

# CONTINUOUS INTEGRATION

G. Molines  
2017-2018



# INTRODUCTION



Université  
Nice  
Sophia Antipolis



POLYTECH<sup>®</sup>  
NICE-SOPHIA

# 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

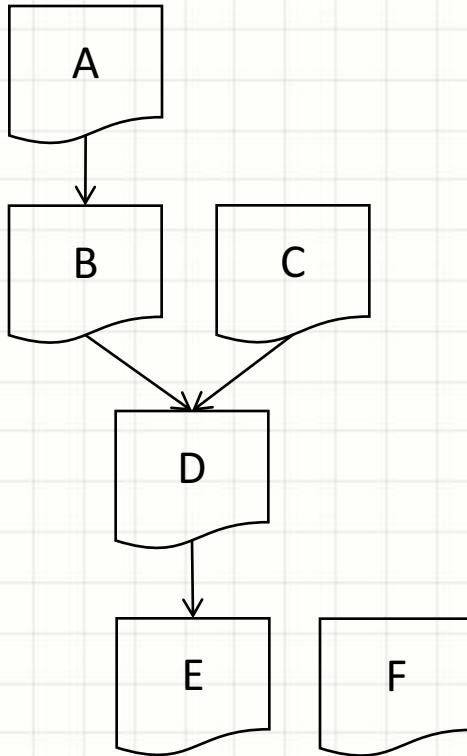
## Tools:

- .
- 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

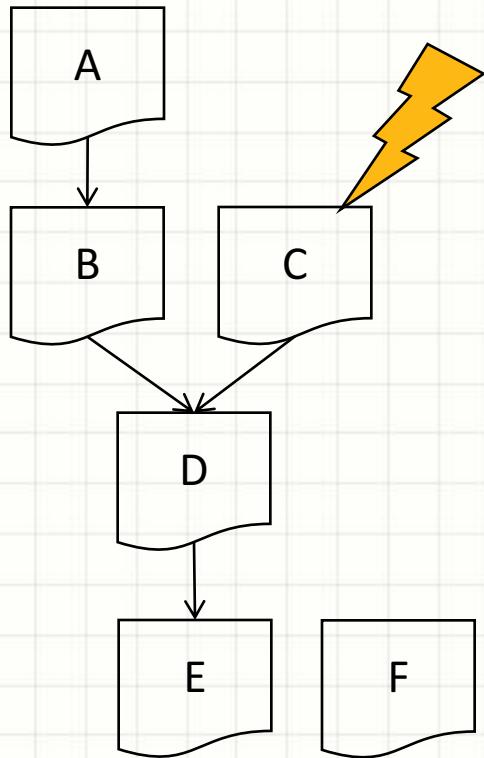
# Module dependency graph



Orchestration:

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

# Code change



Orchestration:

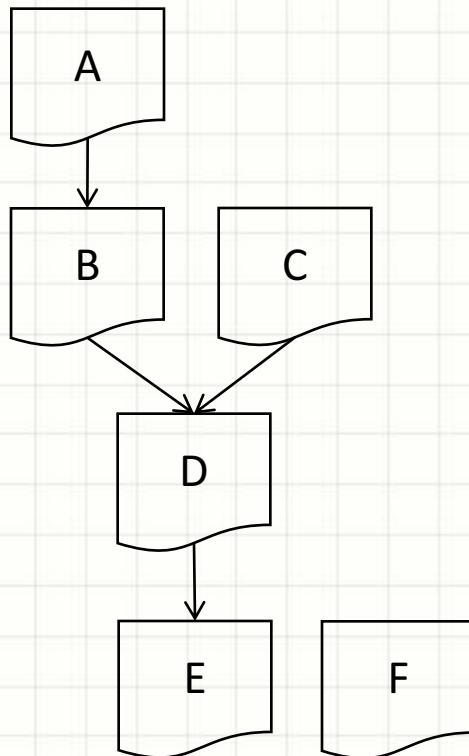
- Build A
- Build B
- Build C
- Build D
- Build E
- Build F

- 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?

