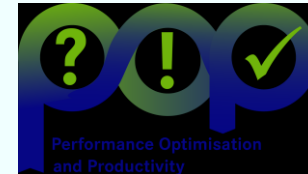


JUBE - A Flexible, Application- and Platform-Independent Environment for Benchmarking

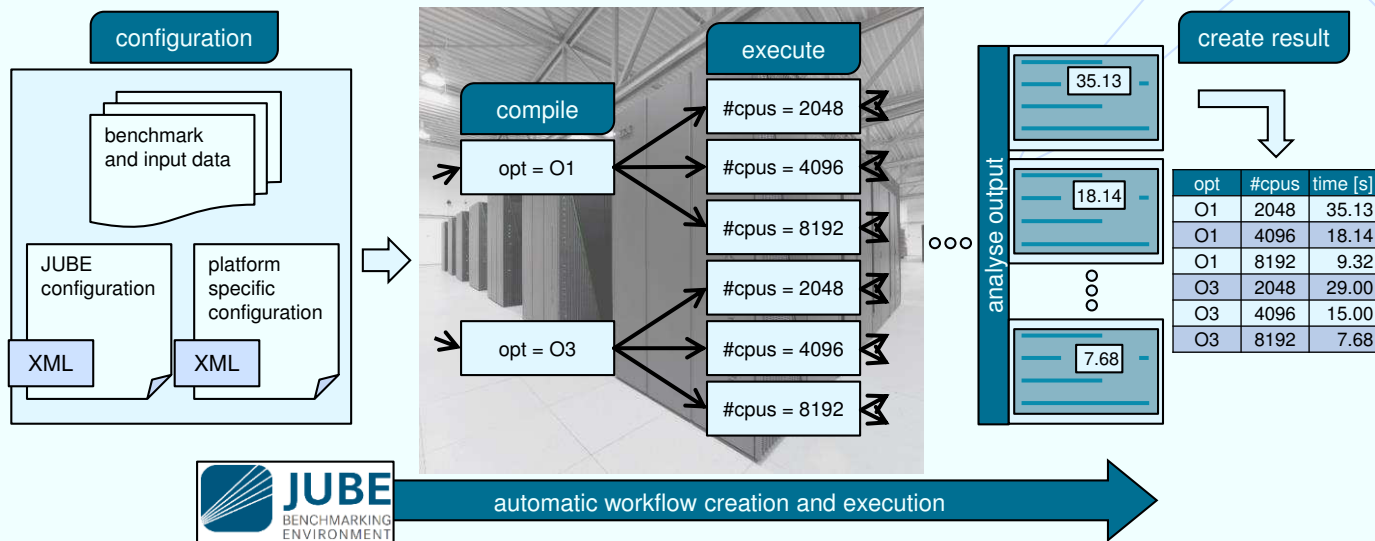
Sebastian Lührs – Jülich Supercomputing Centre
8th - 11th December 2015 – EoCoE - PoP Workshop – Jülich





What is JUBE?

- Generic, configurable environment to run, monitor and analyse benchmarks in a systematic way
- Developed 2008, redesigned 2014
- Also usable for testing or production scenarios





JUBE history

- Development started in 2004
- JUBE version 1
 - Perl based 
 - Used in many European projects like DEISA and PRACE
- 2014 complete new release: JUBE version 2
 - Python based 
 - New, more flexible input file layout
 - New command line options
 - Current version: 2.1.0



