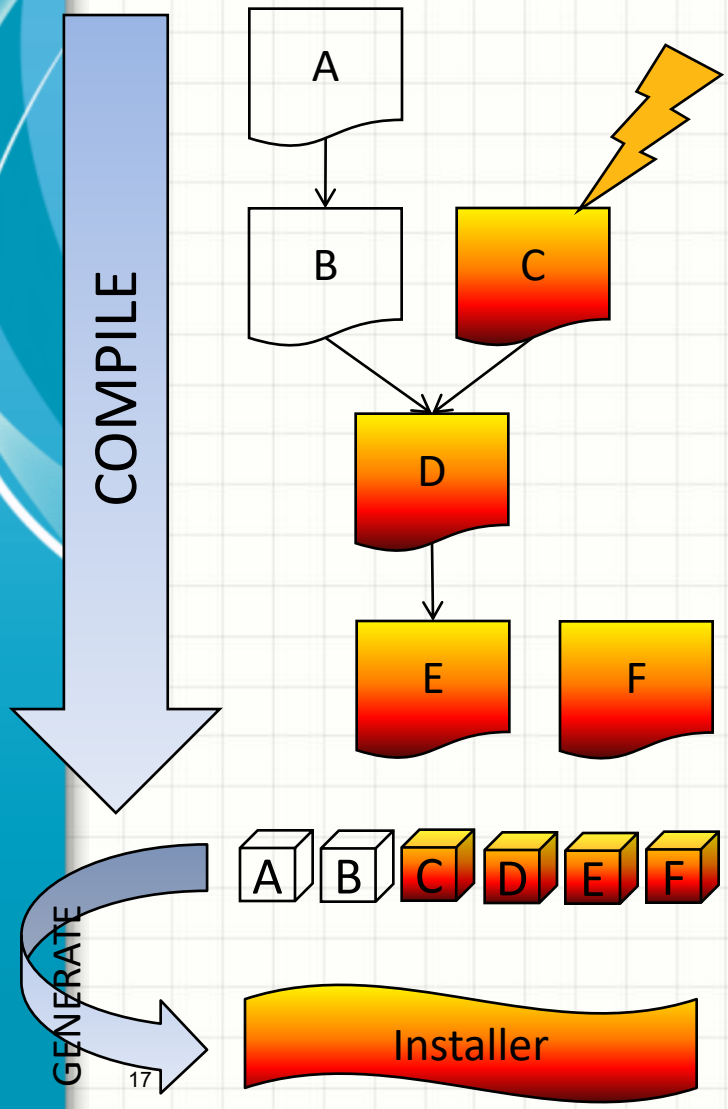
Repeat as fast as possible

The whole process can take hours (or days) !