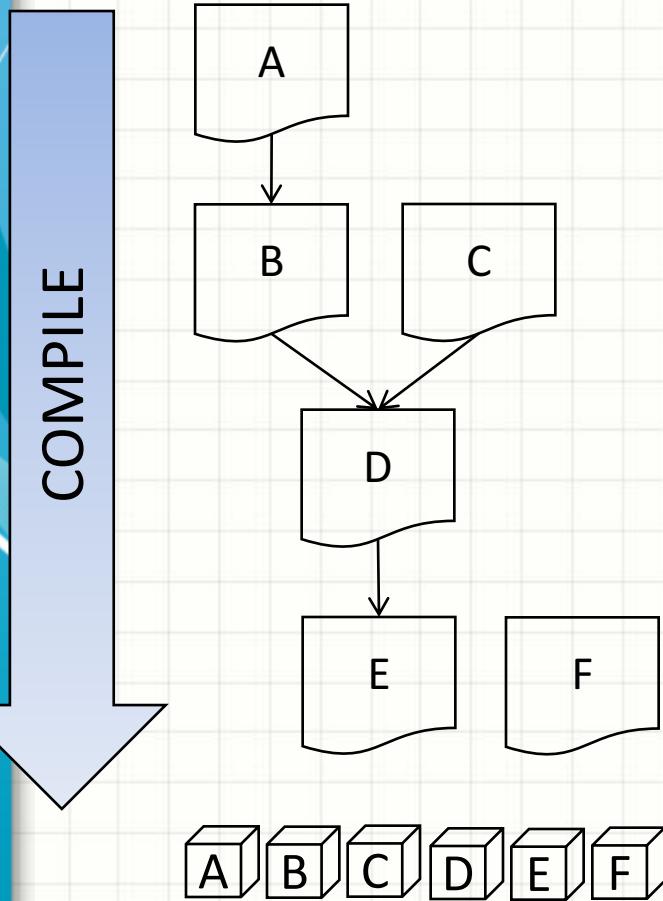
# **DEDICATED BUILD SERVER**



# Classical product build

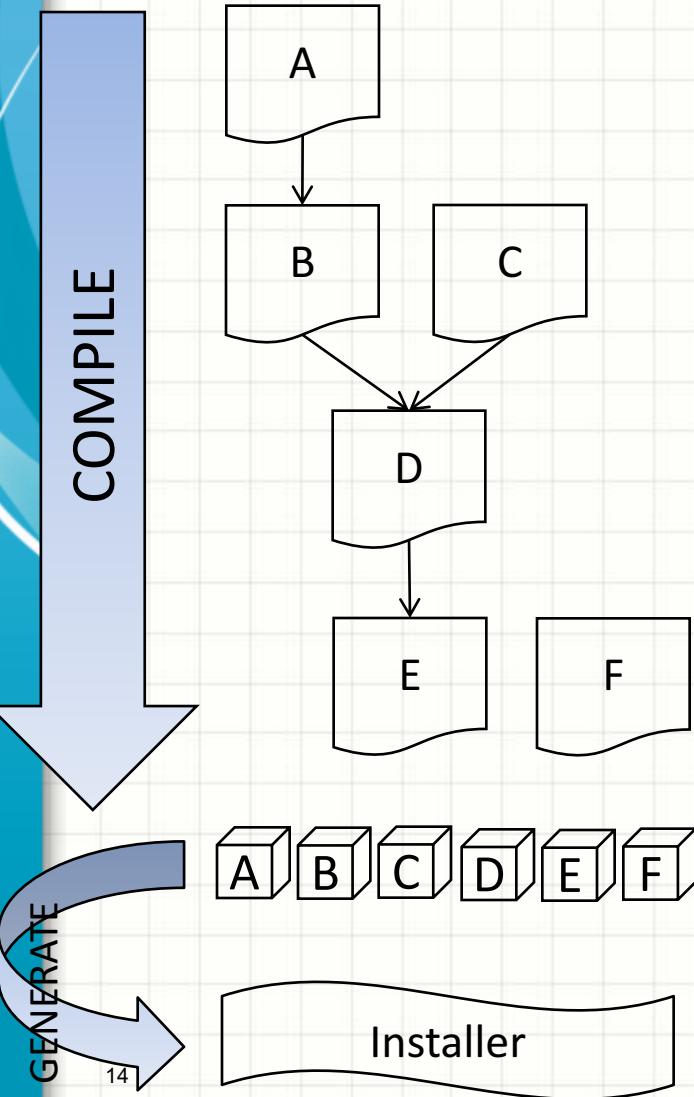


# Classical product build

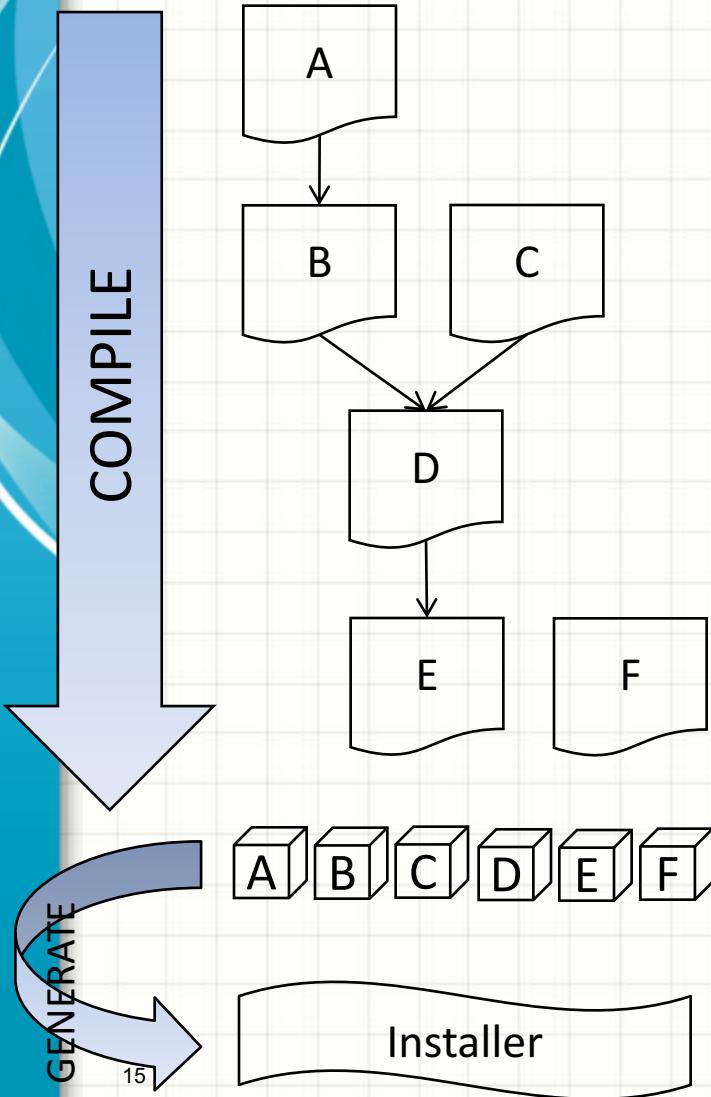


Note: compile really means compile + run unit-tests.  
If a module's unit-tests fail,  
do we consider its  
« compilation » succesfull ?