JUBE 1 file format



JUBE 2 file format

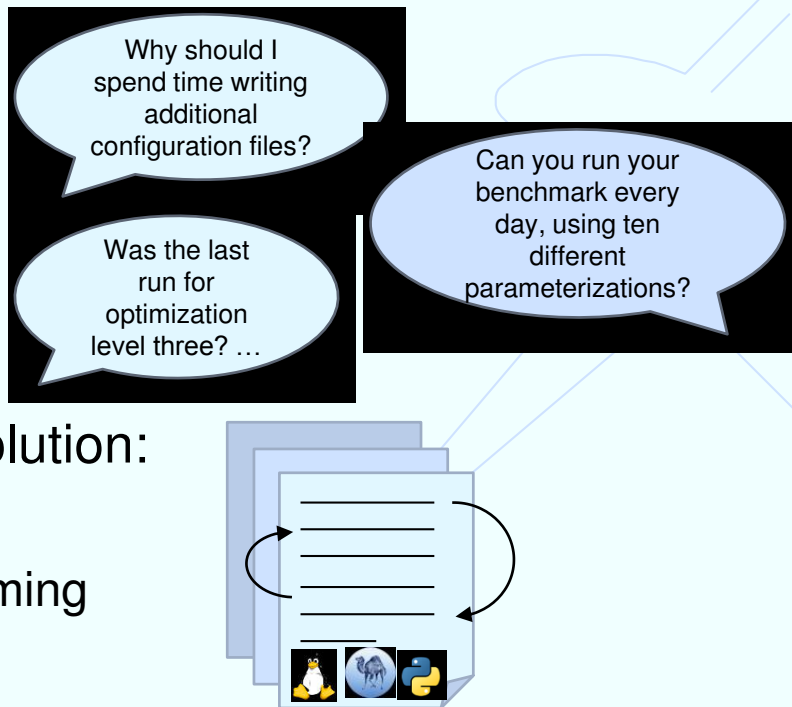




Why JUBE?

Alternatives:

- Manual benchmarking:
 - Easy to use
 - Time-consuming
 - Very error-prone
- Benchmark specific script solution:
 - Optimized
 - Changes can be time-consuming
 - Portability problems

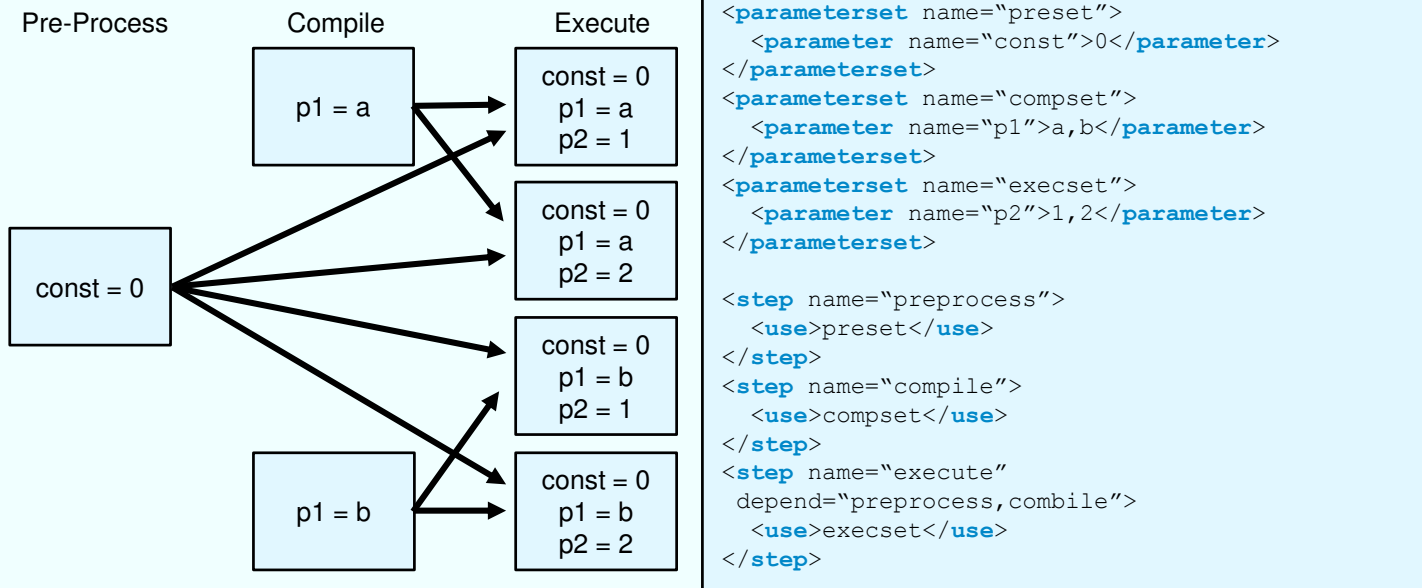


JUBE provides a generic workflow and parameter handling environment, but also supports more flexible and specialised approaches.



