Automated Workflows and Benchmarks with JUBE in IO-SEA and Beyond

Who is who

- We: Jülich Forschungszentrum
  - Yannik Müller
    - *Extensive user and contributor of JUBE*
  - Thomas Breuer
    - *Current maintainer of JUBE*
- You: ?

Scope of this Presentation

- **This talk is for**
  - novice and advanced JUBE users
- **Prerequisites**
  - Basic Knowledge of XML or YAML
  - Basic knowledge of Python and HPC job scripts (slurm) helpful but not required
- **Benefits**
  - Thorough overview of JUBE features and best practices
  - Understanding of use-cases and limitations of JUBE
  - Benefits of JUBE for IO-SEA (and other projects)

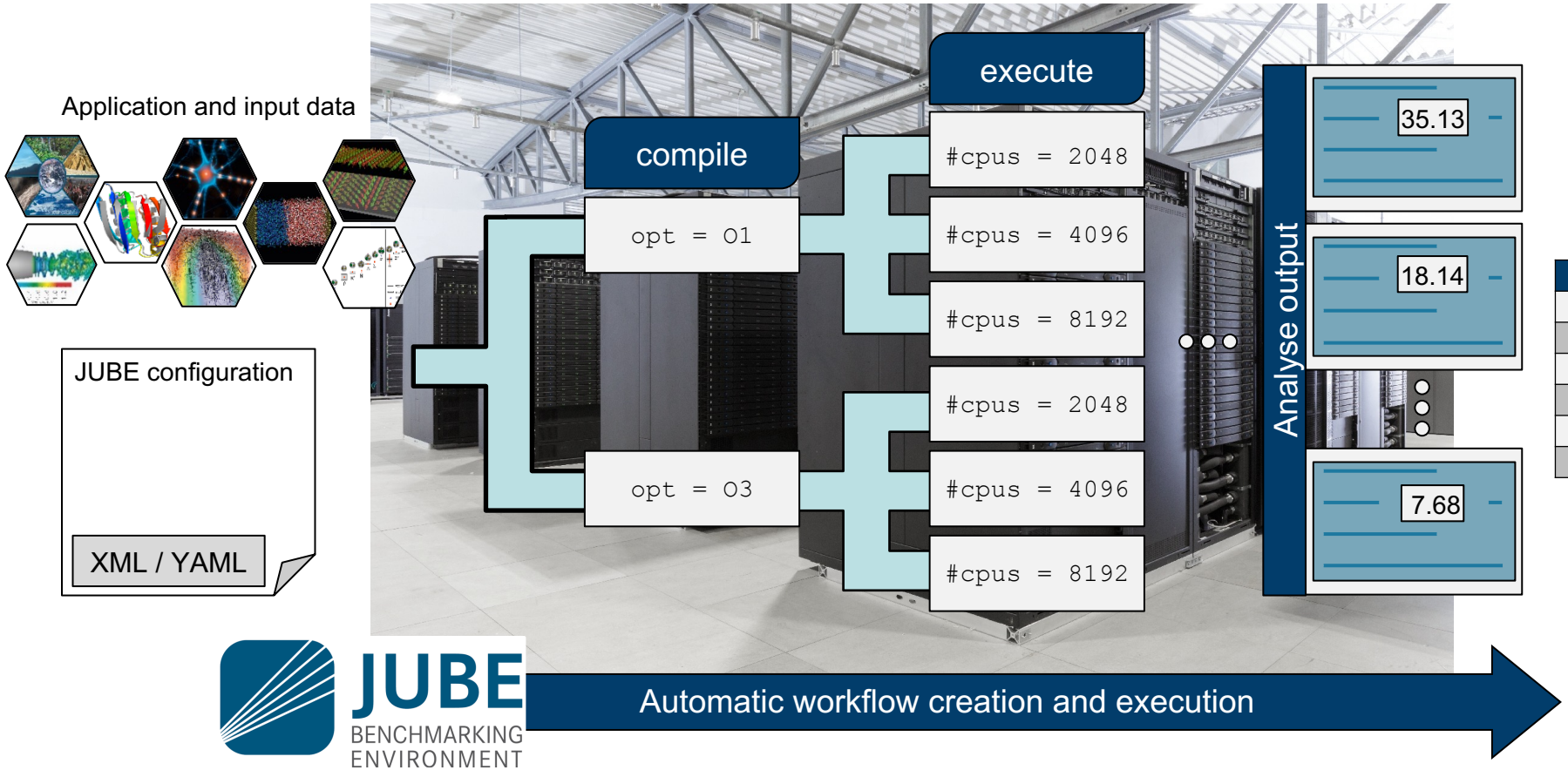
Outline

- Introduction to JUBE and its application in IO-SEA
- Basic Features
- Break
- (Some) Advanced Features
- Advanced JUBE scripts with real examples
- Questions/Discussions