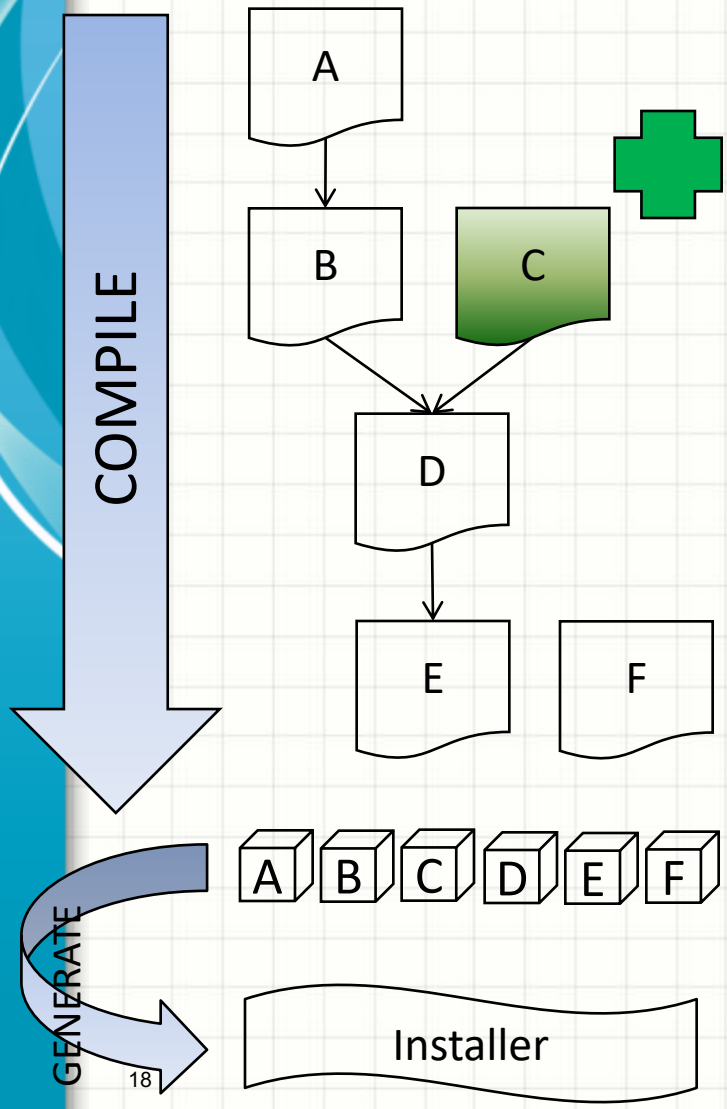
Classical product build



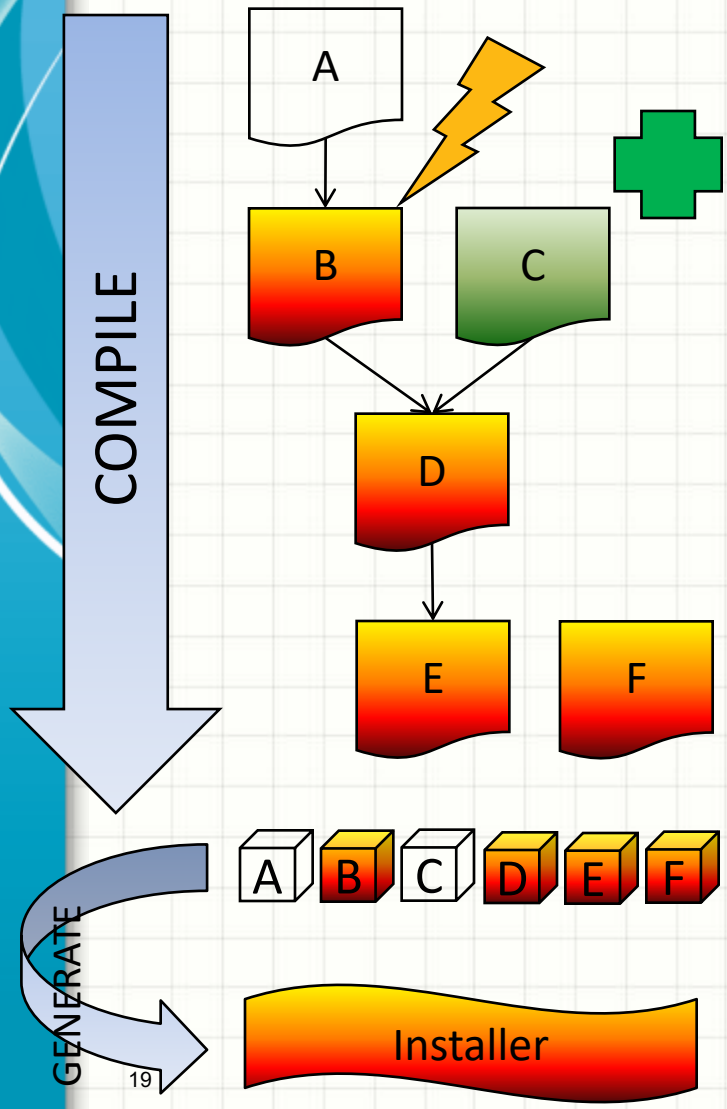
Classical product build



Classical product build



Classical product build



Issues

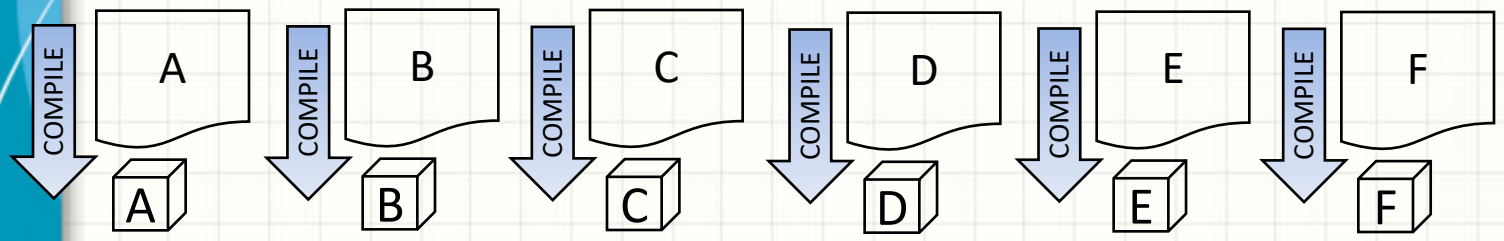
- Build is always broken -> **unreliable**
- Feedback takes days to come -> **slow**
- To ensure end of iteration builds succeed, several days of code freeze -> **unavailable**
- Adding a new module add hours to the build -> **not scalable**

What can we do?