Key Concept: Workflow creation

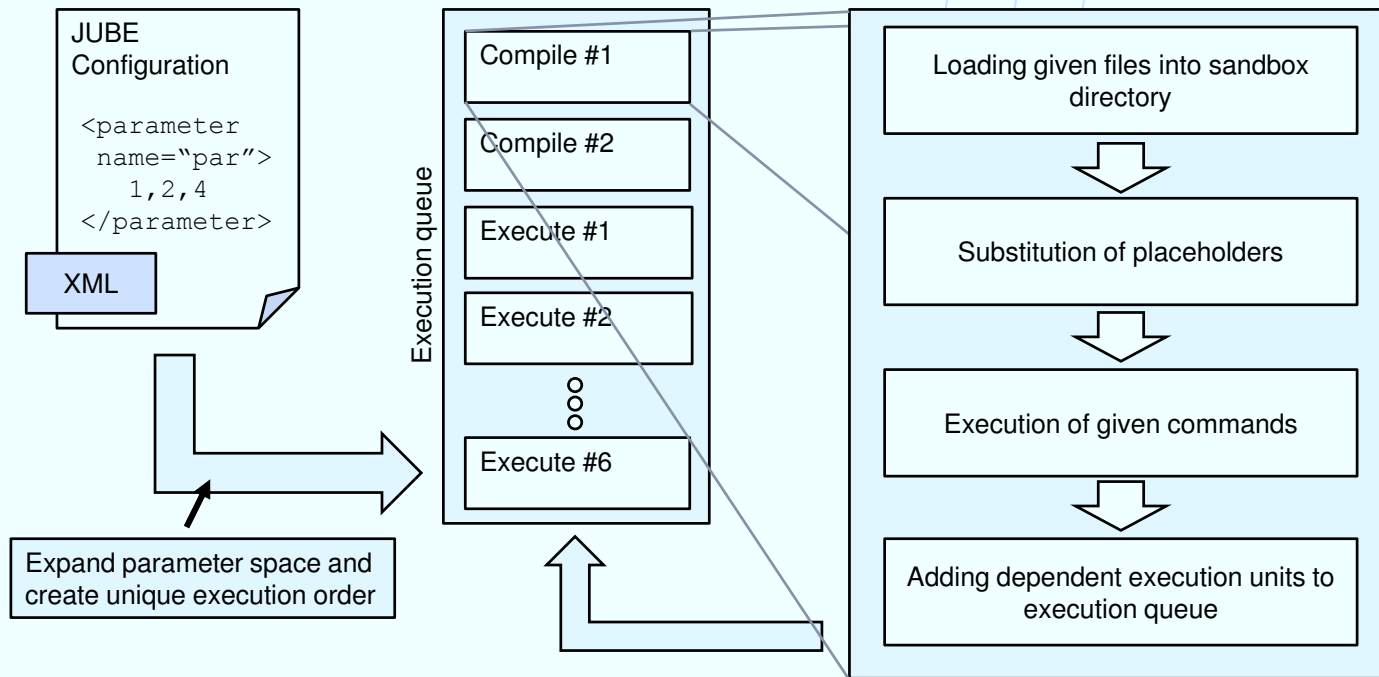
- Dependency driven step structure
- Parameter based expansion of steps





