# INTRODUCTION TO JUBE

## A WORKFLOW AND BENCHMARKING ENVIRONMENT

26.04.2018 I THOMAS BREUER; SEBASTIAN LÜHRS



JUBE
BENCHMARKING
ENVIRONMENT

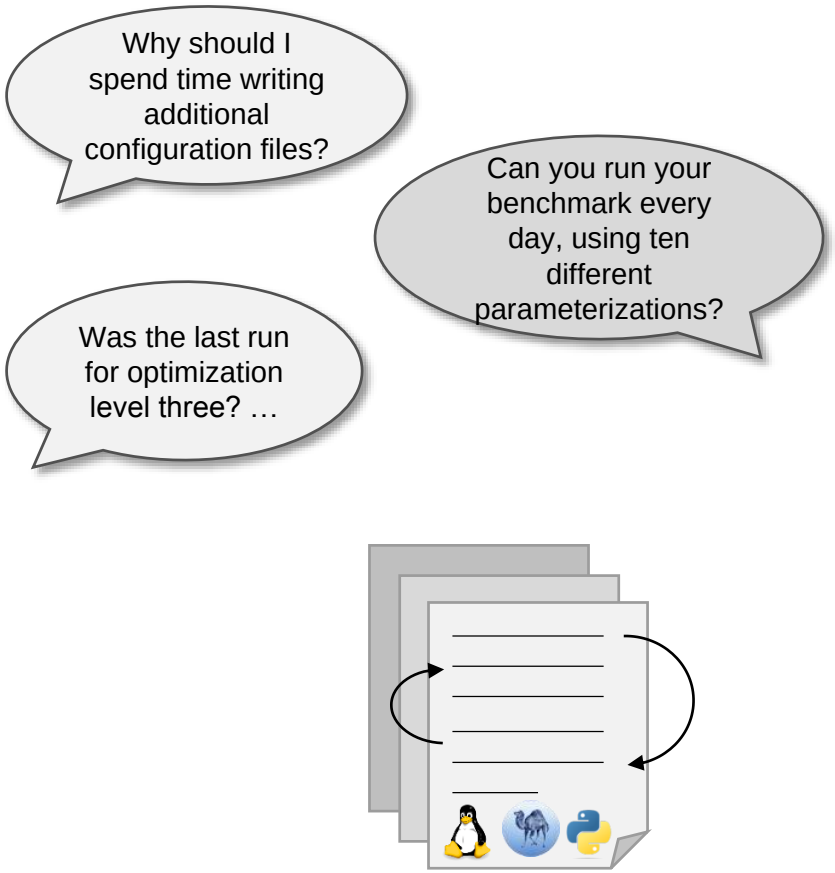
JÜLICH
Forschungszentrum

# OVERVIEW

- Motivation
- JUBE
  - Key concepts
  - Workflow
- Demo

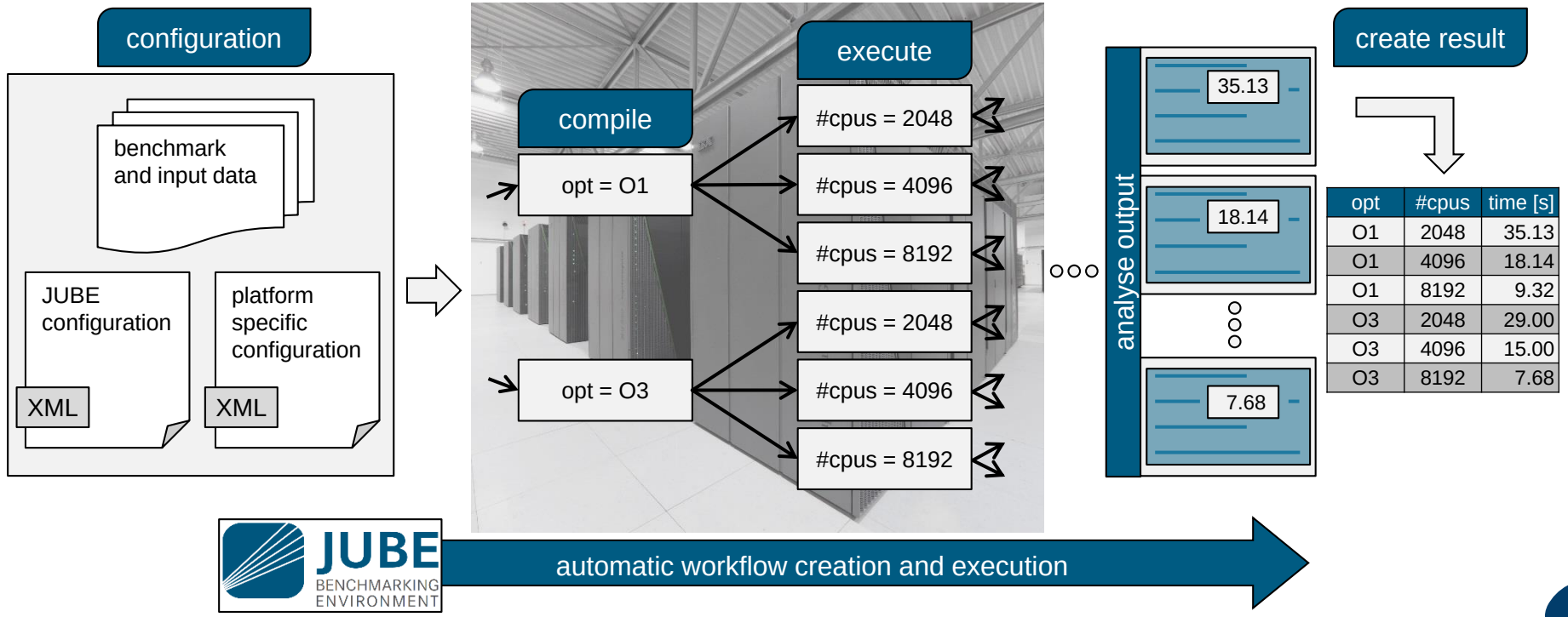
