CI/CD

Master in Artificial Intelligence and Data Engineering
del Politecnico di Milano

Elisabetta Di Nitto and Simone Reale

July 3, 2024

July 16, 2024

About us

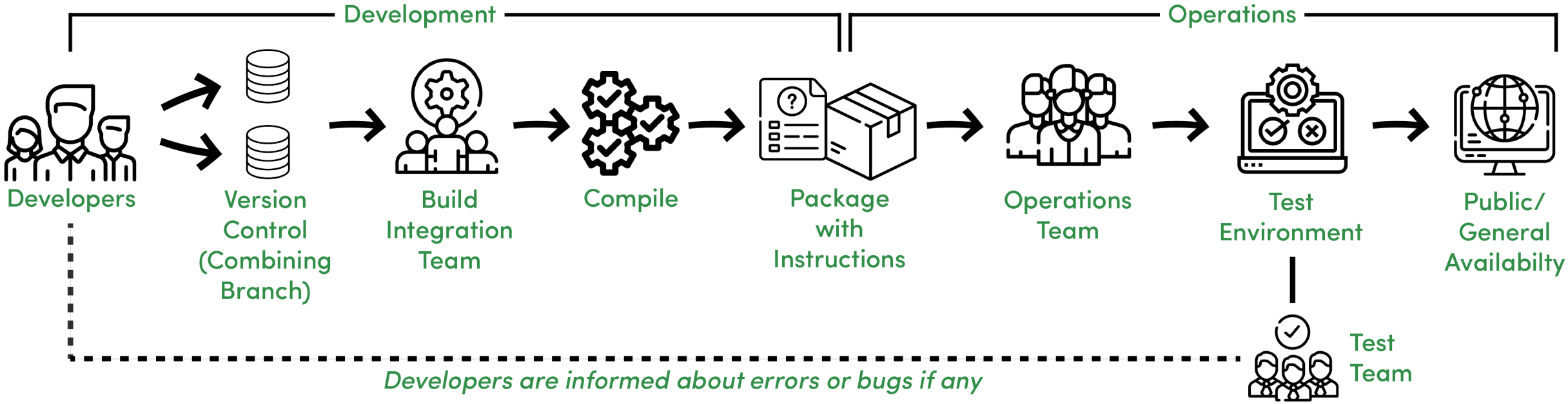
(some years ago… ☺)

- Professor at Politecnico di Milano
- Focus on software engineering, cloud computing, software architectures, more recently, quantum computing

- PhD student at Politecnico di Milano
- Focus on software engineering and quantum computing

CI/CD in a nutshell

- CI/CD = Continuous Integration / Continuous Deployment or Delivery



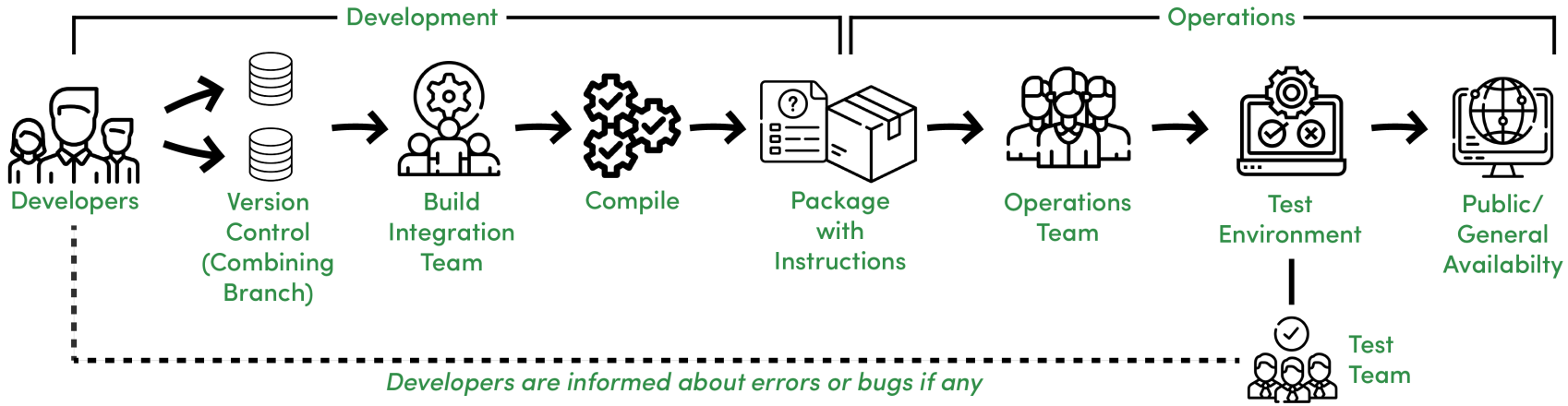
https://www.geeksforgeeks.org/ci-cd-continuous-integration-and-continuous-delivery/

Course learning objectives

POLITECNICO MILANO 1863

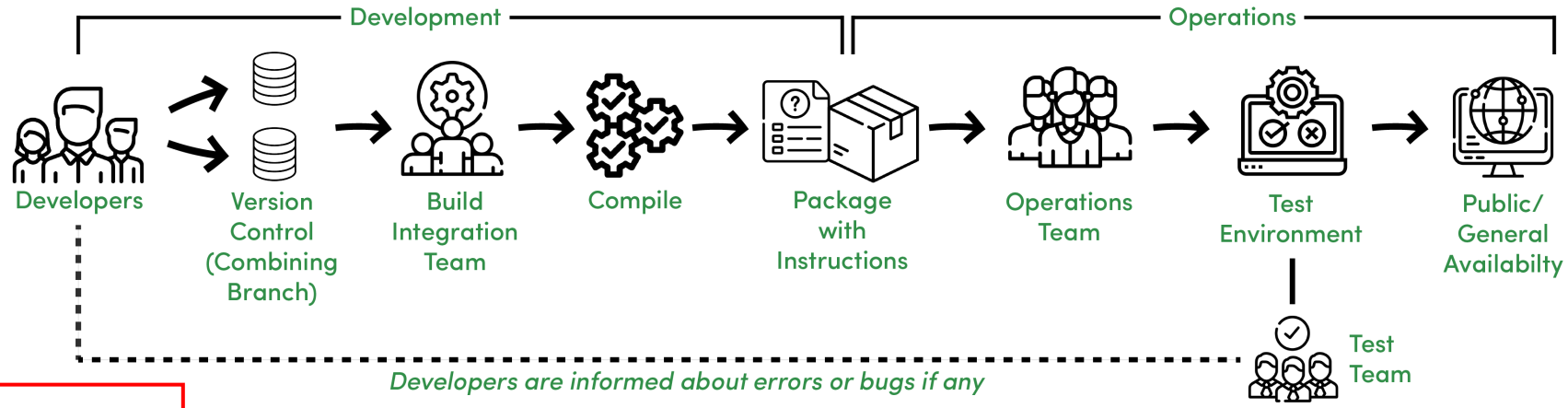
- Learning objectives
  - Be able to design a CI/CD pipeline
  - Understand the most common steps that compose it
  - Understand how to integrate the pipeline in the software development process
- Approach
  - Today more on traditional lecturing with some exercises
  - Next session more practical with hands-on examples

Course plan



- Process
  - Development lifecycles
  - DevOps
- Automation
  - Version control & software configuration management
  - Static analysis
  - Testing automation
  - Pipeline control tools
- Hands-on
  - Testing automation
  - Creating pipelines for C and Python programs using GitHub Actions

Course plan



- Process
  - Development lifecycles
  - DevOps
- Automation
  - Version control & software configuration management
  - Static analysis
  - Testing automation
  - Pipeline control tools
- Hands-on
  - Testing automation
  - Creating pipelines for C and Python programs using GitHub Actions

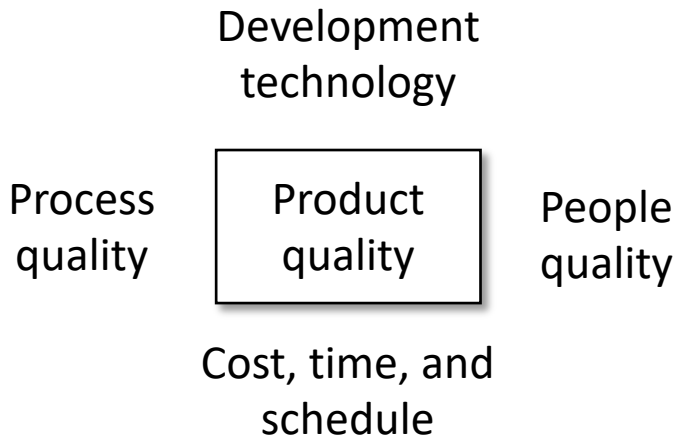
Process and product

POLITECNICO MILANO 1863

- Our goal is to develop **software products**
- The **process** is how we do it
- Both are extremely important, due to the nature of the software product
- Both have qualities
  - in addition, quality of process affects quality of product
  - …even though, other aspects such as the quality of the development team are important as well

The software product

- Different from traditional types of products
  - intangible
    - difficult to describe and evaluate
  - malleable
  - human intensive
    - does not involve any trivial manufacturing process
- Aspects affecting product quality



Software product qualities – ISO/IEC 25010:2011 Software Quality Model

https://iso25000.com/index.php/en/iso-25000-standards/iso-25010

Process qualities: productivity

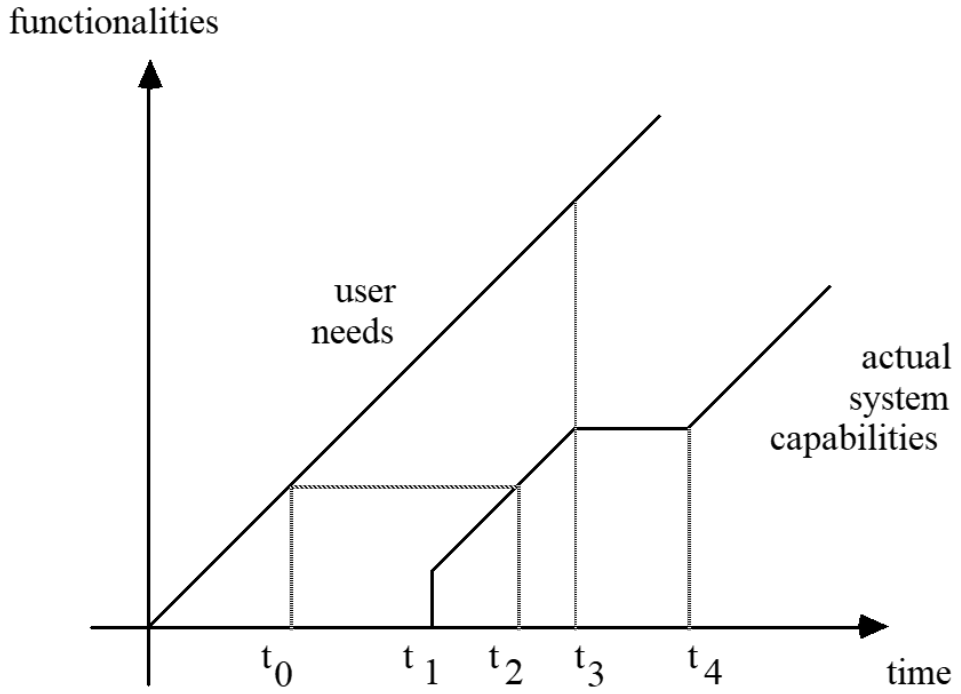
- Ability to produce a “good” amount of product
- How can we measure it? → Delivered items by unit of effort

lines of code (and variations)
function points

person month
WARNING: persons and months cannot be interchanged

Process qualities: timeliness

- Ability to respond to change requests in a timely fashion

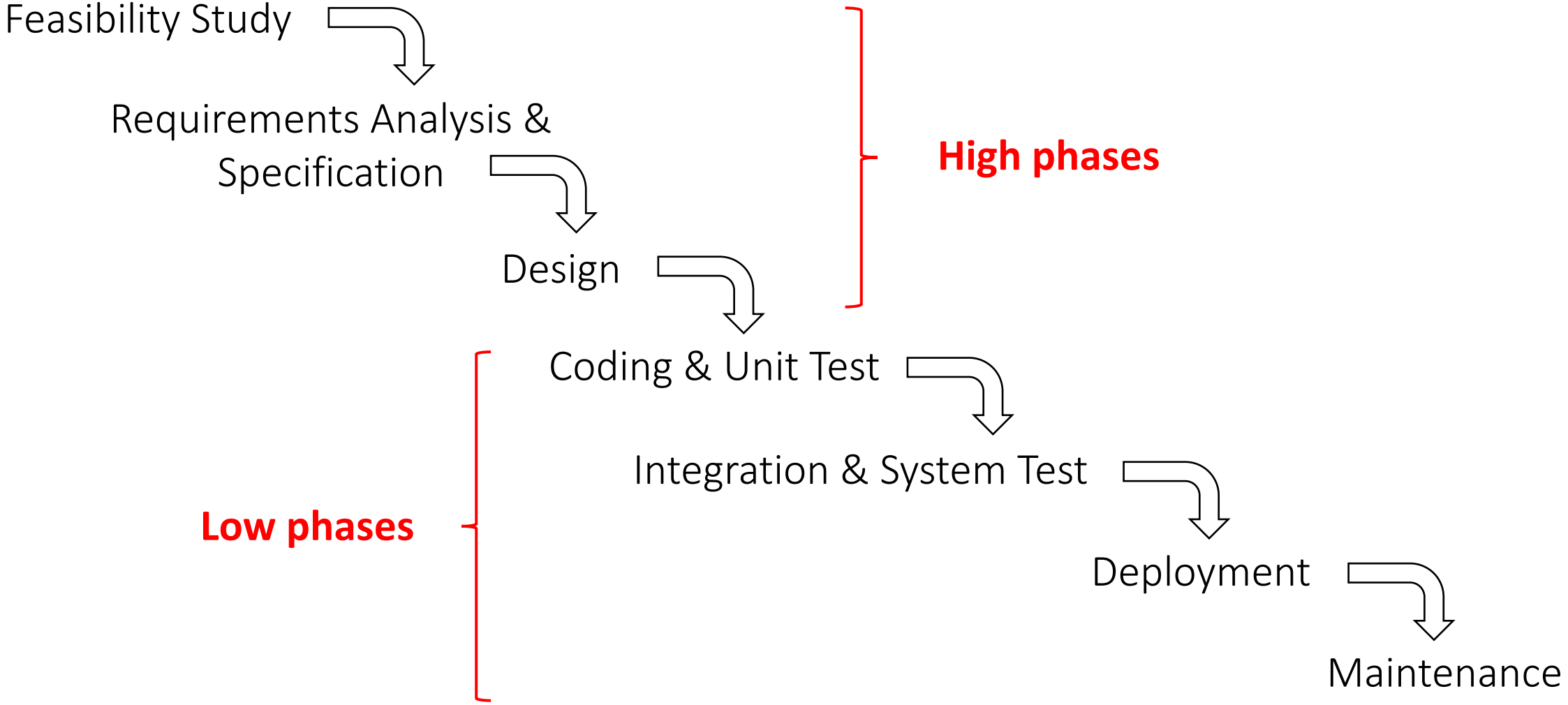


Software lifecycles

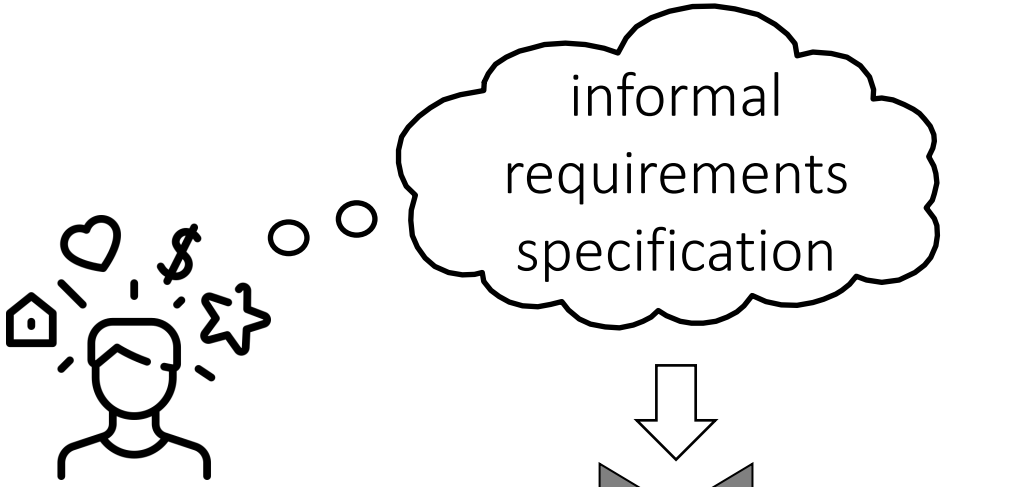
POLITECNICO MILANO 1863

- Initially, no reference model: Just code & fix
- As a reaction to the many problems: traditional “waterfall” model
  - identify phases and activities
  - force linear progression from a phase to the next
  - no returns (they are harmful)
    - better planning and control
  - standardize outputs (artifacts) from each phase
  - Software like manufacturing
- Then, flexible processes: iterative models, agile methods, DevOps

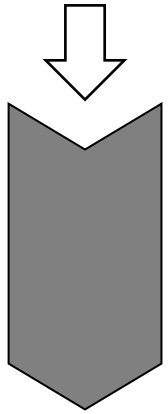
A waterfall organization



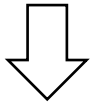
Waterfall is “black box”



informal requirements specification



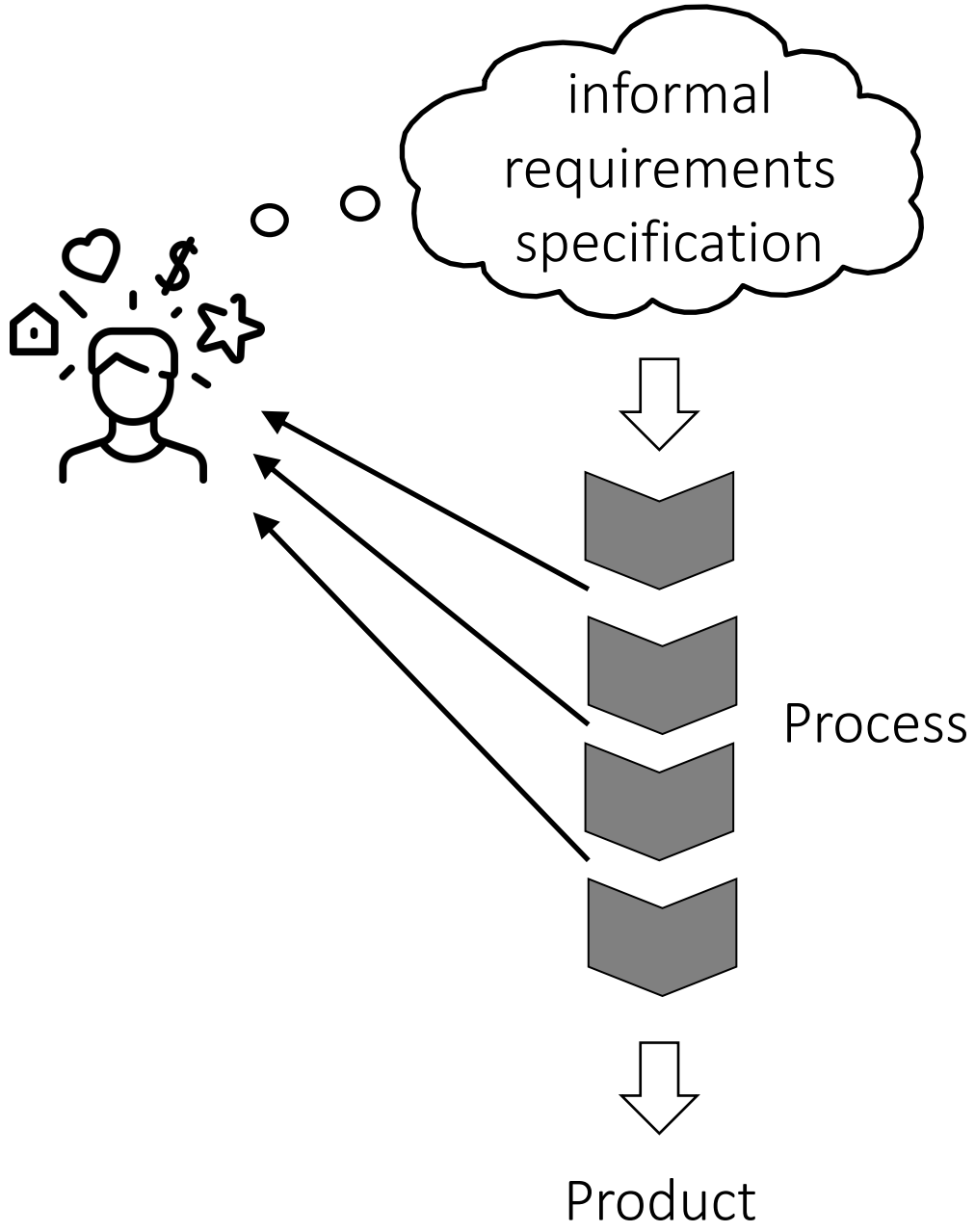
Process



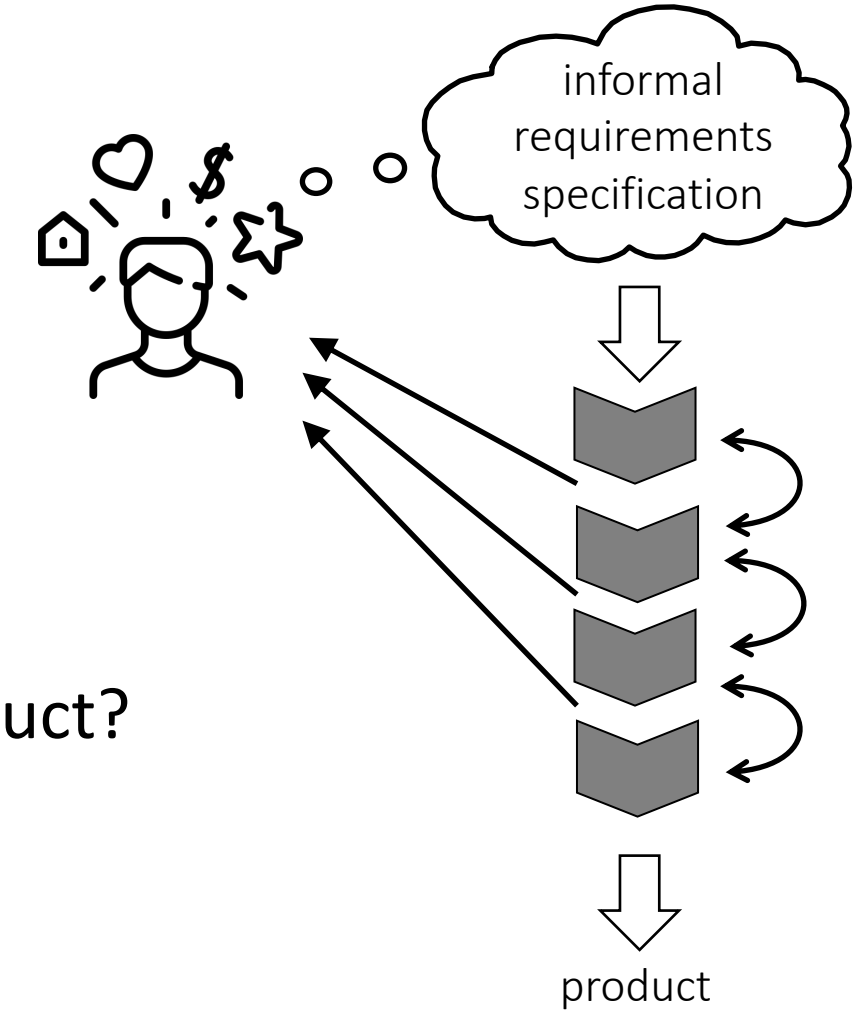
Product

Need for transparency

- Transparency allows early check and change via feedback
- It supports flexibility



Verification and validation



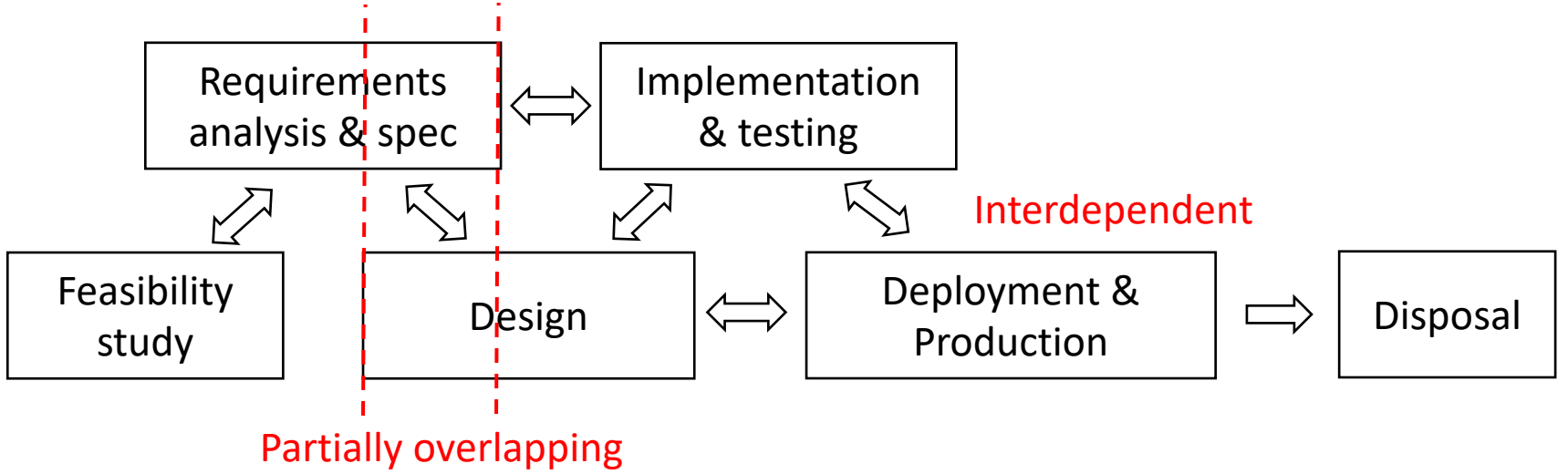
Validation:
are we making
the ”right” product?