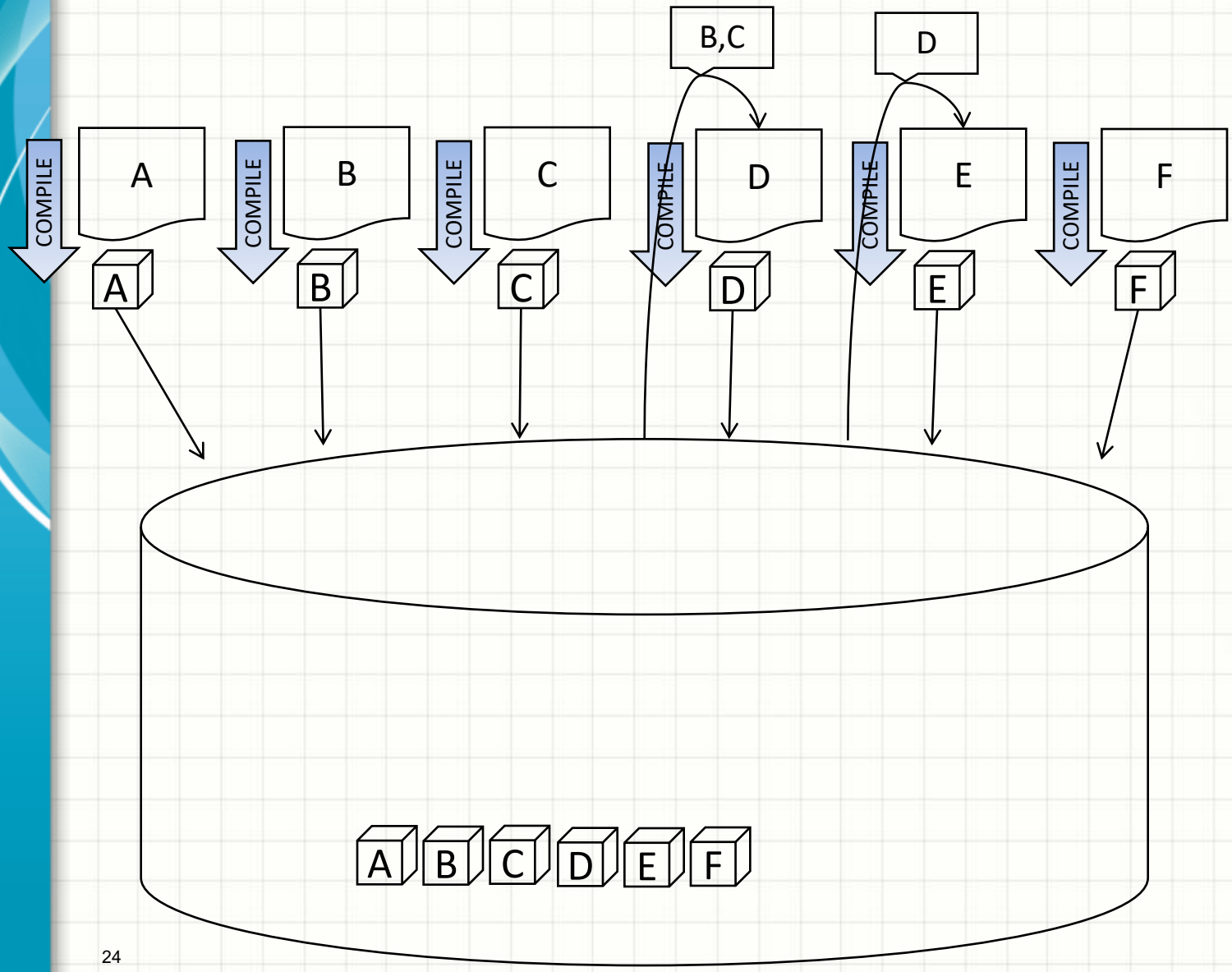
- Build in parallel
- Do not recompile unchanged component
  - By storing / reusing compiled artifacts