# Classical product build



# Classical product build

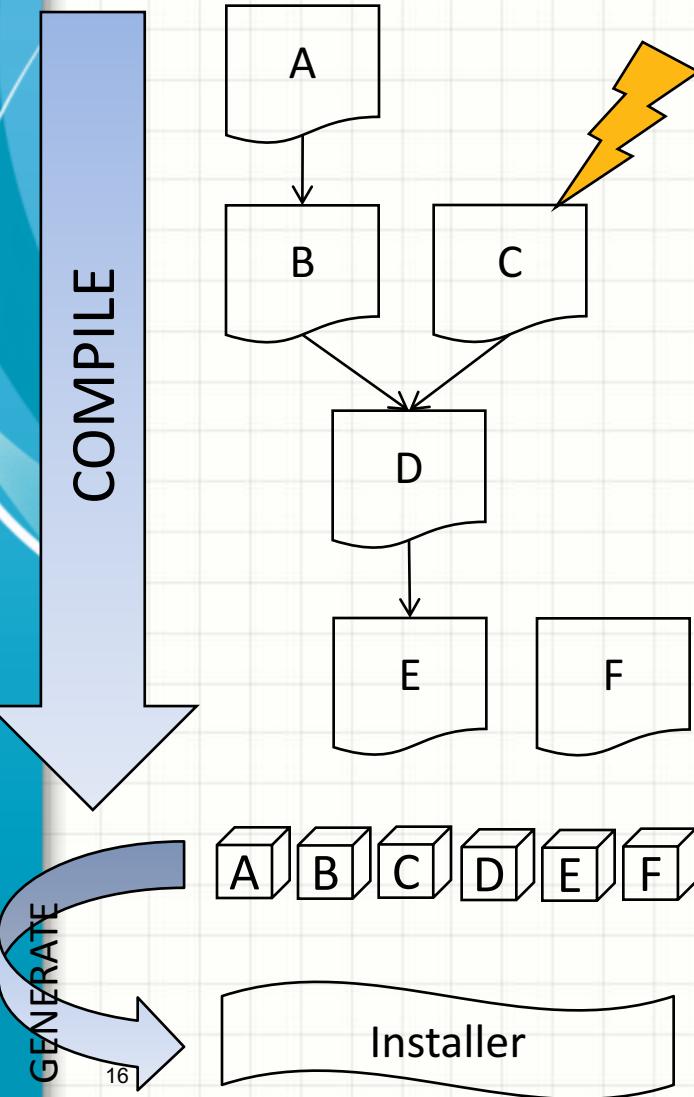


Automate

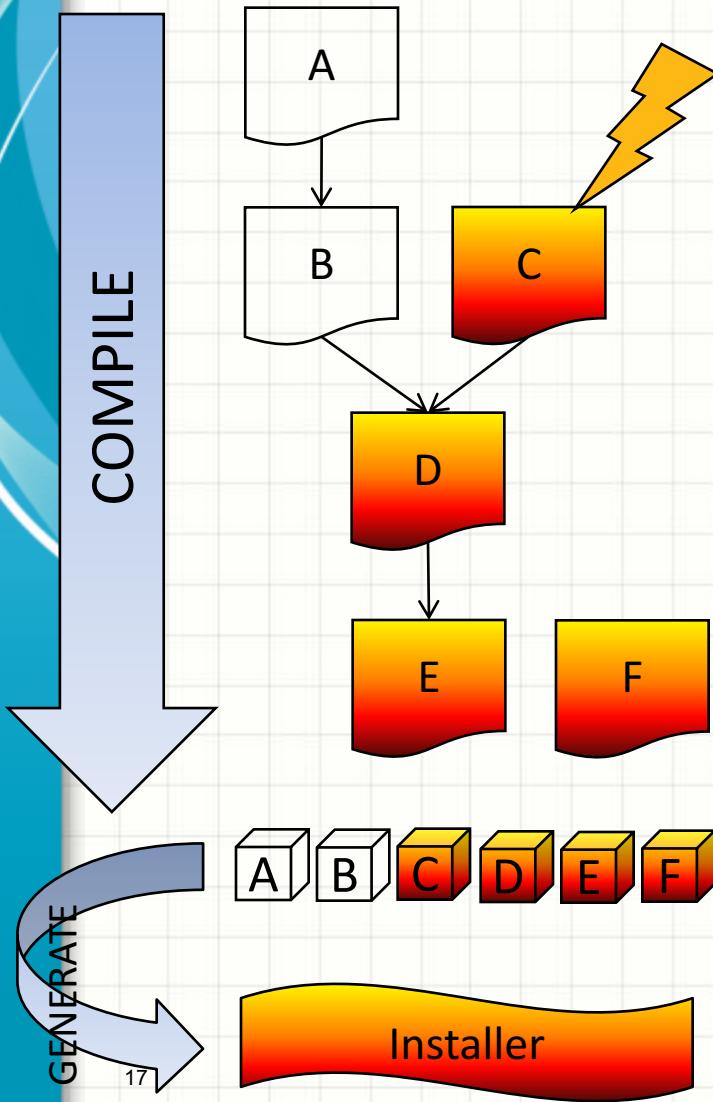
Repeat as fast as possible

The whole process can take hours (or days) !