Verification:
are we doing
the product right?

Flexible processes

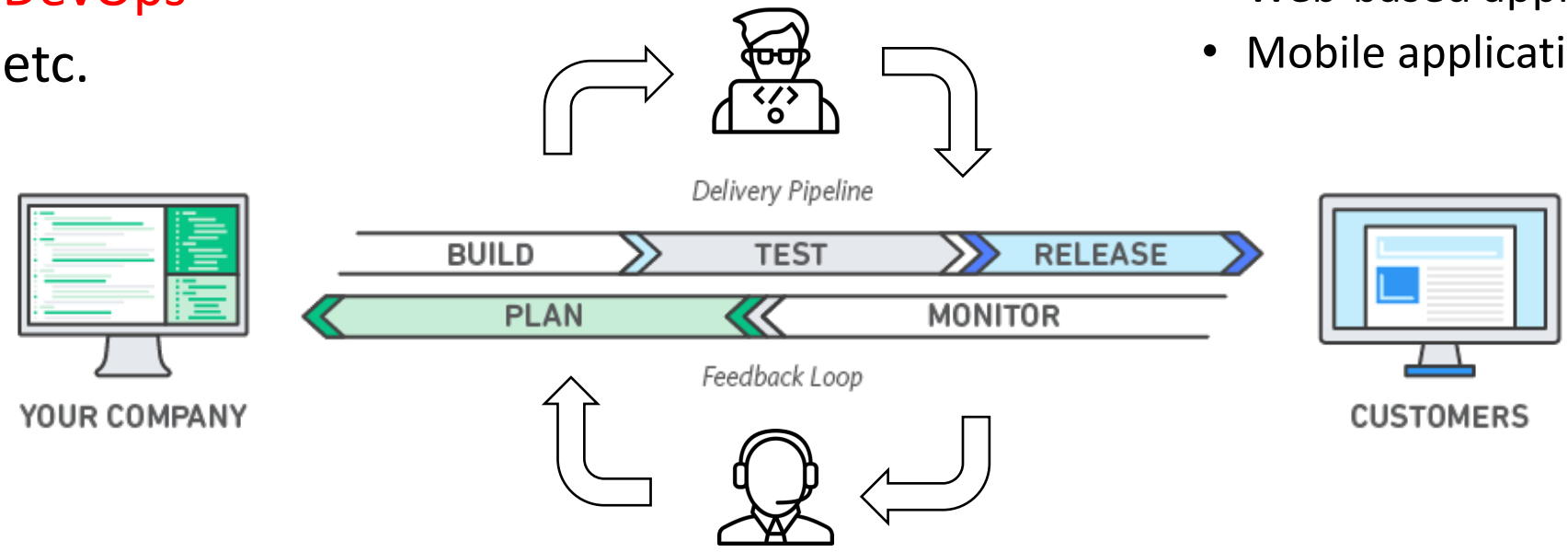
- Main goal: adapt to changes (especially in requirements and specs)
- The very idea:
  - stages are not necessarily sequential
  - processes become iterative and incremental

Example



Flexible processes

- Exist in many forms
  - eXtreme Programming (XP)
  - SCRUM
  - DevOps
  - etc.
- Effective in dynamic contexts
  - Many changes per week
  - Example:
    - Web-based applications
    - Mobile applications



Dev & Ops: the classical roles

POLITECNICO MILANO 1863

Dev focuses on

- Developing features
- Changing requirements
- Releasing products

Ops focuses on

- Offering services
- Providing guarantees and stability

Dev vs Ops

POLITECNICO MILANO 1863

"The system works correctly in the development machine, but it does not on the operation machine"

"10 days after successful deployment of last release, the system is overloaded"

Who is guilty? ***Dev–*** team, or ***–Ops*** team?

The problem: two disconnected groups

- Dev works to apply releases
- Dev does not pay attention to QoS guarantees

- Ops resists to releases
- Ops is aiming at guaranteering QoS

- Changes are fundamental for the business!
- QoS guarantees are needed too!

DevOps

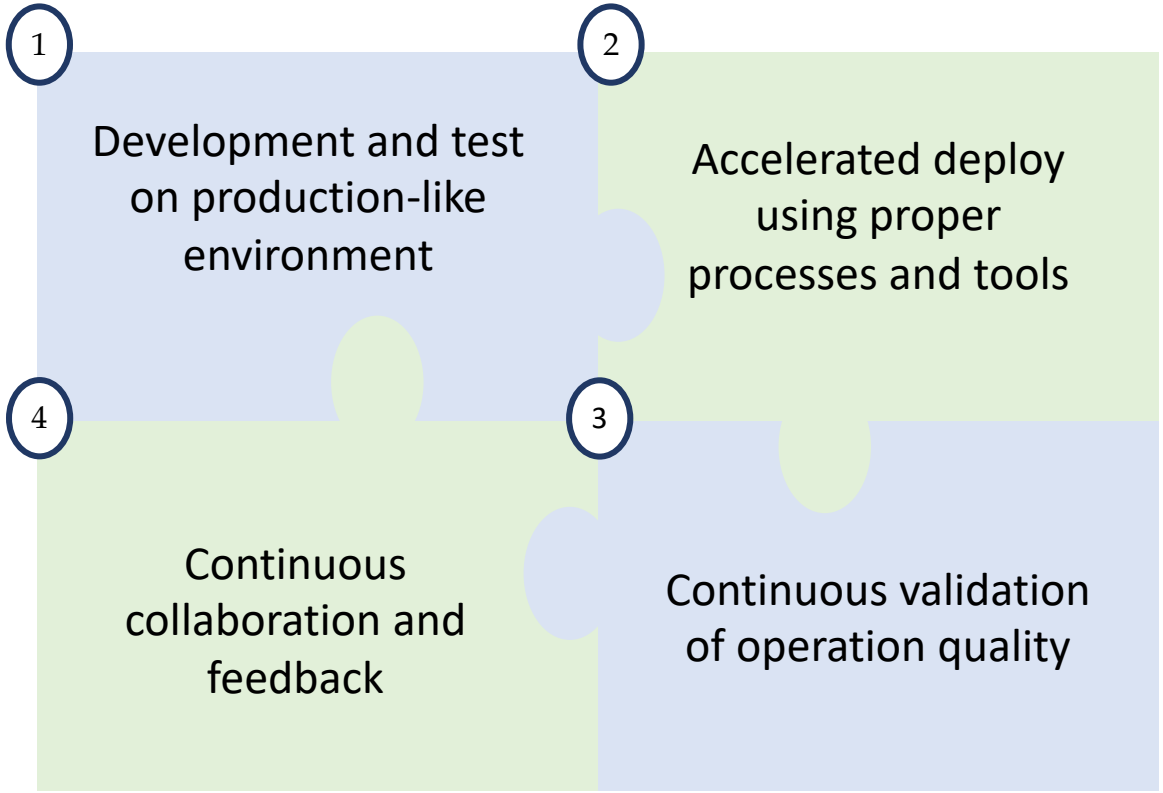
POLITECNICO MILANO 1863

- **What is it:** “Practices or tools that bridge the gap between development and operations”

- **Goal:** Creates a collaborative mindset where a single team performs Dev and Ops
  →the team **must** contain differentiated competences, background, etc.

- **Requires:**
  - Culture management;
  - Automation tools;
  - Organisational as much as technical metrics
  - Continuous sharing artifacts, procedures, languages, approaches…

The four DevOps values



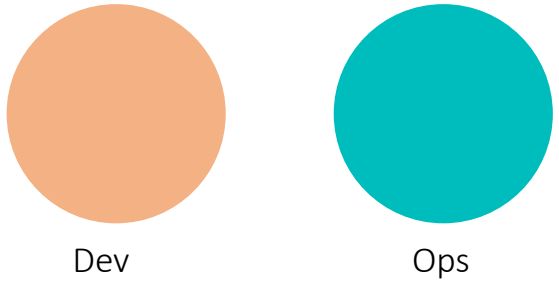
DevOps general advantages

POLITECNICO MILANO 1863

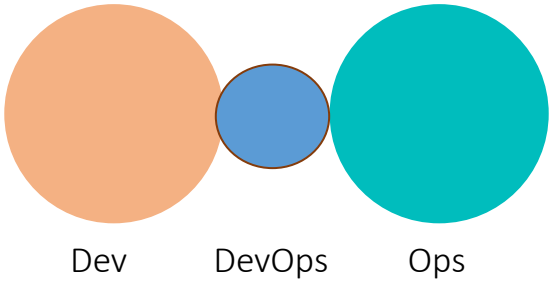
- Shorten time-to-value
- Quick and regular release of software features
- Improve quality
- Mitigate risks
- Increase collaboration in the team

DevOps Organisational Changes

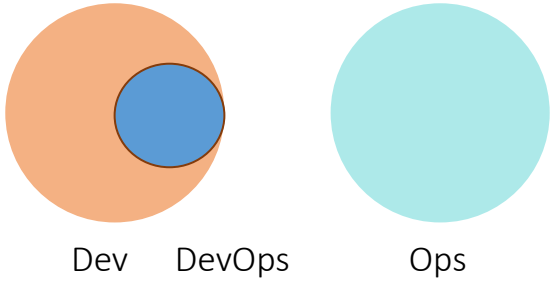
What to avoid: anti-patterns



Separate Silos



Separate DevOps Silo

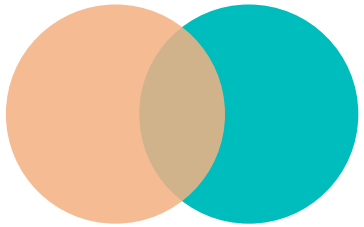


We don’t need Ops

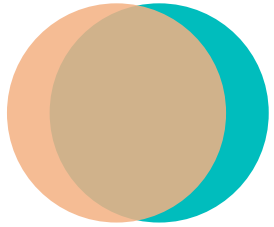
From M. Skelton. What Team Structure is Right for DevOps to Flourish? 2013
https://blog.matthewskelton.net/2013/10/22/what-team-structure-is-right-for-devops-to-flourish/
Further elaborated in http://web.devopstopologies.com

DevOps Organisational Changes

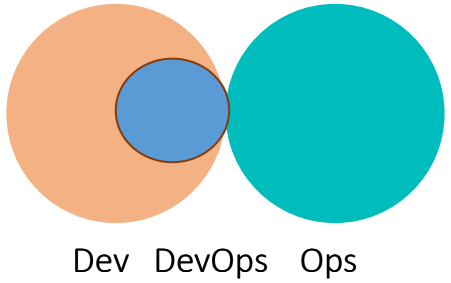
What to do: adoption patterns



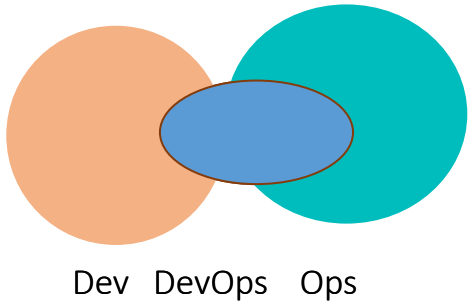
Smooth collaboration



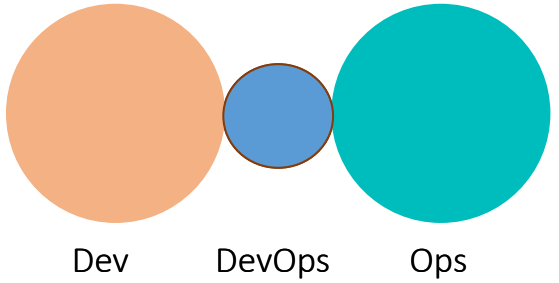
Fully Embedded



Ops as an Infrastructure as a Service



DevOps as a Service



Temporary DevOps Team

From M. Skelton. What Team Structure is Right for DevOps to Flourish? 2013
http://blog.matthewskelton.net/2013/10/22/what-team-structure-is-right-for-devops-to-flourish/
http://web.devopstopologies.com

DevOps processes and toolchain

- Continuous Architecting

Def. *"architect for test, build and deploy, take quality attributes into account, take advantage of feedback from runtime"*

- Continuous Integration

Def. *"m*erge all developer work-copies to a shared mainline frequently"

- Continuous Testing

Def. *"run tests as part the build pipeline so that every check-in and deployment is validated"*

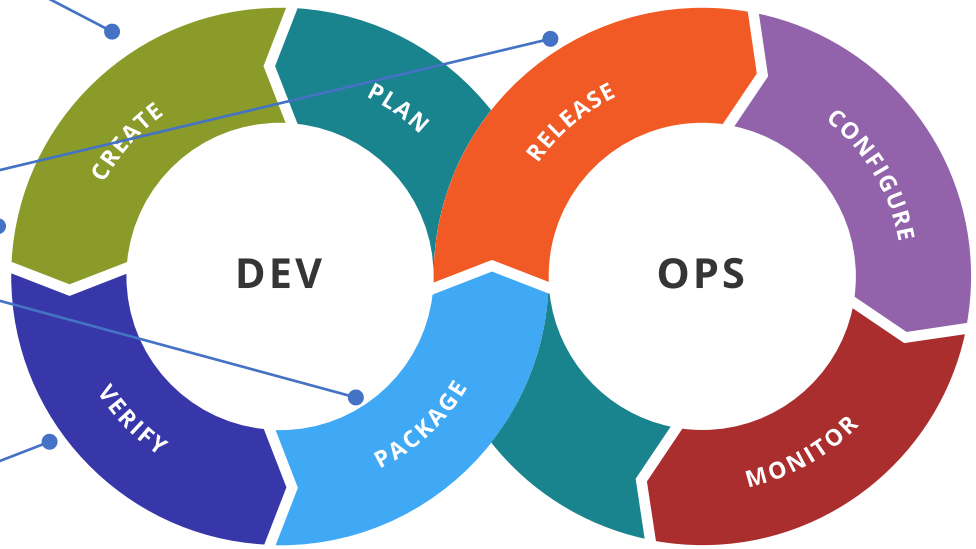
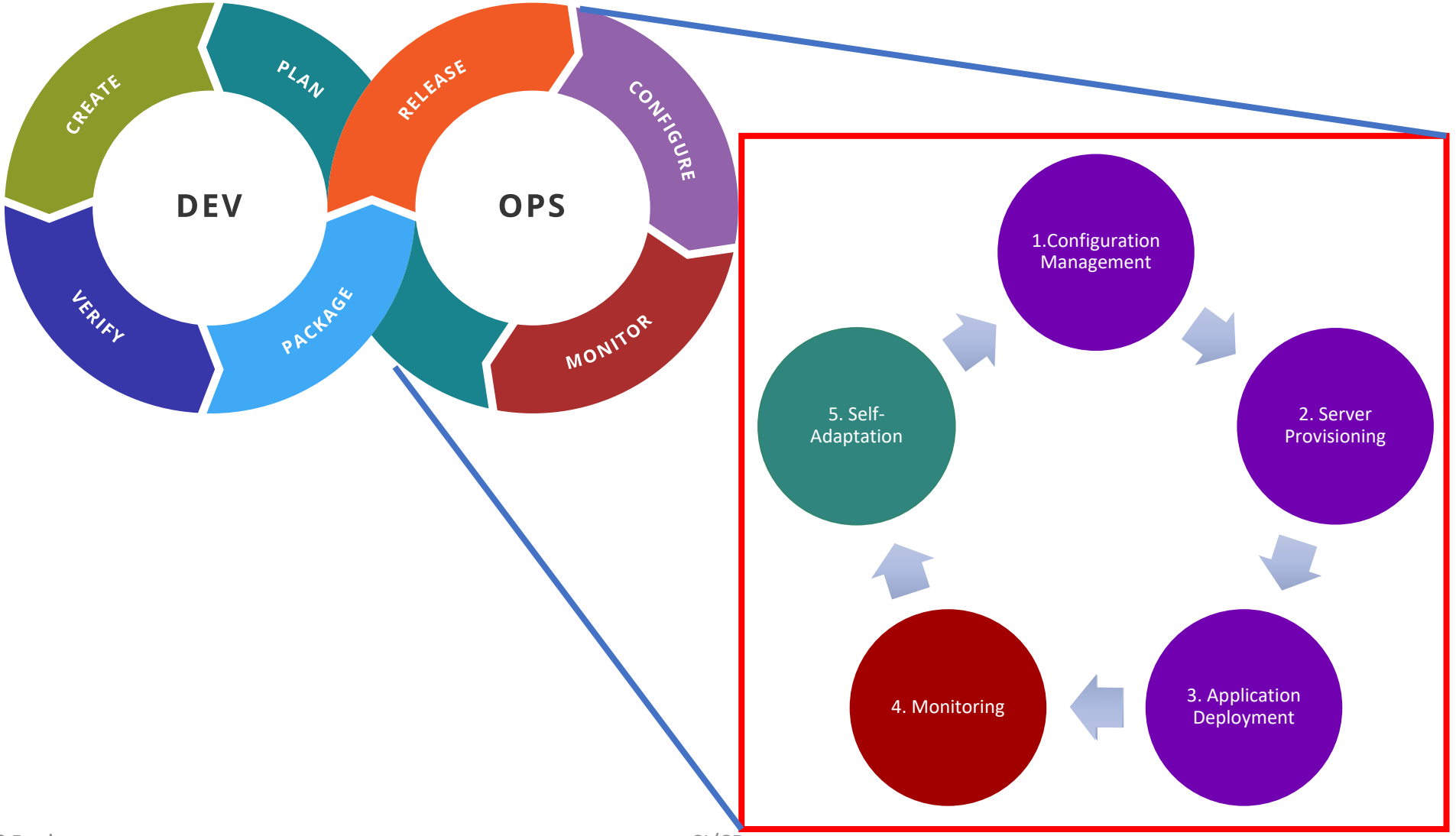


Image by Kharnagy (Own work) [CC BY-SA 4.0 (http://creativecommons.org/licenses/by-sa/4.0)], via Wikimedia Commons

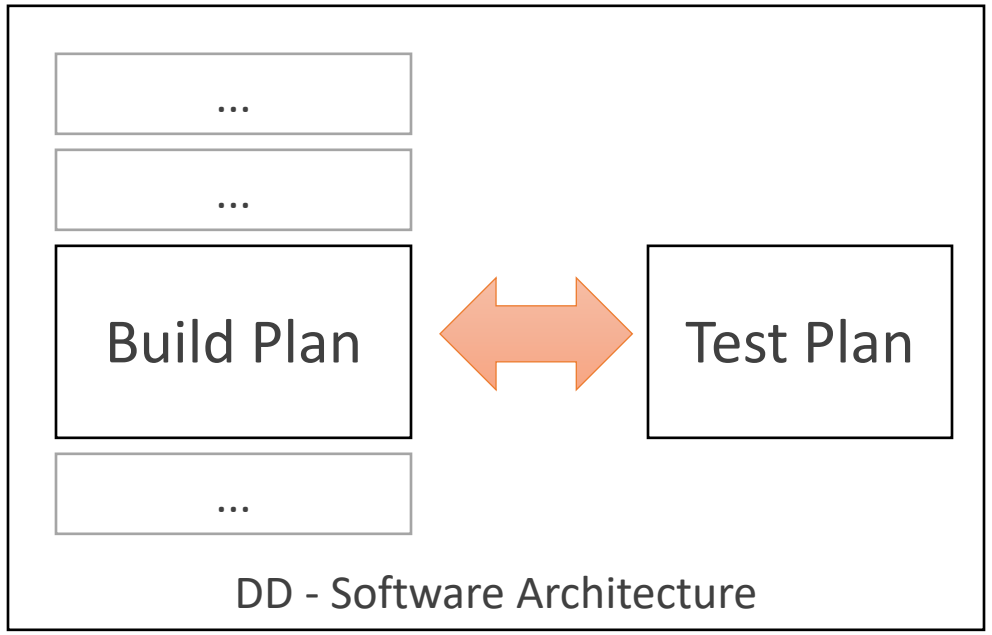
DevOps process and toolchain



A parenthesis on architecting and integration

- Architecting
  - Define the components of your system explicitly, document how they interact with each other => Software architecture
  - Make sure that components are sufficiently autonomous and cohesive
  - Study the cases of high coupling between components
  - Keep track of the mapping between architectural components and your code
- Integration
  - Use the information about the software architecture to define your integration plan

Integration and test plan



- Typically defined by the Design Document
- Build plan = defines the order of the integration
- Test plan = defines how to carry out integration testing
  - Must be consistent with the build plan!

Integration testing: strategies

POLITECNICO MILANO 1863

- Big bang: test only after integrating all modules together (not even a real strategy)
  - Pros
    - Does not require stubs, requires less drivers/oracles
  - Cons
    - Minimum observability, fault localization/diagnosability, efficacy, feedback
    - High cost of repair
      - Recall: Cost of repairing a fault increases as a function of time between the introduction of an error in the code and repair

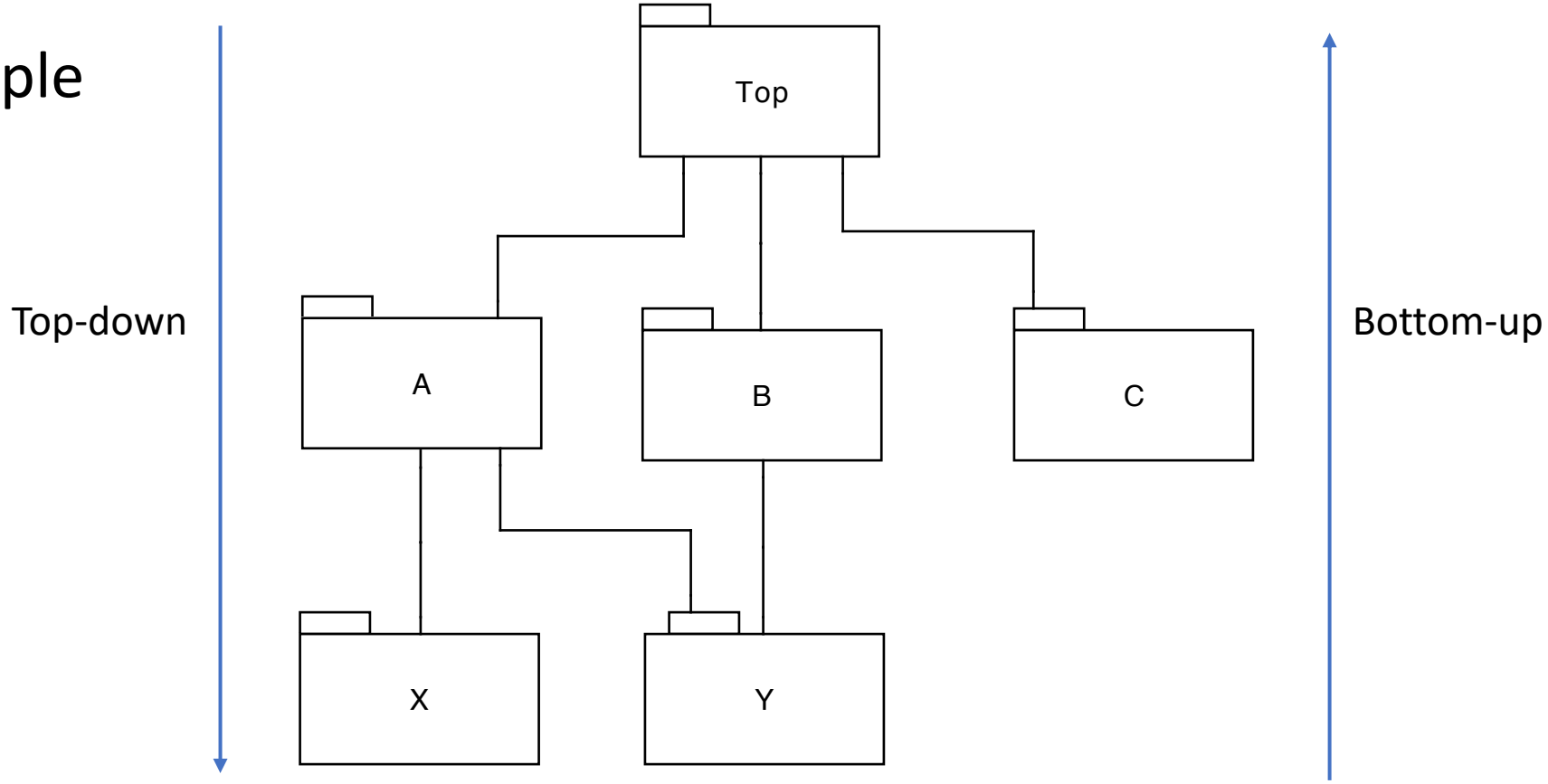
Integration testing: strategies

POLITECNICO MILANO 1863

- Iterative and incremental strategies
  - run as soon as components are released (not just at the end)
  - Hierarchical: based on the hierarchical structure of the system
    - Top-down
    - Bottom-up
  - Threads: a portion of several modules that offers a user-visible function
  - Critical modules