Introduction to

JUBE AND ITS APPLICATION IN IO-SEA

JUBE in a nutshell



opt	#cpus	time [s]
O1	2048	35.13
O1	4096	18.14
O1	8192	9.32
O3	2048	29.00
O3	4096	15.00
O3	8192	7.68

JUBE in a nutshell

What is JUBE

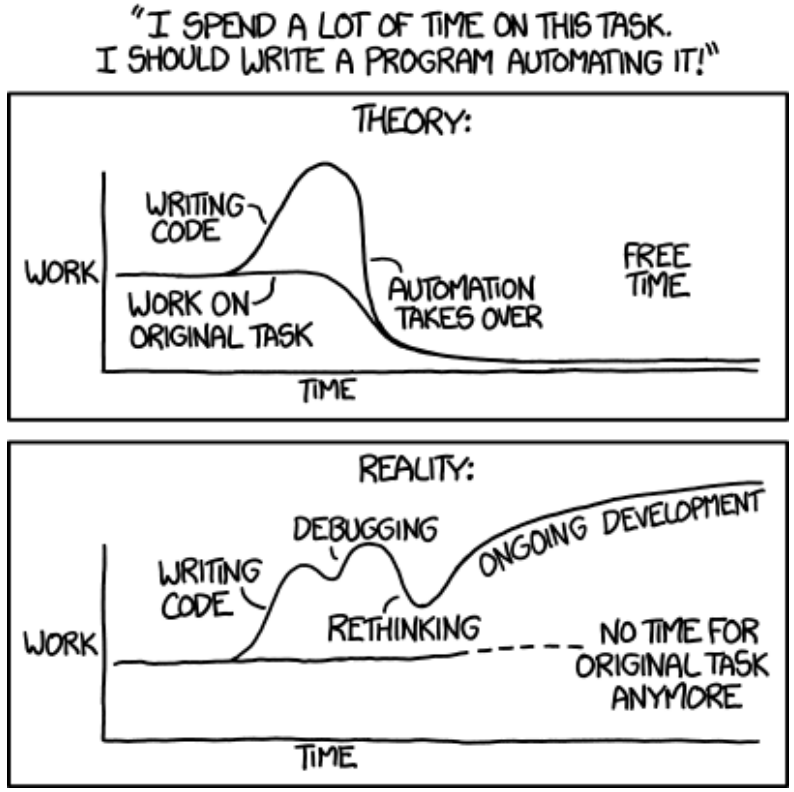
- Generic, lightweight, configurable environment to run, monitor and analyze workflow execution in a systematic way.
- Application-specific script replacement.

Can be used for:

- Benchmarking
- Parameter studies
- Testing
- Production scenarios
- …

What JUBE is not

- A profiling tool: *Metrics are extracted not generated*
- A building tool: *JUBE will follow the same process as done manually*
- Free lunch



Writing Code
=> Doing a study
Debugging
=> Confused about results
Rethinking
=> Finding out stuff has been mixed up

JUBE applications in IO-SEA

- Application developer writes JUBE script
  - For studies
  - For basic stable setup
  - For benchmarking
    - *Benchmarking Team can use JUBE script*
      No knowledge about application needed
    - *CI/CD Pipeline runs JUBE script*
      Simple, because uniform for all applications
- Benchmarking team writes partial or example scripts
  - Application developers can reuse
  - E.g. Workflow Manager interaction

```
Benchmark:
 timeout: 1h
 tags: [io-sea,jacamar,shell]
 script:
    - export BENCH_DIR=<app>_deep_$(date --iso-8601=minutes)
    - jube-autorun -o <app>.xml
    - tar -acf Results/$FILE_NAME.tar.bz2 $BENCH_DIR
    - git config user.name "${REPO_ACCESS_USER}"
    - git config user.email "sc@fz-juelich.de"
    - git add Results/$FILE_NAME.tar.bz2
    - git commit -m "Automated benchmark for commit ${CI_COMMIT_SHORT_SHA} on system
${SYSTEM}"
    - git push
"https://${REPO_ACCESS_USER}:${REPO_ACCESS_TOKEN}@${CI_SERVER_HOST}/${CI_PROJECT_PATH}.git"
"HEAD:${CI_COMMIT_BRANCH}"
```

Uniform for all applications

JUBE

BASIC FEATURES

TODAY'S EXAMPLE: IOR

IOR is a parallel IO benchmark that can be used to test the performance of parallel storage systems using various interfaces and access patterns:
https://github.com/hpc/ior

```
<?xml version="1.0" encoding="UTF-8"?>
<jube>
  <benchmark name="ior" outpath="./runs">
    <step name="execute">
      <do>ior</do>
    </step>
  </benchmark>
</jube>
```