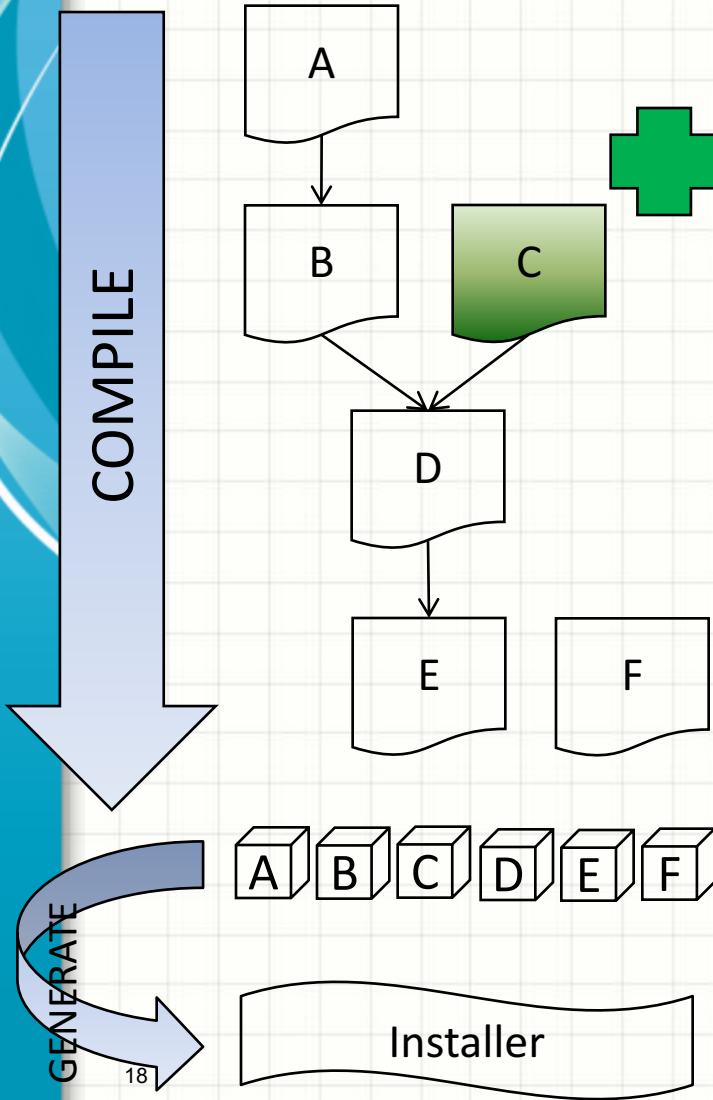
# Classical product build



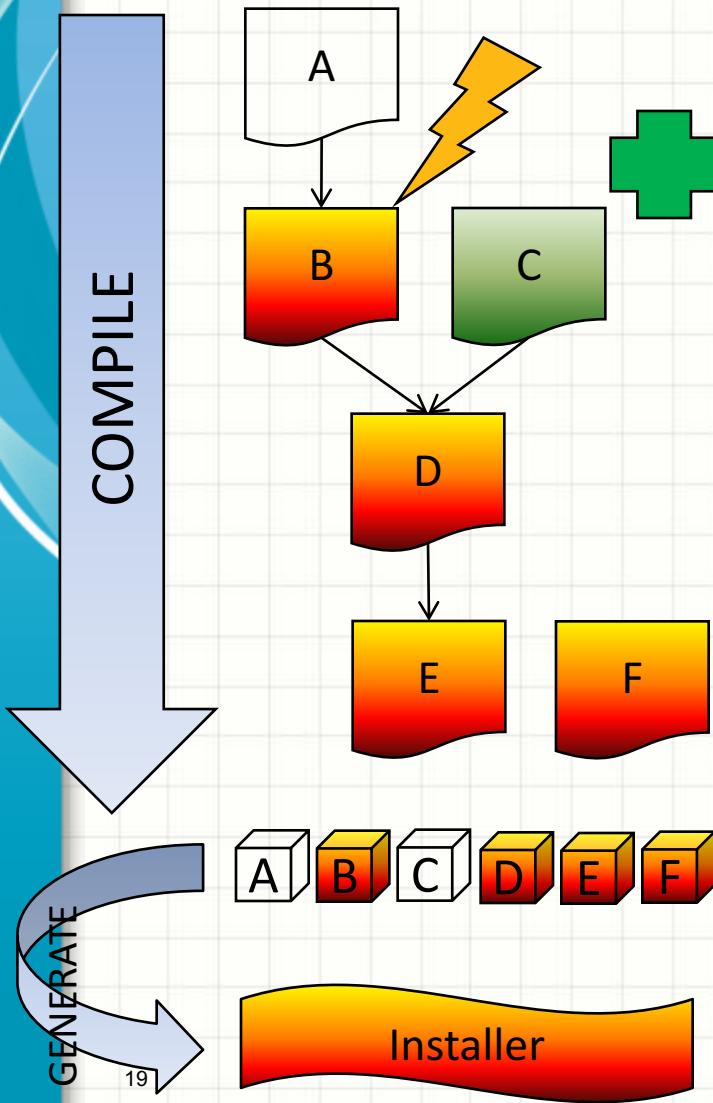
# Classical product build



# 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**

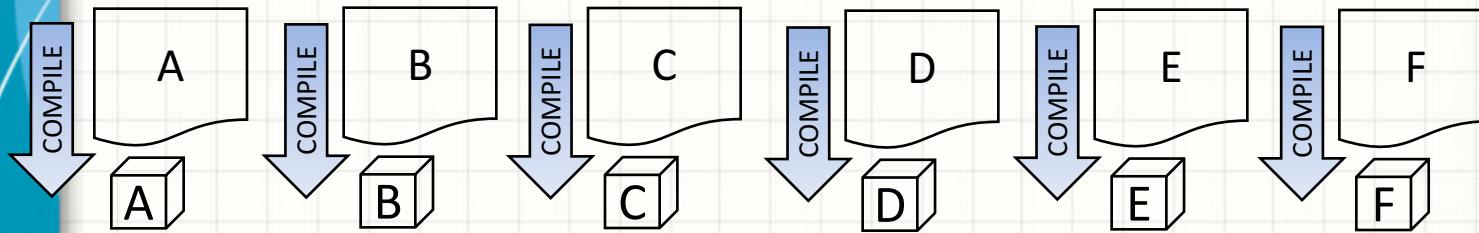
# What can we do?