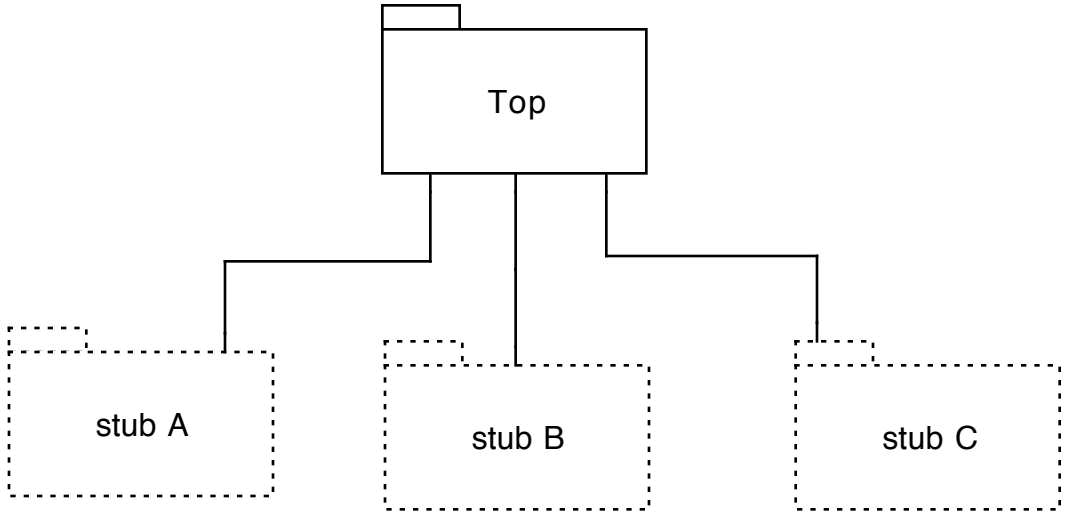
Integration testing: top-down vs bottom up

- Architecture example



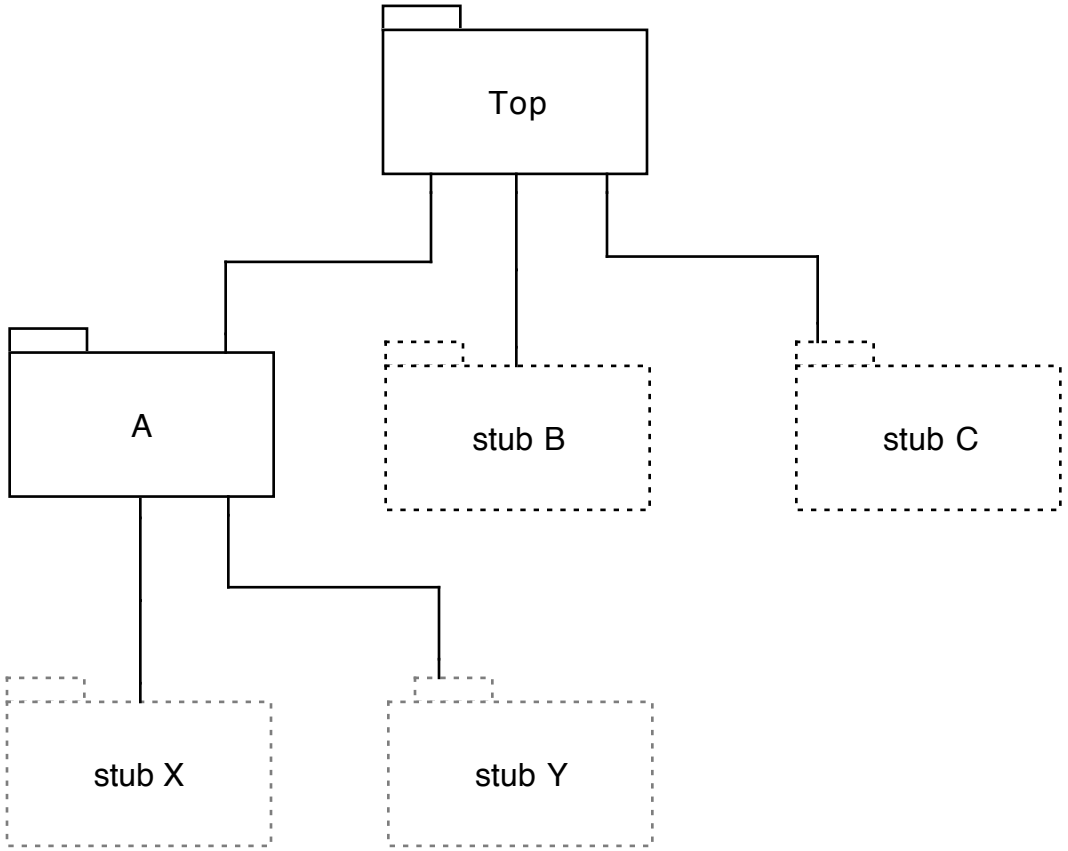
Integration testing: top-down

- Top-down strategy
  - Working from the top level (in terms of “use” or “include” relation) toward the bottom
  - We need stubs of used modules at each step of the process



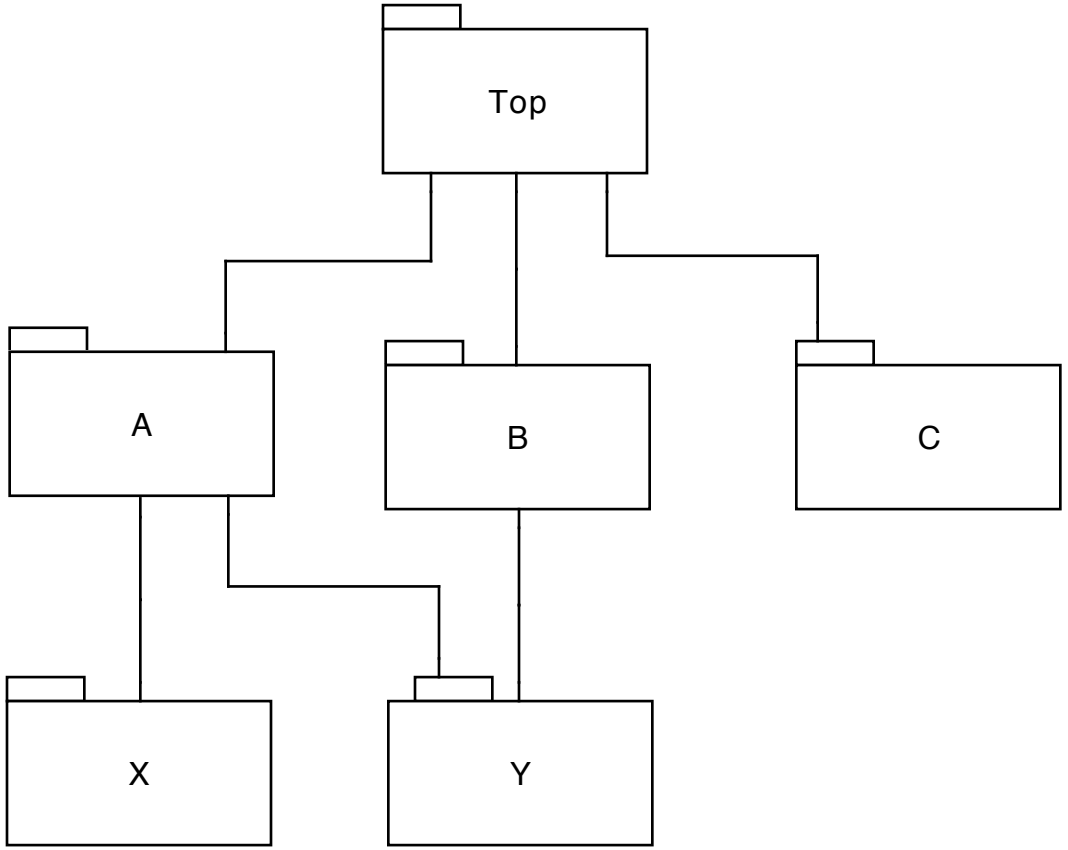
Integration testing: top-down

- Top-down strategy
  - As modules are ready (following the build plan) more functionality is testable
  - We replace some stubs and we need other stubs for lower levels



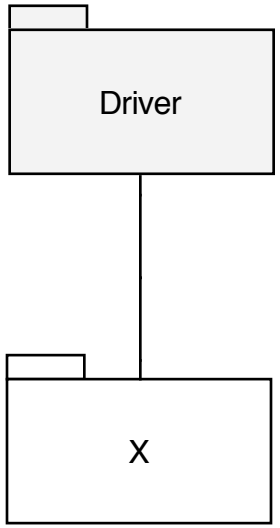
Integration testing: top-down

- Top-down strategy
  - When all modules are incorporated, the whole functionality can be tested



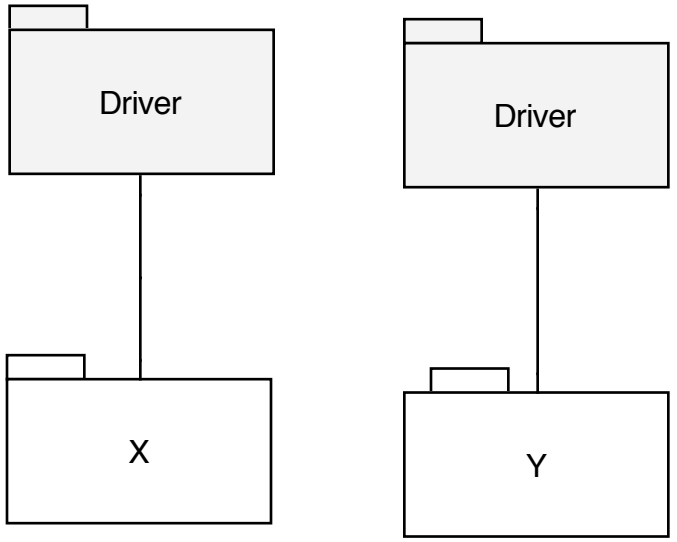
Integration testing: Bottom-up

- Bottom-up strategy
  - Starting from the leaves of the “uses” hierarchy
  - Does not need stubs



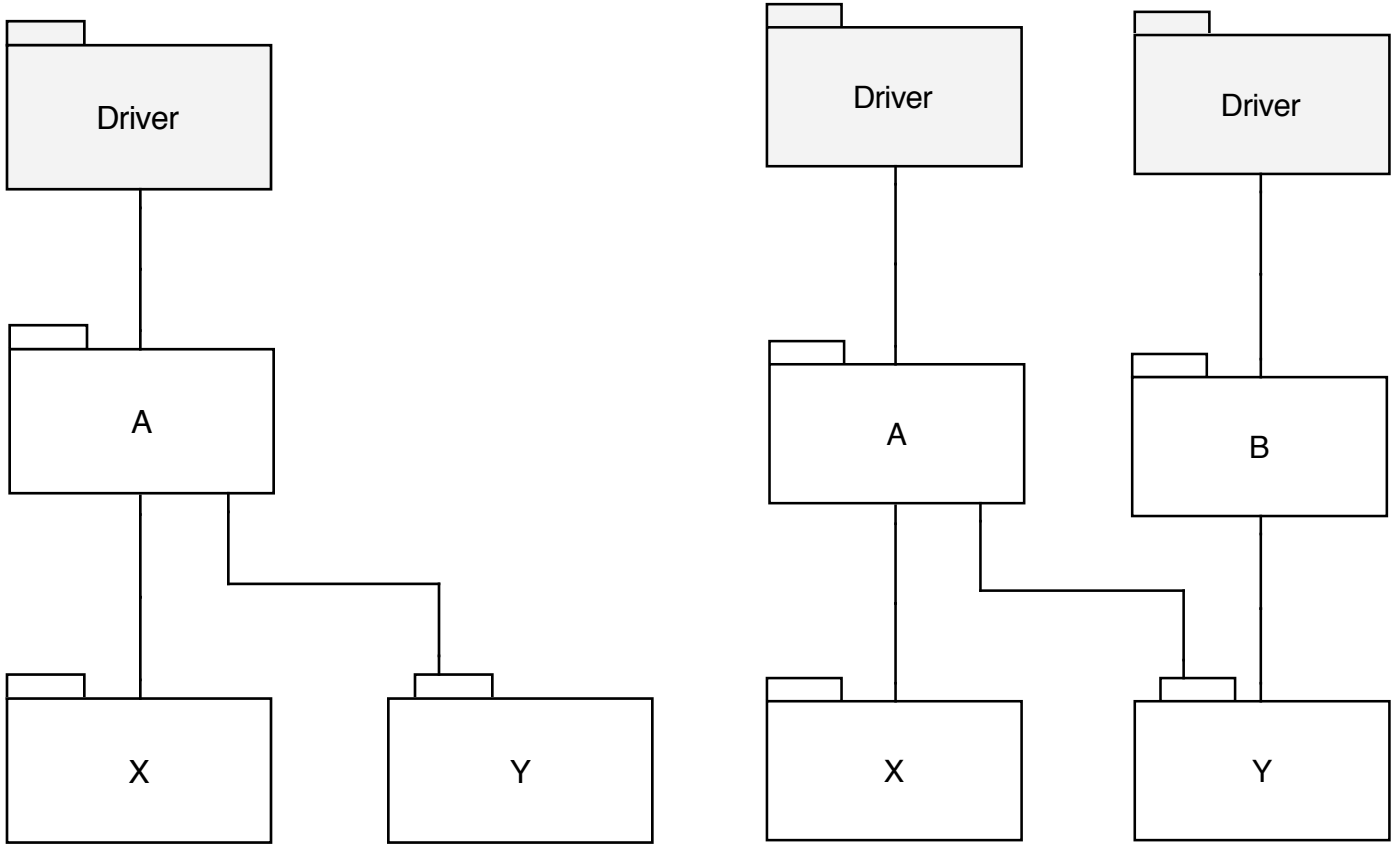
Integration testing: Bottom-up

- Bottom-up strategy
  - Starting from the leaves of the “uses” hierarchy
  - Does not need stubs
  - Typically requires more drivers: one for each module (as in unit testing)



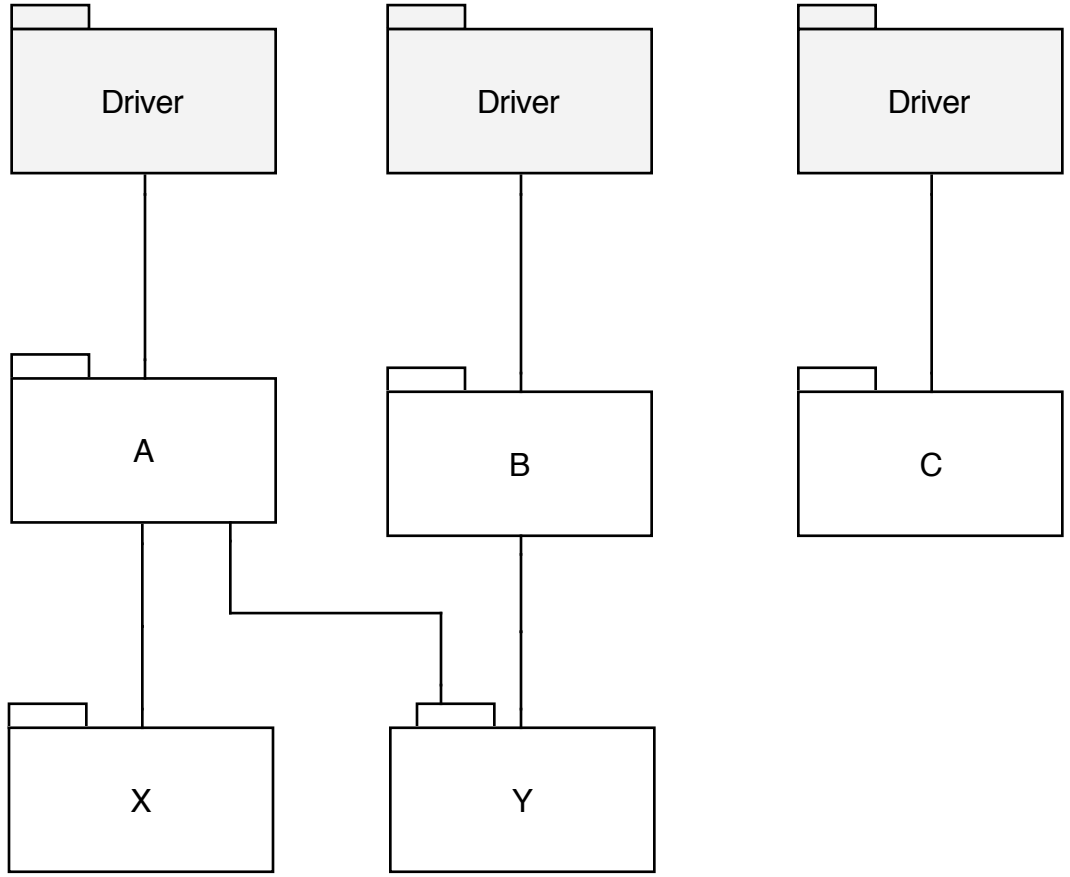
Integration testing: Bottom-up

- Bottom-up strategy
  - Newly developed module may replace an existing driver
  - New modules require new drivers



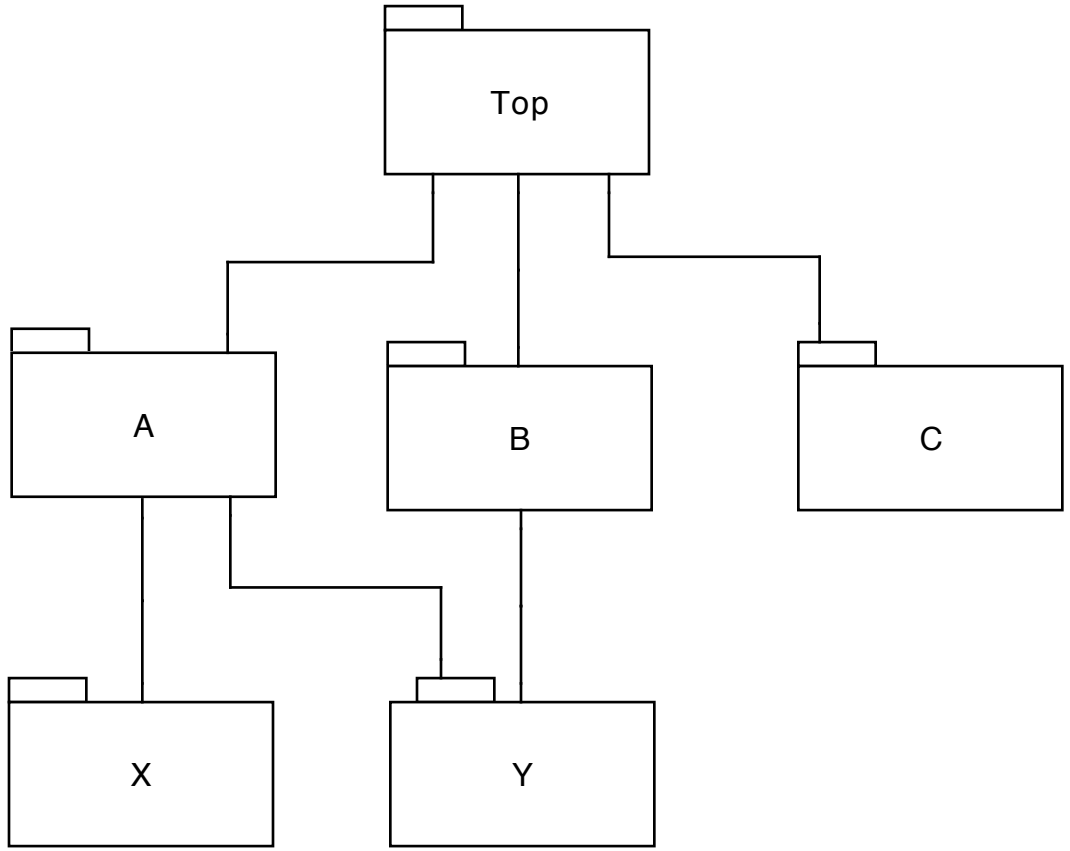
Integration testing: Bottom-up

- Bottom-up strategy
  - It may create several working subsystems



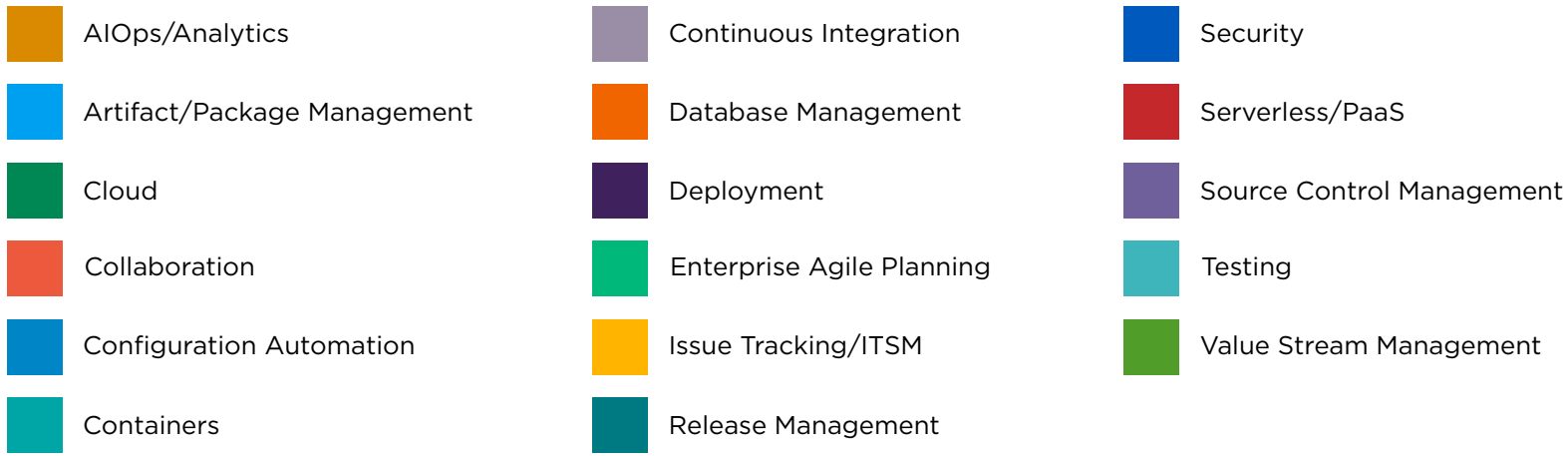
Integration testing: Bottom-up

- Bottom-up strategy
  - Working subsystems are eventually integrated into the final one



Automation - a large number of tools

https://digital.ai/learn/devops-periodic-table/

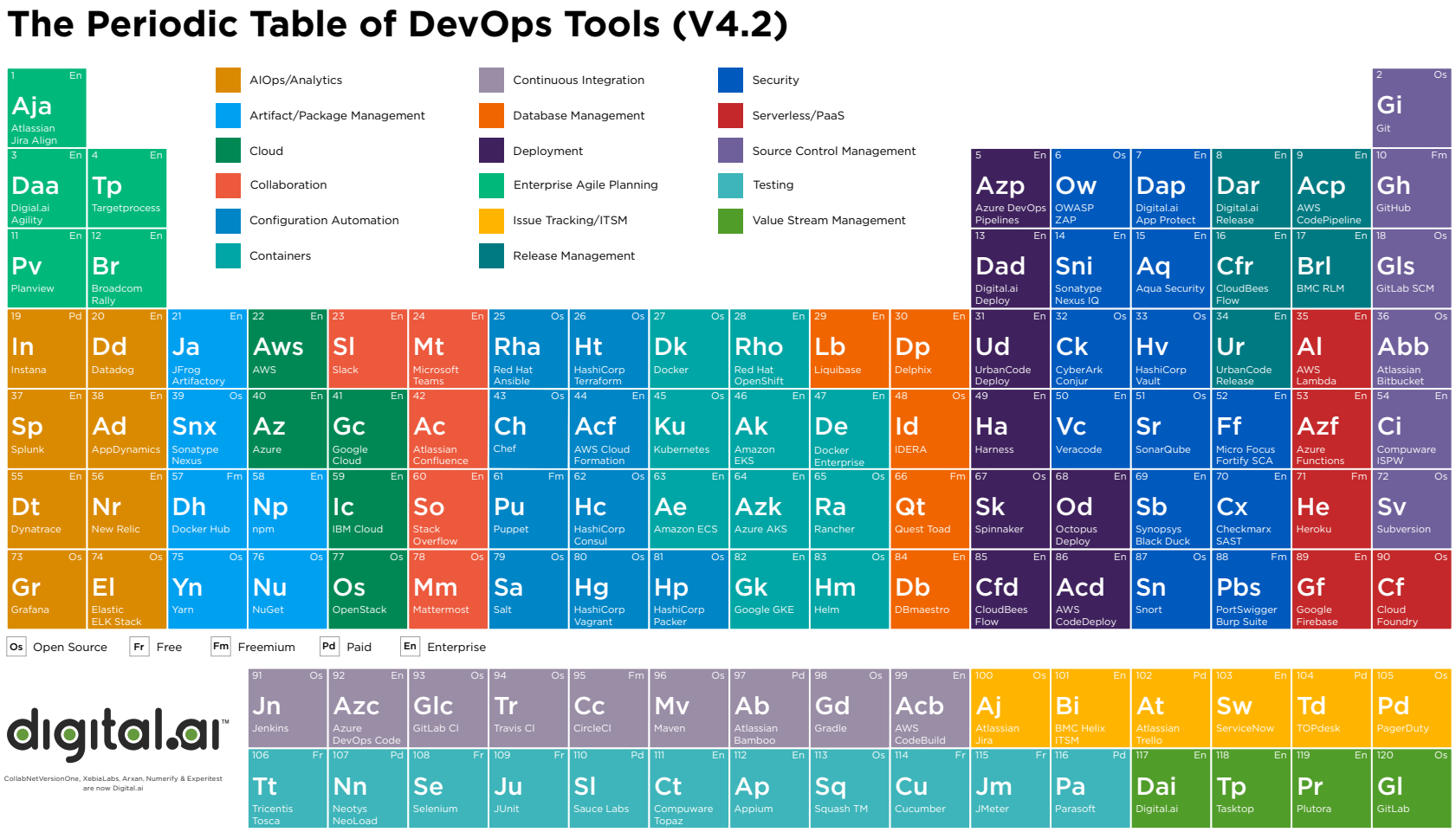


digital.ai™

CollabNetVersionOne, XebiaLabs, Arxan, Numerify & Experitest are now Digital.ai

Automation - a large number of tools

https://digital.ai/learn/devops-periodic-table/

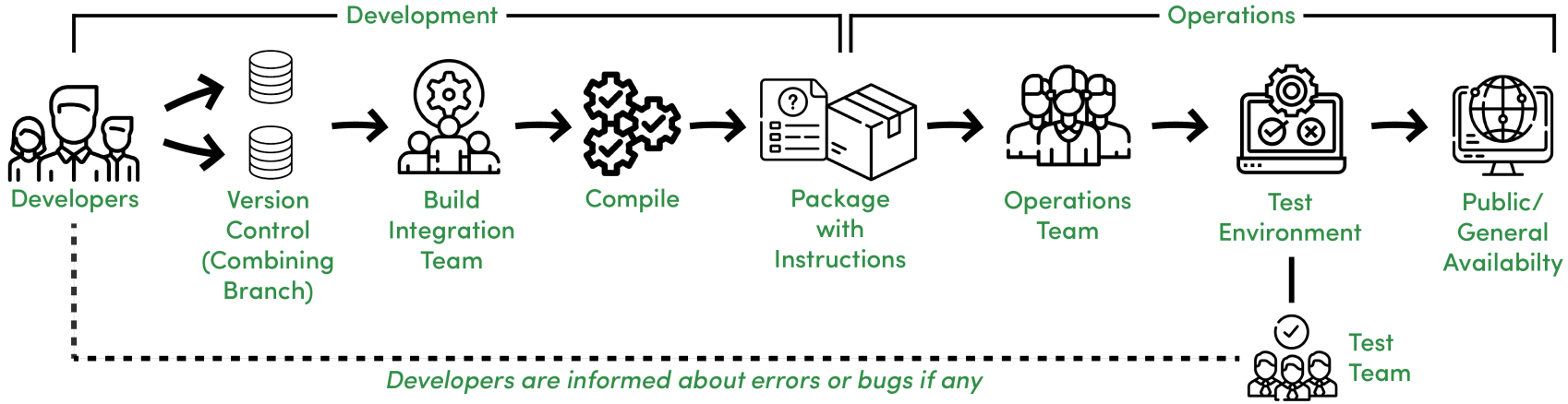


Exercise in breakout rooms

POLITECNICO MILANO 1863

- Think at your own organization and processes
- Answer to the following questions
  - To which DevOps organizational pattern/anti-pattern your organization is close?
  - How do you handle architecting and integration?
  - Do you use any automation tool in your projects?
  - What are the highest priority steps for your organization to fully embrace DevOps?