# WHY JUBE?

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



# WHAT IS JUBE?

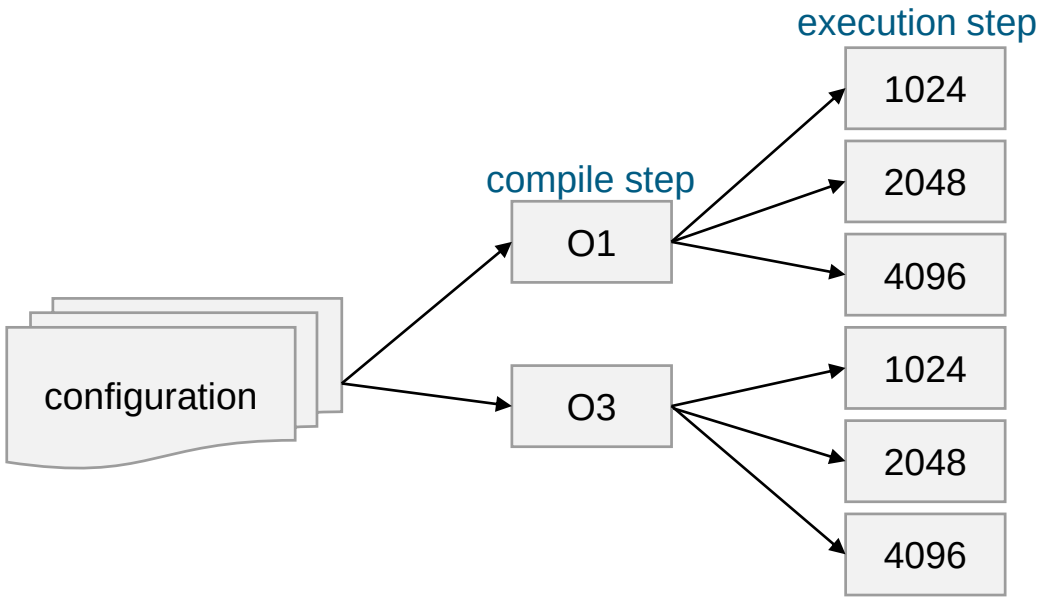
- Generic, configurable environment to run, monitor and analyse benchmarks in a systematic way (command line based)
- Use cases: procurements, monitoring the effects of system and configuration changes, workflow handling, scaling plots