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

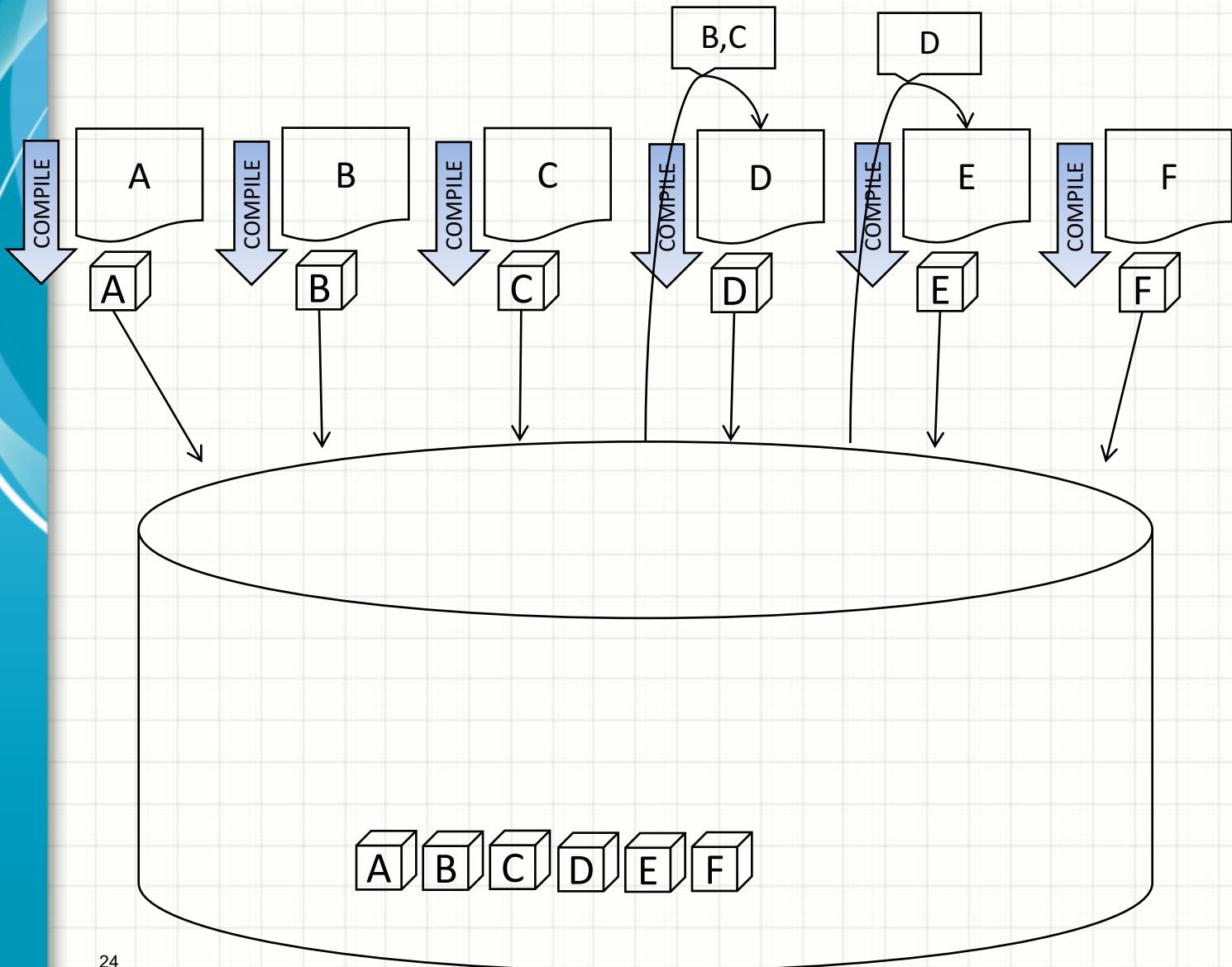
# ARTIFACT REPOSITORY

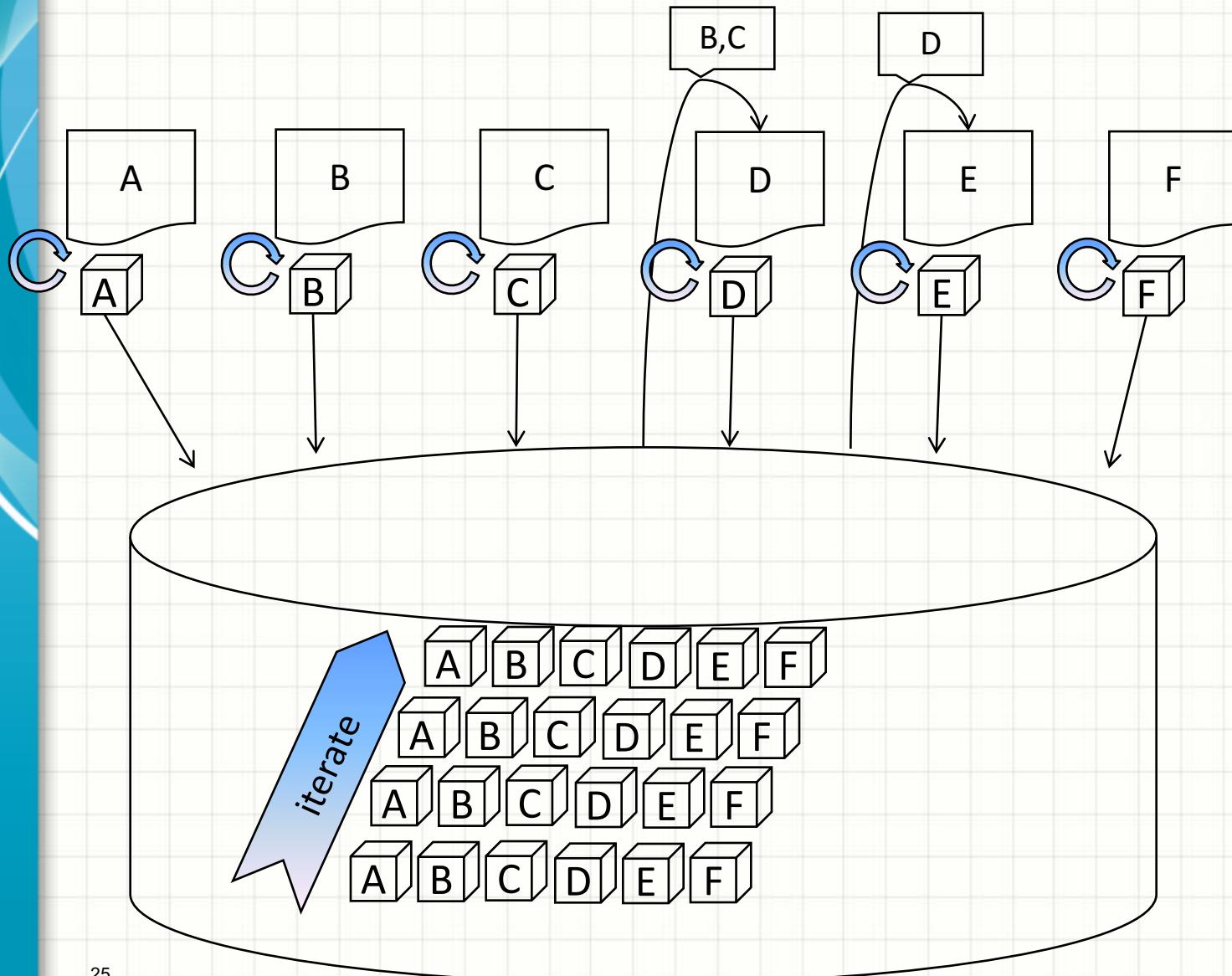