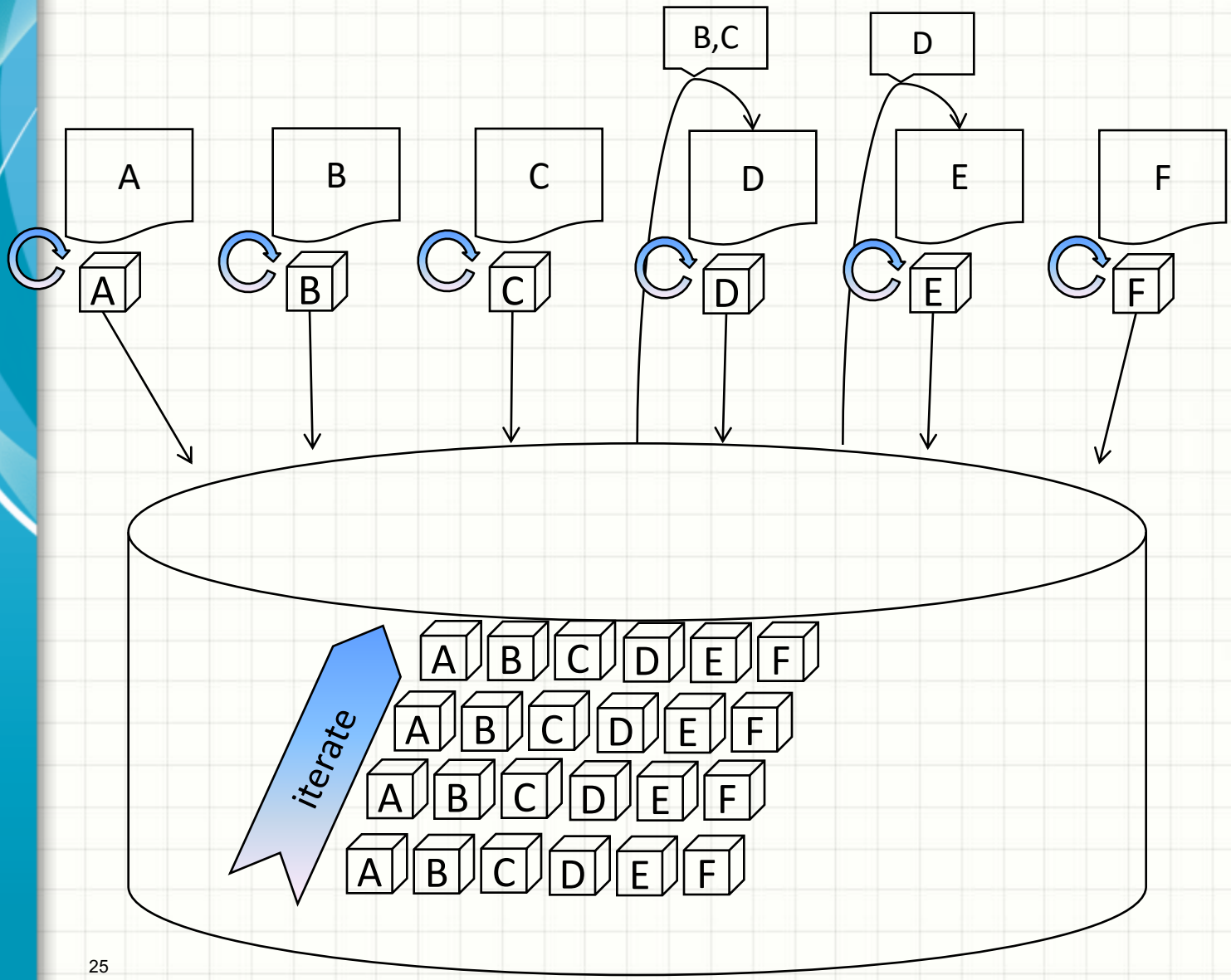
Artifact Repository

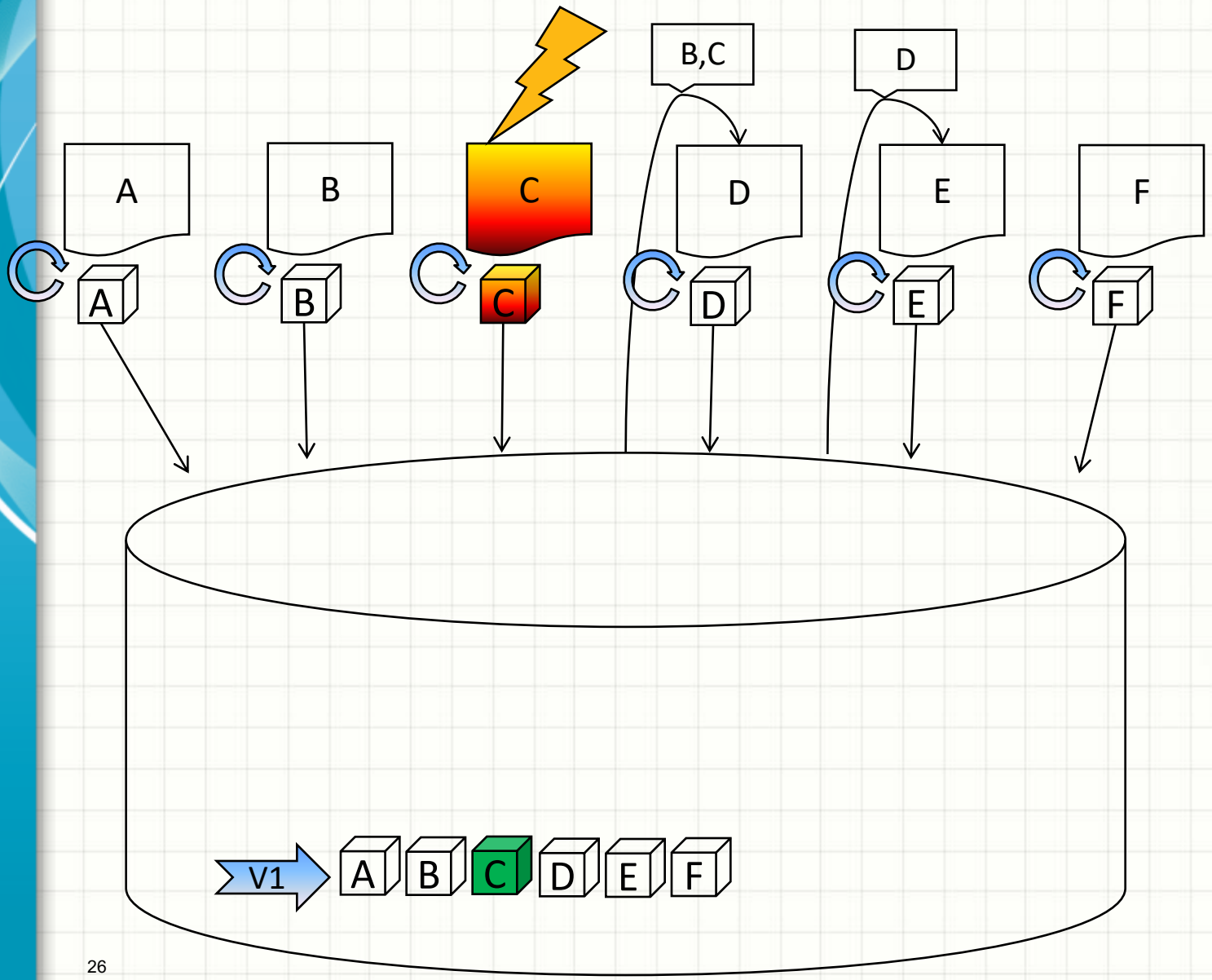
Continuous Integration

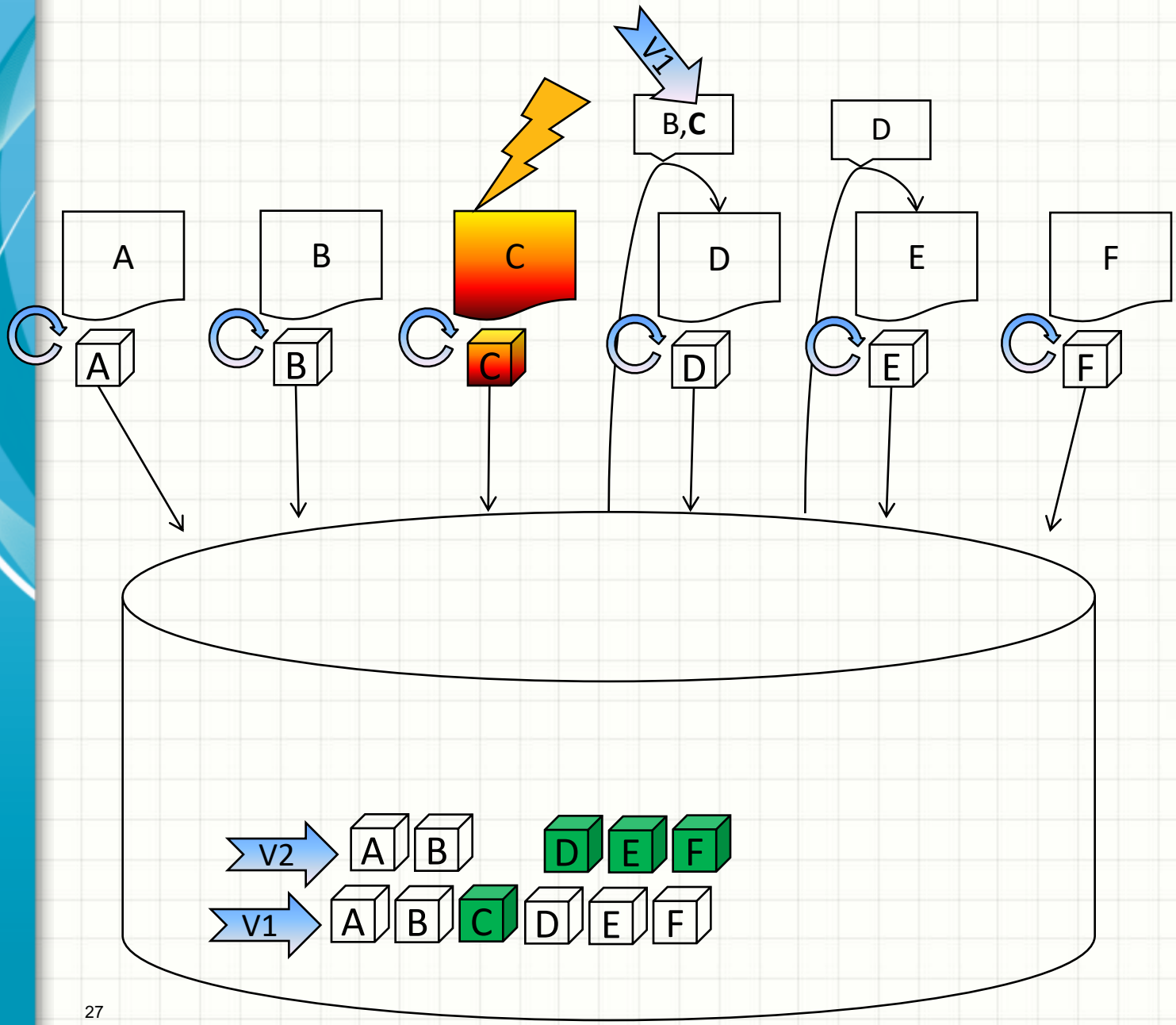