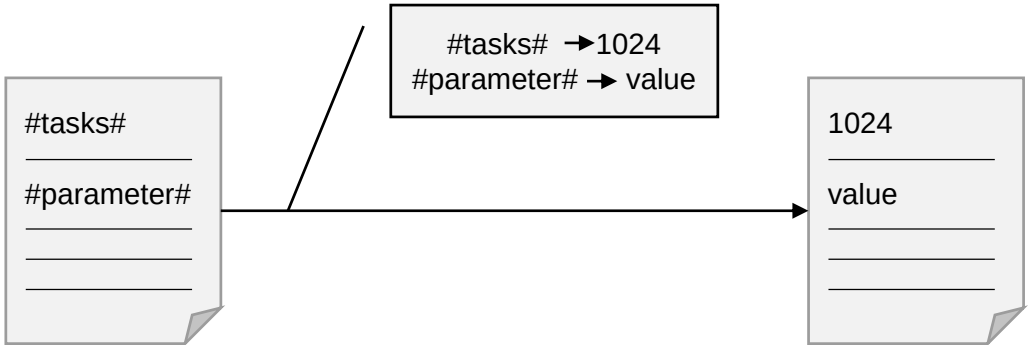
# KEY CONCEPTS

- Easy parameter space creation

```
<parameter name="opt">O1,O3</…>
<parameter name="tasks">1024,2048,4096</…>
```

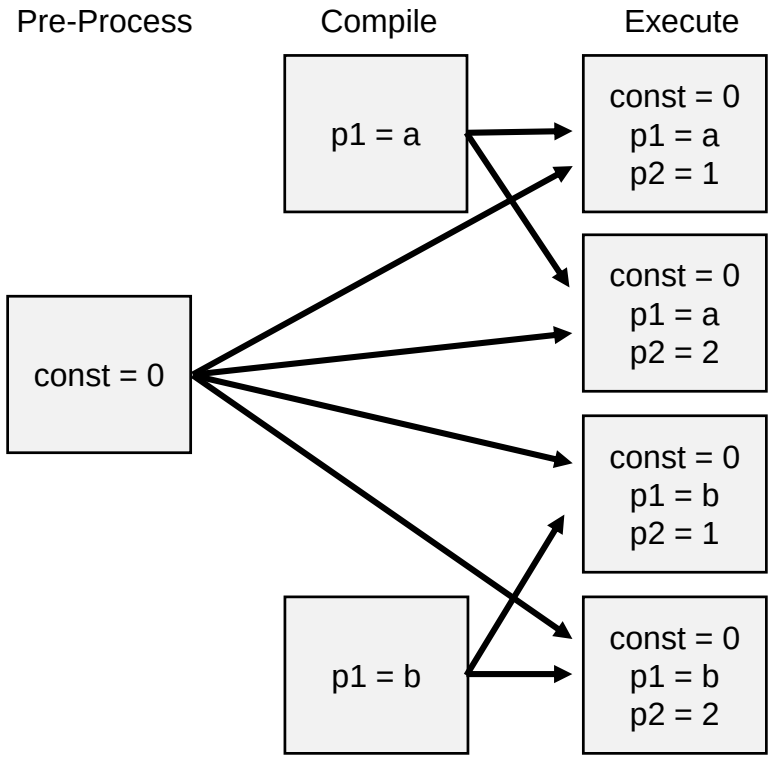



- Substitution of placeholders



# KEY CONCEPTS: WORKFLOW CREATION

- Dependency driven step structure
- Parameter based expansion of steps




Jube snippet

```
<parameterset name="preset">
  <parameter name="const">0</parameter>
</parameterset>
<parameterset name="compset">
  <parameter name="p1">a,b</parameter>
</parameterset>
<parameterset name="execset">
  <parameter name="p2">1,2</parameter>
</parameterset>

<step name="preprocess">
  <use>preset</use>
</step>
<step name="compile">
  <use>compset</use>
</step>
<step name="execute" depend="preprocess,compile">
  <use>execset</use>
</step>
```