Execution order

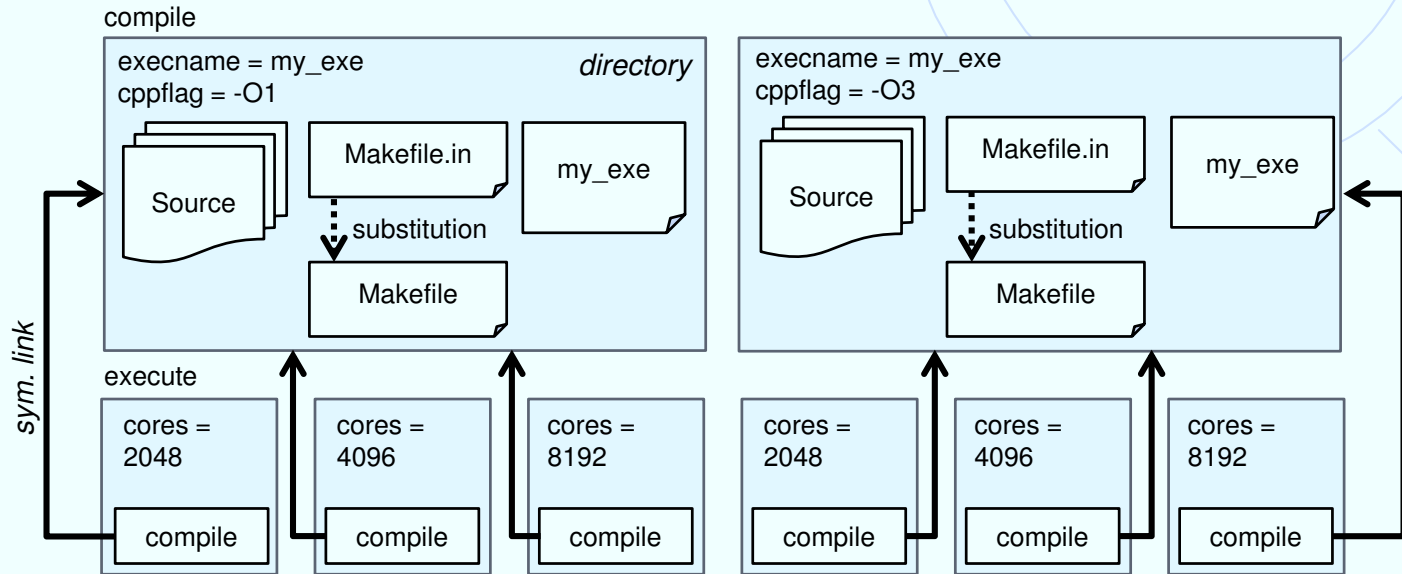
- Internal execution queue creates unique execution order





Key Concept: Directory and data handling

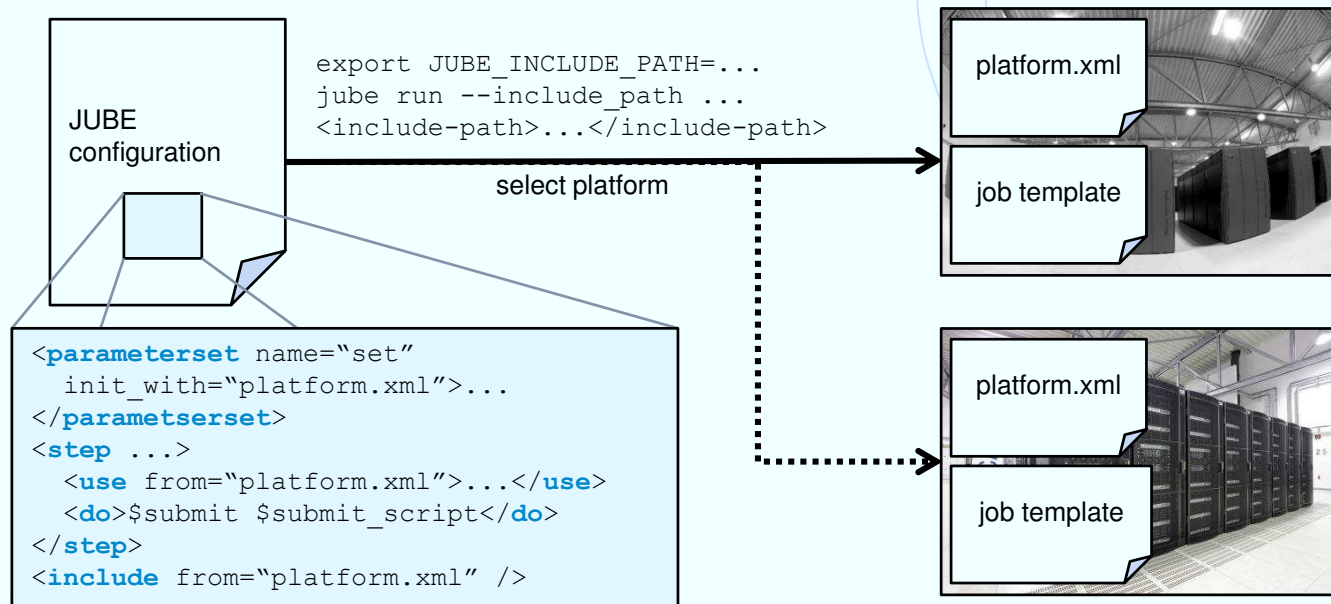
- Each parameter/step combination runs in a separate sandbox directory
- Dependent steps can be accessed using sym. links