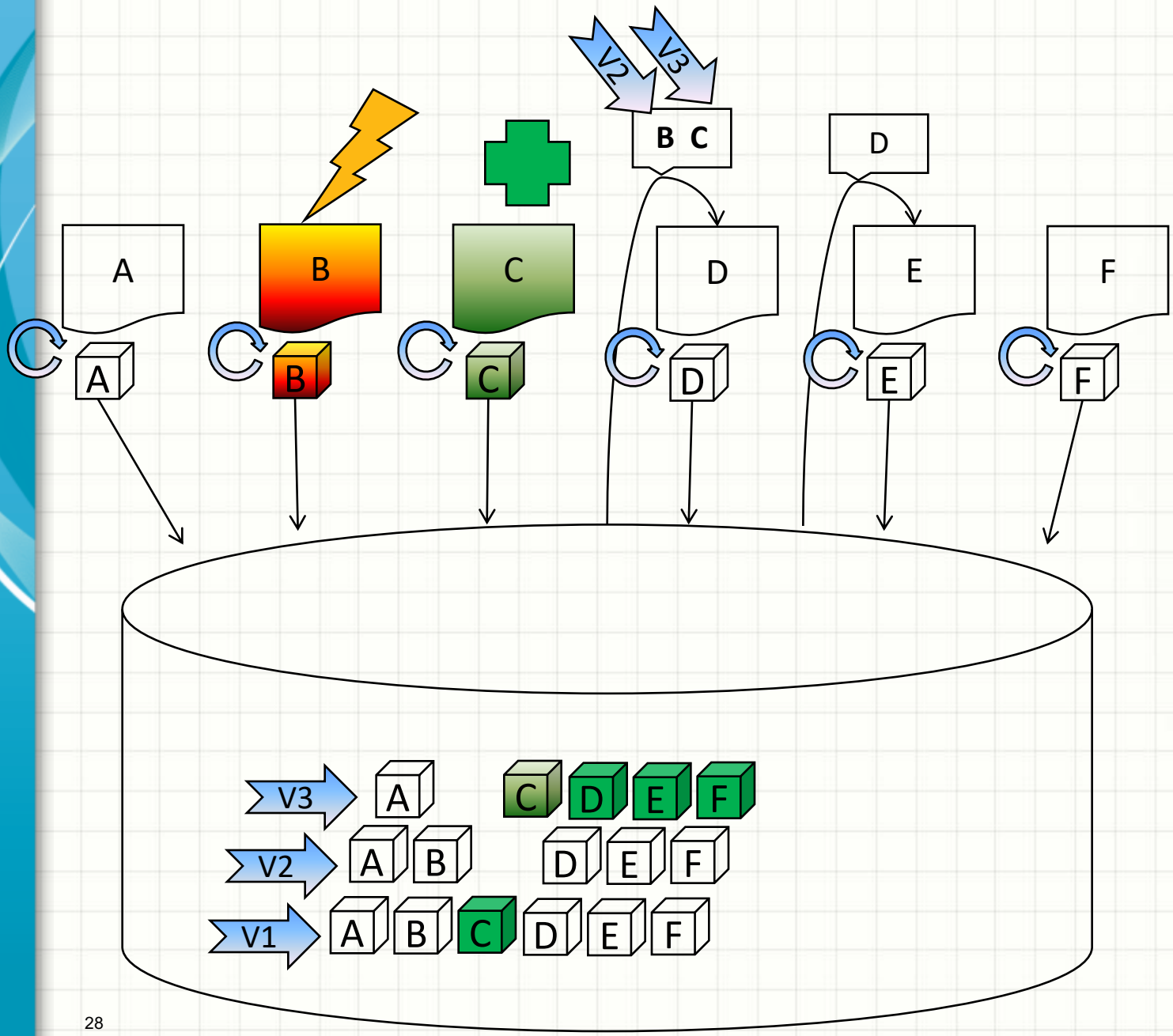
Note: compile really means compile + run unit-tests.
If a module's unit-tests fail, we consider its « compilation » as failed !