Course plan



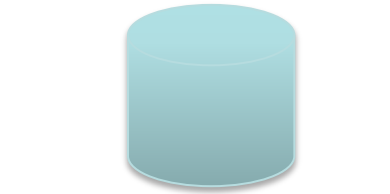
- Process
  - Development lifecycles
  - DevOps
- Automation
  - Version control & software configuration management
  - Static analysis
  - Testing automation
  - Pipeline control tools
- Hands-on
  - Testing automation
  - Creating pipelines for C and Python programs using GitHub Actions

Software Configuration Management

POLITECNICO MILANO 1863

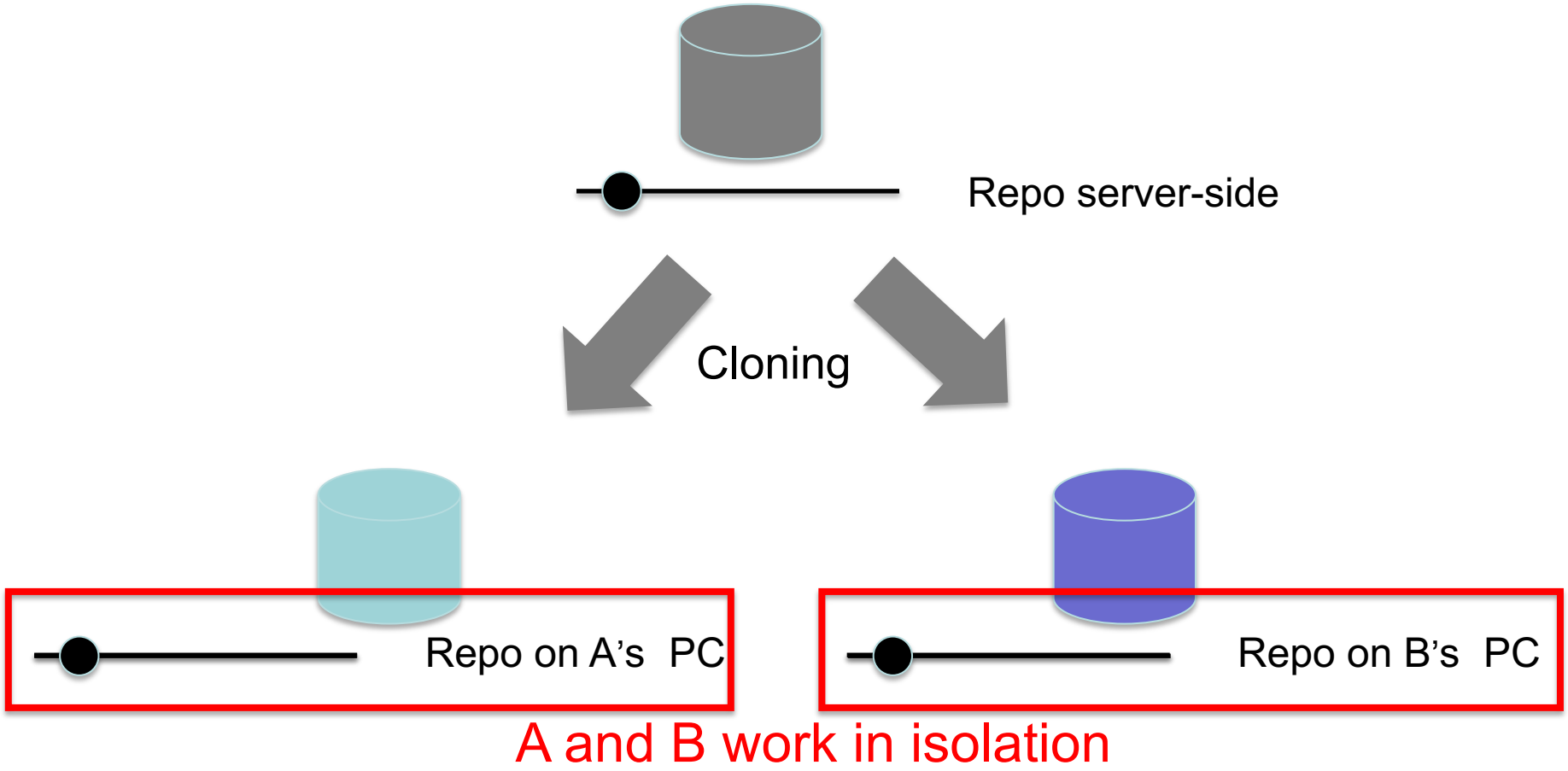
- Roles:
  - Manage change in a controlled manner
  - Sharing software artifacts
- Centralized
  - Central repository, selective access to project tree
  - Tools: CVS, SVN, Team Foundation Server, Rational ClearCase, Rational TeamConcert...
- Decentralized
  - Distributed repositories, replicated repositories
  - Tools: Git, Mercurial

Basic approach to use a decentralized CM

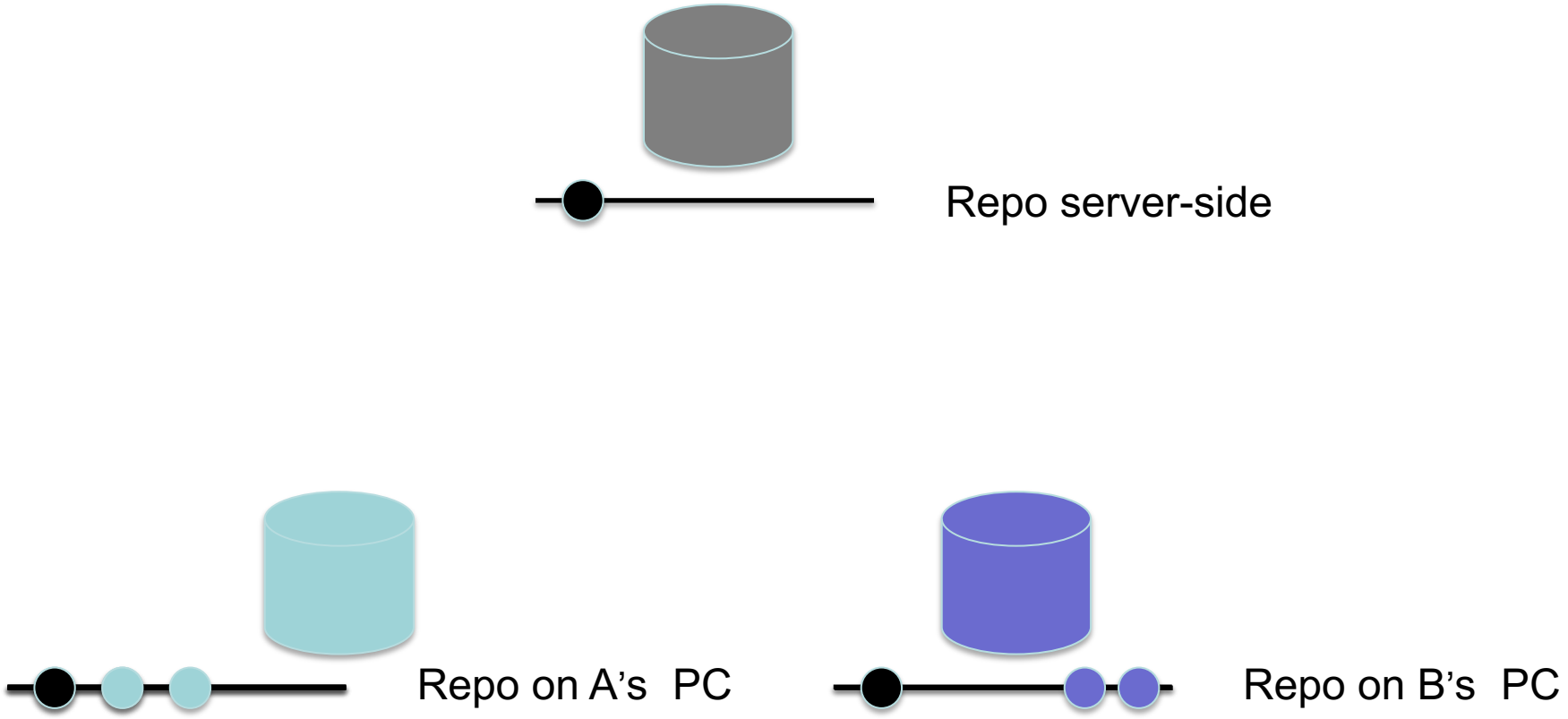


Repo server-side

Cloning to local repositories

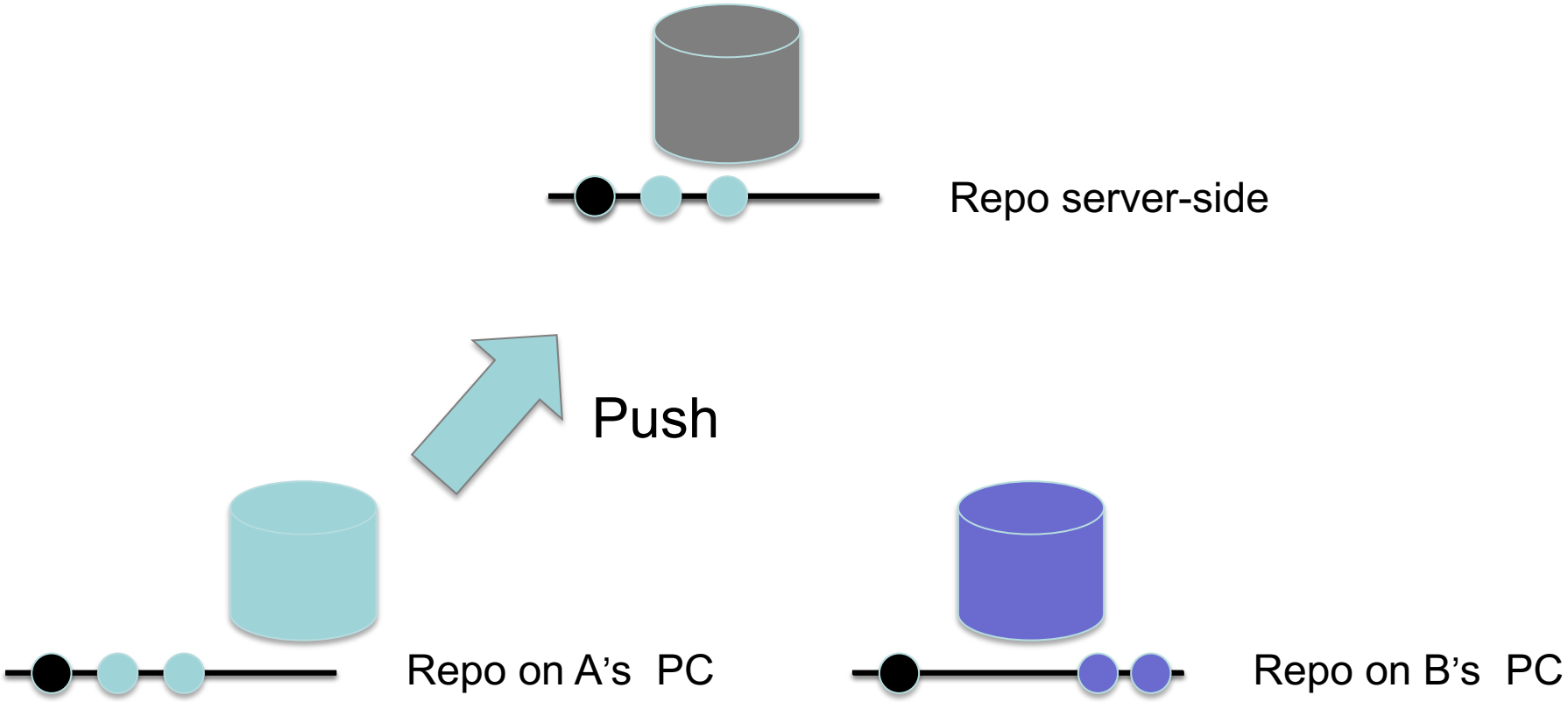


Local work



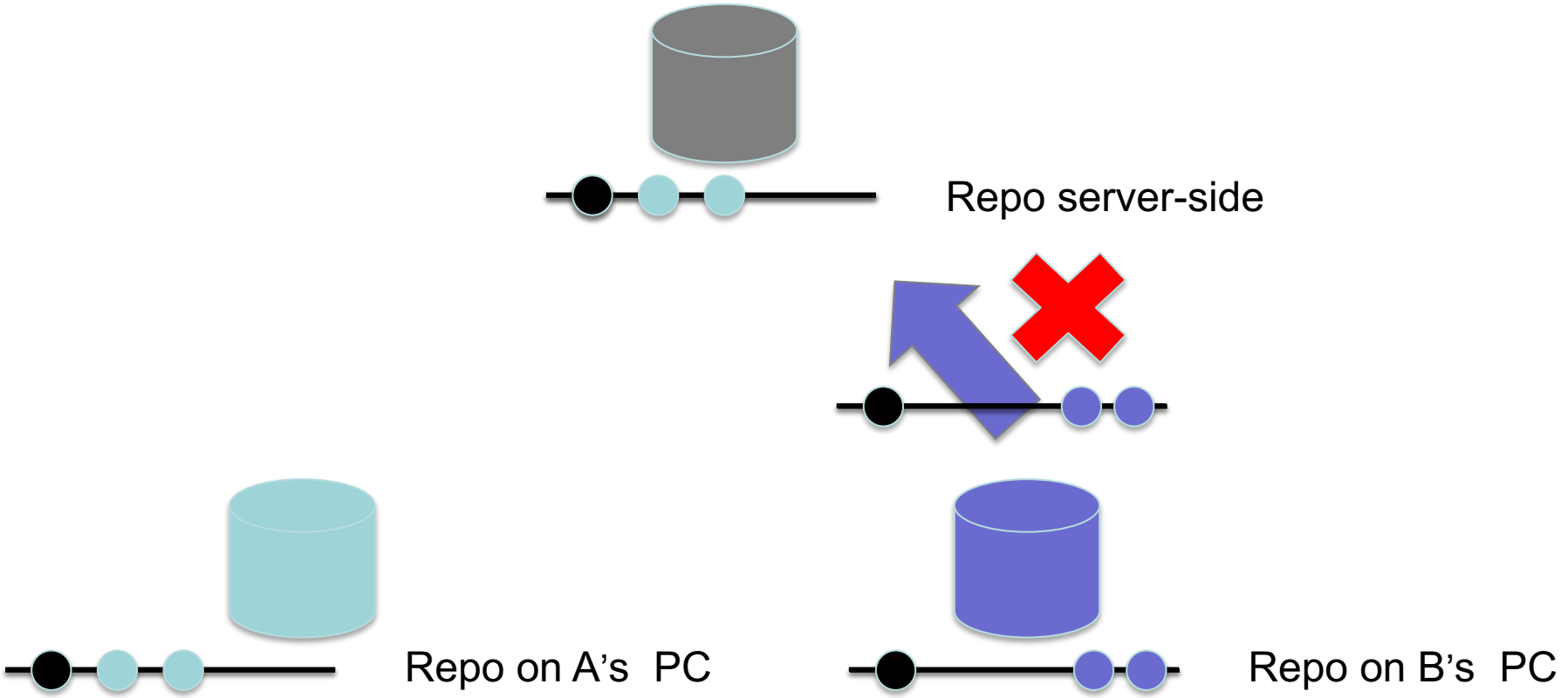
Locally code, compile, test, commit

Integrating changes to central repository



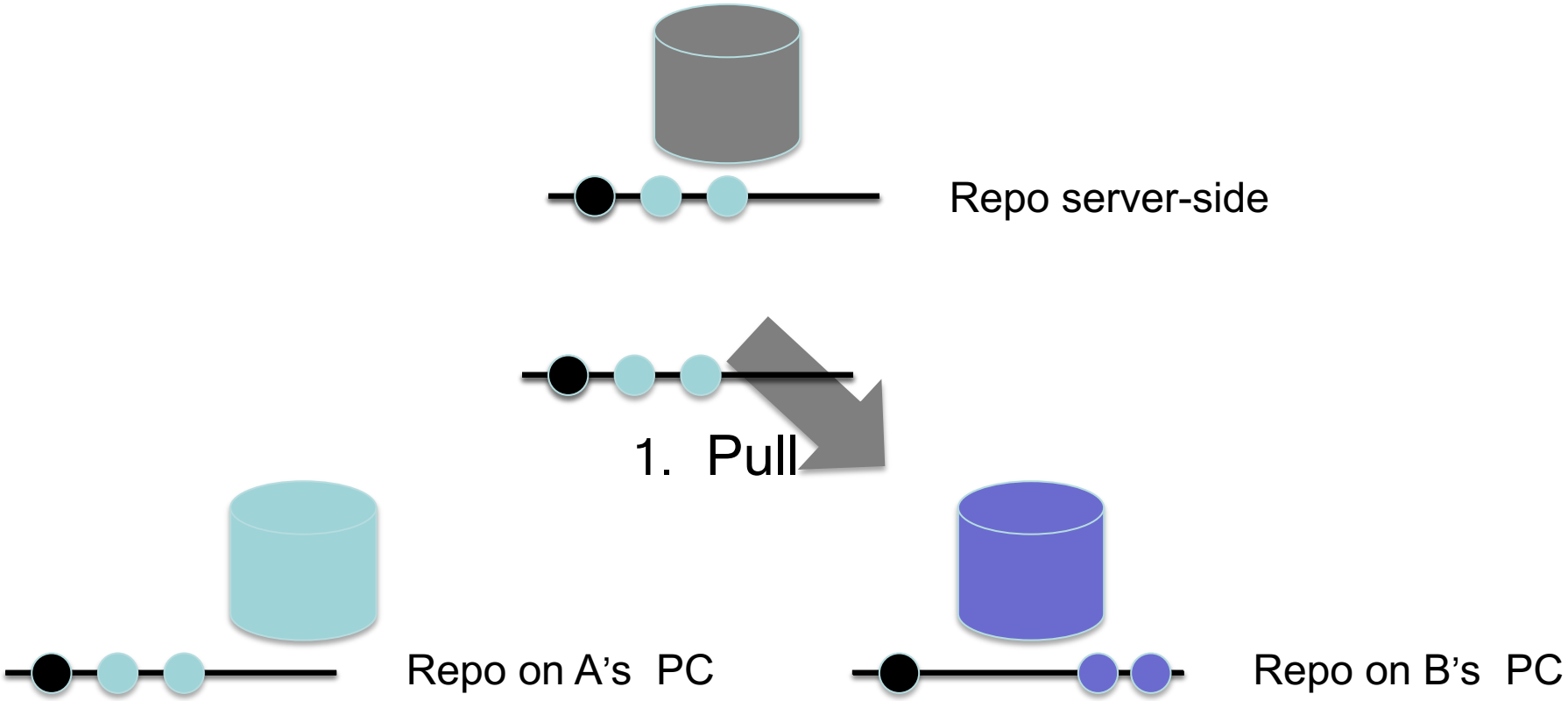
Push changes to central repository master branch

Integrating changes to central repository



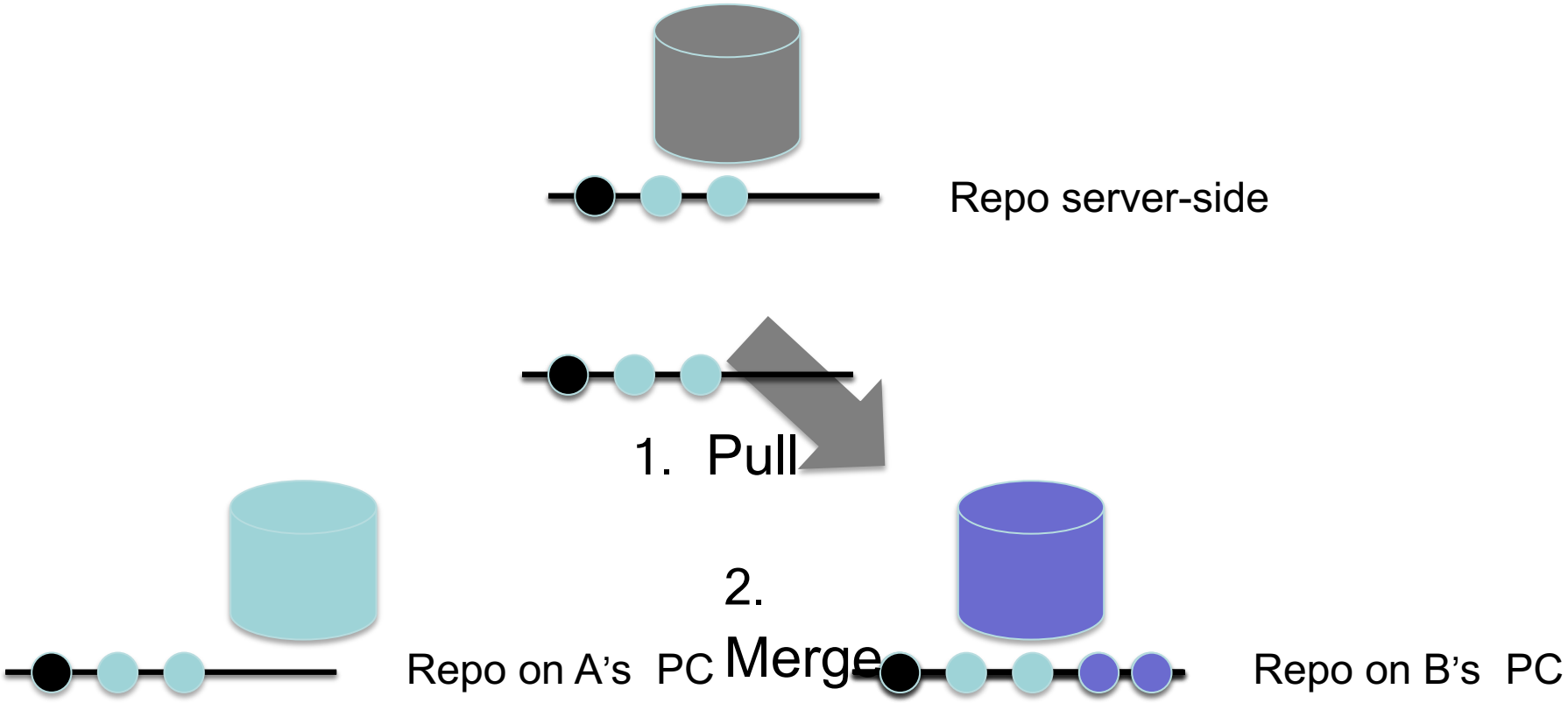
Push cannot be completed due to changes in central repository master thread