Key Concept: Platform independence

- Separation of platform dependent and independent configuration options





JUBE general workflow

```
<jube>
  <benchmark name="bench" outpath="./benchmark_runs">
    <parameterset name="compileset">
      <parameter name="execname">my_exe</parameter>
      <parameter name="cppflagslist">
        -O1,-O2
      </parameter>
    </parameterset>
```

```
    <fileset name="sources">
      <copy>src/*</copy>
    </fileset>
```

```
    <substituteset name="compilesub">
      <iofile in="Makefile.in" out="Makefile" />
      <sub source="#PROGNAME#" dest="$execname" />
    </substituteset>
```

```
    <step name="compile">
      <use>compileset</use>
      <use>sources</use>
      <use>compilesub</use>
      <do>make OPT=$cppflagslist</do>
    </step>
```

```
    <step name="execute" depend="compile">
      ...
    </step>
```

compile

```
execname = my_exe
cppflagslist = -O1
```

Source

Makefile.in

Makefile

my_exe

```
execname = my_exe
cppflagslist = -O2
```

Source

Makefile.in

Makefile

my_exe

execute

compile

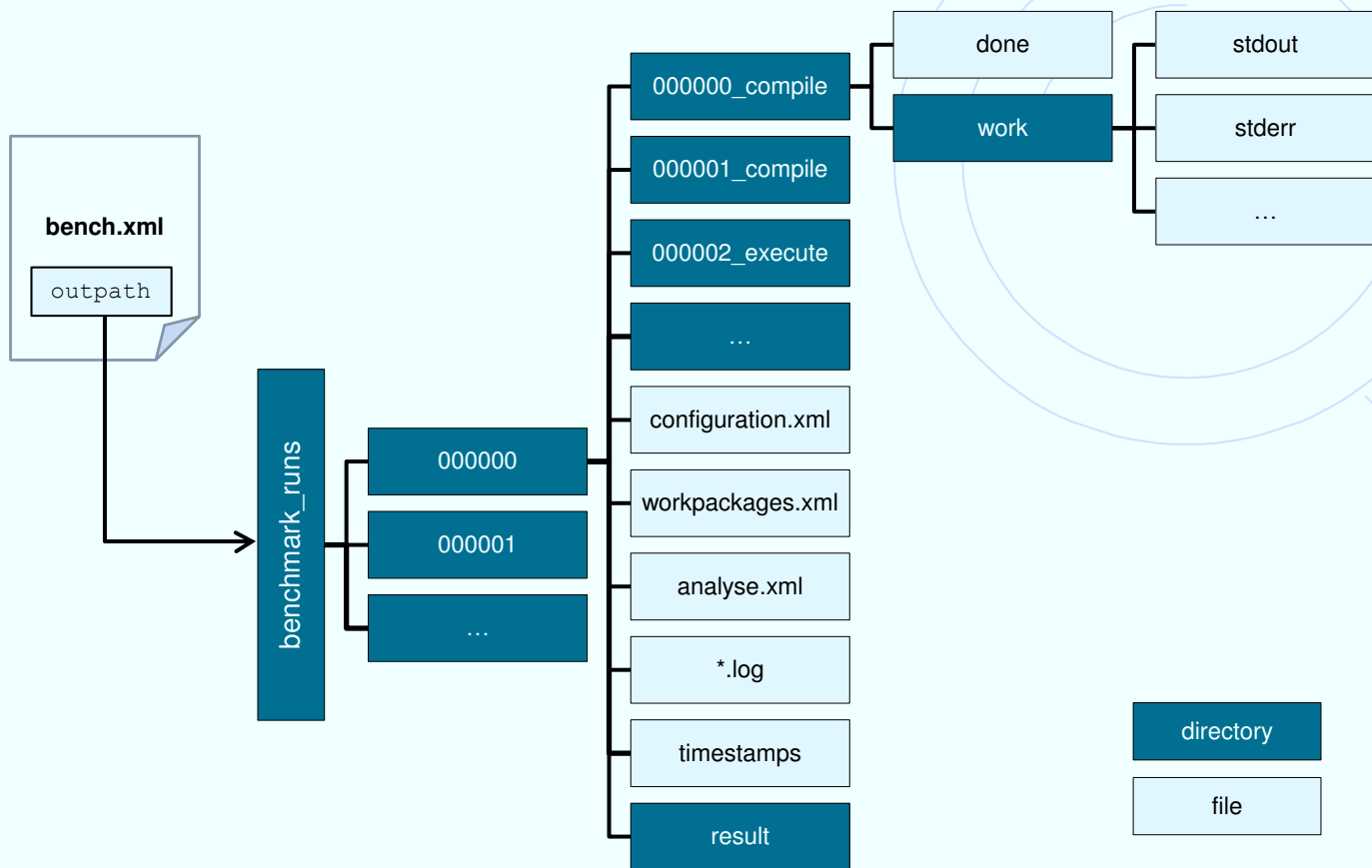
compile

...

analyse and result creation



Directory structure





XML input file format

- XML must be well formed:
 - Only one root element: `<jube>`
 - `<a><b>...</a></b>` not allowed
 - Every tag must be closed (`<a>...</a>` or `<a/>`)
 - `<a attr1="..." attr1="..." />` not allowed
 - `<a attr=... />` not allowed (missing `"`)
- Normal XML comment syntax can be used:
 - `<!-- ... -->`
- JUBE tags can be validated using available DTD, schema, or RELAX NG file





Command line access

Start a new benchmark run

