How to use Workflow modules to implement user functionality

Thursday 12 May 2011
DIRAC User Community Meeting

Ching Bon Lam
Ching.Bon.Lam@cern.ch

UNIVERSITEIT TWENTE.

Background

- Applied Physics master student
- Software development background
- First use of DIRAC for my work at the Linear Collider Detector research group at CERN

Disclaimer

- The Workflow framework is not written by me
- There might be errors in this presentation

Outline

1. Motivation
2. Workflow Framework
3. Simple Example
4. Conclusion

Part I

Motivation

Motivation (1/4)

- Example job using JDL and shell script (from DIRAC tutorial)

```
testJob.sh
----------
#!/bin/bash
/bin/hostname
/bin/date
/bin/ls -la
```

```
JobName       = "LFNInputSandbox";
Executable    = "testJob.sh";
StdOutput     = "StdOut";
StdError      = "StdErr";
InputSandbox  = {"testJob.sh","LFN:/somePath/someFile.txt"};
OutputSandbox = {"StdOut","StdErr"};
OutputSE      = "M3PEC-disk";
OutputData    = {"StdOut"};
```

DIRAC tutorial: http://mareela.in2p3.fr:9200/dirac/wiki/DIRAC/Tutorials/JobManagement

Motivation (2/4)

- Example job using JDL and shell script (from DIRAC tutorial)

```
testJob.sh
----------
#!/bin/bash
/bin/hostname
/bin/date
/bin/ls -la
```

- What if you want to run more complex jobs with multiple applications?

```
JobName       = "LFNInputSandbox";
Executable    = "testJob.sh";
StdOutput     = "StdOut";
StdError      = "StdErr";
InputSandbox  = {"testJob.sh","LFN:/somePath/someFile.txt"};
OutputSandbox = {"StdOut","StdErr"};
OutputSE      = "M3PEC-disk";
OutputData    = {"StdOut"};
```

Motivation (3/4)

- What if you want to run more complex jobs with multiple applications?

1. Basic user: one shell script executing multiple applications

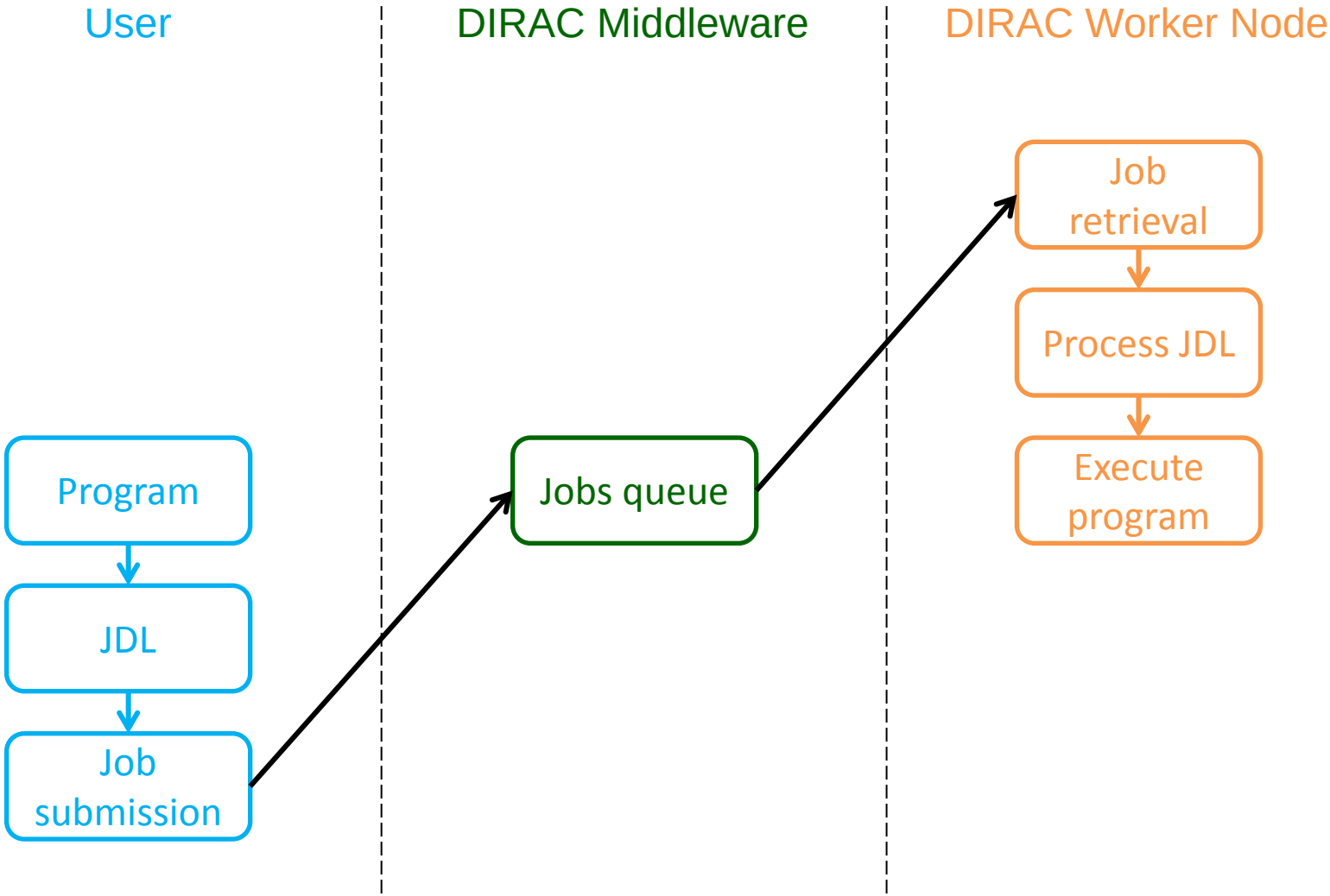
2. Advanced user: one shell script executing shell functions from a library. Each function is a wrapper around an application. In this way it is modular.

- This advanced concept can be implemented in shell scripting or any other programming language, but…

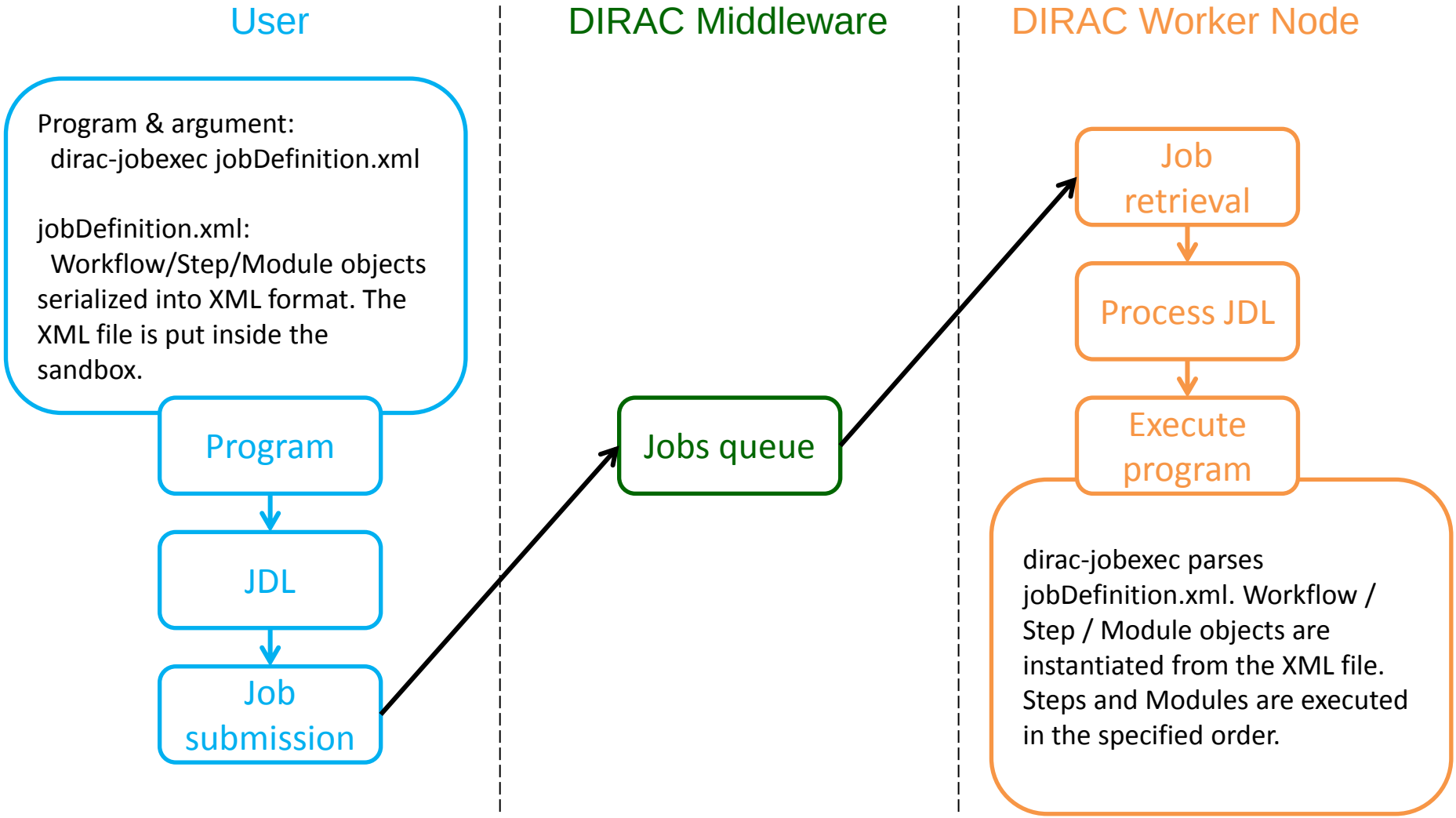
Motivation (4/4)

- This advanced concept can be implemented in shell scripting or any other programming language, but…
- It already exists in DIRAC: Workflow
- The Workflow framework is the product of a natural evolution from simple jobs to complex jobs.