Integrating changes to central repository



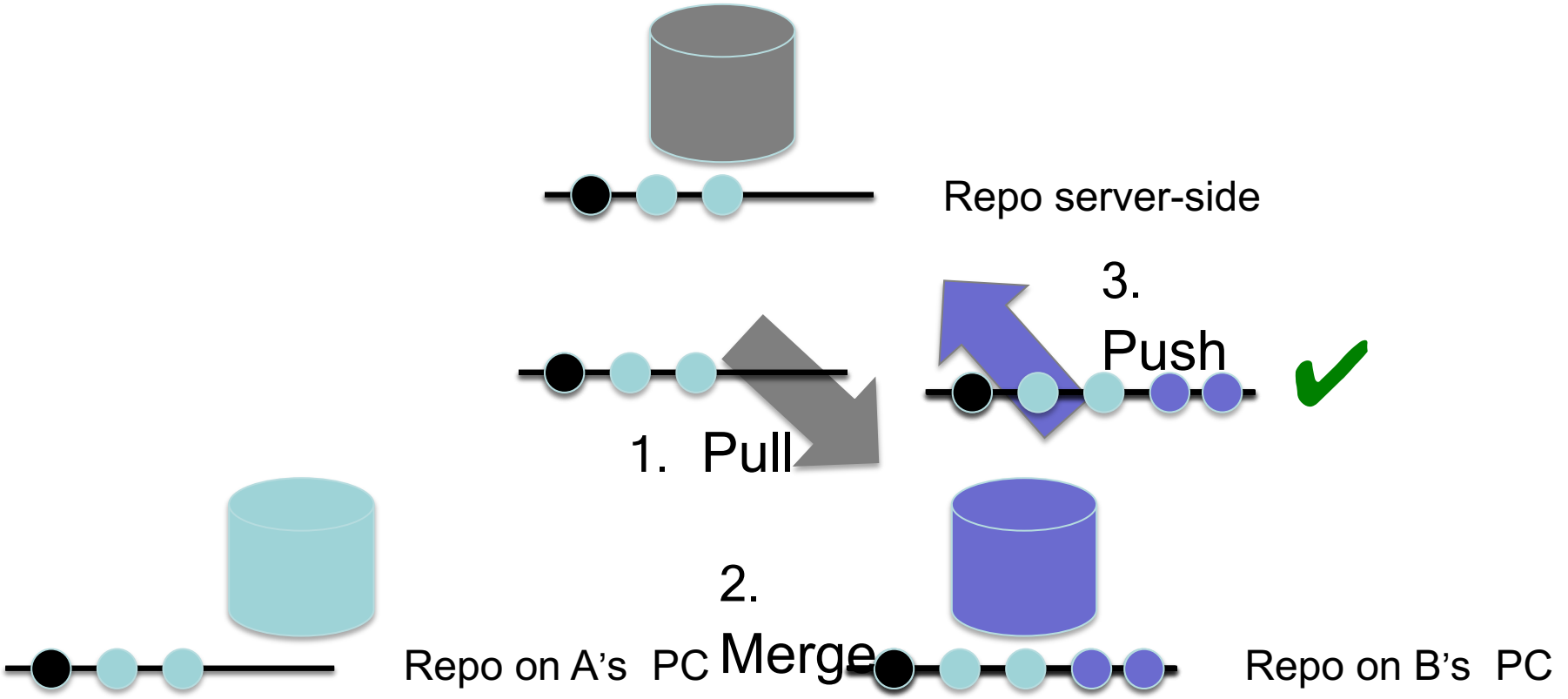
Push cannot be completed due to changes in central repository master thread

Integrating changes to central repository



Push cannot be completed due to changes in central repository master thread

Integrating changes to central repository



Push cannot be completed due to changes in central repository master thread

Main Git commands

POLITECNICO MILANO 1863

- git clone <repository URL>: create a local copy of the repository on the server
- git pull: get new updates from the server and merges them with the local changes
- git add: add modified files to the local repository
- git commit: finalizes the changes in the local repository
- git push: transfers local commits to the server
- git status: it shows the status of the local repository (if there are files not included in the repository, if it not aligned with the server-side one)
- References
  - https://education.github.com/git-cheat-sheet-education.pdf
  - https://git-scm.com/doc

Some examples of free of charge clients

POLITECNICO MILANO 1863

- Atlassian SourceTree
  - OS X, Windows
  - Integrate with any git repository
- Github desktop
  - OS X, Windows
  - Specific for Github
  - Simple to use
- SmartGit (free for non-commercial use)
  - OS X, Windows, Linux
  - Integrate with Github, GitLab, Bitbucket, or Stash
  - Advanced features

Exercise in breakout rooms

POLITECNICO MILANO 1863

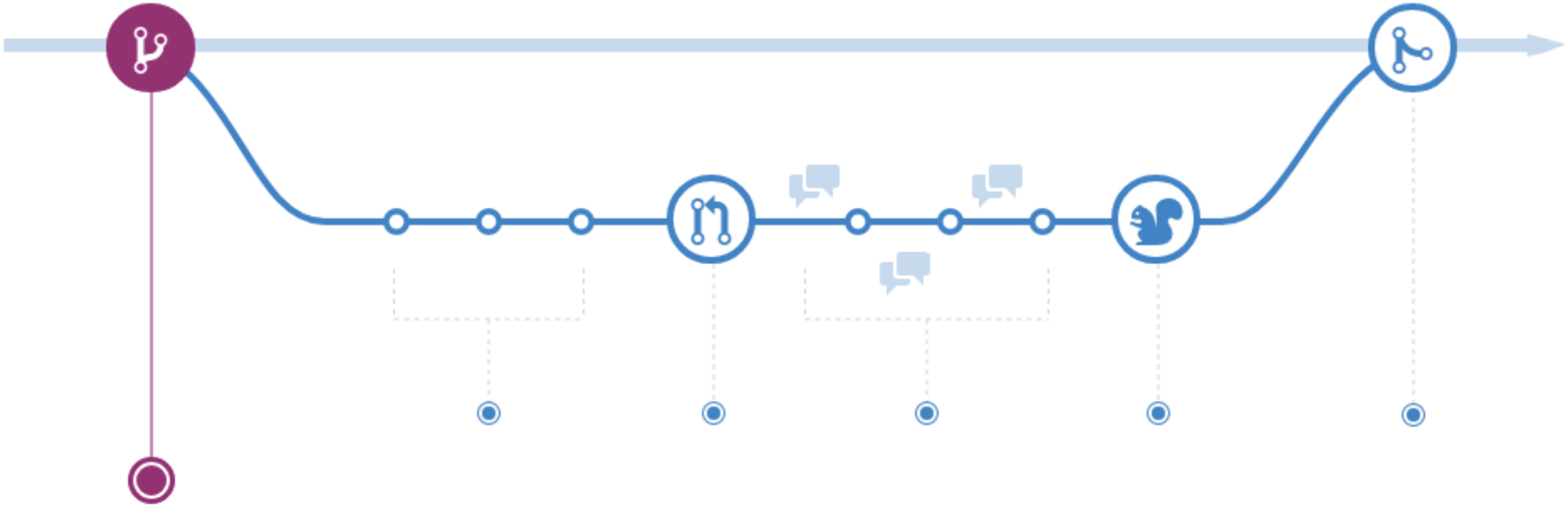
- Create a repository on GitHub or on your preferred SCM
- Share it with your colleagues
- Experiment with concurrent work
  - Work on your local repo, push changes (remember to always pull before push), merge when needed

Workflows

POLITECNICO MILANO 1863

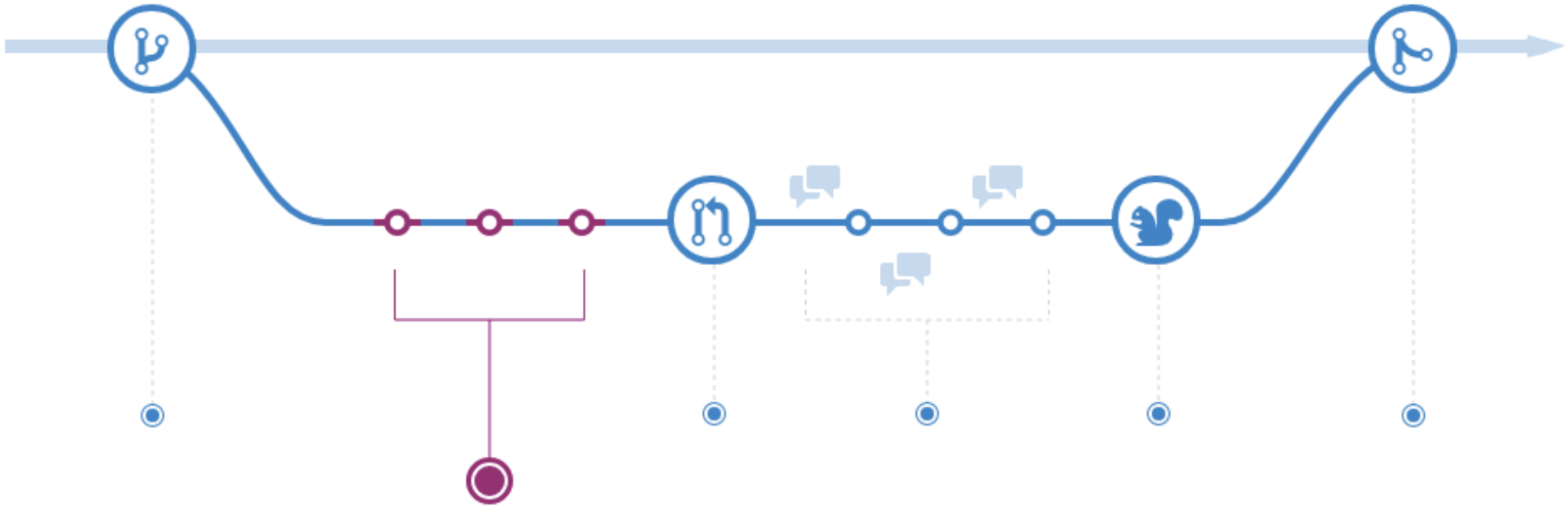
- Define the way an organization uses the CM for a specific project
- Main principles
  - The *master* repository must be kept as clean as possible
  - The distinction between local and central repository is not enough in case of multiple contributors
  - On the central repository we can create one *master* repository and as many *branches* as we want
  - A new contribution becomes part of the master only if the whole team agrees

GitFlow https://guides.github.com/introduction/flow/



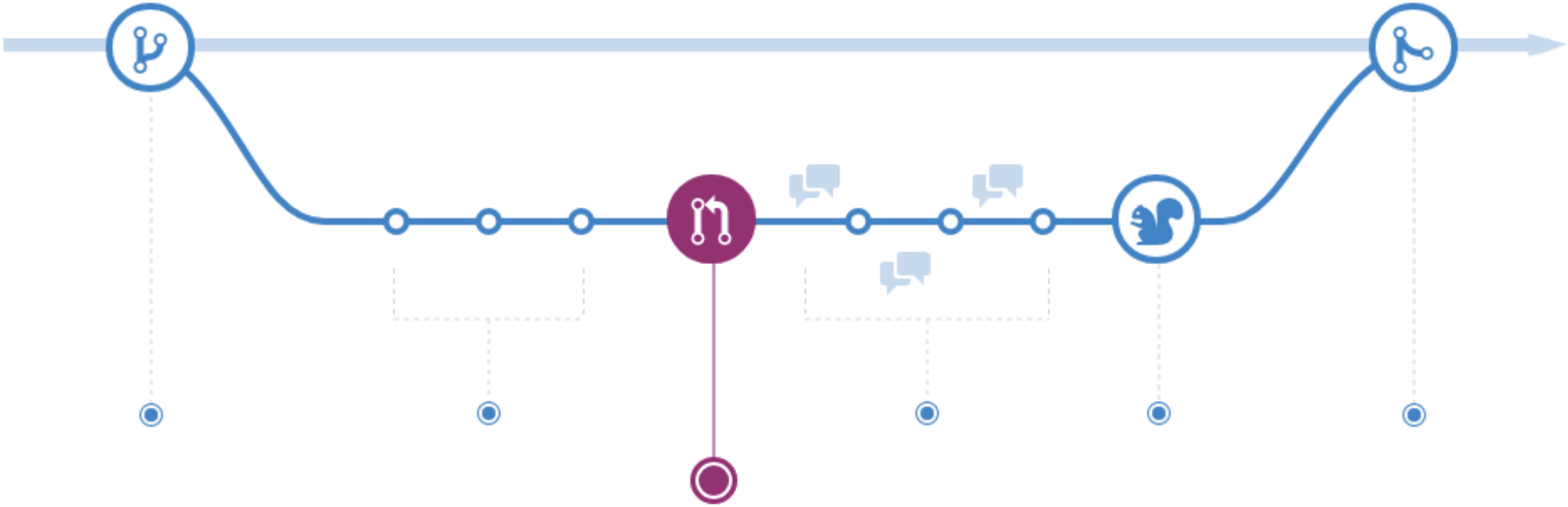
- When you need to develop a new feature or a new idea, create a new *branch*
  - A new independent sub-repository where you can experiment without impacting on what is in the master

GitFlow https://guides.github.com/introduction/flow/



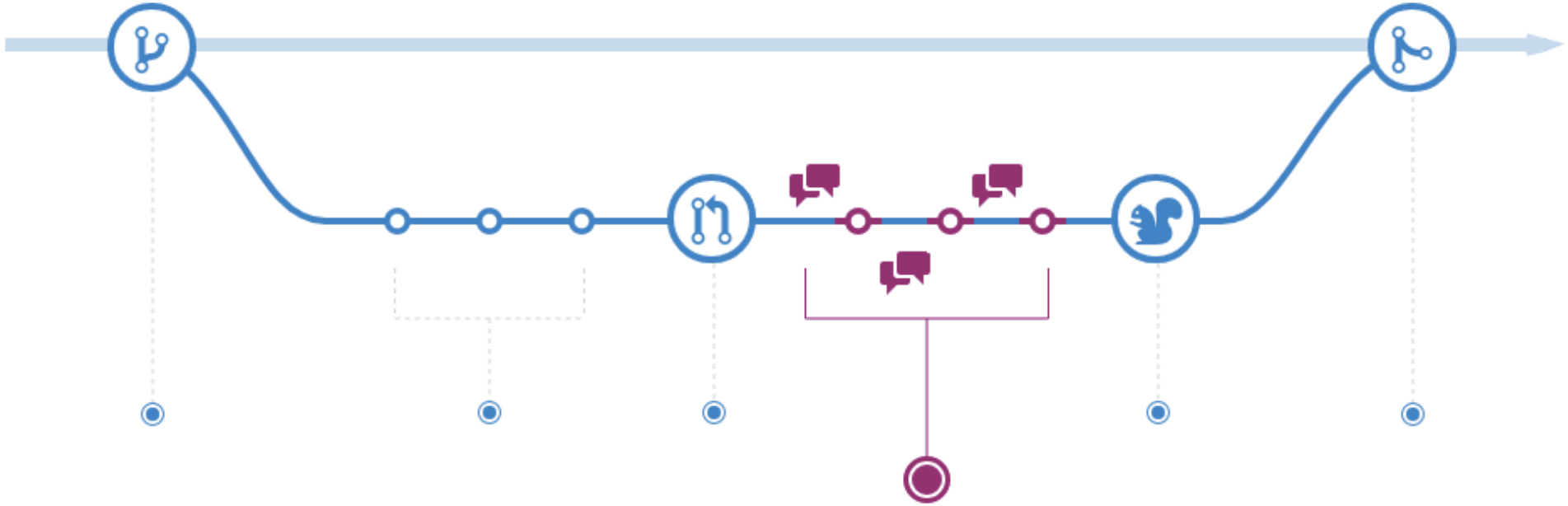
- New versions of software are produced in the branch

GitFlow https://guides.github.com/introduction/flow/



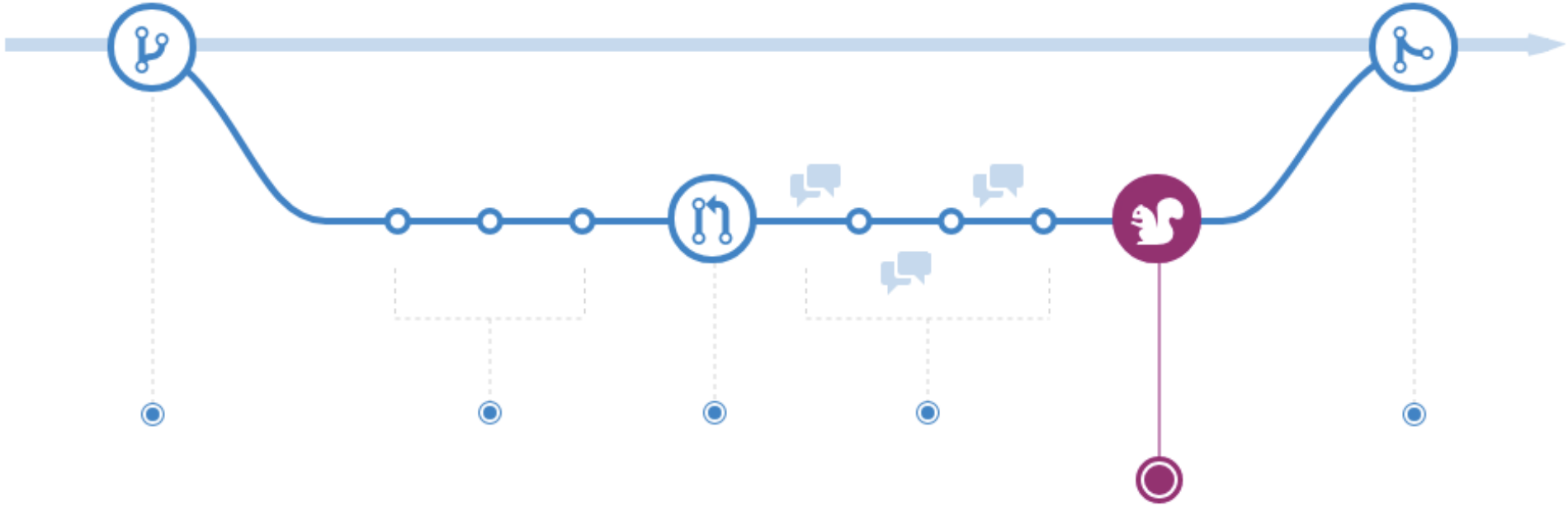
- When the developers are happy, they can issue a *pull request*
  - The sub-team is asking the whole team to review the changes and decide whether to accept them or not
  - The sub-team has already merged any local change in the branch server-side

GitFlow https://guides.github.com/introduction/flow/



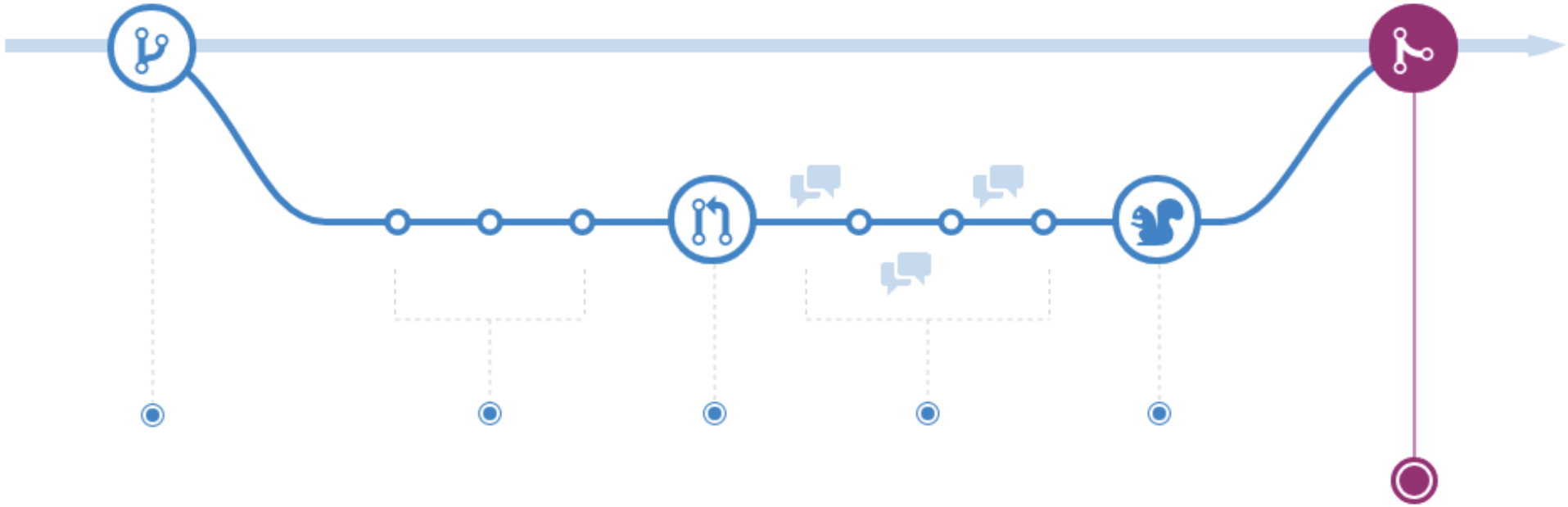
- A discussion starts
- During the discussion, new versions can be created in the branch

GitFlow https://guides.github.com/introduction/flow/



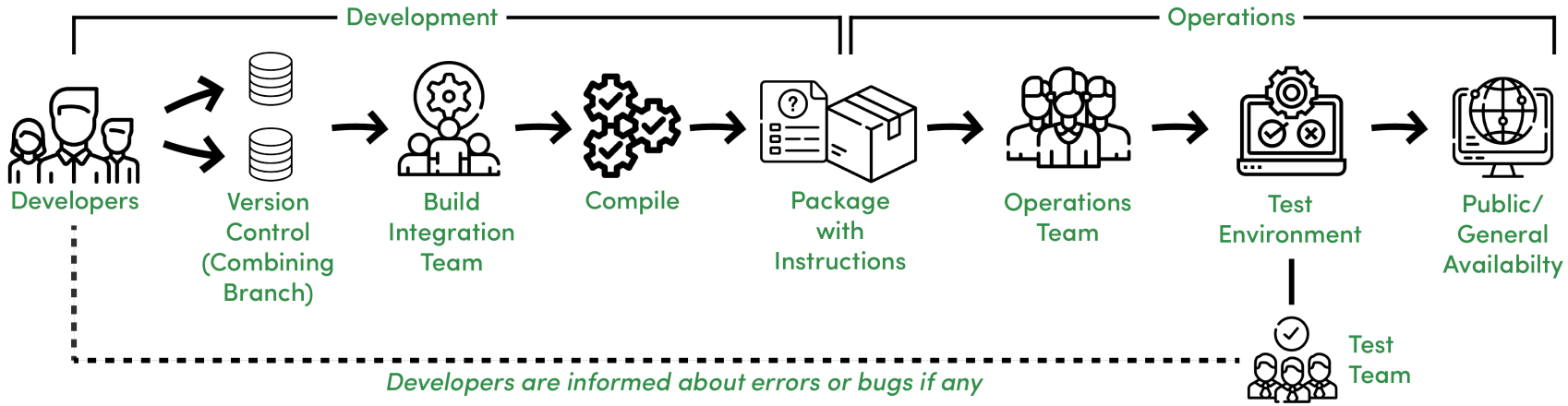
- The code is deployed and tested

GitFlow https://guides.github.com/introduction/flow/



- Then finally, the branch is merged in the master

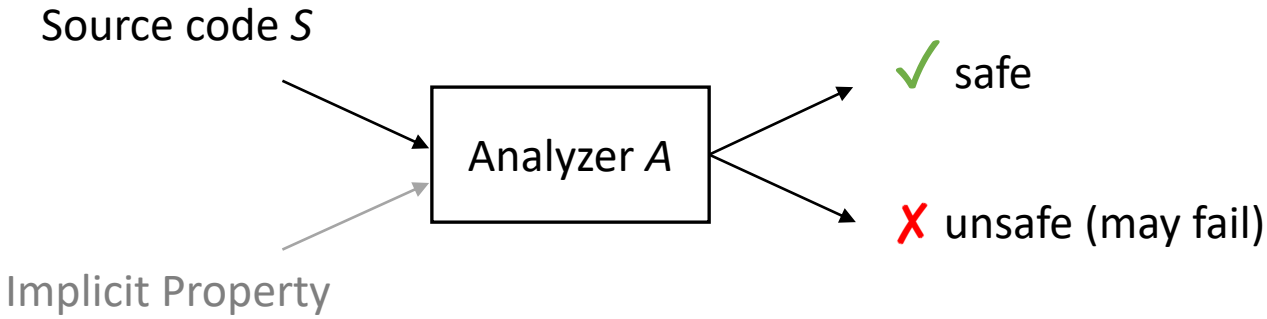
Course plan



- Process
  - Development lifecycles
  - DevOps
- Automation
  - Version control & software configuration management
  - Static analysis
  - Testing automation
  - Pipeline control tools
- Hands-on
  - Testing automation
  - Creating pipelines for C and Python programs using GitHub Actions

Static Analysis

- The very idea
  - Analyzes the source code
  - Each analyzer targets a fixed set of **hard-coded** (pre-defined, not custom) properties
  - Completely **automatic**
  - The output reports
    - **Safe** = no issues
    - **Unsafe** = potential issues



Static Analysis: properties

POLITECNICO MILANO 1863

- Checked properties are often general safety properties (absence of certain conditions that may yield errors)
- Examples:
  - No overflow for integer variables
  - No type errors
  - No null-pointer dereferencing
  - No out-of-bound array accesses
  - No race conditions
  - No useless assignments
  - No usage of undefined variables
  - No execution of specific paths

How to automate these checks?