- `jube run benchmark.xml`

Continue an existing benchmark run

- `jube continue benchmark_dir [--id <id>]`

Analyse the benchmark data

- `jube analyse benchmark_dir [--id <id>]`

Create and show result representation

- `jube result benchmark_dir [--id <id>]`





Help?!

Online documentation and tutorial

- `www.fz-juelich.de/jsc/jube`

Info mode

- `jube info benchmark_dir [--id <id>] [--step <stepname>]`

Command line accessible glossary

- `jube help <keyword>`

Logs

- `jube log benchmark_dir [--id <id>] [--command <cmd>]`

Debug mode

- `jube --debug run|continue|analyse|result ...`

Verbose mode

- `jube -v[vv] run ...`





HowTo: General file layout

```
<?xml version="1.0" encoding="UTF-8" ?>  
<jube>  
  <benchmark name="..." outputpath="...">  
    <parameterset/>  
    <fileset/>  
    <substituteset/>  
    <patternset/>  
  
    <step/>  
  
    <analyse/>  
    <result/>  
  </benchmark>  
</jube>
```

XML header line

JUBE root tag

benchmark area

set definitions

steps and commands

file analysers

result output creation

```
>> jube help general_structure
```



HowTo: Sets

- Main JUBE information storage technique
- Four different types of sets are available
 - `<parameterset>` Parameter storage
 - `<fileset>` Define all available files
 - `<substituteset>` Define file substitution
 - `<patternset>` Define the analyse pattern
- Set names must be unique
- Can be initialised by using an additional configuration file
- Available `<parameterset>`, `<fileset>` and `<substituteset>` are used and combined within a `<step>`
- Available `<patternset>` are used within `<analyse>`

```
>> jube help <setname>_tag
```