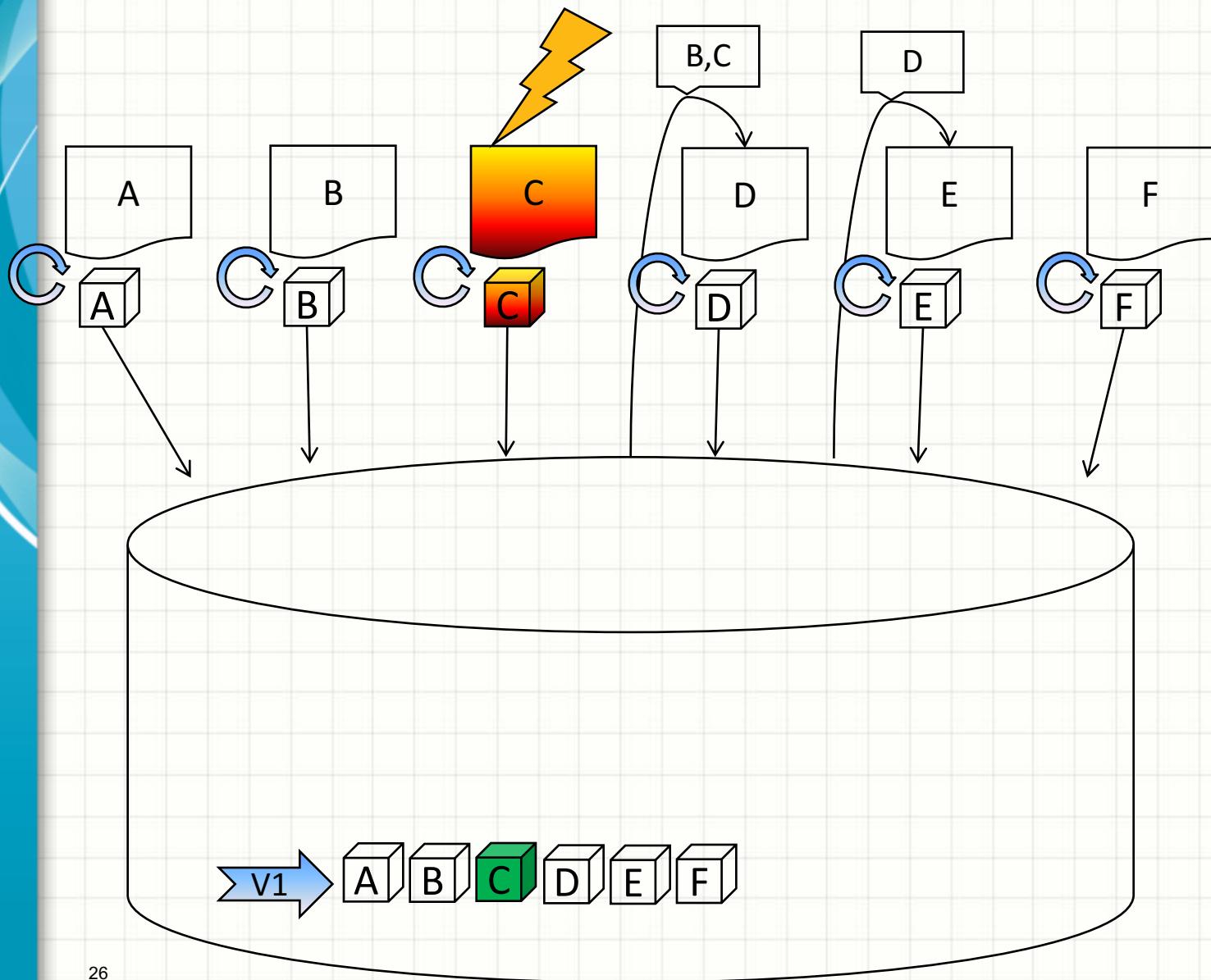
# Continuous Integration

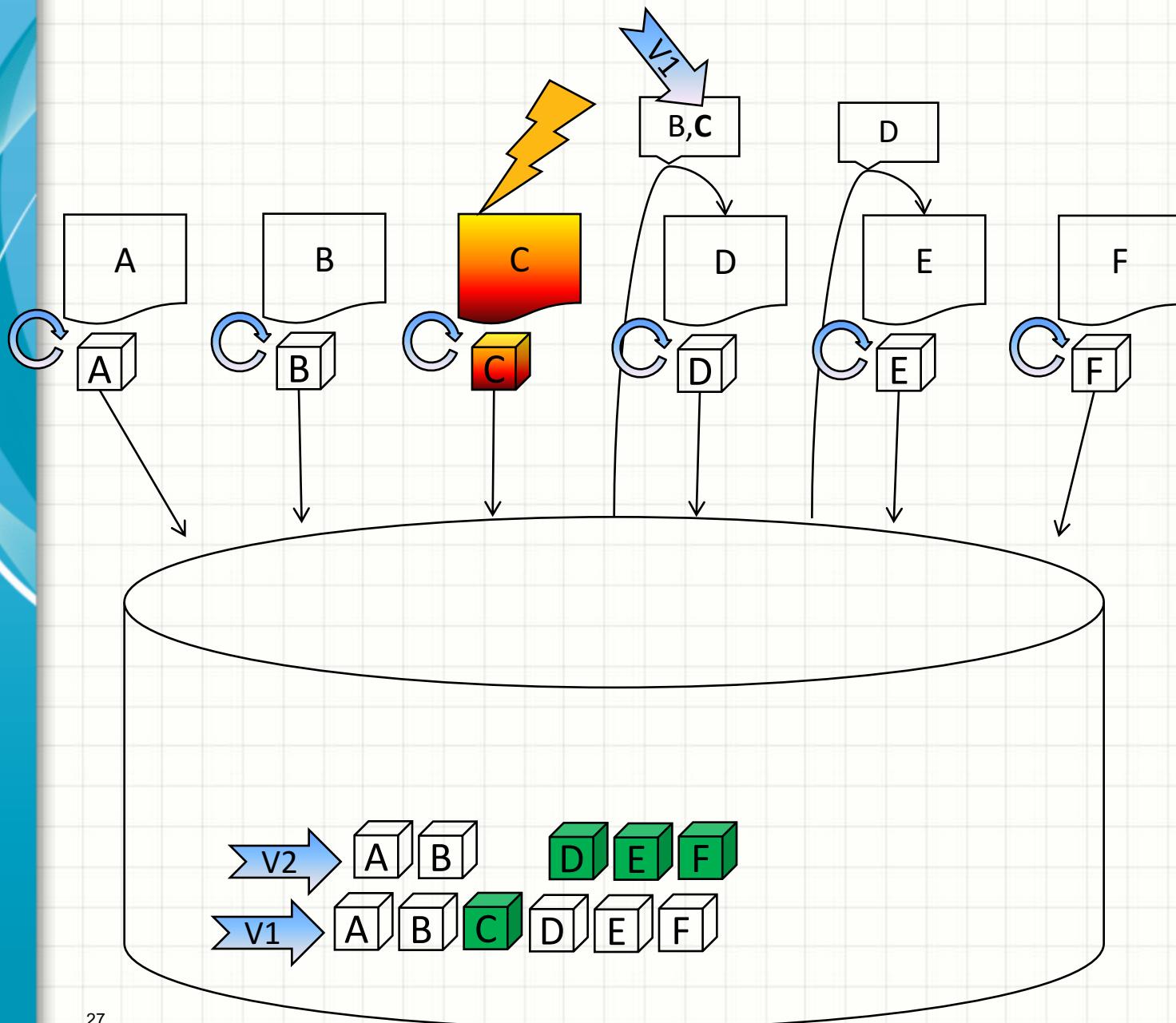


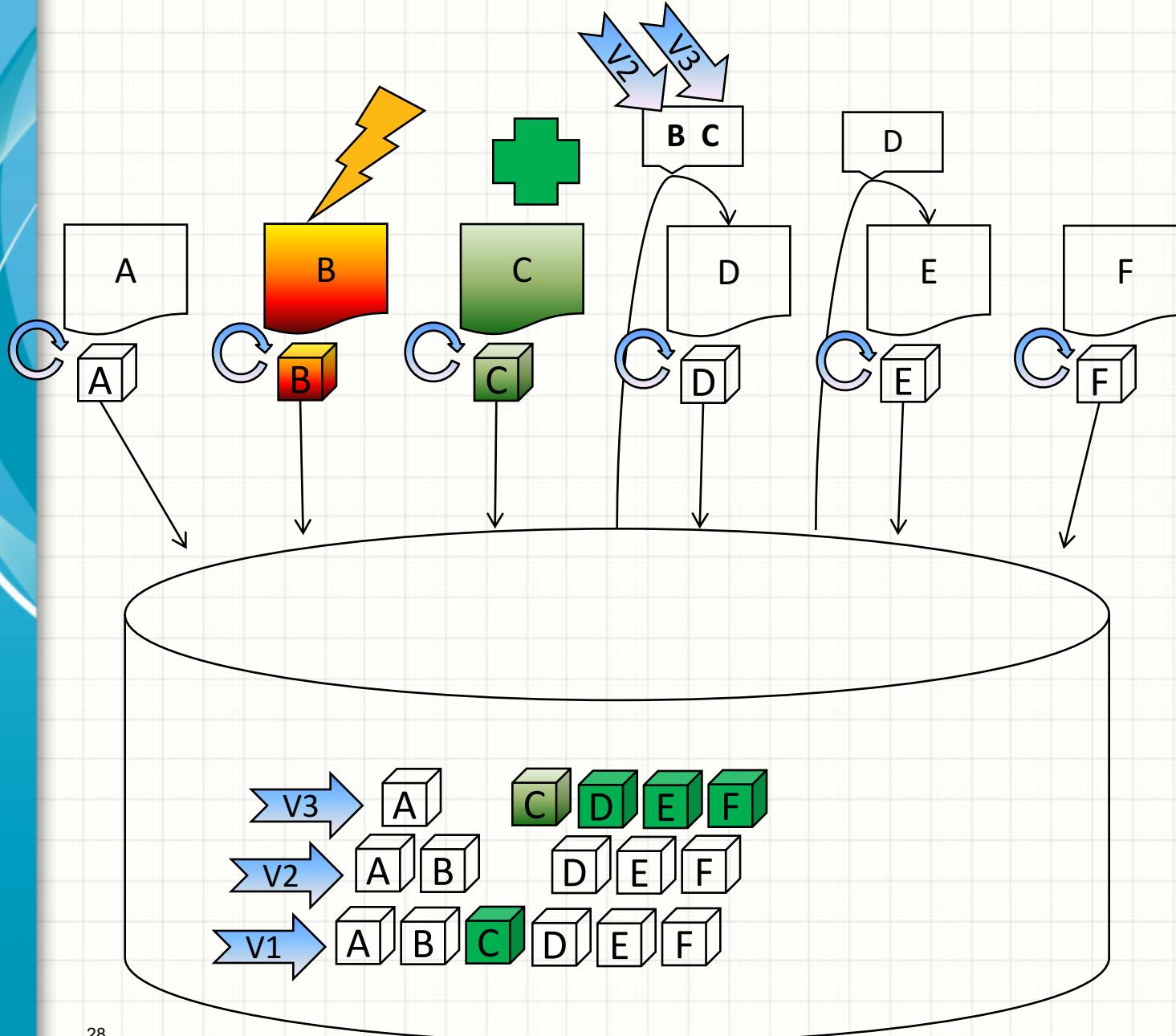
Note: compile really means compile + run unit-tests.  
If a module's unit-tests fail, we consider its « compilation » as failed !