- Derive the control flow diagram
- Identify points where variables are defined and used
  - Note: i++ contains both a use and a definition for i
- Identify def-use pairs

- Example

```
1 int i, k = 0;
2 i = k;
3 while (i<10)
4   i++;
```

def-use pairs for k
<1, 2>
def-use pairs for i
<2, 3>, <2, 4>, <4, 3>, <4, 4>

Def-use pairs through an example

POLITECNICO MILANO 1863

- Consider the following fragment of code:

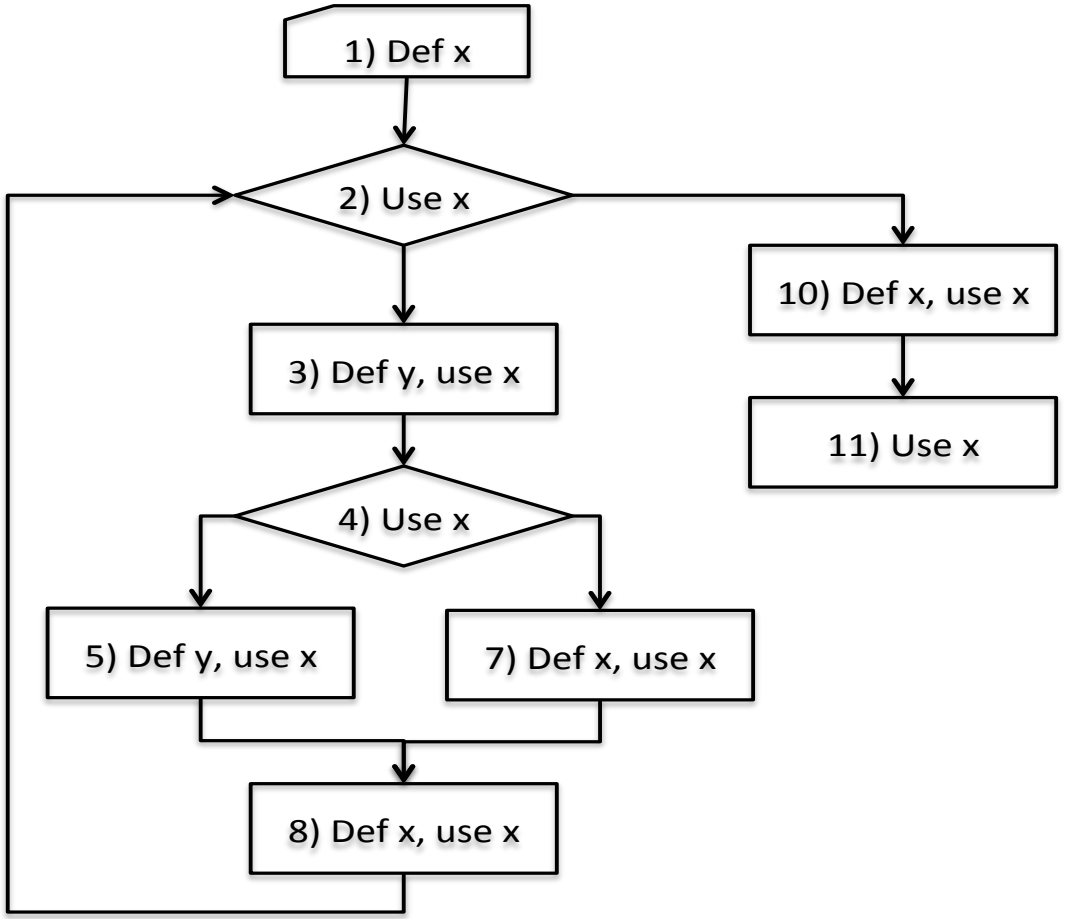
```
int foo() {
1   x = input();
2   while (x > 0) {
3     y = 2 * x;
4     if (x > 10)
5       y = x - 1;
6     else
7       x = x + 2;
8     x = x - 1;
9   }
10  x = x - 1;
11  return x;
}
```

You are to accomplish the following:

1. draw the control flow graph of the program;
2. provide the use-definition information for variables x and y;
3. provide the def-use pairs
4. point out a potential issue with this code that data flow analysis would be able to spot;

1., 2. Control flow graph and def/use

```
int foo() {
1   x = input();
2   while (x > 0) {
3     y = 2 * x;
4     if (x > 10)
5       y = x - 1;
6     else
7       x = x + 2;
8     x = x - 1;
9   }
10  x = x - 1;
11  return x;
}
```



3, 4. Def-use pairs

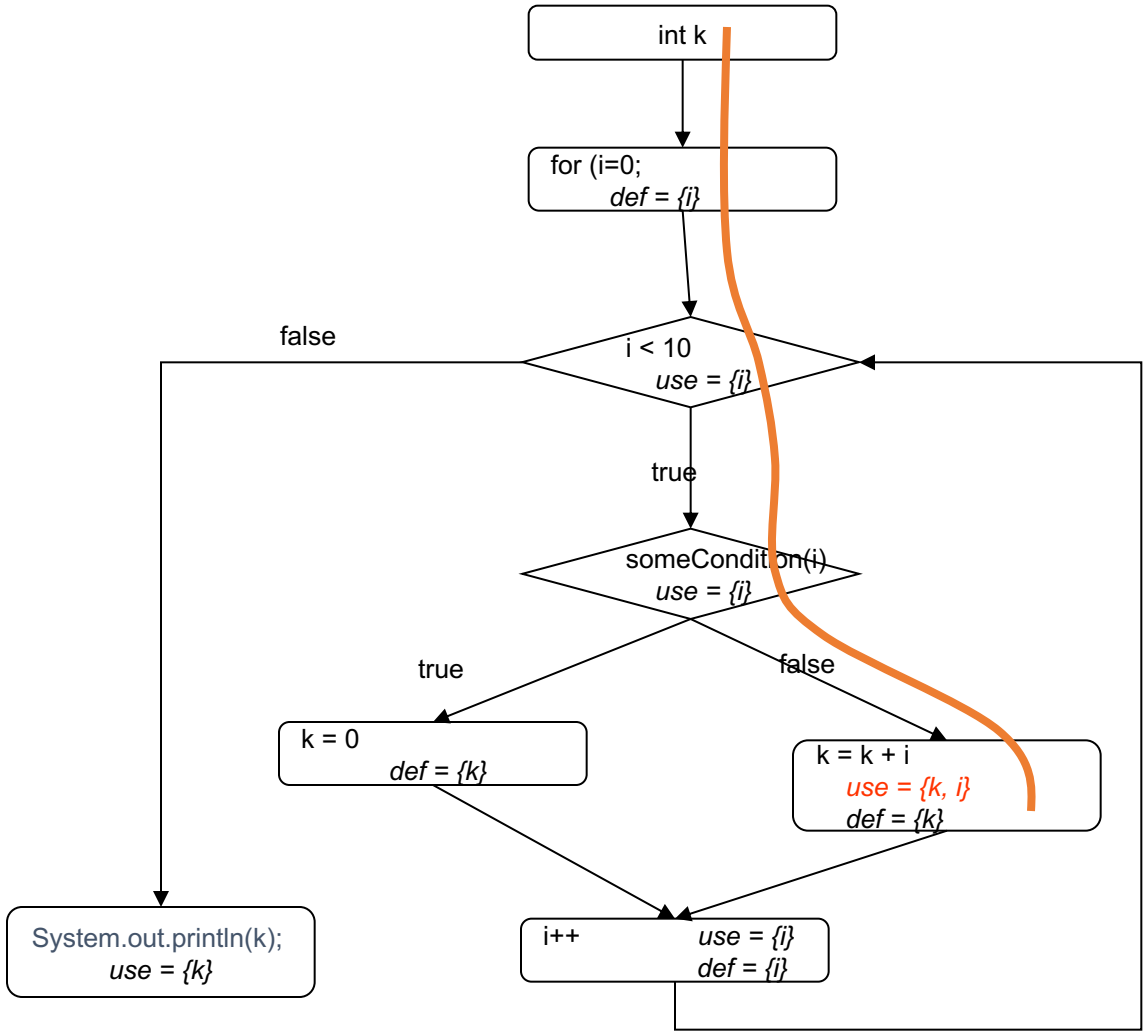
x	**<1, 2> <1, 3> <1, 4> <1, 5> <1, 7> <1, 8> <1, 10> <7,8> <8, 2> <8, 3> <8, 4> <8, 5> <8, 7> <8, 8> <8, 10> <10, 11>**
y	<3, ?>, <5, ?>

The potential issue is a useless assignment

Is it guaranteed that a variable is initialized when used?

```
static void questionable() {
  int k;
  for (int i=0; i<10; i++)
        if (someCondition(i))
              k = 0;
        else k = k+i;
  System.out.println(k);
}
```

- Assuming that someCondition yields false, k is not initialized
- Maybe in practice this does not happen
- Dataflow is a "pessimistic" tool



Static analysis: succesful stories

POLITECNICO MILANO 1863

[2017] *"Our strategy at Uber has been to use static code analysis tools to prevent null pointer exception crashes."*

Engineering NullAway, Uber's Open Source Tool for Detecting NullPointerExceptions on Android
https://www.uber.com/en-IT/blog/nullaway/

[2013] *"Each month, hundreds of potential bugs identified by Facebook Infer are fixed [. . .] before they are [. . .] deployed to people's phones."*

Facebook buys code-checking Silicon Roundabout startup Monoidics
https://www.theguardian.com/technology/2013/jul/18/facebook-buys-monoidics

Static analysis tools

POLITECNICO MILANO 1863

- Various tools available
- The analyses are language-specific but many tools support multiple programming languages
- The first static analysis tool was a Unix utility, Lint, developed in 1978 for C programs. From this, simple static analysis is also called linting
- Lists of currently available tools are available from various sources:
  - https://en.wikipedia.org/wiki/List_of_tools_for_static_code_analysis
  - https://github.com/analysis-tools-dev/static-analysis

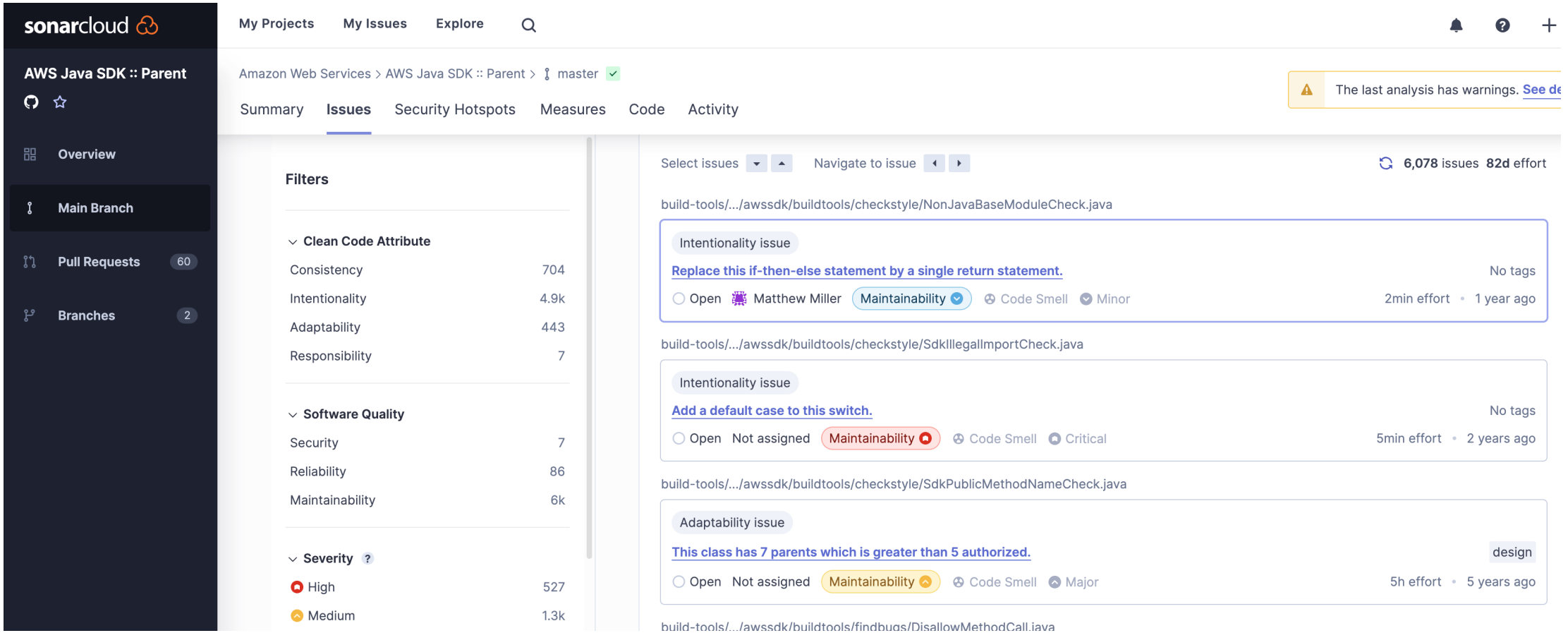
Comparing some static analysis tools

https://www.comparitech.com/net-admin/best-static-code-analysis-tools/

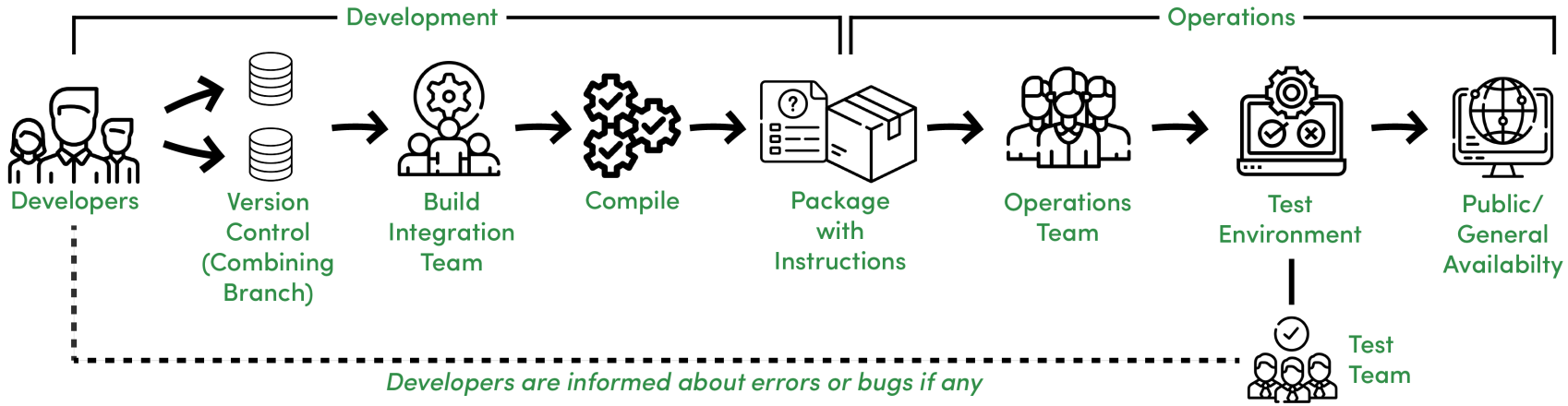
Tool/Features	SonarQube	Checkmarx	Synopsys Coverity	Micro Focus Fortify SCA	Veracode Static Analysis	Snyk Code
Language Support	Multiple	Multiple	Multiple	Multiple	Multiple	Multiple
Integrations	Various IDEs, CI/CD	Various IDEs, CI/CD	Various IDEs, CI/CD	Various IDEs, CI/CD	Various IDEs, CI/CD	Various IDEs, CI/CD
Free Trial	Yes	Yes	No	Yes	Yes	Yes
On-Premises/Cloud	Both	Both	Both	Both	Both	Cloud
Automated Scans	Yes	Yes	Yes	Yes	Yes	Yes
Compliance Reporting	Yes	Yes	Yes	Yes	Yes	Yes
Vulnerability Database	Yes	Yes	Yes	Yes	Yes	Yes
Real-Time Feedback	Yes	Yes	No	No	No	Yes

Example of issues report from SonarCloud/SonarQube

https://sonarcloud.io/project/issues?resolved=false&id=aws_aws-sdk-java-v2

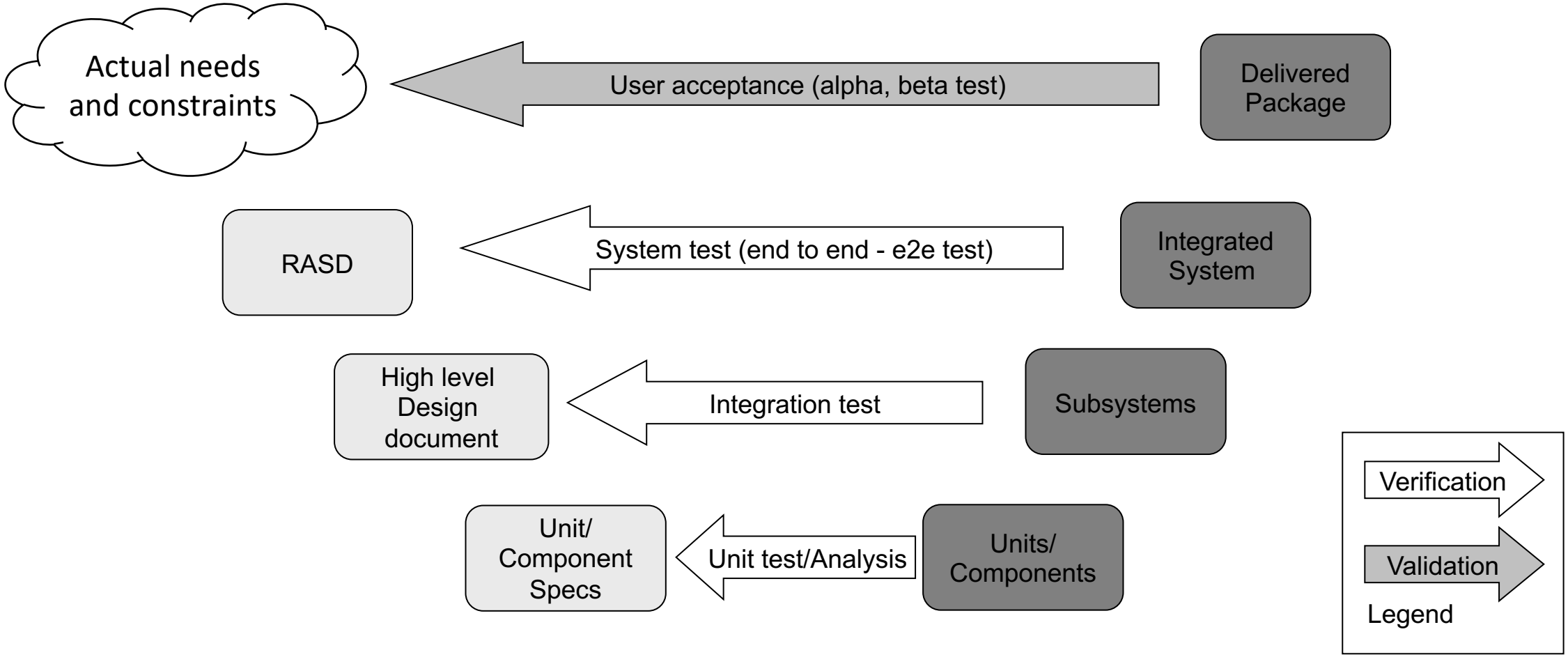


Course plan

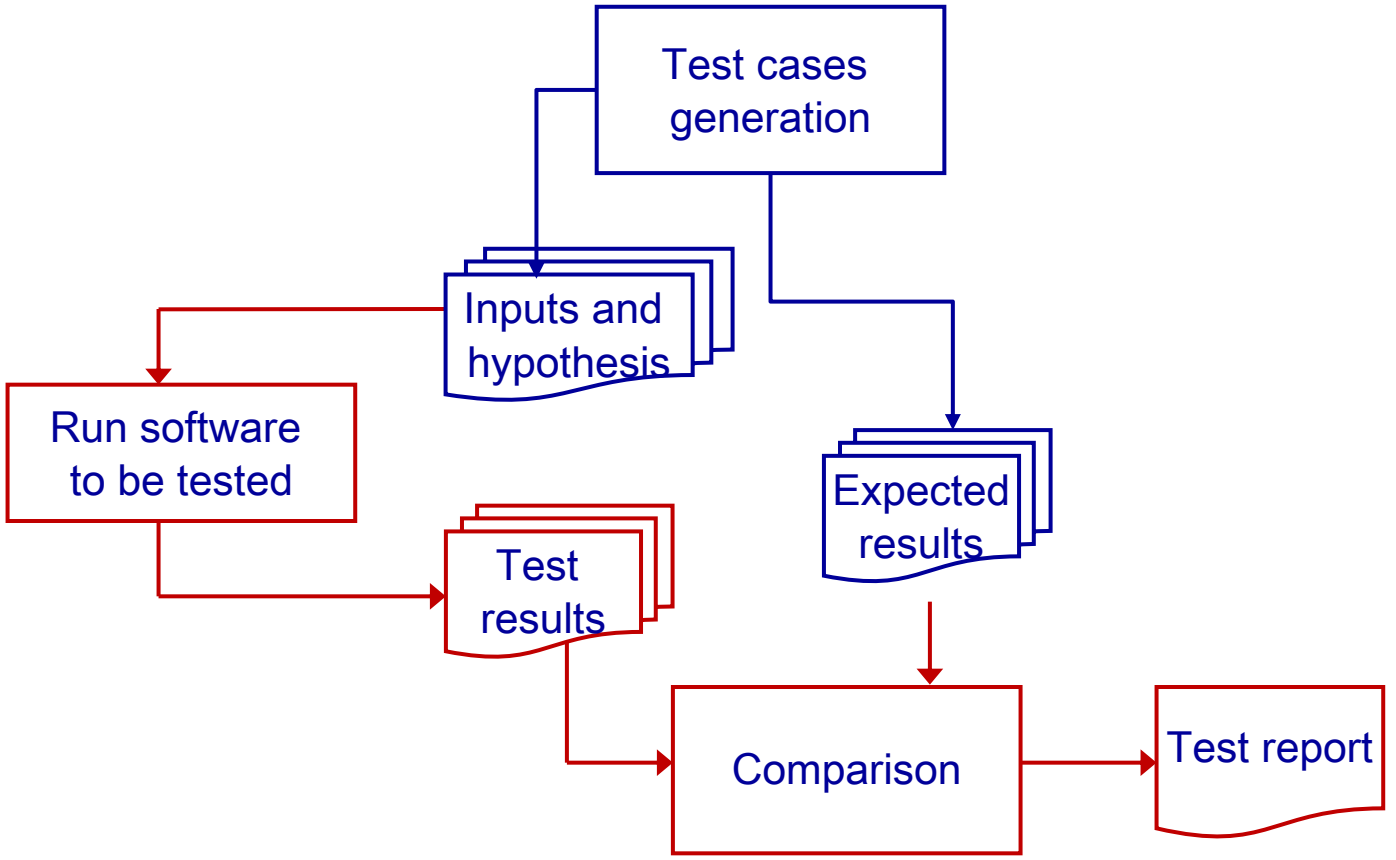


- Process
  - Development lifecycles
  - DevOps
- Automation
  - Version control & software configuration management
  - Static analysis
  - Testing automation
  - Pipeline control tools

Testing activities (the V model)



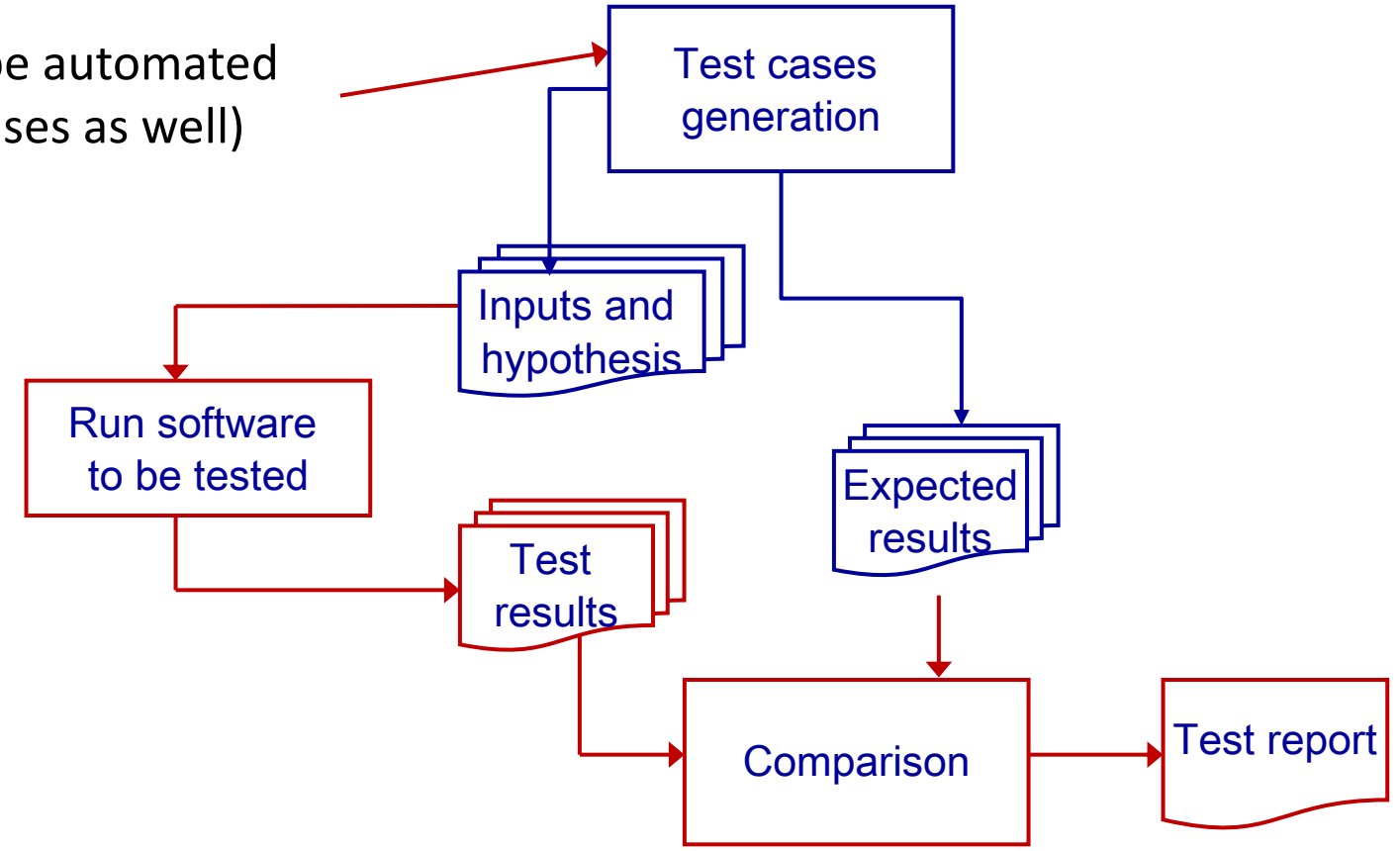
Automation possibilities



Today modern programming languages come with means to automate the execution of red steps!