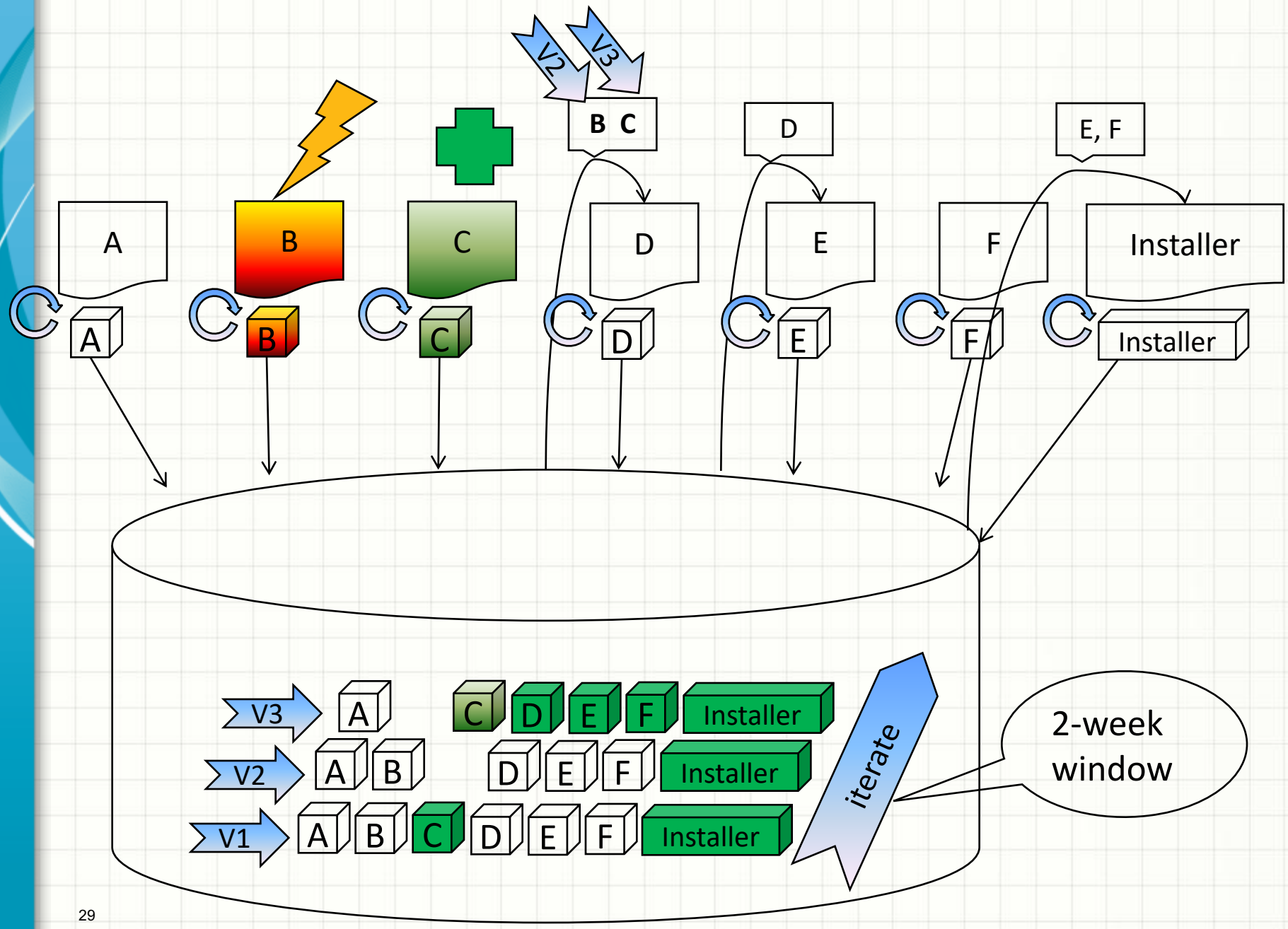

B,C
D
COMPILE
A
COMPILE
B
COMPILE
C
COMPILE
D
COMPILE
E
COMPILE
F
A
B
C
D
E
F
A
B
C
D
E
F


A
B
C
B,C
D
D
E
F
A
B
C
D
E
F
iterate
A B C D E F
A B C D E F
A B C D E F
A B C D E F


A
B
C
B,C
D
D
E
F
A
B
C
D
E
F
V1
A
B
C
D
E
F


V1
B,C
D
A
B
C
D
E
F
A
B
C
D
E
F
V2
A
B
D
E
F
V1
A
B
C
D
E
F


V2
V3
B C
D
A
B
C
D
E
F
A
B
C
D
E
F
V3
A
C
D
E
F
V2
A
B
D
E
F
V1
A
B
C
D
E
F


V2
V3
B C
D
E, F
A
B
C
D
E
F
Installer
A
B
C
D
E
F
Installer
V3
A
C
D
E
F
Installer
V2
A
B
D
E
F
Installer
V1
A
B
C
D
E
F
Installer
iterate
2-week
window

What did we gain?

- More frequent green builds
  - No team blocked
- Integration all along
  - Detect issues early, when they're still easy to fix
- Testing all along
  - Regressions spotted early
- Scalability

At what cost?