Flow of execution (1/2)



Flow of execution (2/2)



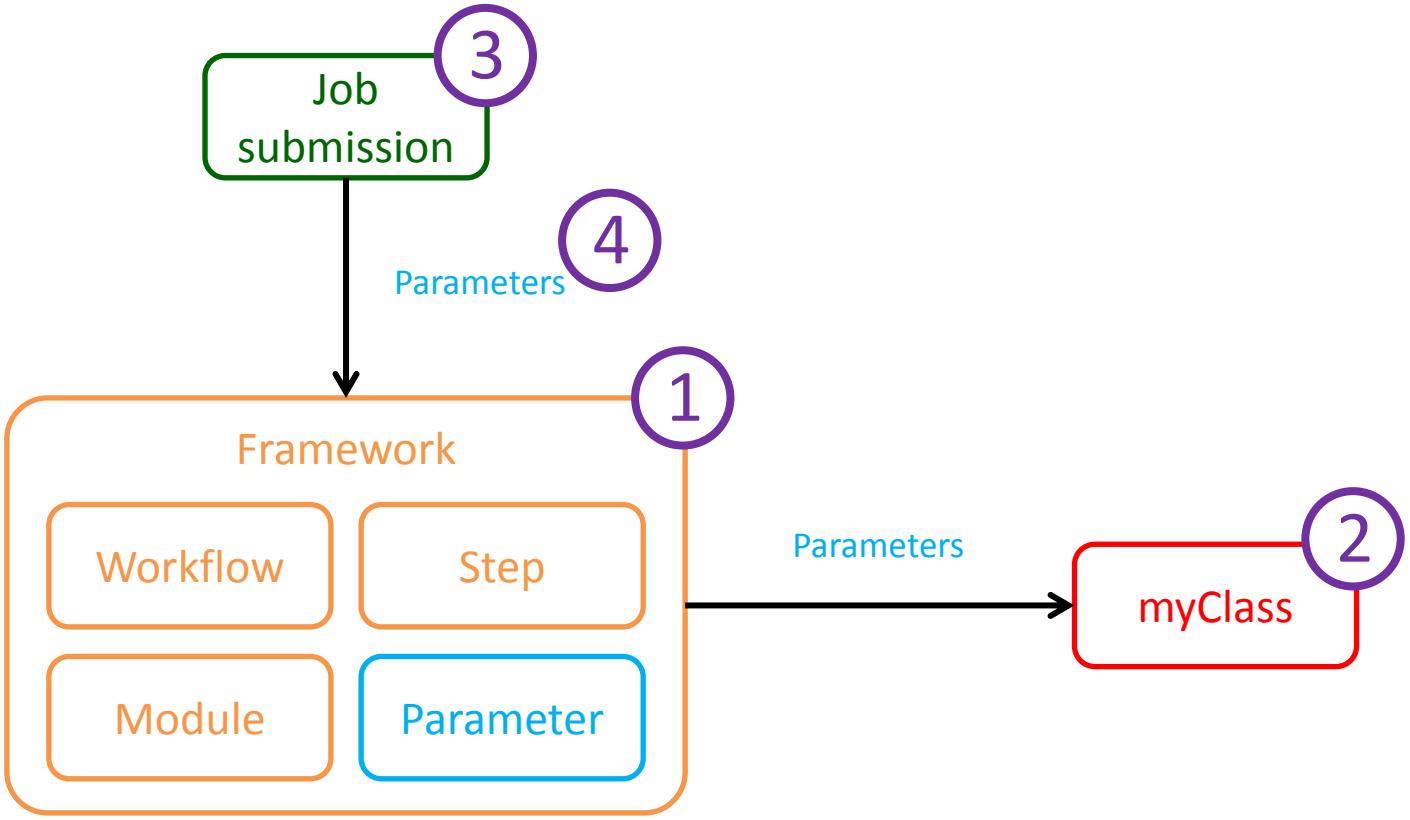
Part II
Workflow framework

Framework Overview: Workflow, Step, Module

My description:

A framework for loading code and execute it in

the specified order with the specified parameters



1 Framework – Introduction (1/5)

Example:

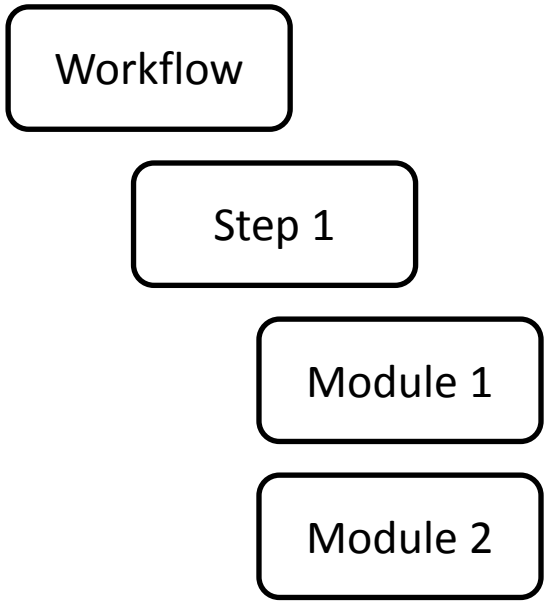
- User functionalities are implemented inside myClass1 and myClass2
- myClass1 and myClass2 are executed on the GRID

myClass1

myClass2

1 Framework – Introduction (2/5)

Job submission



- A Workflow is a container for Steps
- A Step is a container for Modules

User functionalities

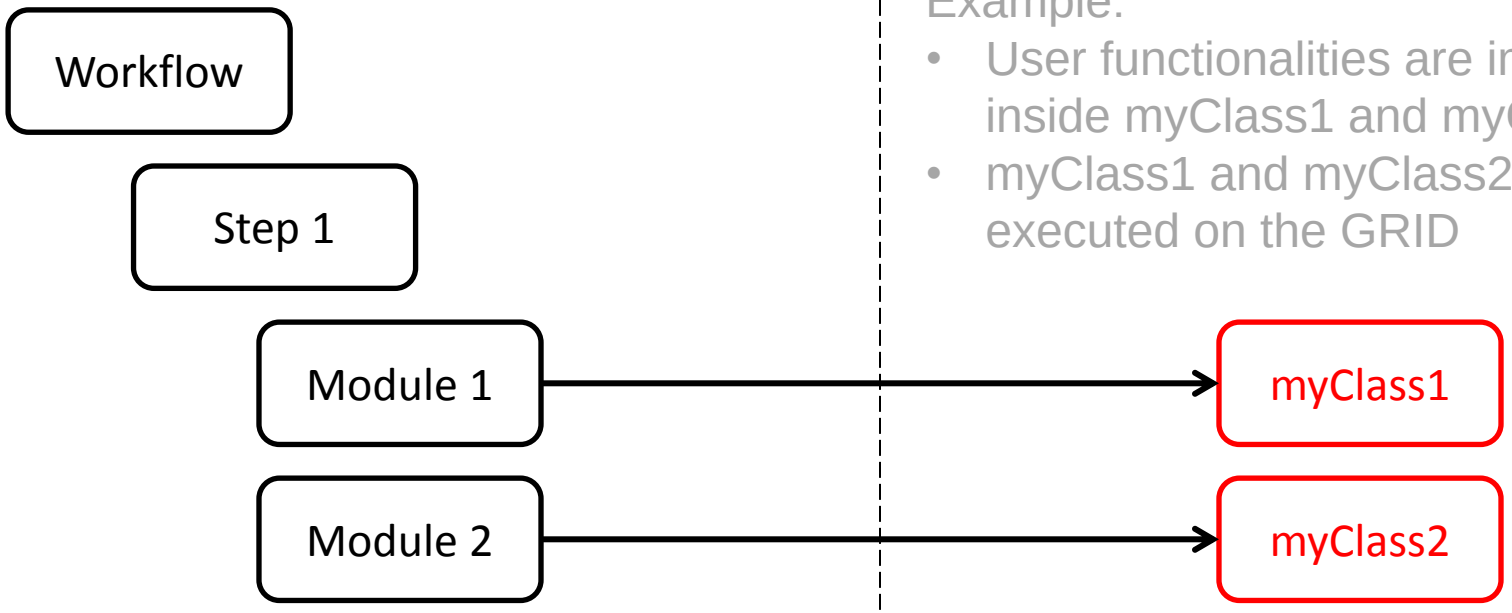
Example:

- User functionalities are implemented inside myClass1 and myClass2
- myClass1 and myClass2 are executed on the GRID

1 Framework – Introduction (3/5)

Job submission

User functionalities

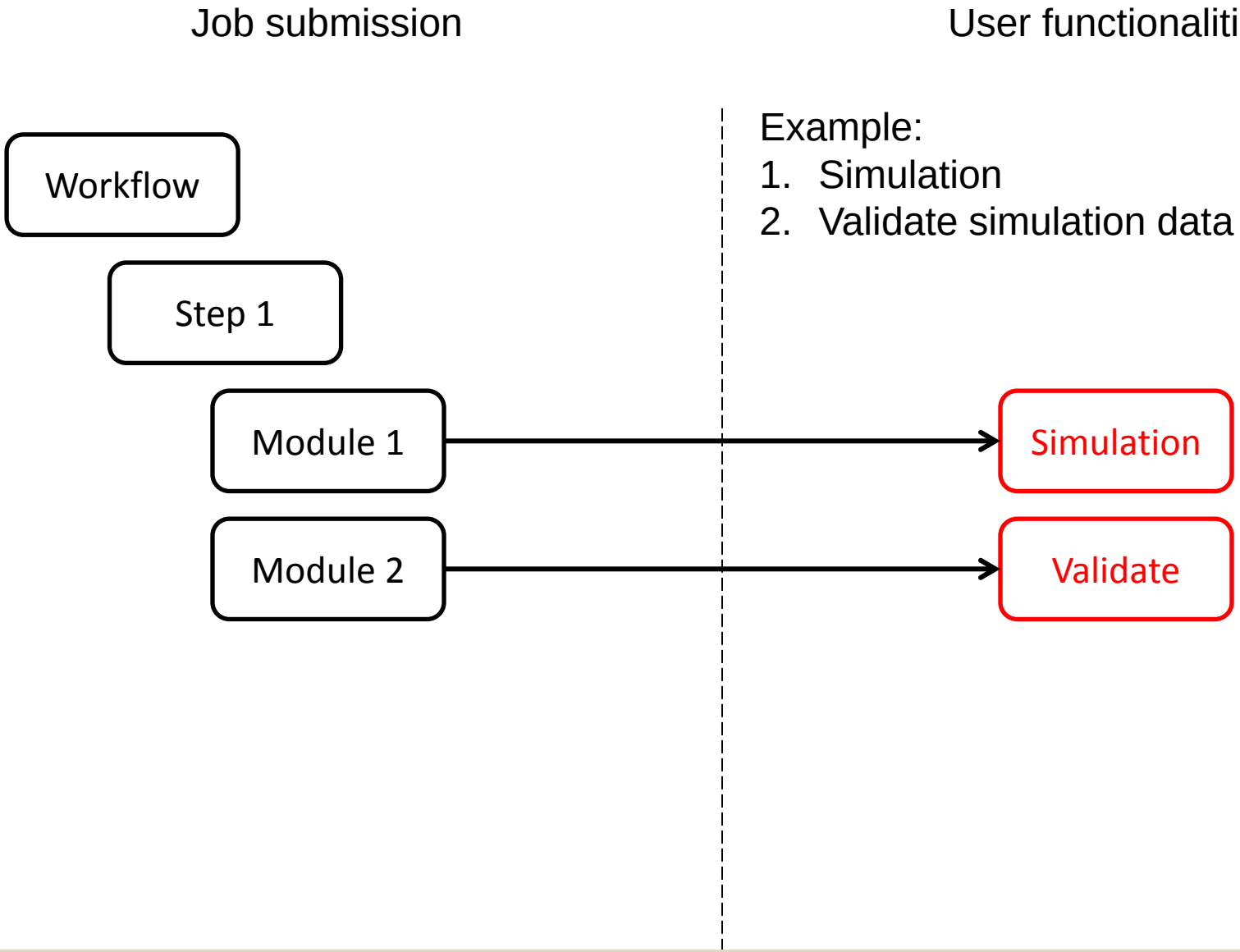


Example:

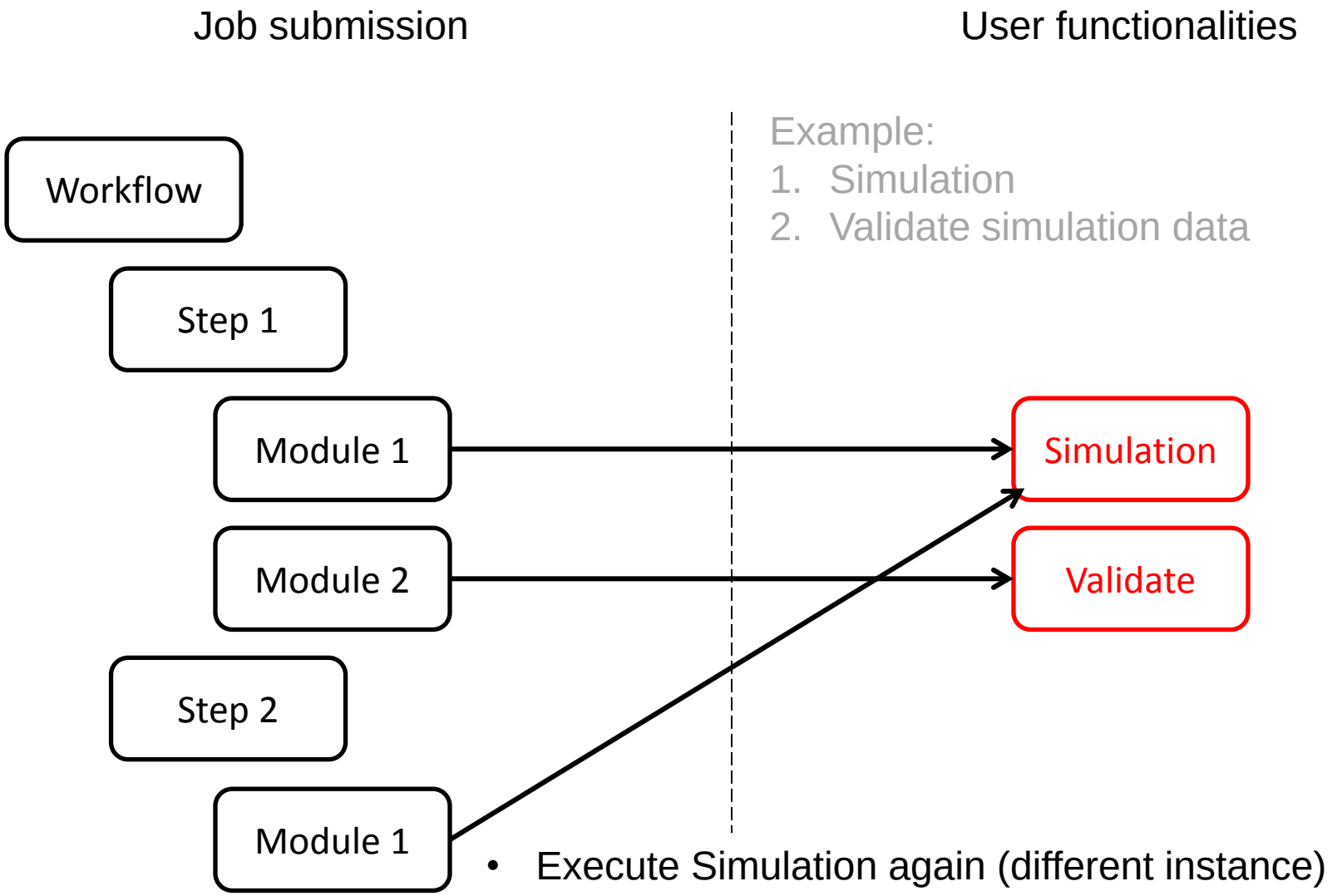
- User functionalities are implemented inside myClass1 and myClass2
- myClass1 and myClass2 are executed on the GRID

- Steps and Modules are used to specify the **execution order** on the GRID
- Workflows are not limited in the number of steps or modules

1 Framework – Introduction (4/5)



1 Framework – Introduction (5/5)

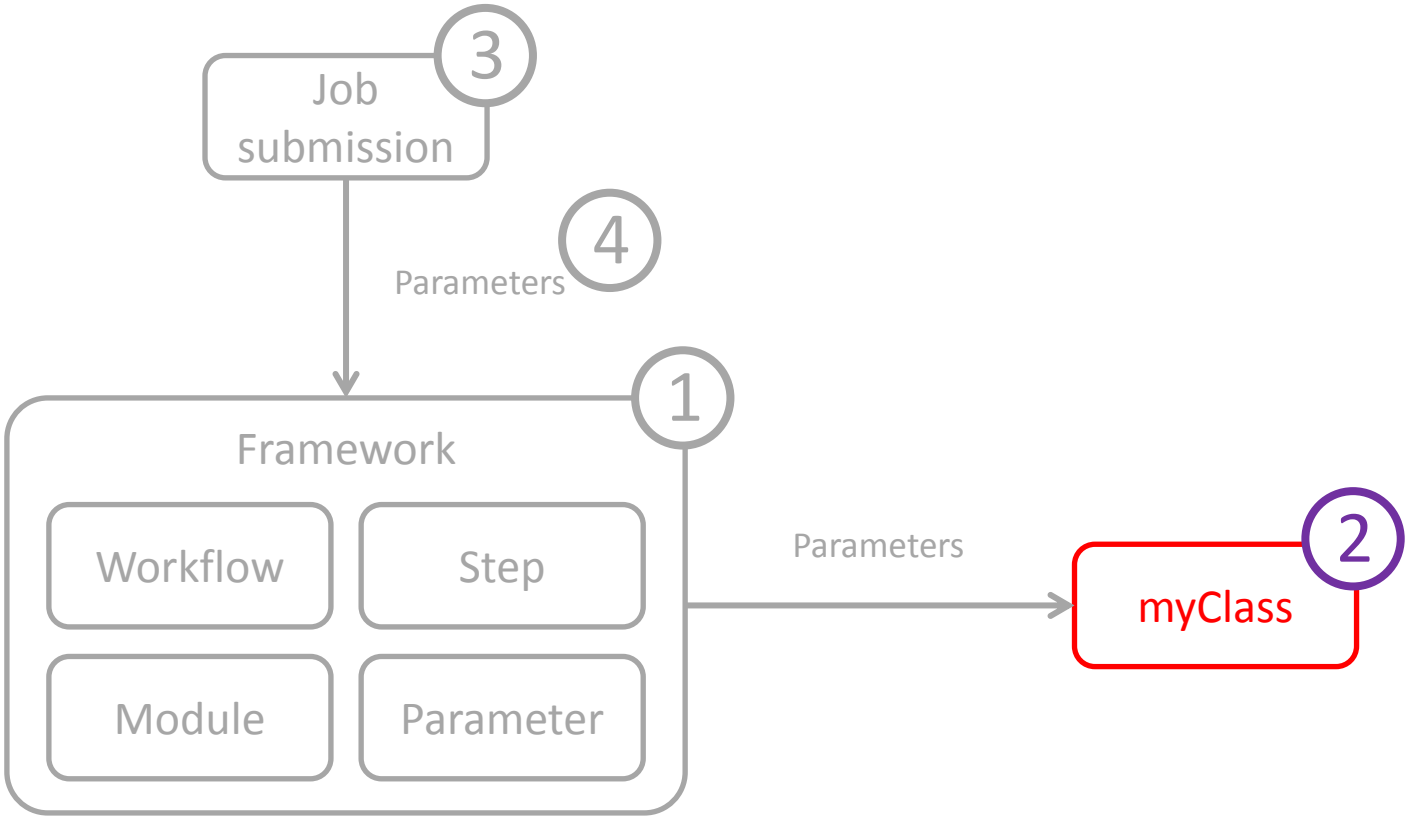


Framework Overview: Workflow, Step, Module

My description:

A framework for loading code and execute it in

the specified order with the specified parameters



2 myClass – Introduction (1/3)

myClass

2 myClass – Introduction (2/3)

User functionalities

- The implementation of the method "execute" is the **sole** requirement for myClass

myClass

2 myClass – Introduction (3/3)