# WORKFLOW

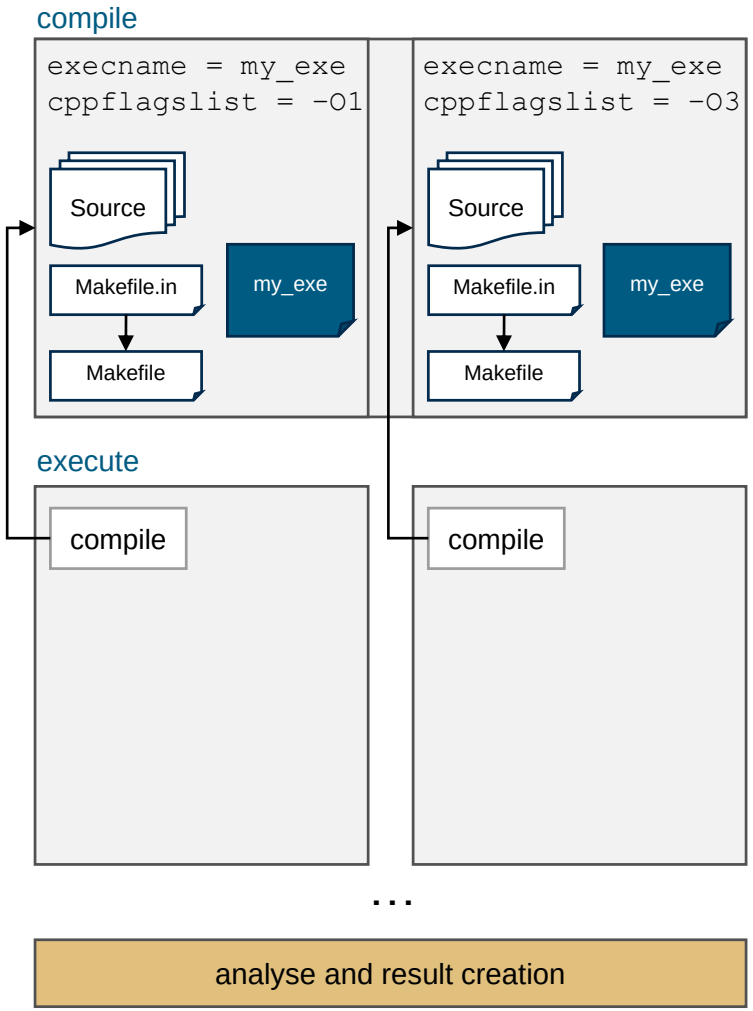
```
<jube>
  <benchmark name="bench" outpath="./benchmark_runs">
    <parameterset name="compileset">
      <parameter name="execname">my_exe</parameter>
      <parameter name="cppflagslist">
        -O1,-O3
      </parameter>
    </parameterset>

    <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">
      …
```