- Rigorous approach
- Need clear interfaces / contracts
- Need more hardware
- Need actual **tests**!!!

Maven repositories

- Maven central repository
- Maven internal repository
- Maven local repository (on dev. Hard drive)

Real life example

As Part of the Build

Code complexity

- Lines of code
  - With/without comments
- # methods per class, #lines per file

- Cyclomatic complexity

Static code analysis

- Detection of specific patterns
- On source or bytecode
- Useful for
  - Vulnerabilities
  - Copyright infringement
  - Defect prevention

Coverage

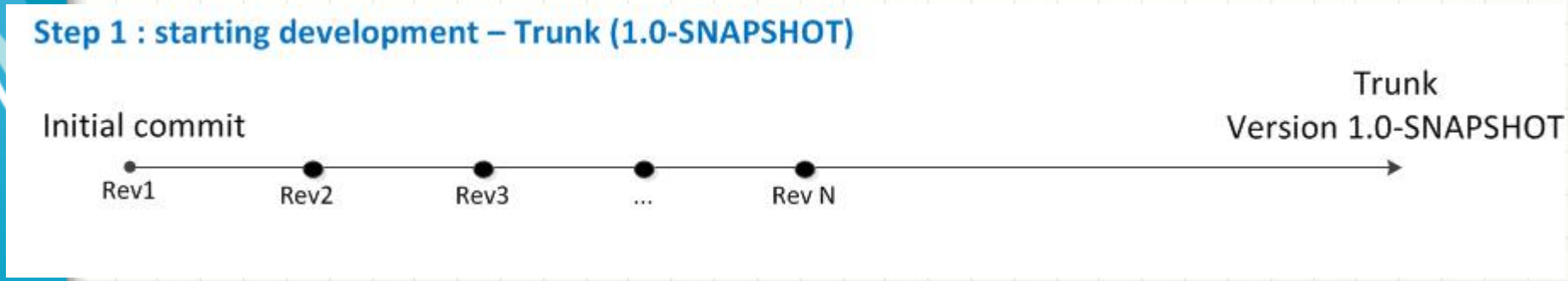
- Code exercised by tests
- Drives
  - testing strategy
  - dataset selection
- But… easy to trick

Multitools

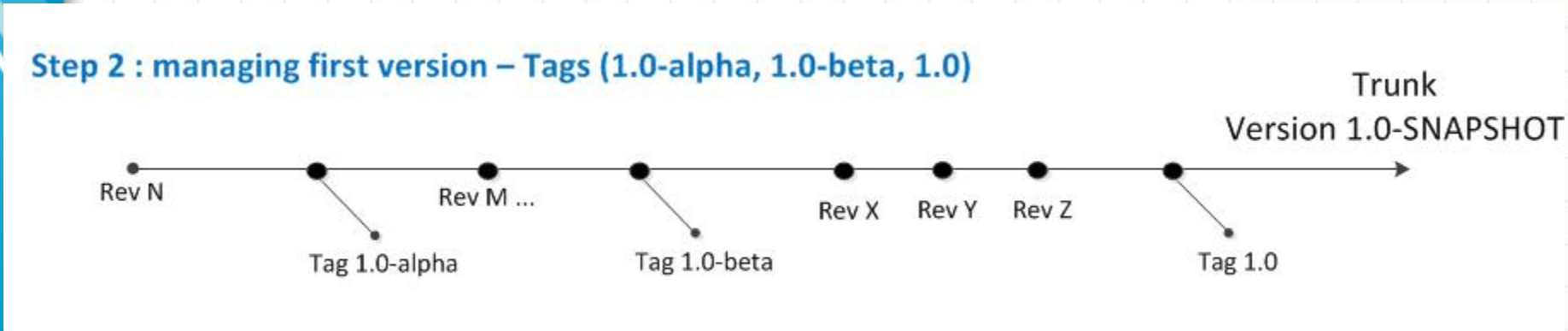
- Eg: SonarQube
- Defects, code smells
- Security vulnerability
- General code “health”