User functionalities

- The implementation of the method "execute" is the **sole** requirement for myClass

myClass

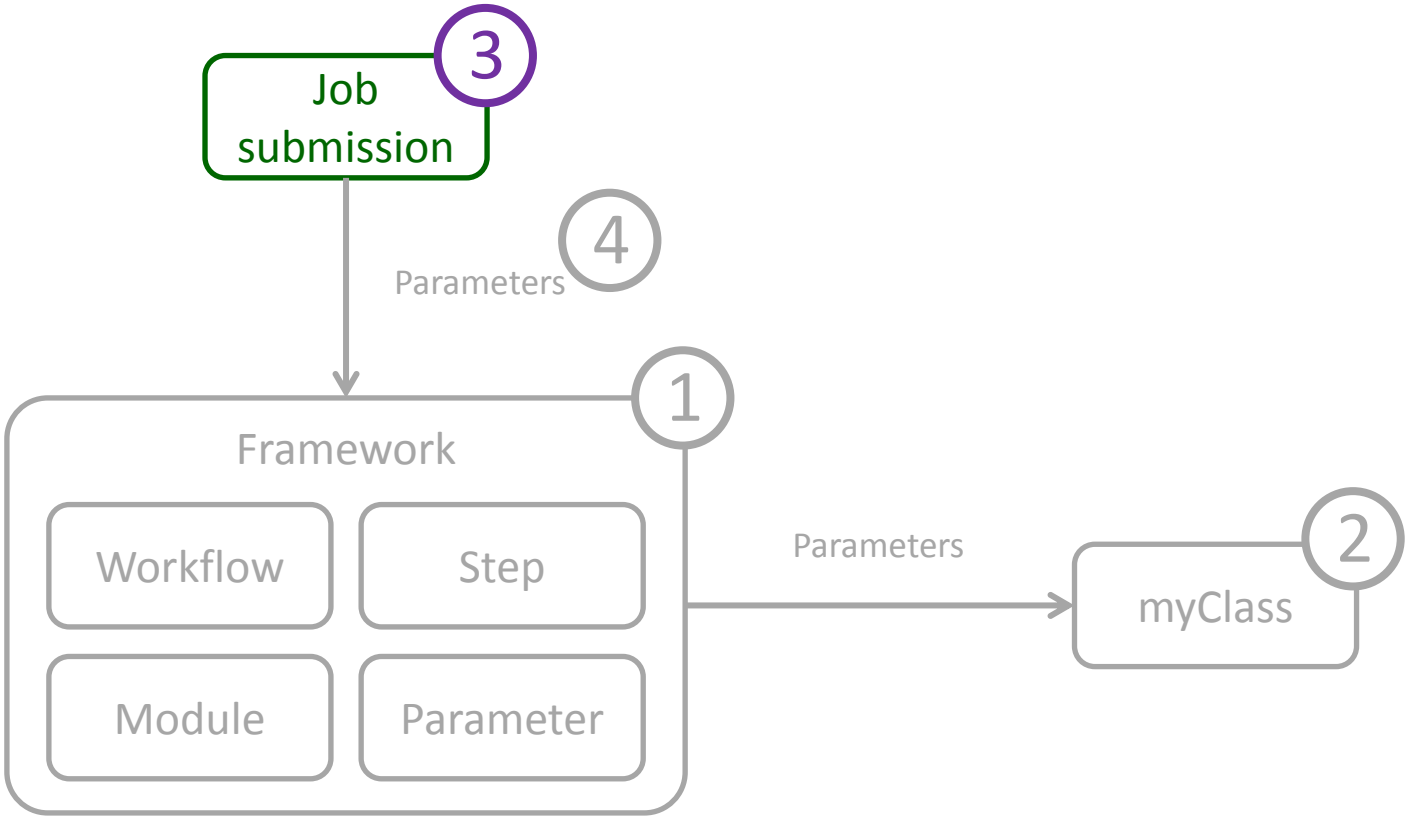
```
$DIRAC/ILCDIRAC/Workflow/Modules/myClass.py
-------------------------------------------

class myClass( .. ):

  def execute(self):
    your own python code
```

Framework Overview: Workflow, Step, Module

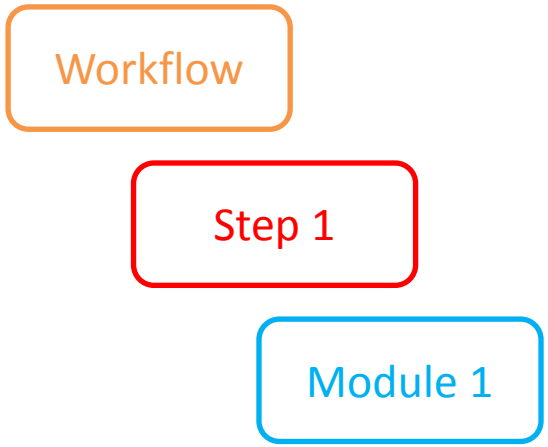
My description:

A framework for loading code and execute it in

the specified order with the specified parameters

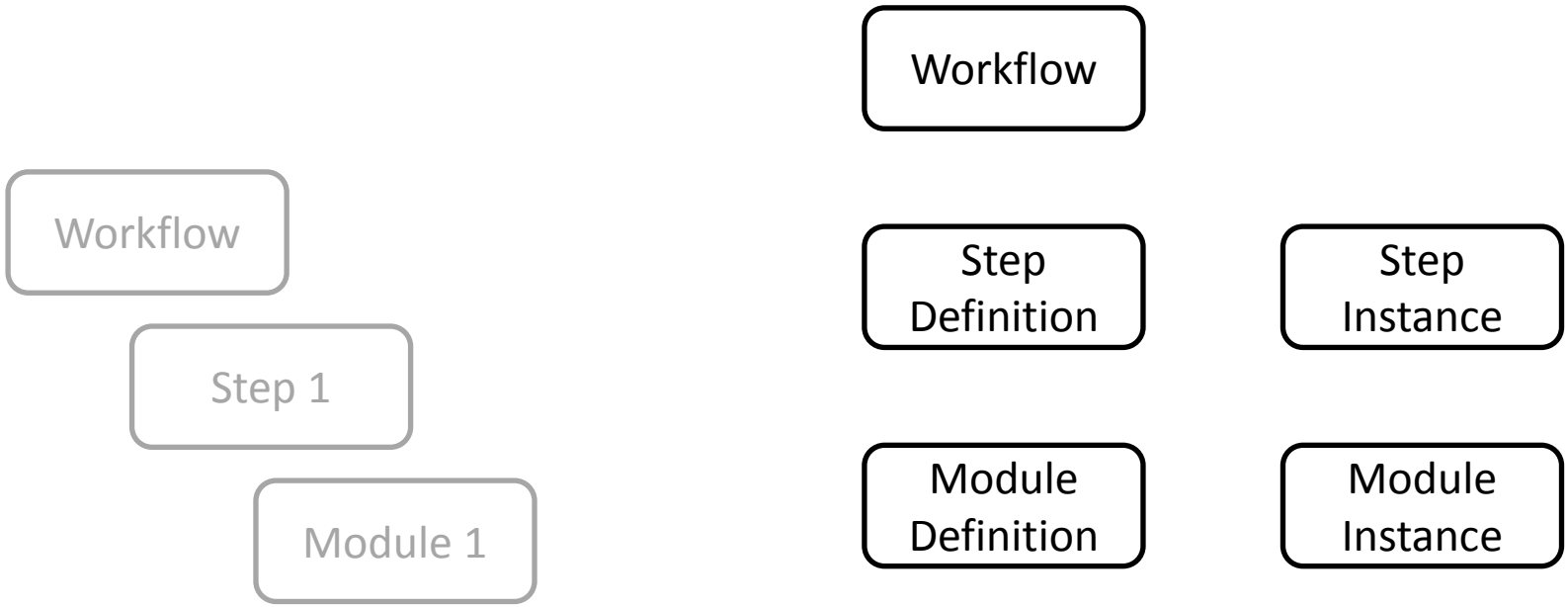


3 Job submission – Introduction (1/5)



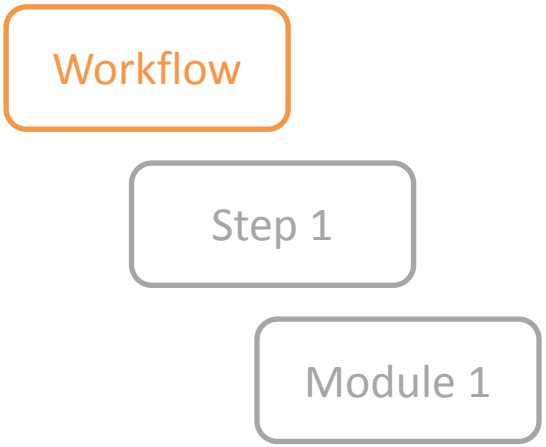
Example: simplest case with
- one step
- one module

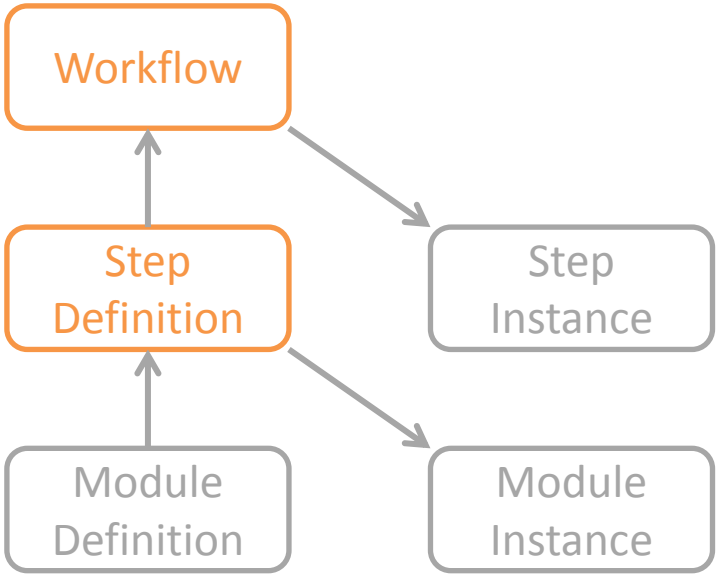
3 Job submission – Introduction (2/5)



Example: simplest case with
- one step
- one module

3 Job submission – Introduction (3/5)





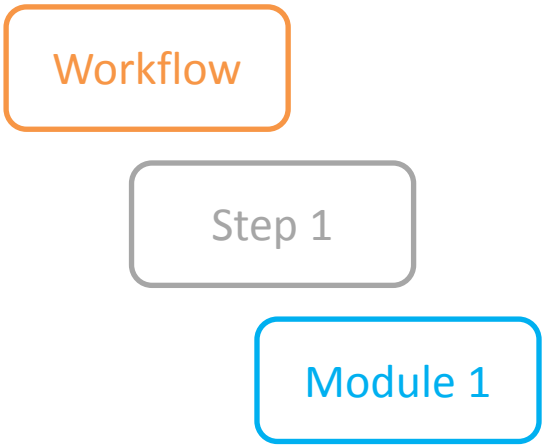
Example: simplest case with
- one step
- one module