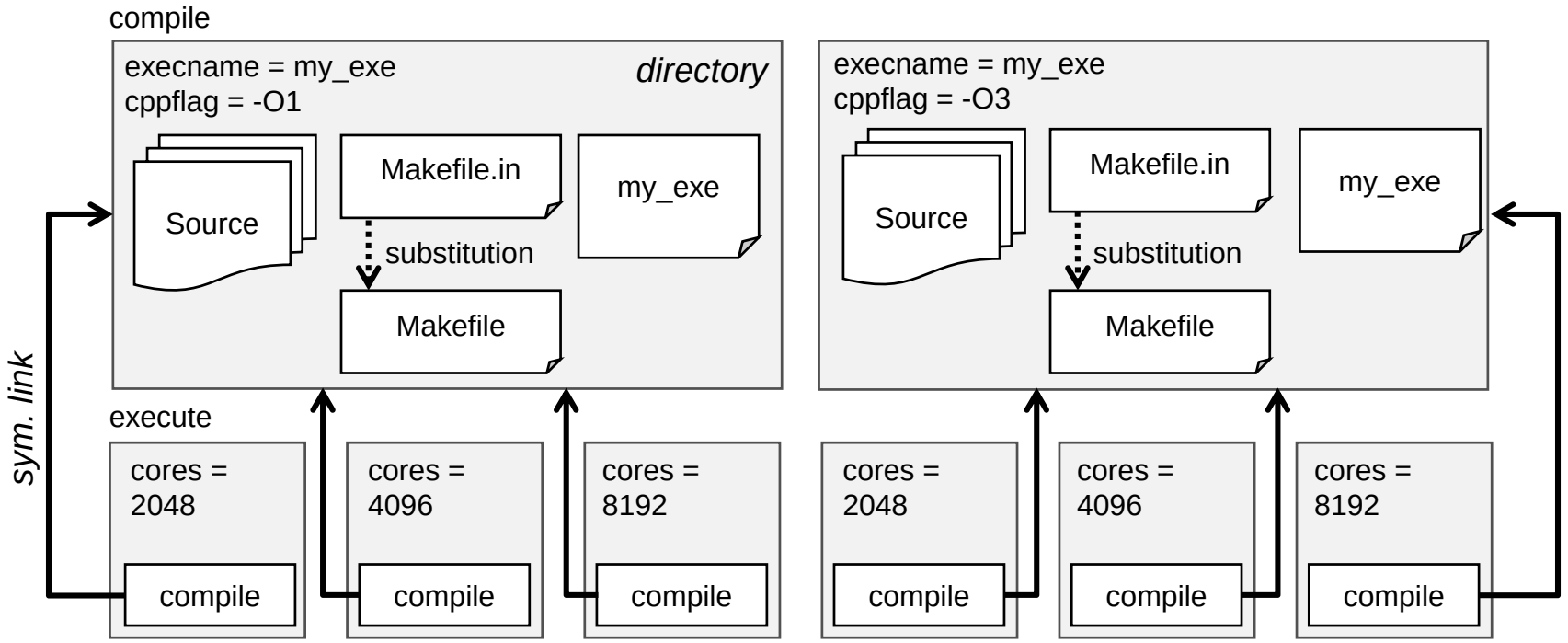





JÜLICH Forschungszentrum

# DIRECTORY AND DATA HANDLING

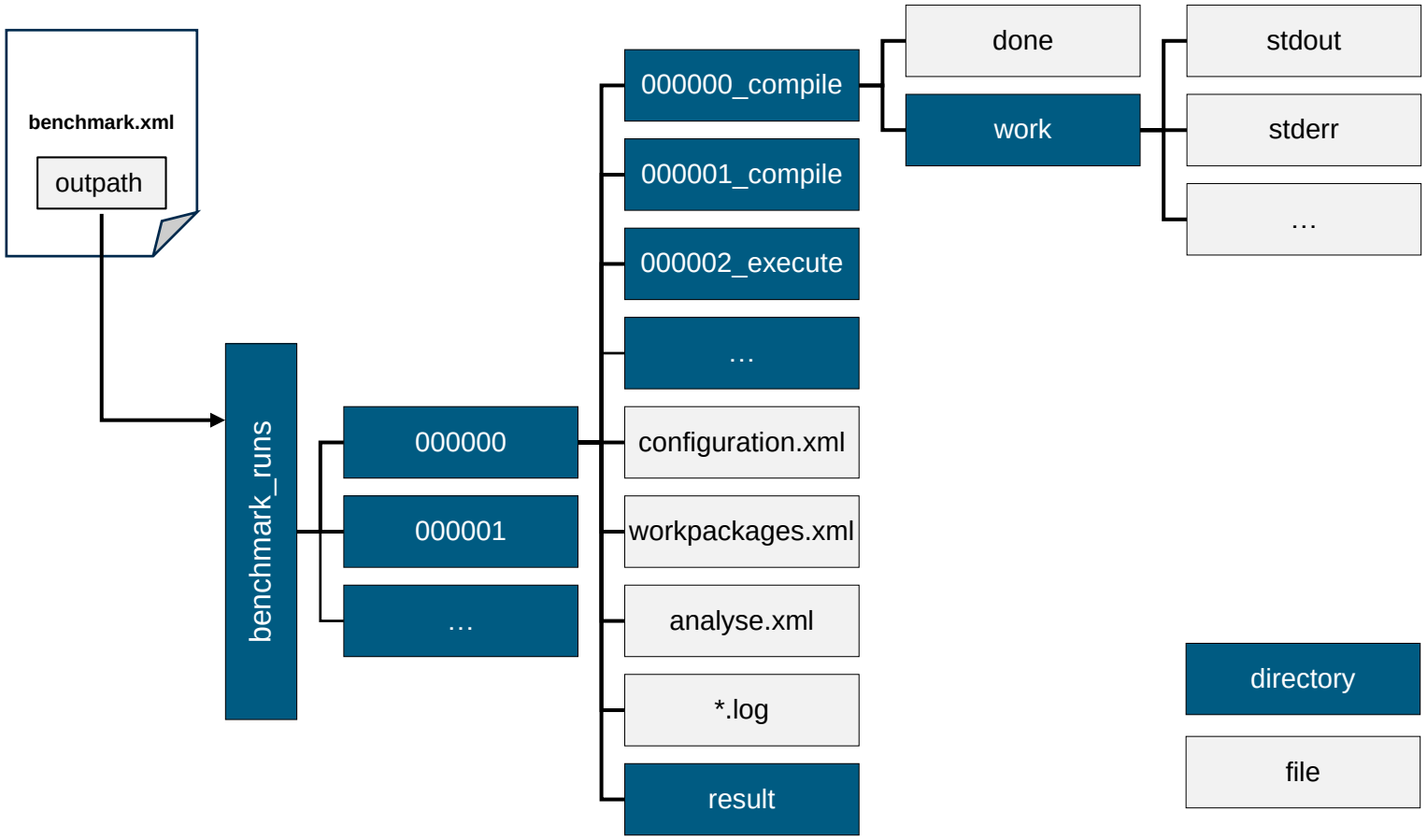
- Each parameter/step combination runs in a separate sandbox directory
- Dependent steps can be accessed using sym. links