HowTo: Command execution

- `<do>...</do>` holds the executable commands
- All commands must use SHELL syntax (they will be executed by using `/bin/sh`)
- JUBE parameter can be used by using `$parametername`
- Parameter will be expanded in a pre-processing step
- Environment parameter can also be used
- JUBE stops execution if the command's return code fails
- Commands will only be executed once
- All `<do>` within the same `<step>` shares the same environment



HowTo: Command execution

```
<step name="..." depend="..."> ←  
  <use>...</use> ←  
  <use>...</use> ←  
  <do>...</do> ←  
  <do>...</do> ←  
</step>
```

name and dependencies

used sets

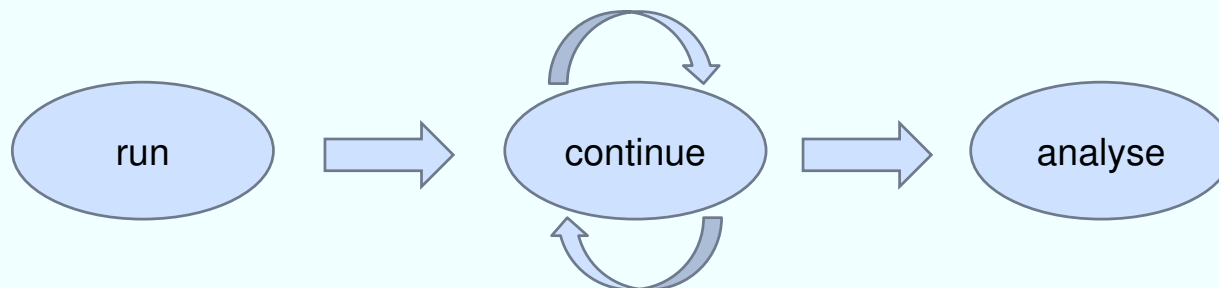
Shell commands

```
>> jube help step_tag  
>> jube help do_tag
```