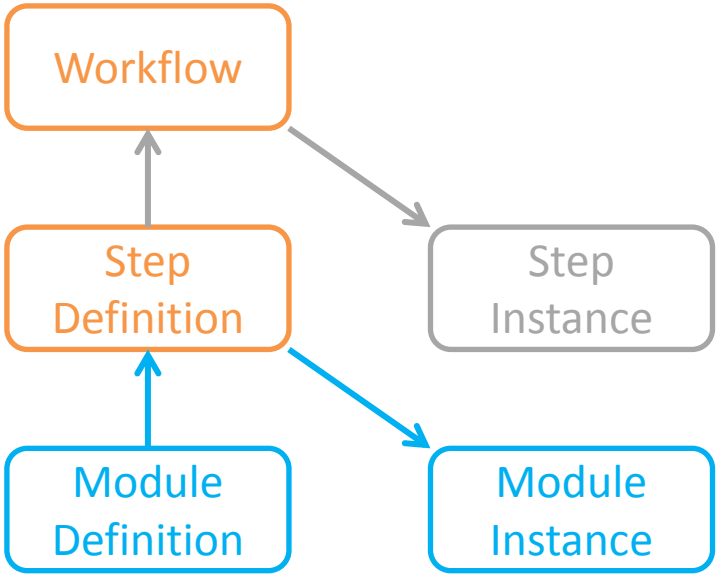
1. Create Workflow
2. Create StepDefinition

3. Create ModuleDefinition
4. Add ModuleDefinition to StepDefinition
5. Create ModuleInstance

6. Add StepDefinition to Workflow
7. Create StepInstance

3 Job submission – Introduction (4/5)





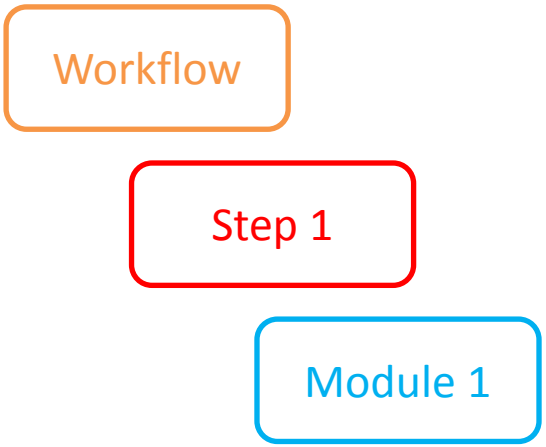
Example: simplest case with
- one step
- one module

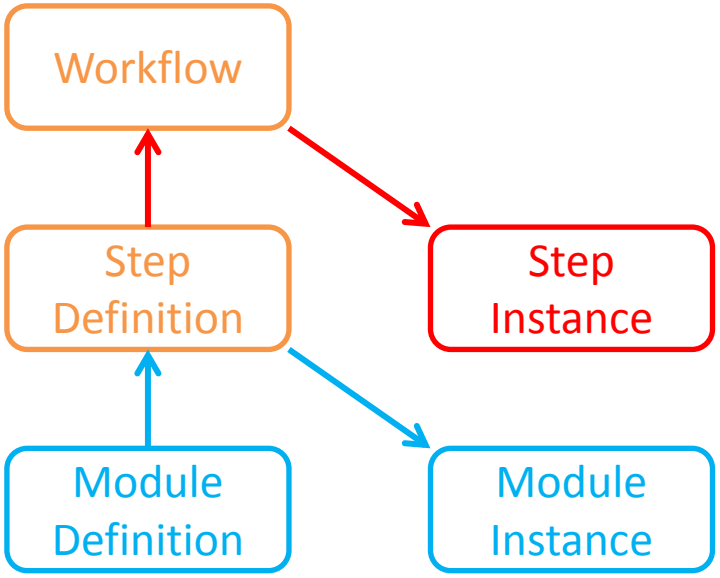
1. Create Workflow
2. Create StepDefinition

3. Create ModuleDefinition
4. Add ModuleDefinition to StepDefinition
5. Create ModuleInstance

6. Add StepDefinition to Workflow
7. Create StepInstance

3 Job submission – Introduction (5/5)





Example: simplest case with
- one step
- one module

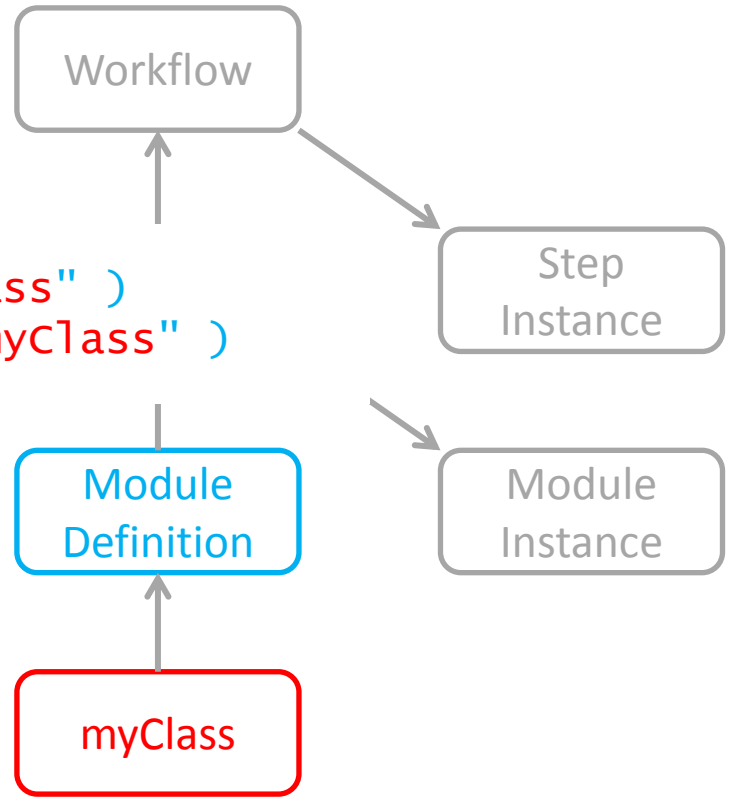
1. Create Workflow
2. Create StepDefinition

3. Create ModuleDefinition
4. Add ModuleDefinition to StepDefinition
5. Create ModuleInstance

6. Add StepDefinition to Workflow
7. Create StepInstance

3 Job submission – Module execution (1/3)

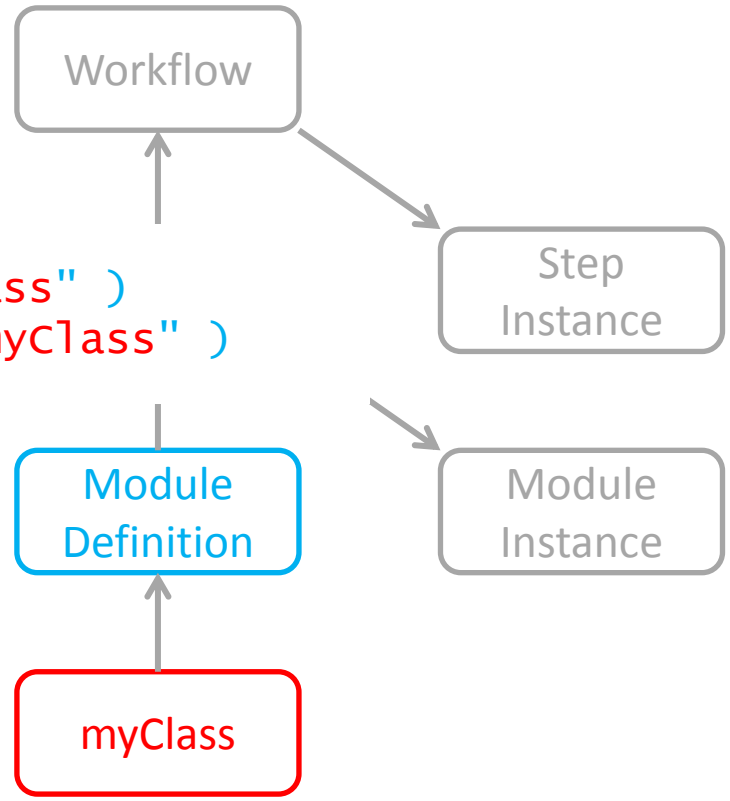
```
moduleD = ModuleDefinition( "myClass" )
moduleD.setBody( "from .. import myClass" )
```



3 Job submission – Module execution (2/3)

```
moduleD = ModuleDefinition( "myClass" )
moduleD.setBody( "from .. import myClass" )
```

1. moduleD.Body is executed to load the specified python module
2. An object of a type with name "myClass" is instantiated

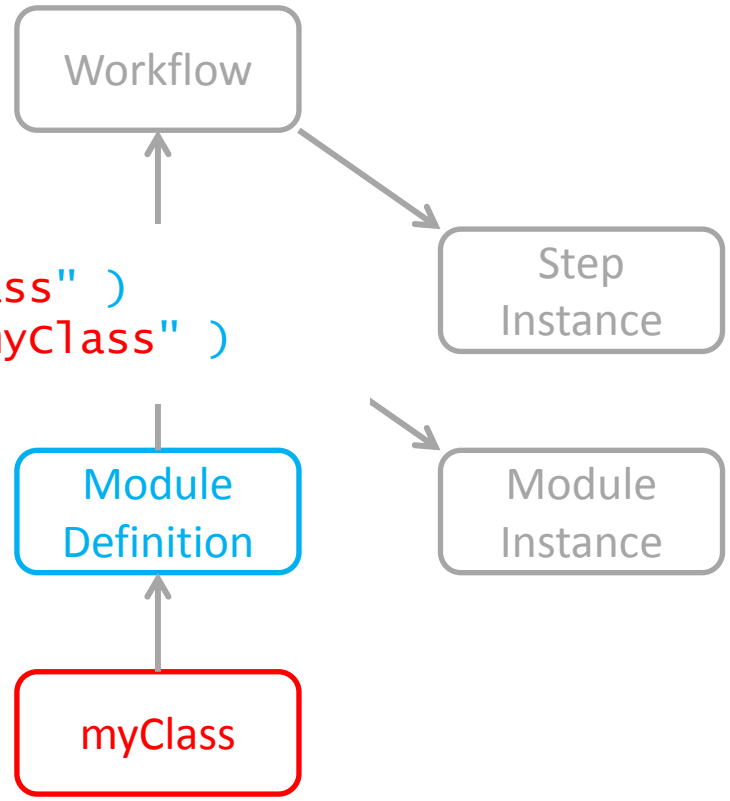


3 Job submission – Module execution (3/3)

```
moduleD = ModuleDefinition( "myClass" )
moduleD.setBody( "from .. import myClass" )
```