JÜLICH Forschungszentrum

# DIRECTORY STRUCTURE



# COMMAND LINE ACCESS

1. Start a new benchmark run
   - `jube run benchmark.xml`
2. Continue an existing benchmark run
   - `jube continue benchmark_dir [--id <id>]`
3. Analyse the benchmark data
   - `jube analyse benchmark_dir [--id <id>]`
4. Create and show result representation
   - `jube result benchmark_dir [--id <id>]`

# HELP?!

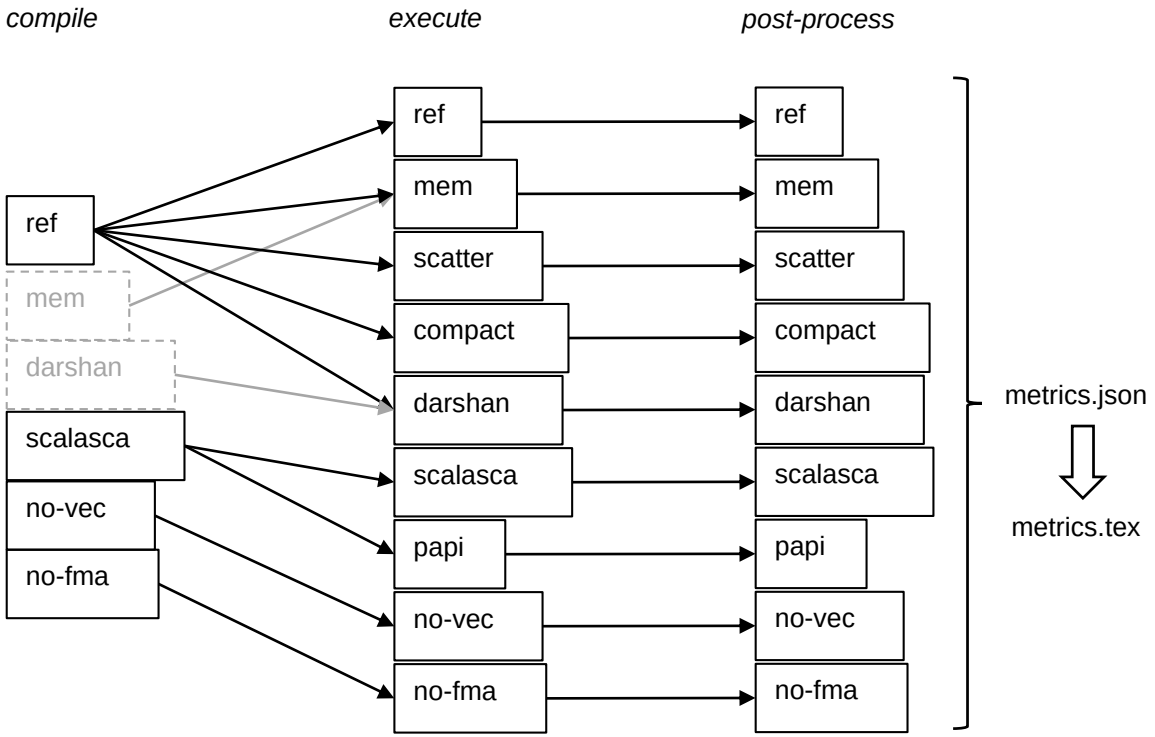
1. Online documentation and tutorial
   - `www.fz-juelich.de/jsc/jube`
2. Info mode
   - `jube info benchmark_dir [--id <id>] [--step <stepname>]`
3. Command line accessible glossary
   - `jube help <keyword>`
4. Logs
   - `jube log benchmark_dir [--id <id>]`
5. Debug mode
   - `jube --debug run|continue|analyse|result …`