BRANCHING

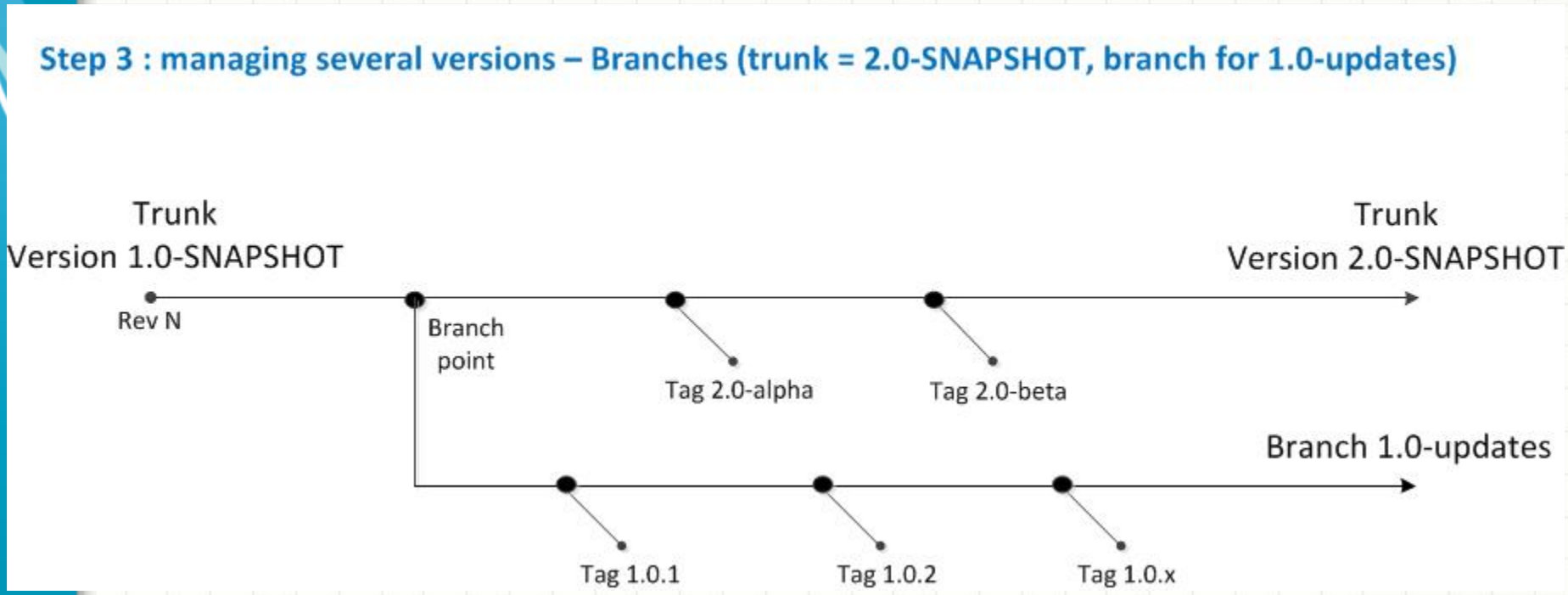
Start a project



First release



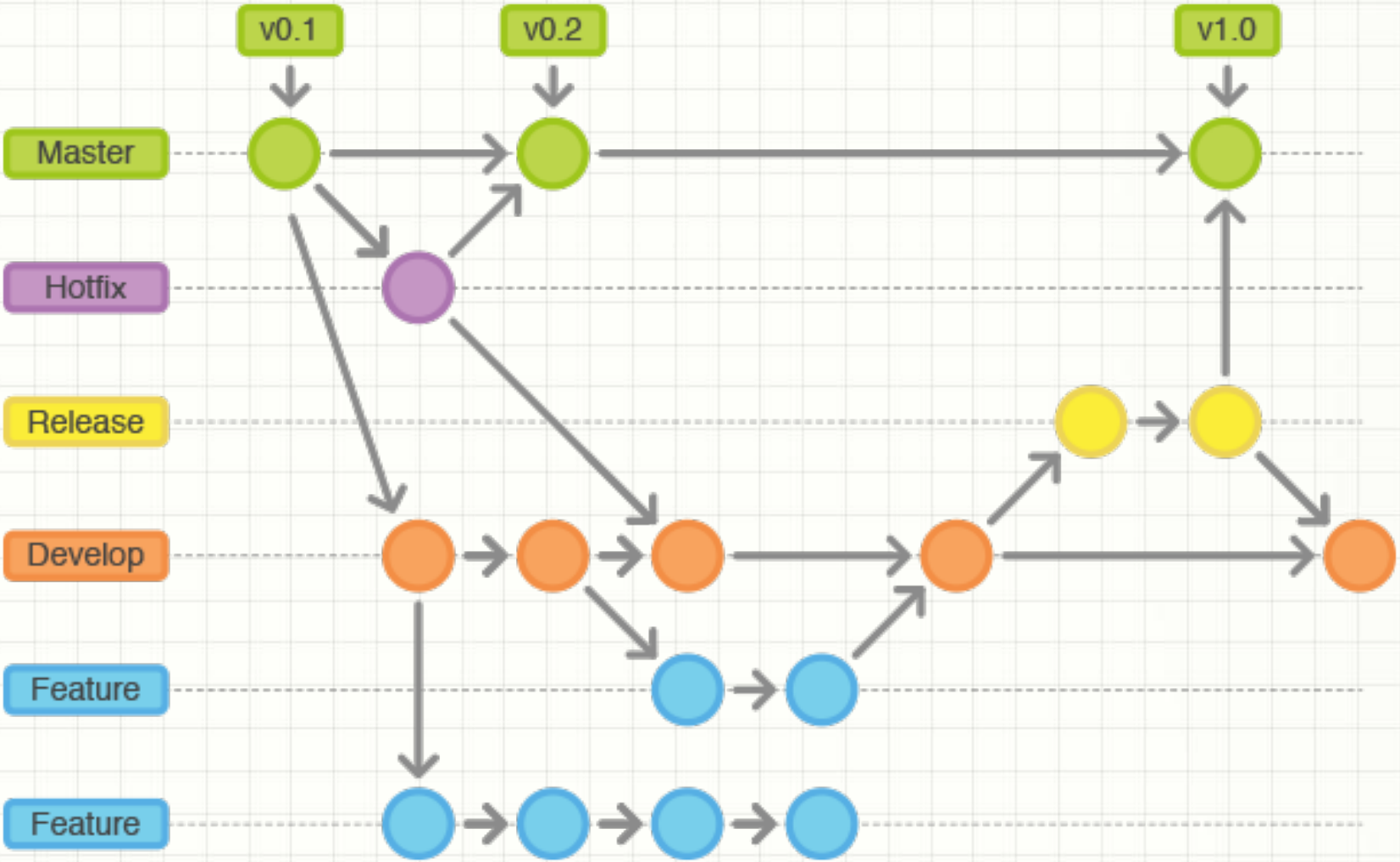
Several releases



Strategies

- Release branch
- Feature branch
- Team branch

Git Flow



QUESTIONS?

TD

- For your project DroneDelivery
  - Decide how to split it into components
  - Establish dependencies
  - Split the build script
- **Goal:** at the end of today, should be able to build each component independently on *one* machine (no repository yet)

APPENDIX