1. moduleD.Body is executed to load the specified python module

2. An object of a type with name "myClass" is instantiated

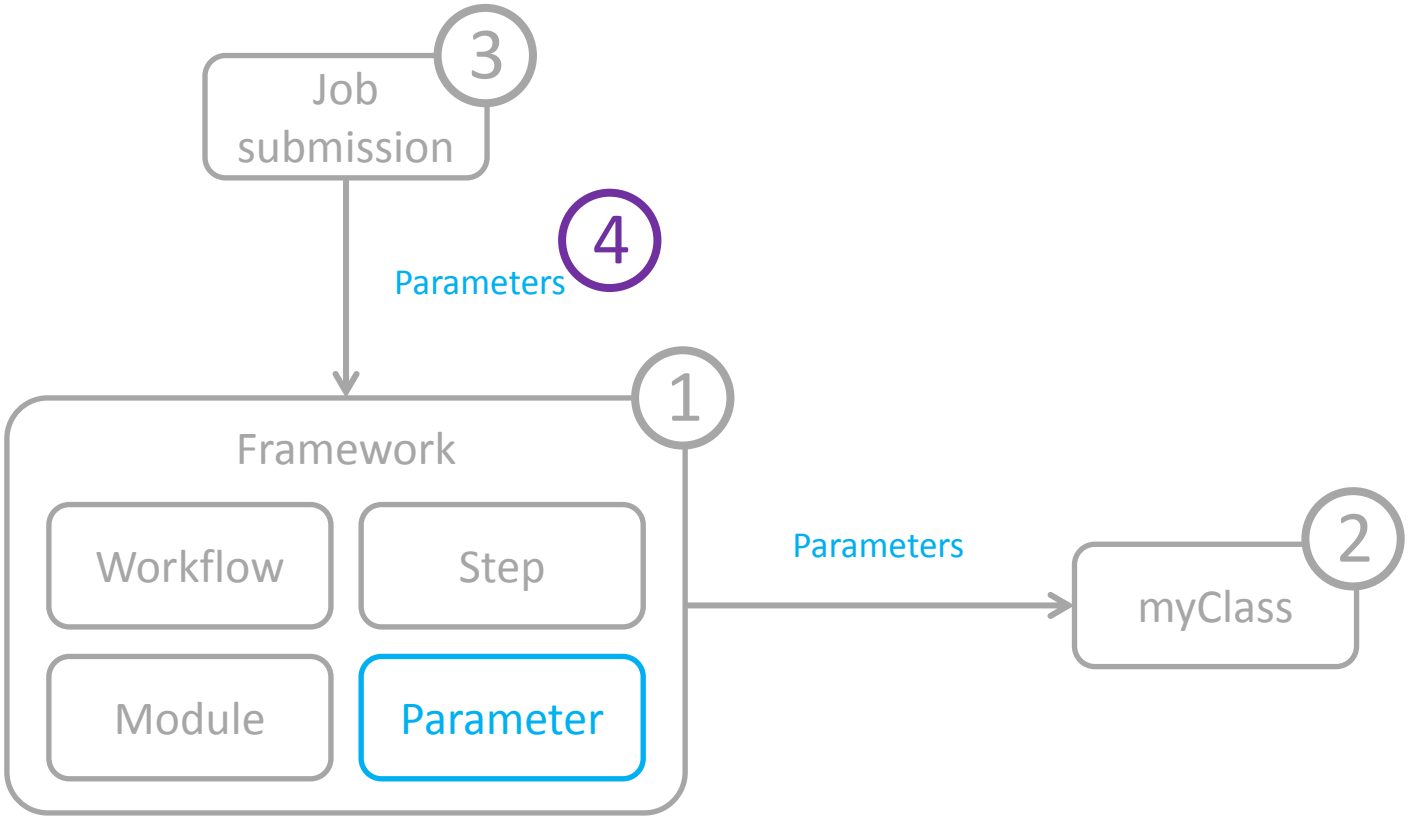


Framework Overview: Workflow, Step, Module

My description:

A framework for loading code and execute it in

the specified order with the specified parameters



4 Parameters – Introduction (1/8)

Problem:

- User's functionality needs parameters to run

myClass

4 Parameters – Introduction (2/8)

- Parameters are used to pass values from job submission to job execution

Parameter

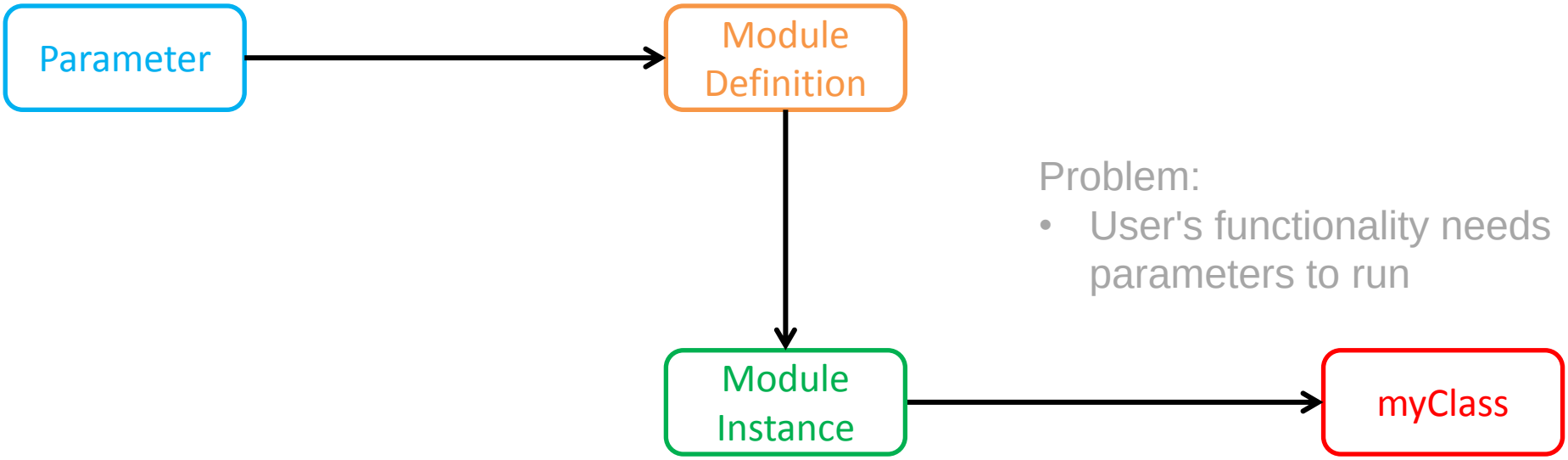
Problem:
- User's functionality needs parameters to run