# WHERE TO START?

- Website (download and tutorials):
  - **`www.fz-juelich.de/jsc/jube`**
- Versions:
  - **2.x** new release, Python based
- Prerequisites:
  - OS: Linux
  - Python 2.6*, Python 2.7, Python 3.2 (or a more recent version)
- Publication:
  - S.Luehrs et al., Flexible and Generic Workflow Management
    DOI: 10.3233/978-1-61499-621-7-431

# PERFORMANCE MEASUREMENT EXAMPLE

- Reproducible, comparable setup to extract performance metrics for different applications
- Orchestration of different performance evaluation tools and techniques



# PERFORMANCE MEASUREMENT EXAMPLE

| | Metric name | fayalite.json | fayalite_io.json |
|---|---|---|---|
| Global | Total Time (s) | 25 | 19 |
| | Time IO (s) | 0.32 | 0.13 |
| | Time MPI (s) | 23.20 | 15.03 |
| | Memory vs Compute Bound | 1.00 | 1.06 |
| | Load Imbalance (%) | 55.56 | 43.88 |
| IO | IO Volume (MB) | 197.73 | 197.73 |
| | Calls (nb) | 3113003 | 27519 |
| | Throughput (MB/s) | 616.35 | 1571.13 |
| | Individual IO Access (kB) | 0.06 | 7.70 |
| MPI | P2P Calls (nb) | 15352 | 15352 |
| | P2P Calls (s) | 0.34 | 0.33 |
| | P2P Calls Message Size (kB) | 0 | 0 |
| | Collective Calls (nb) | 8062 | 8062 |
| | Collective Calls (s) | 21.94 | 13.88 |
| | Coll. Calls Message Size (kB) | 2357 | 2357 |
| | Synchro / Wait MPI (s) | 21.79 | 13.72 |
| | Ratio Synchro / Wait MPI (%) | 92.74 | 90.32 |
| Node | Time OpenMP (s) | N.A. | N.A. |
| | Ratio OpenMP (%) | N.A. | N.A. |
| | Synchro / Wait OpenMP (s) | N.A. | N.A. |
| | Ratio Synchro / Wait OpenMP (%) | N.A. | N.A. |
| Mem | Memory Footprint | 148336kB | 122632kB |
| | Cache Usage Intensity | 0.97 | 0.97 |
| Core | IPC | 0.66 | 0.66 |
| | Runtime without vectorisation (s) | 25 | 20 |
| | Vectorisation efficiency | 1.00 | 1.05 |
| | Runtime without FMA (s) | 25 | 20 |
| | FMA efficiency | 1.00 | 1.05 |

# DEMO

# DEMO

- Execute the following commands:
  - source /home/hpc/a2c06/lu23vov/load-jube
  - cp –rv /home/hpc/a2c06/lu23vov/jube_btmz_demo $SCRATCH_LEGACY
  - cd $SCRATCH_LEGACY/jube_btmz_demo
- Start your first JUBE job:
  - jube –vvv run btmz.xml
- Wait until job is done, then:
  - jube continue bench_run
  - jube result bench_run -a