Automation possibilities

(This can be automated in some cases as well)



Today modern programming languages come with means to automate the execution of red steps!

Automated testing in Python: PyUnit

POLITECNICO MILANO 1863

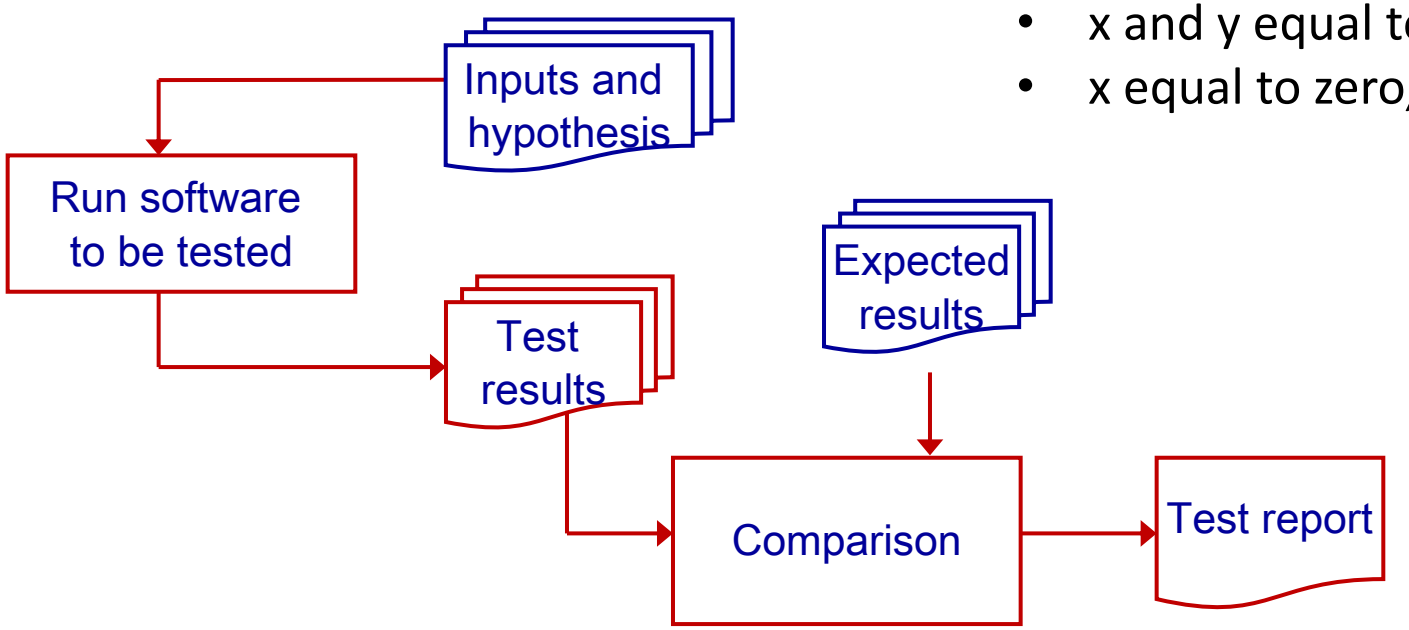
- Python unit testing framework also called **unittest**
- Derived from jUnit (first automated testing tool developed for Java)
- Developed by Kent Beck and Eric Gamma

First simple example

```
#this is file simpleAdd
def add(x,y):
    return x + y
```

Possible test cases (inputs and hypothesis):

- Positive values for x and y: 4, 5
- Negative values for x and y: -4, -5
- x positive and y negative: 4, -5
- x negative and y positive: -4, 5
- x and y equal to zero: 0, 0
- x equal to zero, y not: 0, -5



First simple example

```
#this is file simpleAdd
def add(x,y):
    return x + y
```

```
import unittest
import simpleAdd
class SimpleTest(unittest.TestCase):
    def testadd (self):
        self.assertEqual(simpleAdd.add(4,5),9)
        self.assertEqual(simpleAdd.add(-4,-5),-9)
        self.assertEqual(simpleAdd.add(4,-5),-1)
        self.assertEqual(simpleAdd.add(-4,5),1)
        self.assertEqual(simpleAdd.add(0,0),0)
        self.assertEqual(simpleAdd.add(0,-5),-5)

if __name__ == '__main__':
    unittest.main()
```

Possible test cases:

- Positive values for x and y: 4, 5
- Negative values for x and y: -4, -5
- x positive and y negative: 4, -5
- x negative and y positive: -4, 5

zero: 0, 0

y not: 0, -5

unittest concepts

POLITECNICO MILANO 1863

- **test case**:
  - unittest provides a base class, **TestCase**, which may be used to create new test cases.
  - This checks for a specific response to a particular set of inputs.
- **test suite:**
  - It is a collection of test cases, test suites, or both.
  - This is used to aggregate tests that should be executed together.
  - Test suites are implemented by the **TestSuite** class.

unittest concepts

POLITECNICO MILANO 1863

- **test fixture:**
  - This represents the preparation needed to perform one or more tests, and any associate cleanup actions.
  - This may involve, for example, creating temporary or proxy databases, directories, or starting a server process.

- **test runner:**
  - This is a component which orchestrates the execution of tests and provides the outcome to the user.
  - The runner may use a graphical interface, a textual interface, or return a special value to indicate the results of executing the tests.

Test case design

POLITECNICO MILANO 1863

1. Select the unit of code to be tested
2. Identify the input data for the test and the initial state your running code should be
3. Define the oracle, that is, the piece of code that asserts what should be true at the end of the test
4. Write the implementation of a test case:
   - Test fixtures can be used to ensure the unit of code is in the initial state before testing
   - Write the code that calls the unit of code under test with the parameters you have identified
   - Write the oracle exploiting instructions such as assertEquals

unittest API

POLITECNICO MILANO 1863

- **setUp():** Method called to prepare the test fixture. This is called immediately before calling the test method
- **tearDown():** Method called immediately after the test method has been called and the result recorded. This is called even if the test method raised an exception.
- **setUpClass():** A class method called before tests in an individual class run.
- **tearDownClass():** A class method called after tests in an individual class have run.
- **run(result = None):** Run the test, collecting the result into the test result object passed as result.
- **skipTest(reason):** Calling this during a test method or setUp() skips the current test.
- **debug():** Run the test without collecting the result.
- **shortDescription():** Returns a one-line description of the test.

Second example

```
#this is file SimpleTest2
import unittest
import addSub
class SimpleTest2(unittest.TestCase):
    def setUp(self):
      name = self.shortDescription()
      if name == "Add":
         self.a = 10
         self.b = 20
         print(name, self.a, self.b)
      if name == "Sub":
         self.a = 50
         self.b = 60
         print(name, self.a, self.b)
    def tearDown(self):
      print('\nend of test',self.shortDescription())
    def testadd(self):
      """Add"""
      self.assertEqual(addSub.add(self.a, self.b), self.a+self.b)
    def testsub(self):
      """Sub"""
      self.assertEqual(addSub.sub(self.a, self.b), self.a-self.b)

if __name__ == '__main__':
   unittest.main()
```

```
#this is file addSub
def add(x, y):
    return x+y
def sub(x, y):
    return x-y
```

TestSuites

POLITECNICO MILANO 1863

Test case instances can be grouped together according to the features they test

- **Step 1** – Create an instance of TestSuite class.
  ```
  suite = unittest.TestSuite()
  ```
- **Step 2** – Add tests inside a TestCase class in the suite.
  ```
  suite.addTest(unittest.makeSuite(testcase class))
  ```
- **Step 3** – Individual tests can also be added in the suite.
  ```
  suite.addTest(testcaseclass('testmethod')
  ```
- **Step 4** – Create an object of the TestTestRunner class.
  ```
  runner = unittest.TestTestRunner()
  ```
- **Step 5** – Call the run() method to run all the tests in the suite
  ```
  runner.run (suite)
  ```

Example of test suite

```
import unittest
import SimpleTest2
import SimpleTest

def suite():
    suite = unittest.TestSuite()
    suite.addTest(unittest.makeSuite(SimpleTest2))
    suite.addTest(unittest.makeSuite(SimpleTest.SimpleTest))
    return suite

if __name__ == '__main__':
    runner = unittest.TextTestRunner()
    test_suite = suite()
    runner.run (test_suite)
```

Basic AssertionTypes

POLITECNICO MILANO 1863

- `assertEqual(arg1, arg2, msg = None)`: Test that arg1 and arg2 are equal. If the values do not compare equal, the test will fail.
- `assertNotEqual(arg1, arg2, msg = None)`: Test that arg1 and arg2 are not equal. If the values do compare equal, the test will fail.
- `assertTrue(expr, msg = None)`: Test that expr is true. If false, test fails
- `assertFalse(expr, msg = None)`: Test that expr is false. If true, test fails
- `assertIs(arg1, arg2, msg = None)`: Test that arg1 and arg2 evaluate to the same object.
- `assertIsNot(arg1, arg2, msg = None)`: Test that arg1 and arg2 don't evaluate to the same object.

Basic Assertion Types

POLITECNICO MILANO 1863

- `assertIsNone(expr, msg = None)`: Test that expr is None. If not None, test fails
- `assertIsNotNone(expr, msg = None)`: Test that expr is not None. If None, test fails
- `assertIn(arg1, arg2, msg = None)`: Test that arg1 is in arg2.
- `assertNotIn(arg1, arg2, msg = None)`: Test that arg1 is not in arg2.
- `assertIsInstance(obj, cls, msg = None)`: Test that obj is an instance of cls
- `assertNotIsInstance(obj, cls, msg = None)`: Test that obj is not an instance of cls

Trivial example of assertion usage

```
import unittest

class SimpleTest(unittest.TestCase):
   def test1(self):
      self.assertEqual(4 + 5,9)
   def test2(self):
      self.assertNotEqual(5 * 2,10)
   def test3(self):
      self.assertTrue(4 + 5 == 9,"The result is False")
   def test4(self):
      self.assertTrue(4 + 5 == 10,"assertion fails")
   def test5(self):
      self.assertIn(3,[1,2,3])
   def test6(self):
      self.assertNotIn(3, range(5))

if __name__ == '__main__':
   unittest.main()
```

Testing web apps – simple example 1

```
from flask import Flask

# Create the application instance
app = Flask(__name__)

# Create a URL route in our application for "/"
@app.route('/')
def home():
  return 'Hello!!!!'

# If we're running in stand alone mode, run the application
if __name__ == '__main__’:
  app.run(debug=True)
```

Testing web apps – test case for simple example 1

```
import unittest

from simpleWebApp import app

class AppTestCase(unittest.TestCase):
def setUp(self):
  self.ctx = app.app_context()
  self.ctx.push()
  self.client = app.test_client()

def tearDown(self):
  self.ctx.pop()

def test_home(self):
  response = self.client.get("/")
  assert "Hello!!!!" == response.get_data(as_text=True)

if __name__ == "__main__":
  unittest.main()
```

Testing web apps – simple example 2

```
from flask import (
Flask,
  render_template
)

# Create the application instance
app = Flask(__name__, template_folder="templates")

@app.route('/')
def home():
  return render_template('home.html')
```

```
@app.route('/viewMap')
def viewMap():
  return render_template('viewMap.html')

@app.route('/reserveBike')
def reserveBike():
  return 'Congratulations, you have reserved a bike!'

if __name__ == '__main__’:
  app.run(debug=True)
```

Testing web apps – test case for simple example 2

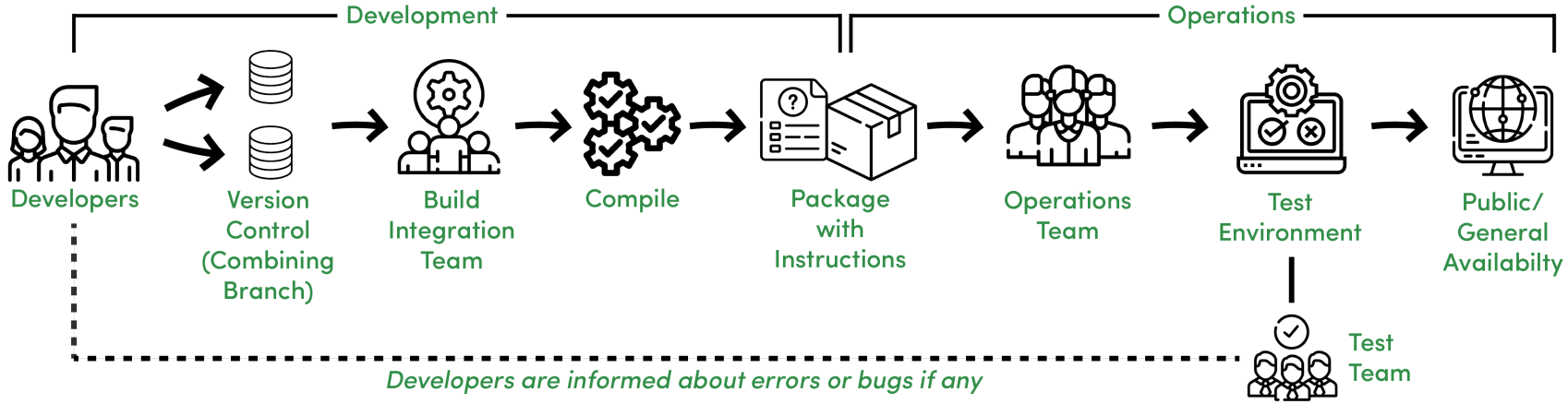
```
import unittest

from simpleWebApp2 import app

class AppTestCase(unittest.TestCase):
def setUp(self):
  self.ctx = app.app_context()
  self.ctx.push()
  self.client = app.test_client()

def tearDown(self):
  self.ctx.pop()

def test_home(self):
  response = self.client.get("/")
  self.assertEqual(response.status_code, 200)
  html = response.data.decode()
  assert "You can do the following:" in html

def test_viewMap(self):
  response = self.client.get("/viewMap")
  assert response.status_code == 200
  html = response.data.decode()
  assert "This is the view map page" in html

if __name__ == "__main__":
  unittest.main()
```

Exercise in breakout rooms

POLITECNICO MILANO 1863

- Clone the repository available here https://github.com/dinitto/pyTestExamples
- Inspect the examples
- Run tests (it requires Python, unittest and Flask to be installed)
- Introduce some errors in the code and run again the tests
- Modify the tests and run them again

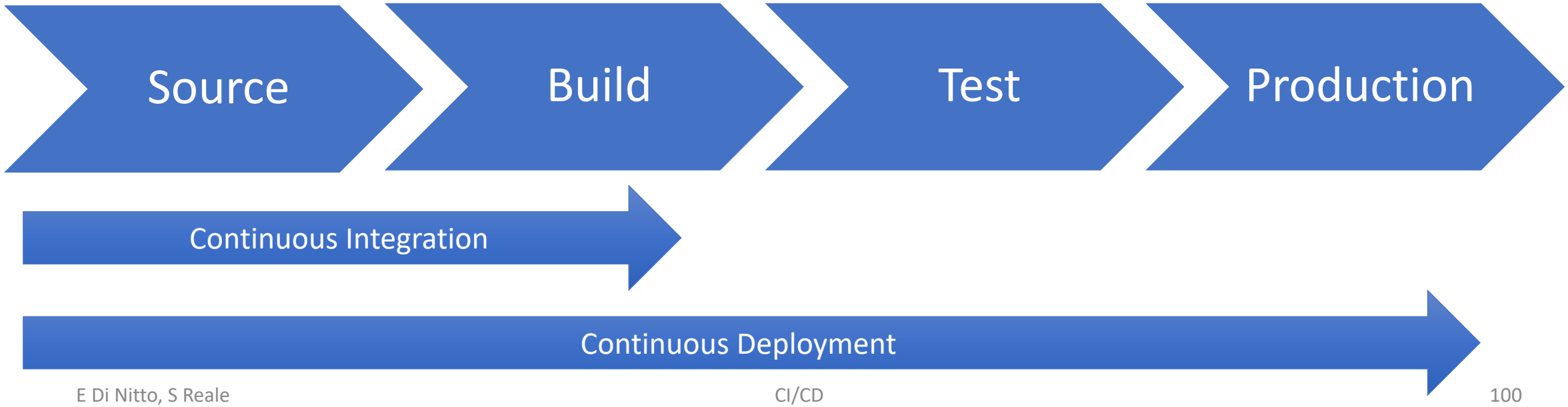
Course plan



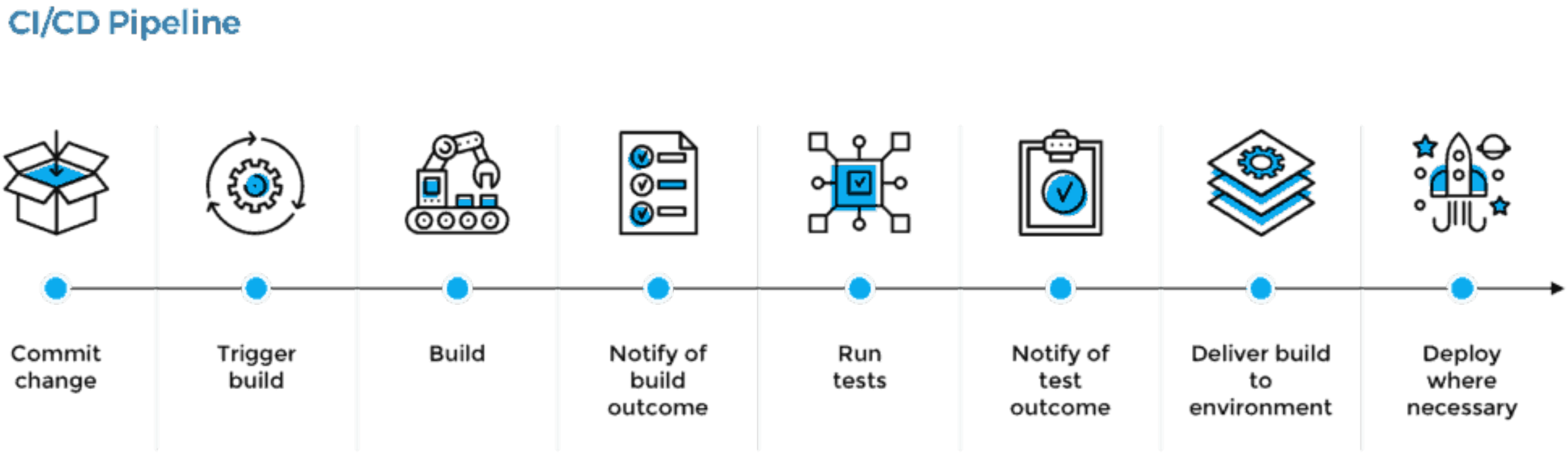
- Process
  - Development lifecycles
  - DevOps
- Automation
  - Version control & software configuration management
  - Static analysis
  - Testing automation
  - Pipeline control tools
- Hands-on
  - Testing automation
  - Creating pipelines for C and Python programs using GitHub Actions

CI/CD

- CI: Continuous Integration
  - Integrate whenever there is a new version of a component available
- CD: Continuous Delivery/Deployment
  - Deliver/Deploy new versions as soon as they are ready

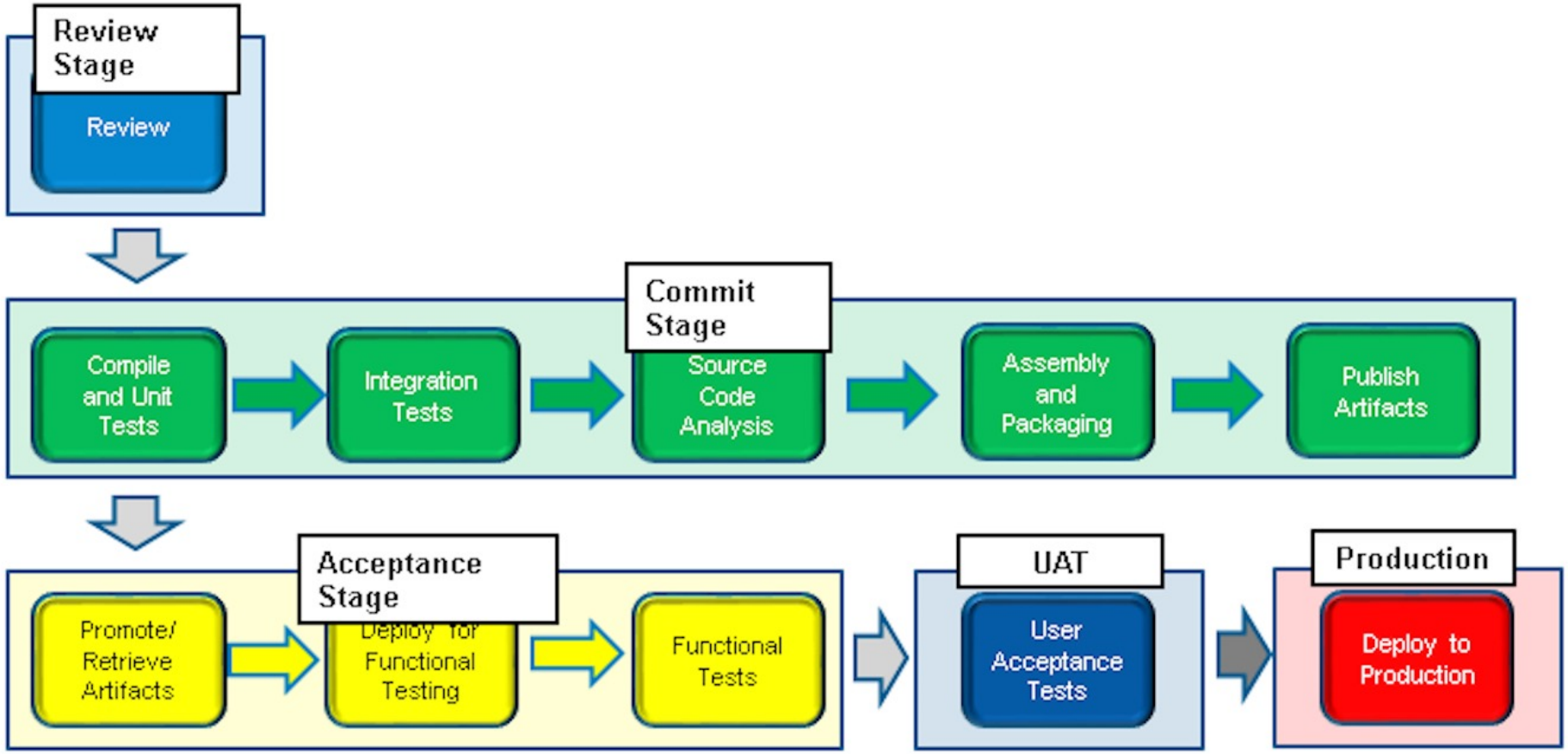


The CI/CD pipeline



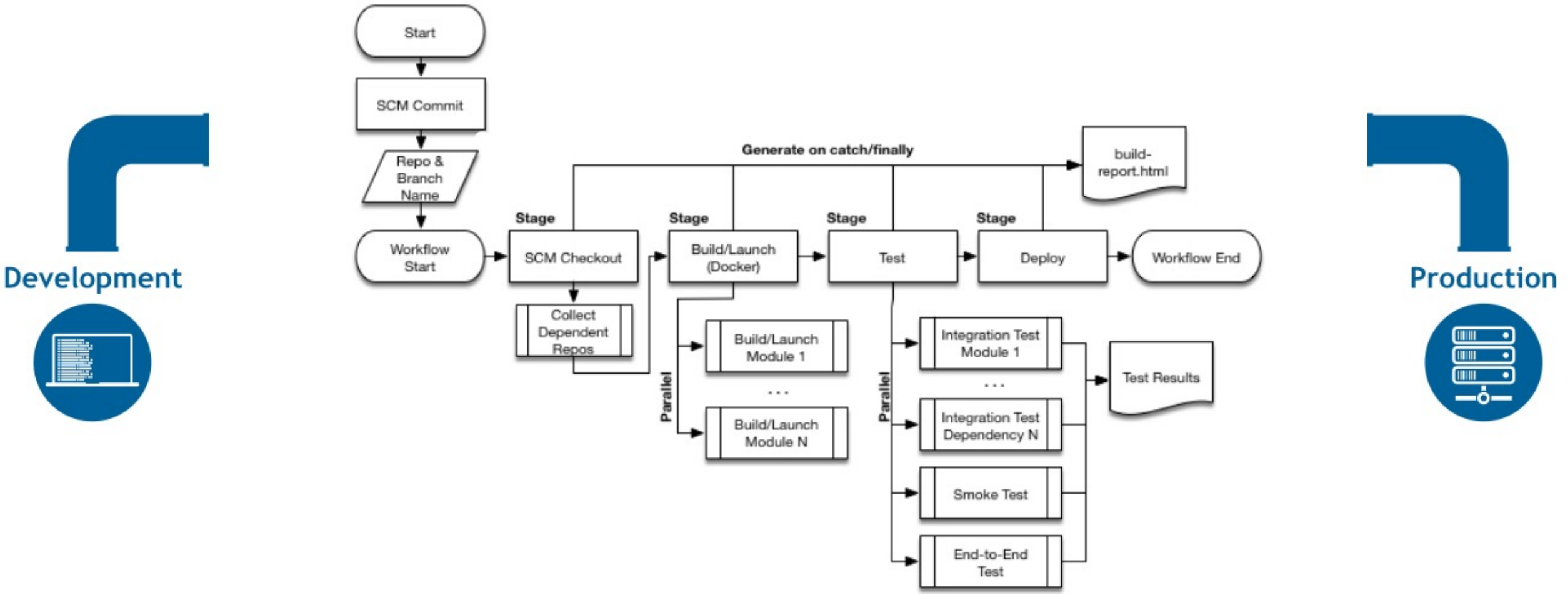
https://www.plutora.com/blog/understanding-ci-cd-pipeline