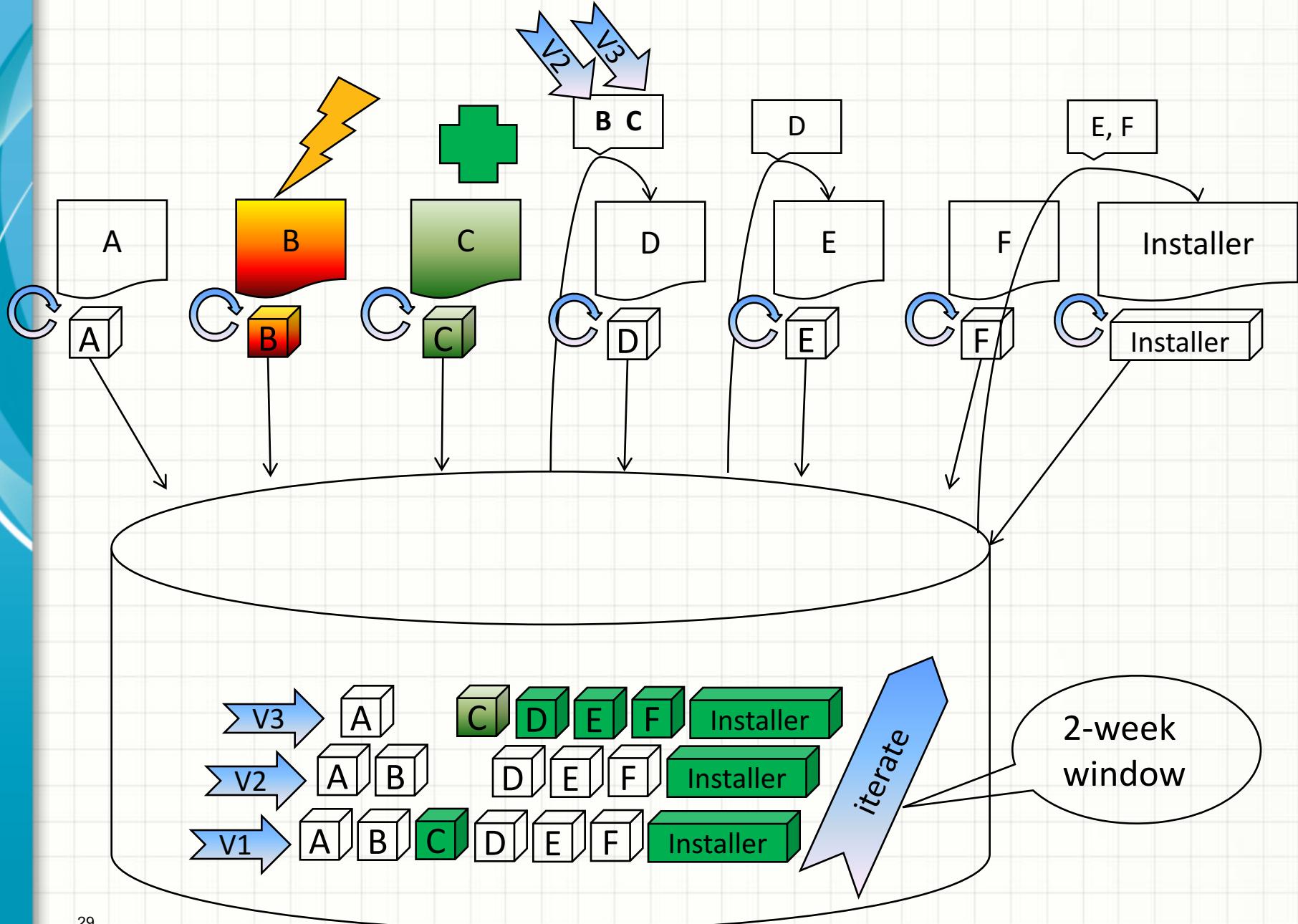












# What did we gain?

- 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

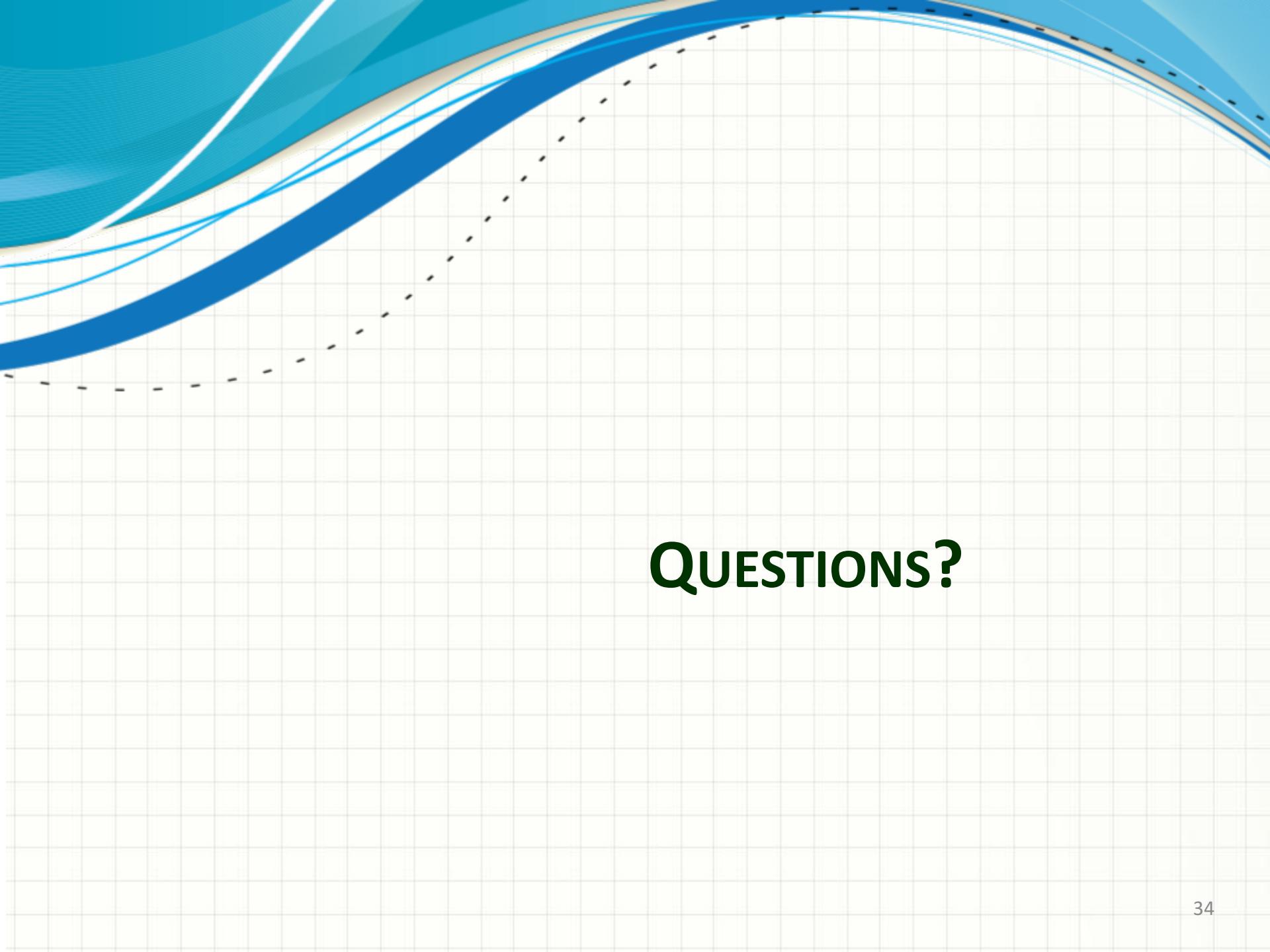
# At what cost?

- 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



**QUESTIONS?**

# TD

- For your project Polyevent
  - 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