myClass

4 Parameters – Introduction (3/8)

- Parameters are used to pass values from job submission to job execution

Example:
- Module parameter

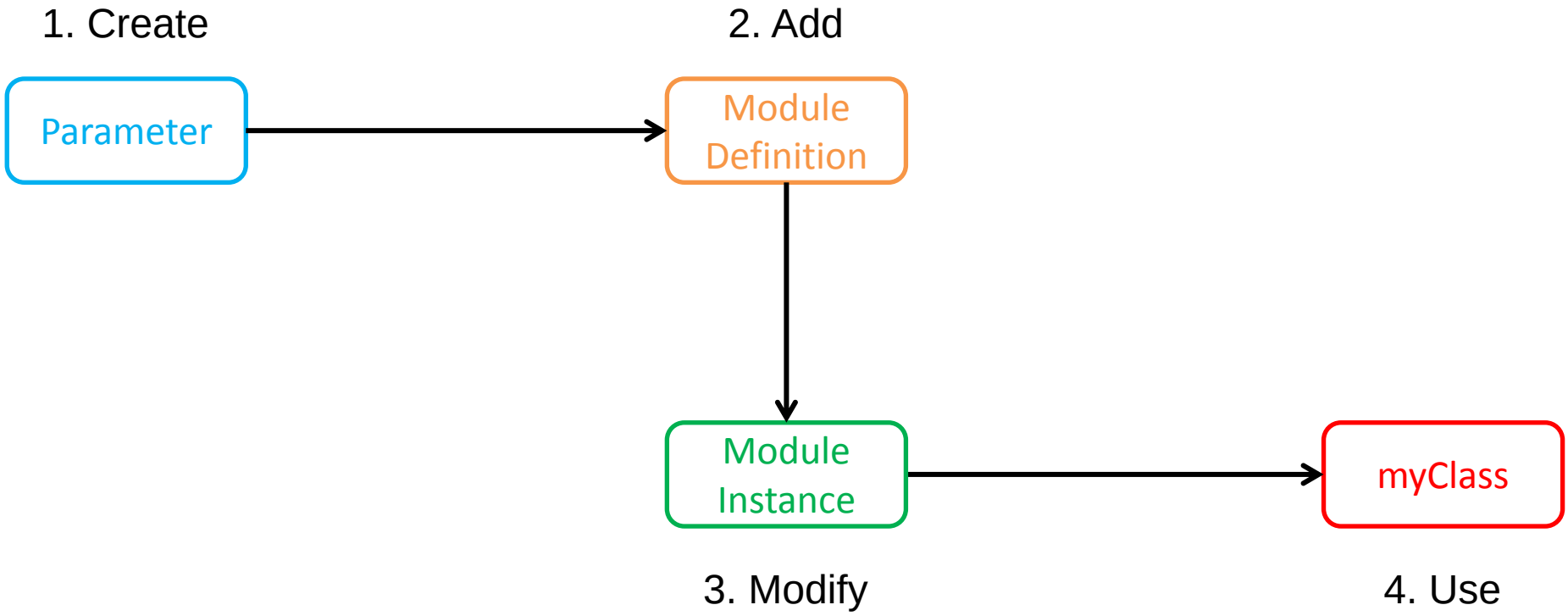


Problem:
- User's functionality needs parameters to run

4 Parameters – Introduction (4/8)

- Parameters are used to pass values from job submission to job execution

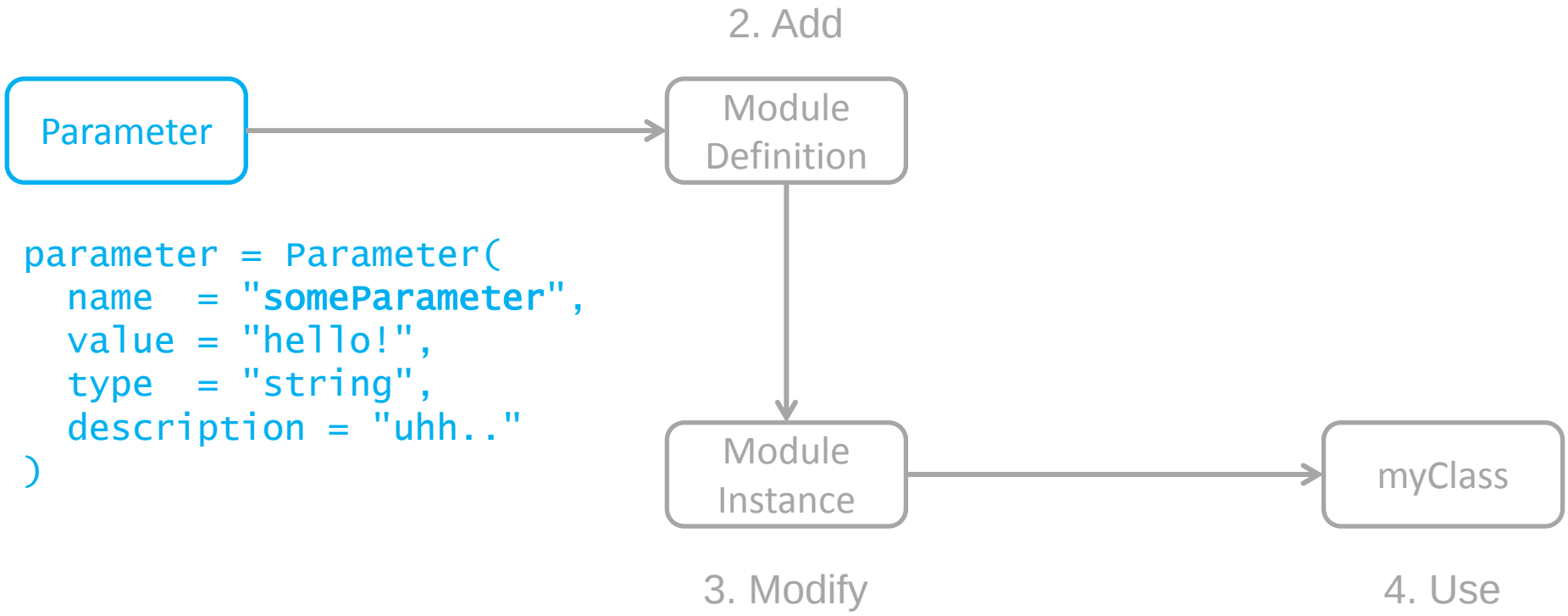
Example:
- Module parameter



4 Parameters – Introduction (5/8)

- Parameters are used to pass values from job submission to job execution

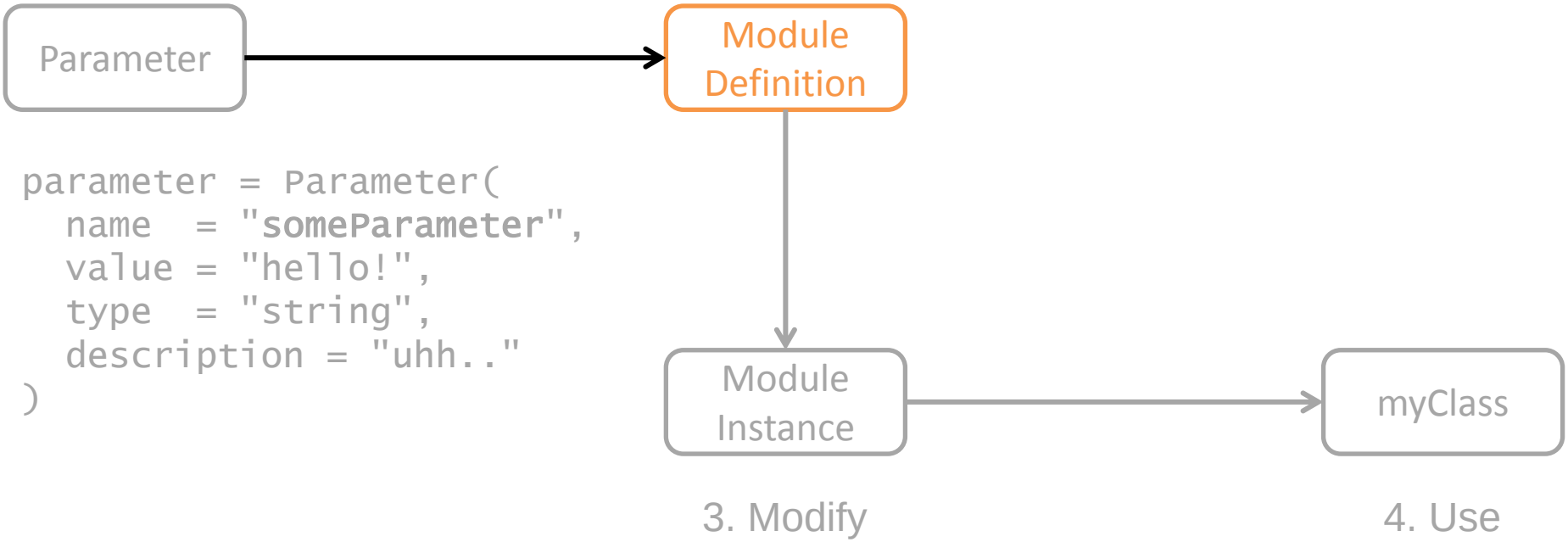
Example:
- Module parameter



4 Parameters – Introduction (6/8)

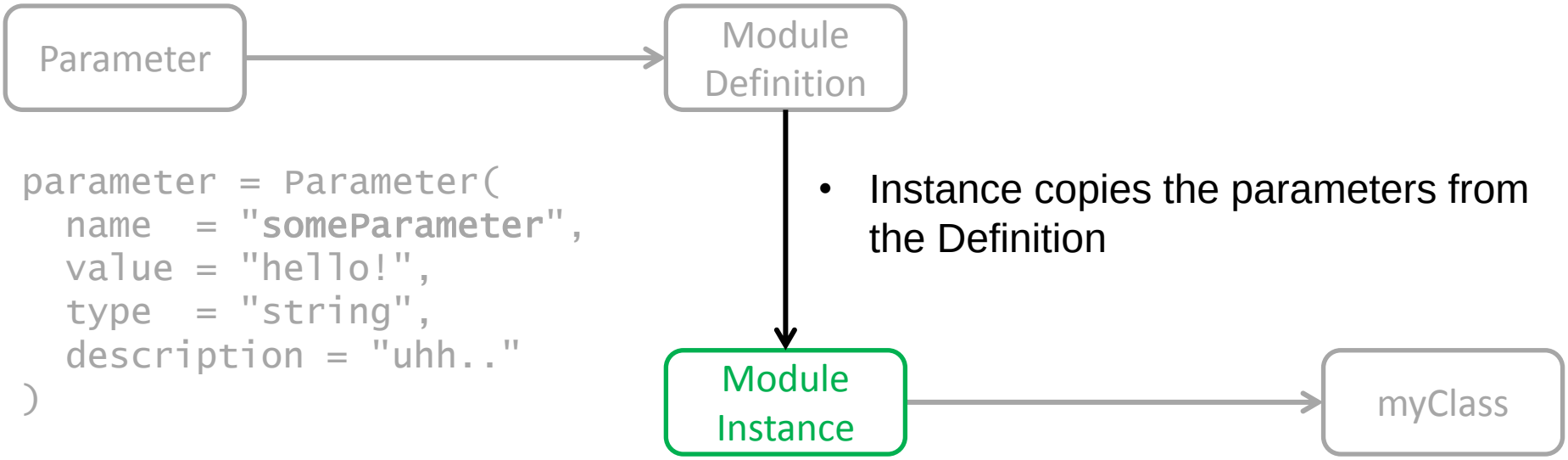
- Parameters are used to pass values from job submission to job execution

```
moduleDefinition.addParameter(parameter)
```



4 Parameters – Introduction (7/8)

```
moduleDefinition.addParameter(parameter)
```



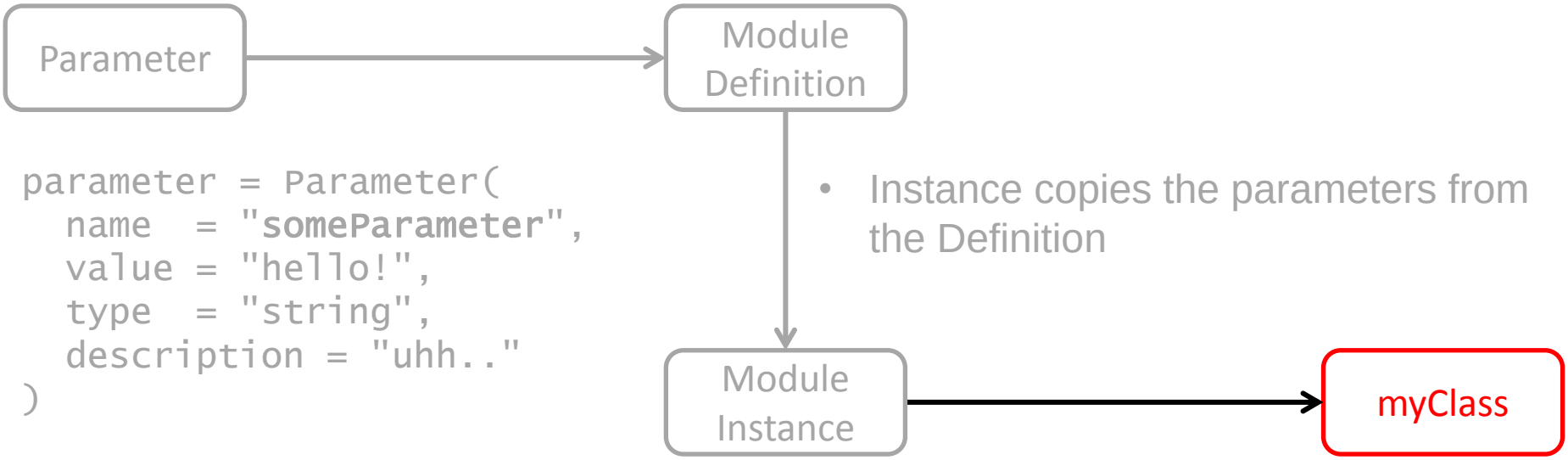

```
parameter = Parameter(
  name  = "someParameter",
  value = "hello!",
  type  = "string",
  description = "uhh.."
)
```

- Instance copies the parameters from the Definition

```
moduleInstance.setValue(
  "someParameter",
  "hi!"
)
```