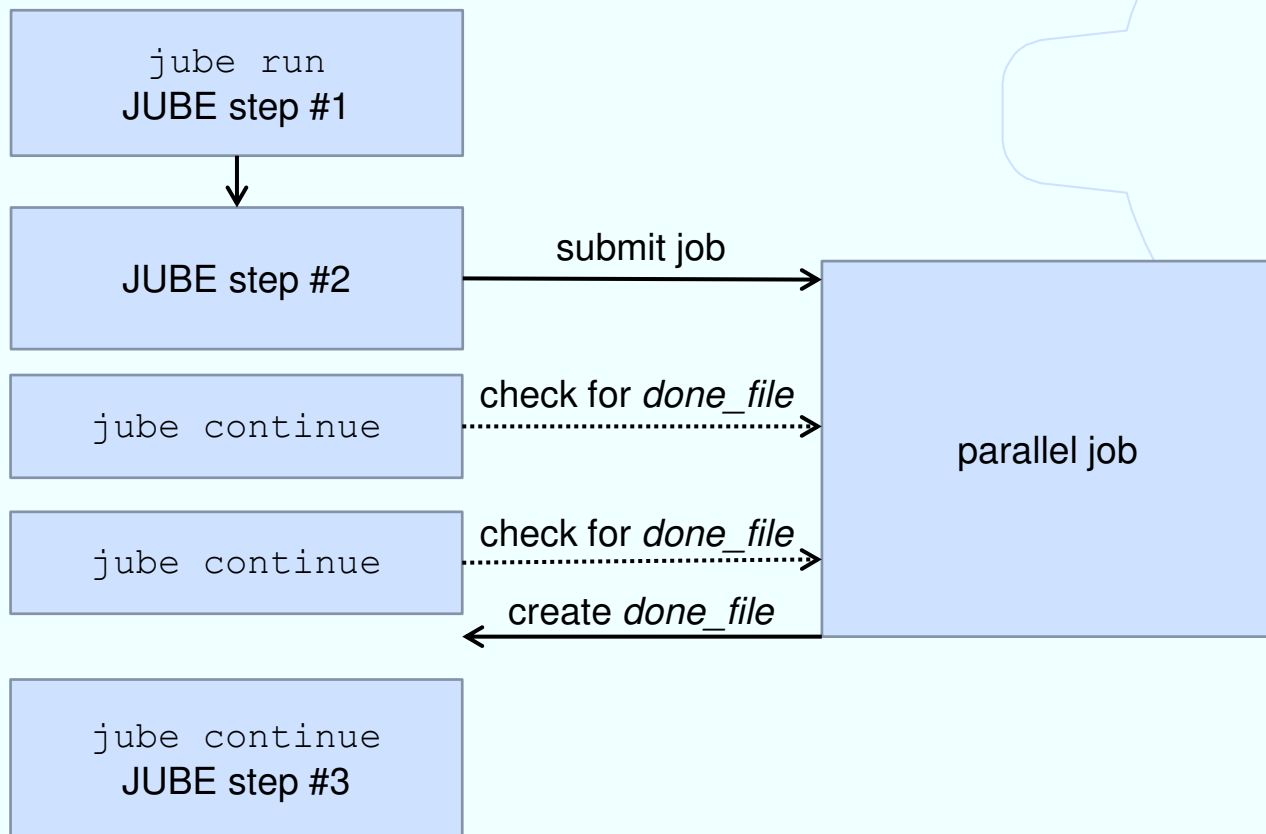
HowTo: Job submission

- A job template and substitution can be used to generalize the job submission process
- `<do>...</do>` is used to submit the job
- `<do>` returns immediately after the job was submitted. To wait for its execution use: `<do done_file=„...“>`
- The marker file, given by `done_file`, must be generated by the job script after the parallel part was executed
- `continue` triggers JUBE to check all available marker files





HowTo: Job submission





HowTo: Analyse

- Files will be analysed by using regular expressions which are defined by the given patterns
- Multiple occurrences of the same pattern create statistical values (average, minimum, maximum etc.)

```
<analyser name="...">
  <use>...</use>
  <analyse step="...">
    <file>...</file>
  </analyse>
</analyser>
```

analyser area

used patternset

step which should be
analysed

list of files

```
>> jube help analyser_tag
```



HowTo: Result creation

```
<result> <←  
  <use>...</use> <←  
  <table name="..."> <←  
    <column>...<column> <←  
  </table>  
</result>
```

result area

used analyser

table result type definition

column definition

```
>> jube help result_tag  
>> jube help table_tag
```



Example Benchmark Configurations

- **HPL:** High-Performance Linpack Benchmark
`www.netlib.org/benchmark/hpl`
- **IOR:** (InterleavedOrRandom) I/O benchmark
`sourceforge.net/projects/ior-sio`
- **mdtest:** Metadata test benchmark
`sourceforge.net/projects/mdtest`

Publicly available and extensible JUBE configuration file repository:

`https://github.com/FZJ-JSC/jube-configs`



Outlook and upcoming Features

- Extend job system interaction
 - Allow JUBE to monitor submitted jobs more easily
- Add another configuration file input format
 - Not everyone likes writing XML files by hand





Where to start?

Download and Tutorials:

- [`www.fz-juelich.de/jsc/jube`](http://www.fz-juelich.de/jsc/jube)
- Open Source (GPLv3)

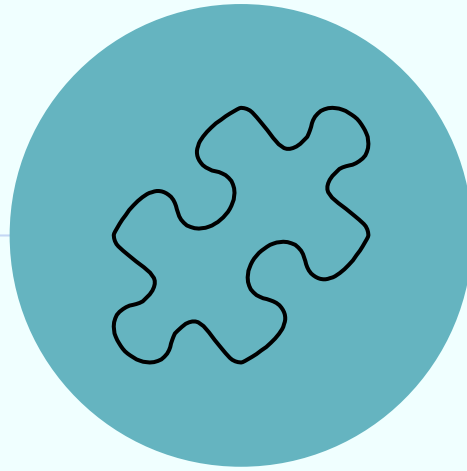
Prerequisites:

- OS: Linux
- Python 2.6, Python 2.7, Python 3.2
(or a more recent version)

Contact:

- [`jube.jsc@fz-juelich.de`](mailto:jube.jsc@fz-juelich.de)





DEMO