The CI/CD pipeline – another view



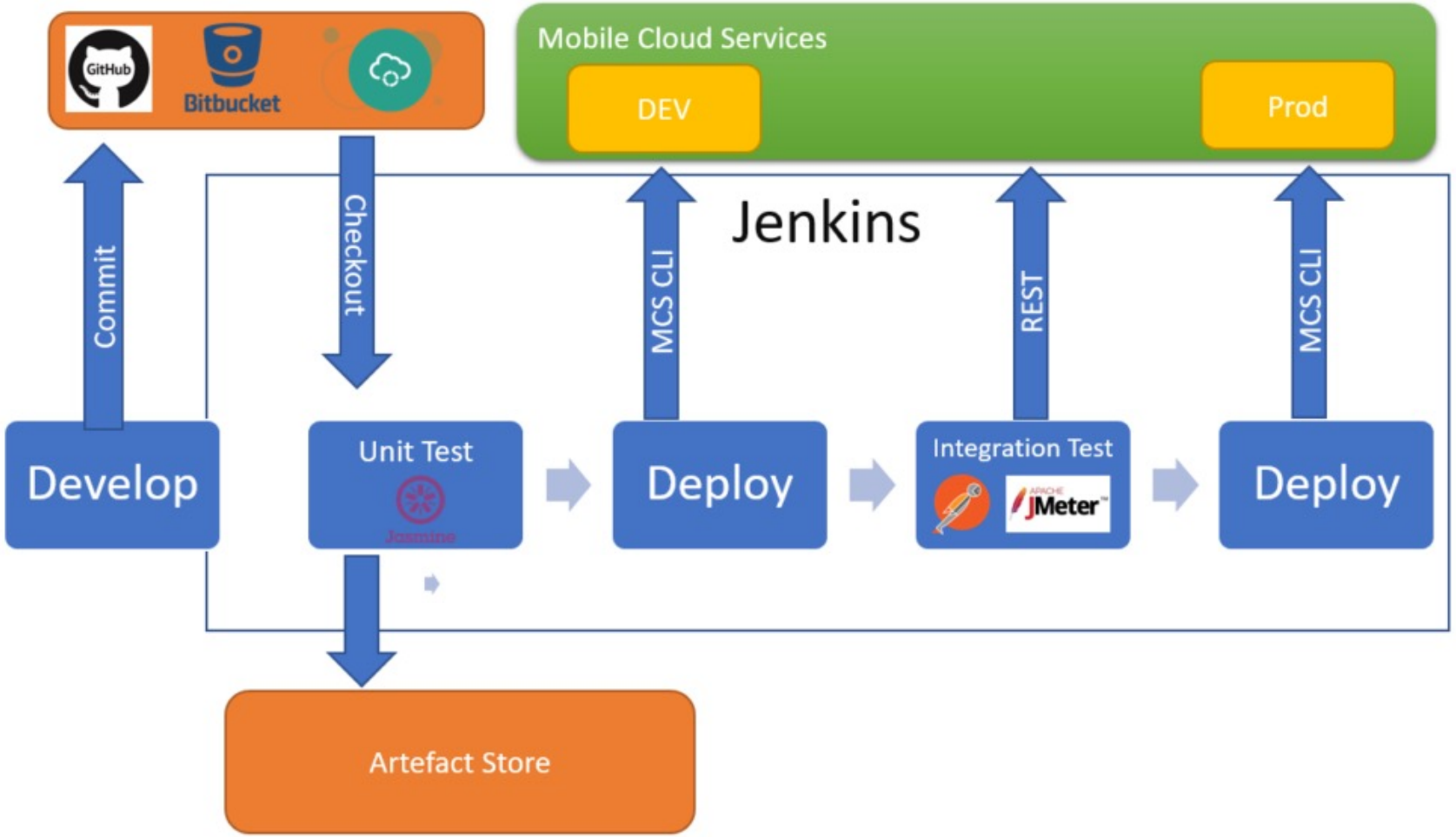
https://www.oreilly.com/content/configuring-a-continuous-delivery-pipeline-in-jenkins/

Another possible workflow



https://www.jenkins.io/doc/book/pipeline/

An example of CI/CD pipeline



https://www.ateam-oracle.com/creating-a-cicd-pipeline-between-jenkins-and-mobile-cloud-services

CI/CD best tools

https://thectoclub.com/tools/best-ci-cd-tools/

Tools	GitLab CI/CD	GitHub Actions	Azure DevOps	Spinnaker	Argo CD	CircleCI	OpenShift Pipelines	Travis CI	Jenkins	Terraform
	Website	Website	Website	Website	Website	Website	Website	Website	Website	Website
Price	From $29/user/month	Pricing upon request	From $52/user/month	Free	Free	From $15/month for 5 users	Free	From $34/user/month	Free To Use	Pricing upon request
Best for	Best maturity feedback	Best for small teams	Best for Azure development	Best for custom integrations	Best for Kubernetes development	Best for enterprise development	Best open-source option	Best for on-premise deployments	Best for scaling companies	Best repeatable code
Trial info	Free plan available	Free plan available	Free plan available	Free	Free plan available	Free plan available	Free plan available	Free trial available	Free	Free plan available
Pros	☆ Pipeline templates ☆ Supports DevSecOps ☆ Detailed maturity feedback	☆ Actions are isolated, minimizing conflicts and compatibility issues ☆ Wide range of events to link to actions ☆ Easy to use	☆ Robust repository management ☆ Includes project management solutions for scrum and agile ☆ Combines CI/CD with DevOps	☆ Supports canary analysis ☆ Active developer community for support ☆ Very configurable	☆ User-friendly UI ☆ Supports GitOps ☆ Kubernetes native	☆ Scalable ☆ SSH debugging ☆ Detailed metrics with insights	☆ Serverless architecture ☆ Kubernetes native ☆ Flexible configuration options	☆ Provides preconfigured customizable build images ☆ Multipurpose GitHub integration ☆ Straightforward setup	☆ Highly scalable ☆ Extensible with hundreds of plugins ☆ Active developer community for support	☆ IAC features that work across most platforms ☆ Robust automation capabilities ☆ Strong code management features
Cons	⊖ No standalone version ⊖ Significantly underpowered free tier	⊖ Poor support for actions originating outside the core development team ⊖ Built entirely around repositories	⊖ Limited customization options ⊖ Poor integration with third-party services	⊖ Relies heavily on third-party tools ⊖ CD only	⊖ Limited to Kubernetes environments ⊖ Requires significant Kubernetes expertise	⊖ Visit WebsiteOpens new window ⊖ Support teams often take long to respond ⊖ Expensive	⊖ Requires extensive configuration ⊖ Doesn't work as well in non-Kubernetes environments	⊖ Not as configurable as other options ⊖ Reporting is too light	⊖ It's very dependent on plugins ⊖ Dated UI	⊖ Relies heavily on third-party tools for full functionality ⊖ HCL takes a while to learn

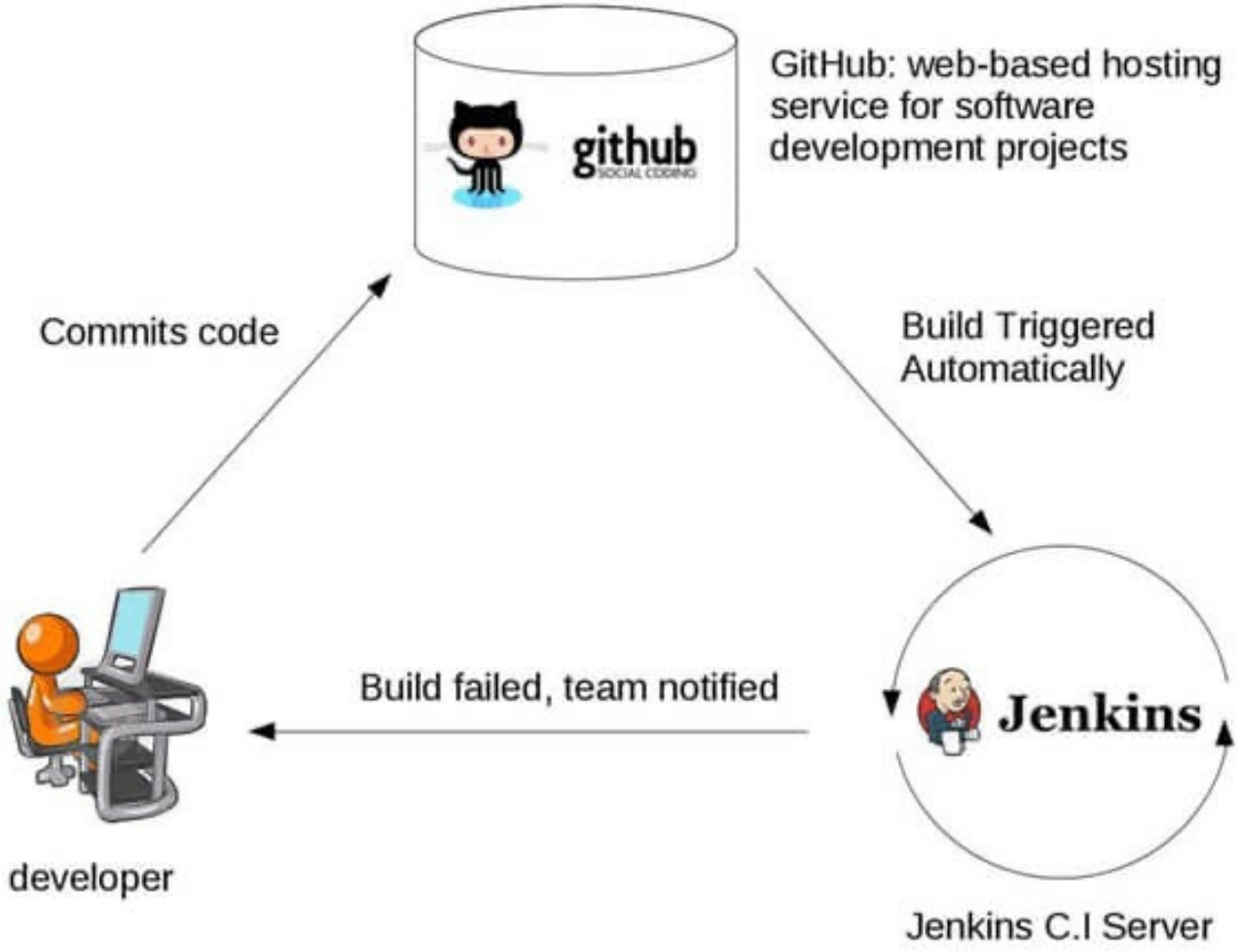
Jenkins

POLITECNICO MILANO 1863

- Originally conceived by Kohsuke Kawaguchi to support build and testing of Java code
- It was called Hudson
- In 2011 part of the Hudson community decided to split and developed Jenkins
- It is a workflow management system able to run several different tools as part of a *pipeline*

Jenkins

https://www.jenkins



A simple Jenkins pipeline

```
pipeline {
    agent any

    tools {
        // Install the Maven version configured as "M3" and add it to the path.
        maven "M3"
    }

    stages {
        stage('Build') {
            steps {
                // Get some code from a GitHub repository
                git 'https://github.com/dinitto/simple-maven-project-with-tests.git'

                // Run Maven on a Unix agent.
                sh "mvn -Dmaven.test.failure.ignore=true clean package"

                // To run Maven on a Windows agent, use
                // bat "mvn -Dmaven.test.failure.ignore=true clean package"
            }

            post {
                // If Maven was able to run the tests, even if some of the test
                // failed, record the test results and archive the jar file.
                success {
                    junit '**/target/surefire-reports/TEST-*.xml'
                    archiveArtifacts 'target/*.jar'
                }
            }
        }
    }
}
```

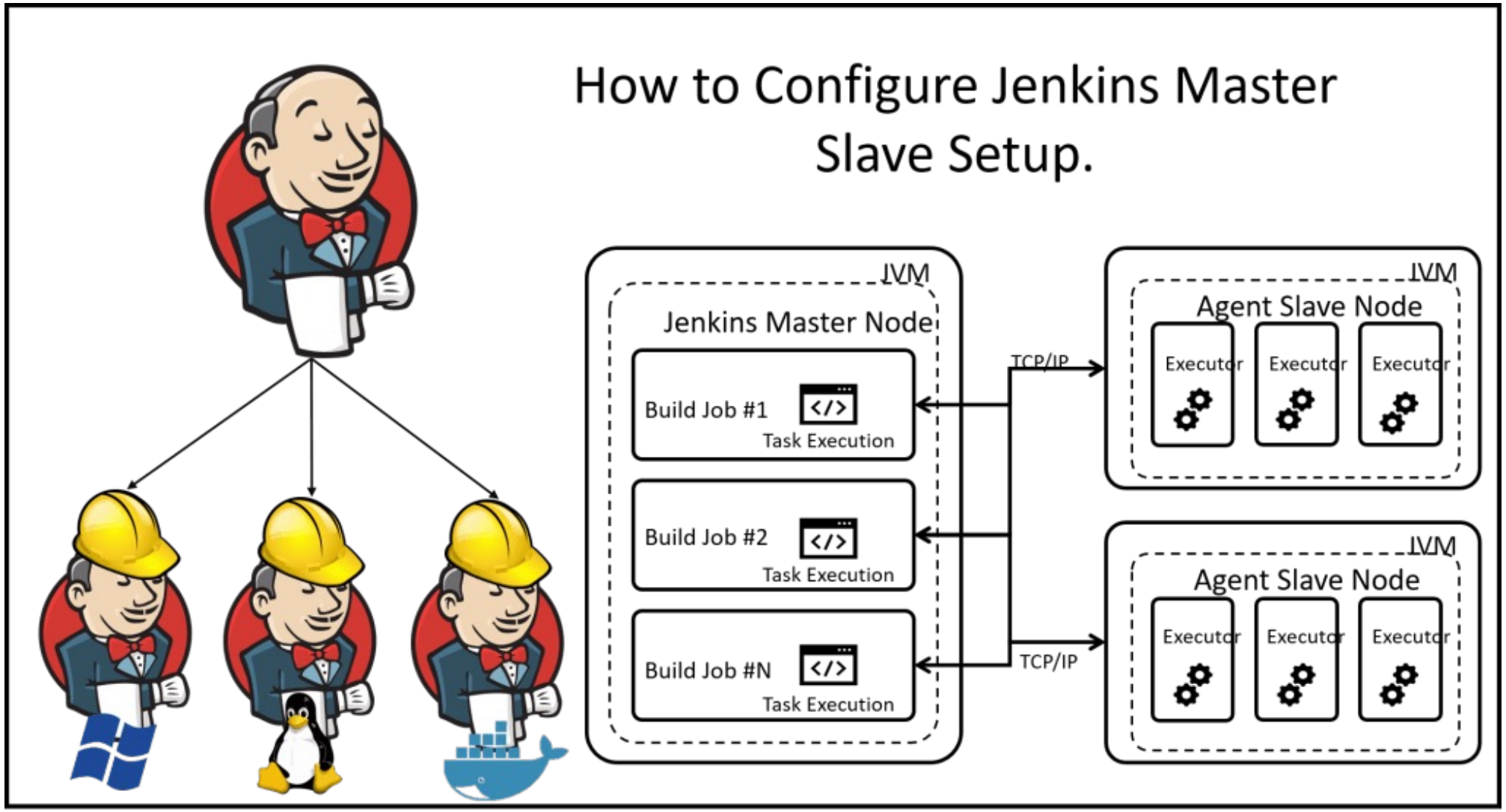
Jenkins pipeline – multiple stages

```
pipeline {
        environment {xxx}
        stages {
                stage ('Pull repo code from github') {xxx}
                stage ('Build the code with Maven') {xxx}
                stage('Sonar analysis') {xxx}
                stage ('Trigger a build of defect-prediction') {xxx}
                stage('Build docker images') {xxx}
                stage('Push Reasoner to DockerHub') {xxx}
                stage('Push graphdb to DockerHub') {xxx}
                stage('Install dependencies') {xxx}
                stage('Deploy to openstack') {xxx}
        }
        post {
                failure {xxx}
                fixed {xxx}
}
}
```

Complete code available here
https://github.com/SODALITE-EU/semantic-reasoner/blob/master/Jenkinsfile

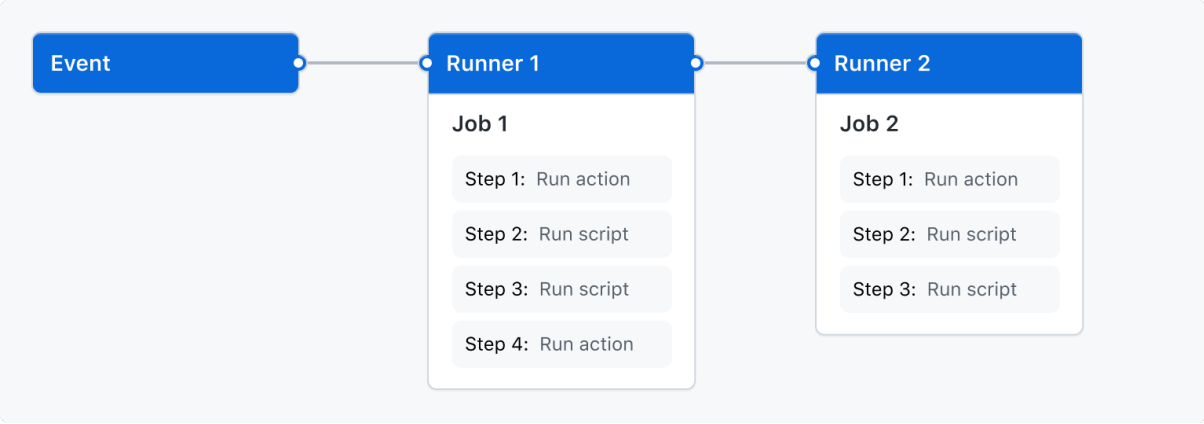
Jenkins master-slave approach

https://digitalvarys.com/how-to-configure-jenkins-master-slave-setup/



Github actions – main concepts

- Workflow: configurable automated process that will run one or more jobs.
- Event: triggers the execution of workflows.
- Job: a set of steps in a workflow.
- Step: can be an action (reusable in different contexts) or a shell script.
- Runner: a VM where a job is run.

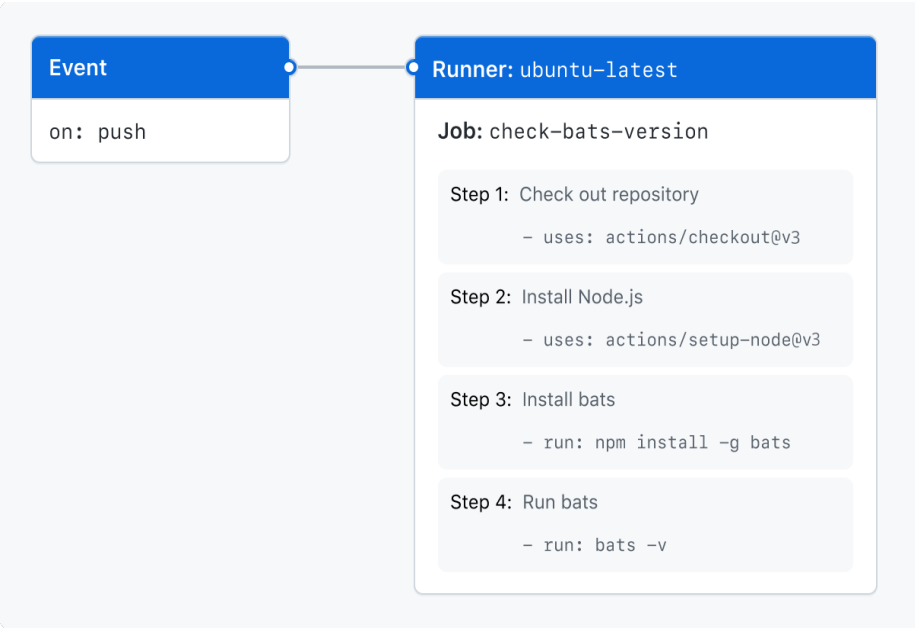


Workflow

- Defined in the `.github/workflows/` directory within a repository

Example

```yaml
name: learn-github-actions
run-name: ${{ github.actor }} is learning GitHub Actions
on: [push]
jobs:
  check-bats-version:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - uses: actions/setup-node@v4
        with:
          node-version: '20'
      - run: npm install -g bats
      - run: bats -v
```



An example of CI workflow using cmake

https://github.com/actions/starter-workflows/ci/cmake-single-platform.yml

```yaml
name: CMake on a single platform
on: [push, pull_request]
env:
  BUILD_TYPE: Release
jobs:
  build:
    runs-on: ubuntu-latest
    steps:
    - uses: actions/checkout@v4
    - name: Configure CMake
      run: cmake -B ${{github.workspace}}/build -DCMAKE_BUILD_TYPE=${{env.BUILD_TYPE}}
    - name: Build
      run: cmake --build ${{github.workspace}}/build --config ${{env.BUILD_TYPE}}
    - name: Test
      working-directory: ${{github.workspace}}/build
      run: ctest -C ${{env.BUILD_TYPE}}
```

An example of CI workflow using Maven

https://github.com/actions/starter-workflows/ci/maven.yml

```yaml
name: Java CI with Maven
on:
  push:
    branches: [ $default-branch ]
  pull_request:
    branches: [ $default-branch ]
jobs:
  build:
    runs-on: ubuntu-latest
    steps:
    - uses: actions/checkout@v4
    - name: Set up JDK 17
      uses: actions/setup-java@v3
      with:
        java-version: '17'
        distribution: 'temurin'
        cache: maven
    - name: Build with Maven
      run: mvn -B package --file pom.xml
```

Triggering events

POLITECNICO MILANO 1863

- Events that occur in your workflow's repository
- Events that occur outside of GitHub and trigger a repository_dispatch event on GitHub
- Scheduled times
- Manual
- Complete list https://docs.github.com/en/actions/using-workflows/events-that-trigger-workflows

Triggering events

- Defined after the on keyword

```yaml
on: push
```

- Multiple events can be specified

```yaml
on: [push, pull_request]
```

```yaml
on:
  push:
    branches:
    - main
  pull_request:
    branches: [ $default-branch ]
  label:
    types:
    - created
```

Filter

Activity type

- NB: A workflow instance starts for every event occurred

Actions

POLITECNICO MILANO 1863

- Building blocks that can be used in a workflow
- Can be defined in
  - Your repository
  - Any public repository
  - Directory of publicly available actions: GitHub Marketplace https://github.com/marketplace/actions/
- Are pieces of code written in JavaScript or in any other language if made available as a container
- Each action must have a metadata file to define the inputs, outputs and main entrypoint: `action.yaml` or `action.yml`
- Publicly available actions should have a readme describing it

Workflow execution

- The occurrence of an event with the specified action type and fulfilling the filter triggers the execution of a workflow
- A runner is spooned for each job in the workflow

```yaml
jobs:
  build1:
    runs-on: ubuntu-latest
  …
  build2:
    runs-on: macos-latest
  …
```

```yaml
jobs:
  test:
    runs-on: [self-hosted, linux]
```

- Steps within each job are executed sequentially