4 Parameters – Introduction (8/8)

```
moduleDefinition.addParameter(parameter)
```



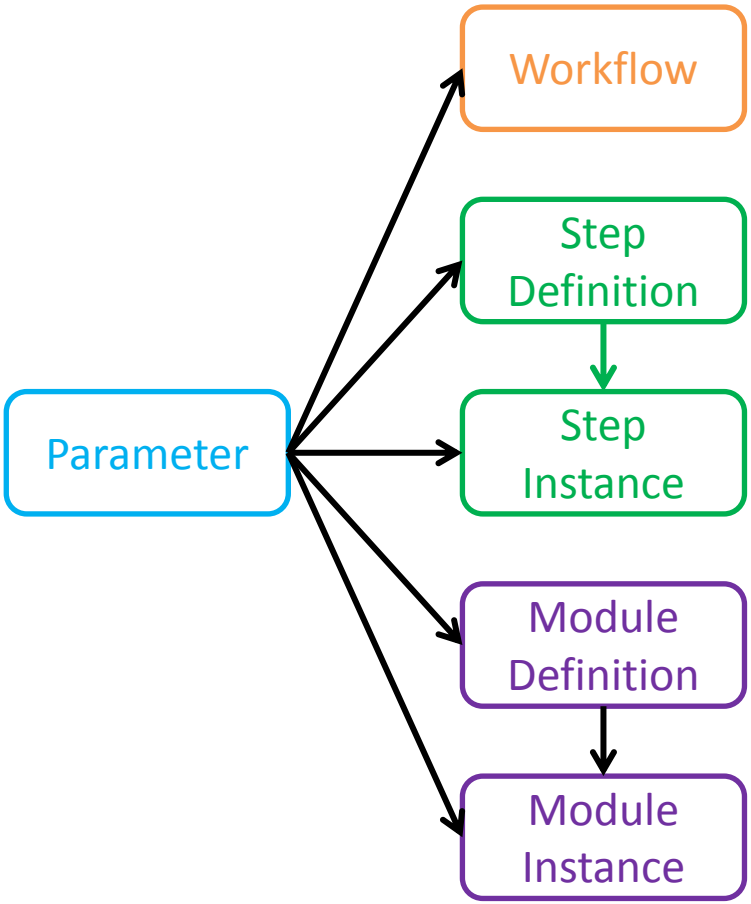

```
parameter = Parameter(
  name  = "someParameter",
  value = "hello!",
  type  = "string",
  description = "uhh.."
)
```

- Instance copies the parameters from the Definition

```
moduleInstance.setValue(
  "someParameter",
  "hi!"
)
```

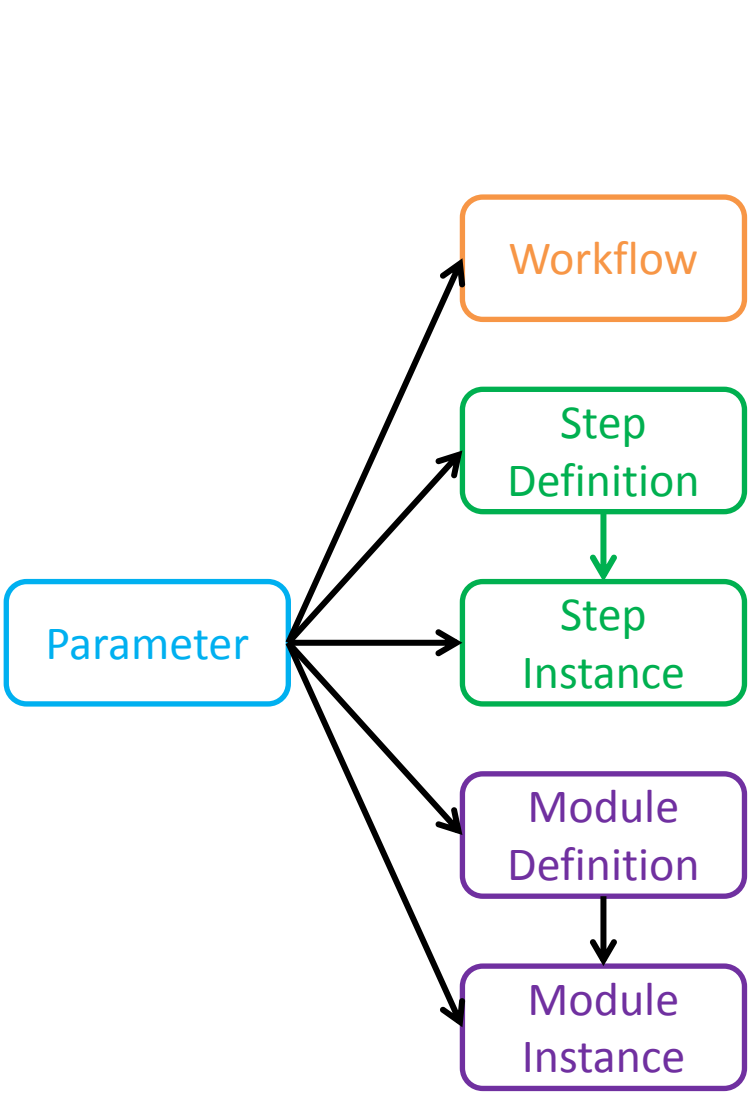
```
// accessible by
myClass.someParameter
```

4 Parameters – Overview (1/2)



- Parameters can be added to Workflow, StepDefinition, StepInstance, ModuleDefinition and ModuleInstance.
- Instance copies the parameters from the Definition

4 Parameters – Overview (2/2)




```
myClass.workflow_commons["parameterName"]
```

- Parameters can be added to Workflow, StepDefinition, StepInstance, ModuleDefinition and ModuleInstance.

```
myClass.step_commons["parameterName"]
```

- Instance copies the parameters from the Definition

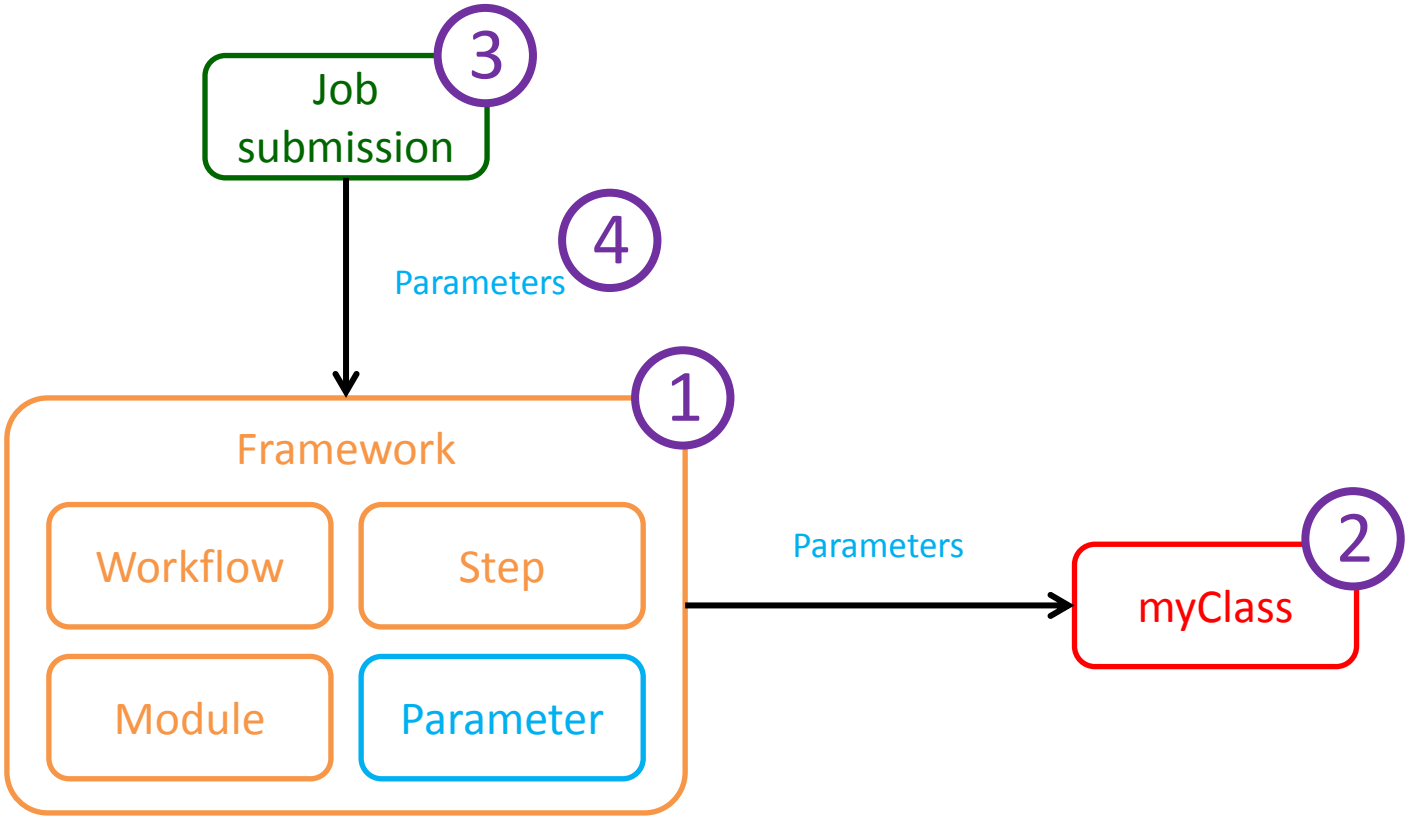
```
myClass.parameterName
```

Framework Overview: Workflow, Step, Module

My description:

A framework for loading code and execute it in

the specified order with the specified parameters



Part III

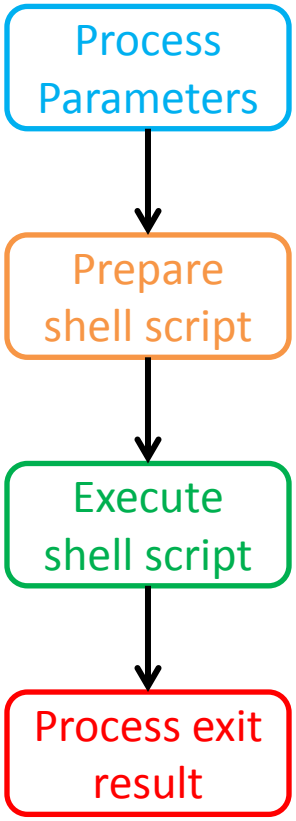
Simple example

Simple example (1/2)

https://svnweb.cern.ch/trac/dirac/browser/ILCDIRAC/trunk/ILCDIRAC/Workflow/Modules

Simple example (2/2)

https://svnweb.cern.ch/trac/dirac/browser/ILCDIRAC/trunk/ILCDIRAC/Workflow/Modules



Part IV
Conclusion

Conclusion

The Workflow framework is the product of
a natural evolution from simple jobs to complex jobs,
and it already exists in DIRAC

My description:
A framework for loading code and execute it in
the specified order with the specified parameters