Yes it is XML

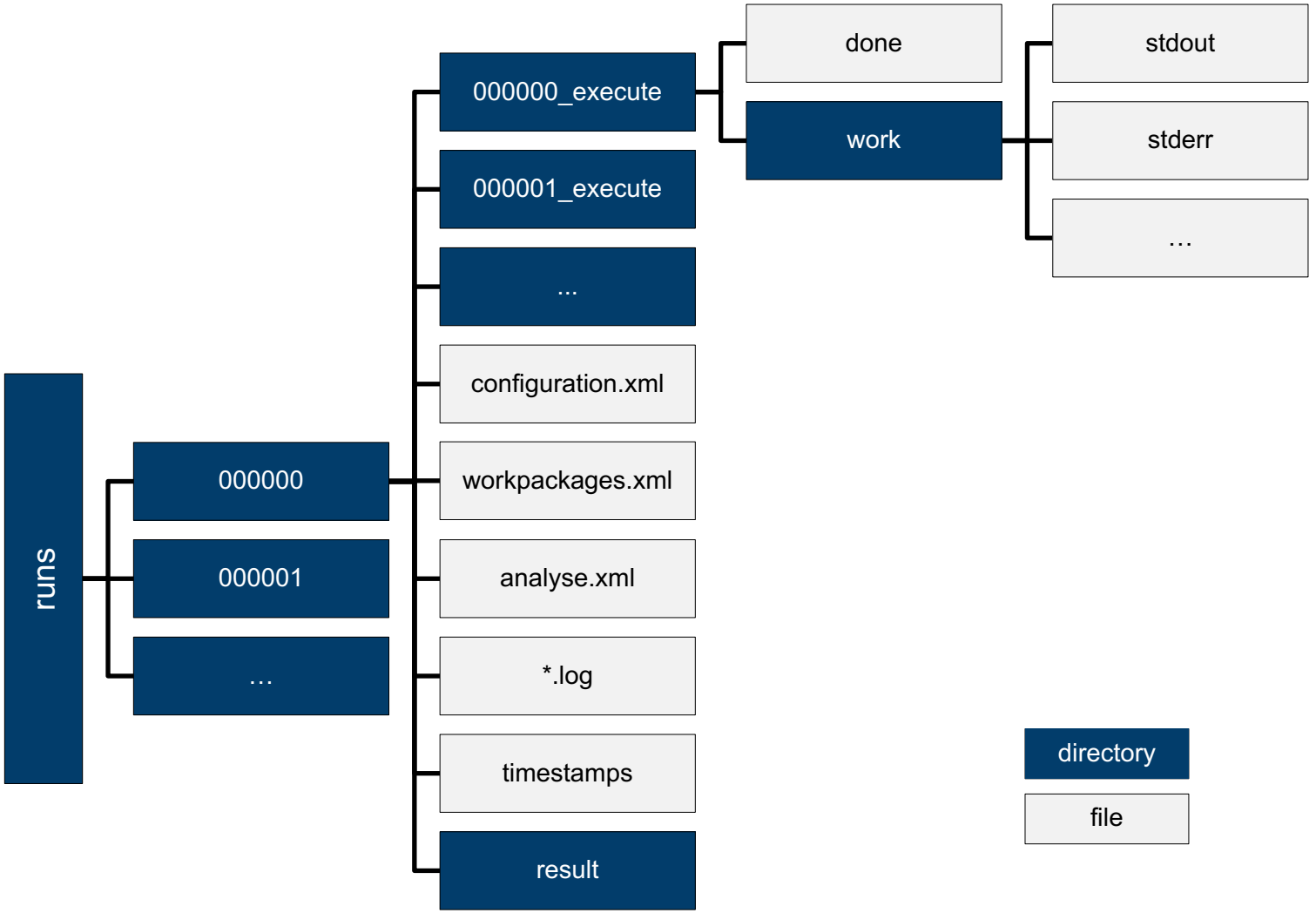
Benchmark output directory

Assuming `ior` is sitting in `$PATH`

```
> jube run ior.xml

################################################################################
# benchmark: ior
# id: 0
#
#
################################################################################

Running workpackages (#=done, 0=wait, E=error):
################################################################ (  1/  1)

  stepname | all | open | wait | error | done
  ---------+-----+------+------+-------+-----
   execute |   1 |    0 |    0 |     0 |    1

>>>> Benchmark information and further useful commands:
>>>>       id: 0
>>>>   handle: runs
>>>>      dir: runs/000000
>>>>  analyse: jube analyse runs --id 0
>>>>   result: jube result runs --id 0
>>>>     info: jube info runs --id 0
>>>>      log: jube log runs --id 0
################################################################################
```

Individual run ID

Directory structure



```xml
<?xml version="1.0" encoding="UTF-8"?>
<jube>
  <benchmark name="ior" outpath="./runs">

    <parameterset name="ior_parameter">
      <parameter name="blocksize">100m</parameter>
      <parameter name="transfersize">
            1m,10m,100m
      </parameter>
    </parameterset>

    <step name="execute">
      <use>ior_parameter</use>
      <do>ior -b $blocksize -t $transfersize</do>
    </step>
  </benchmark>
</jube>
```

Three configurations to test

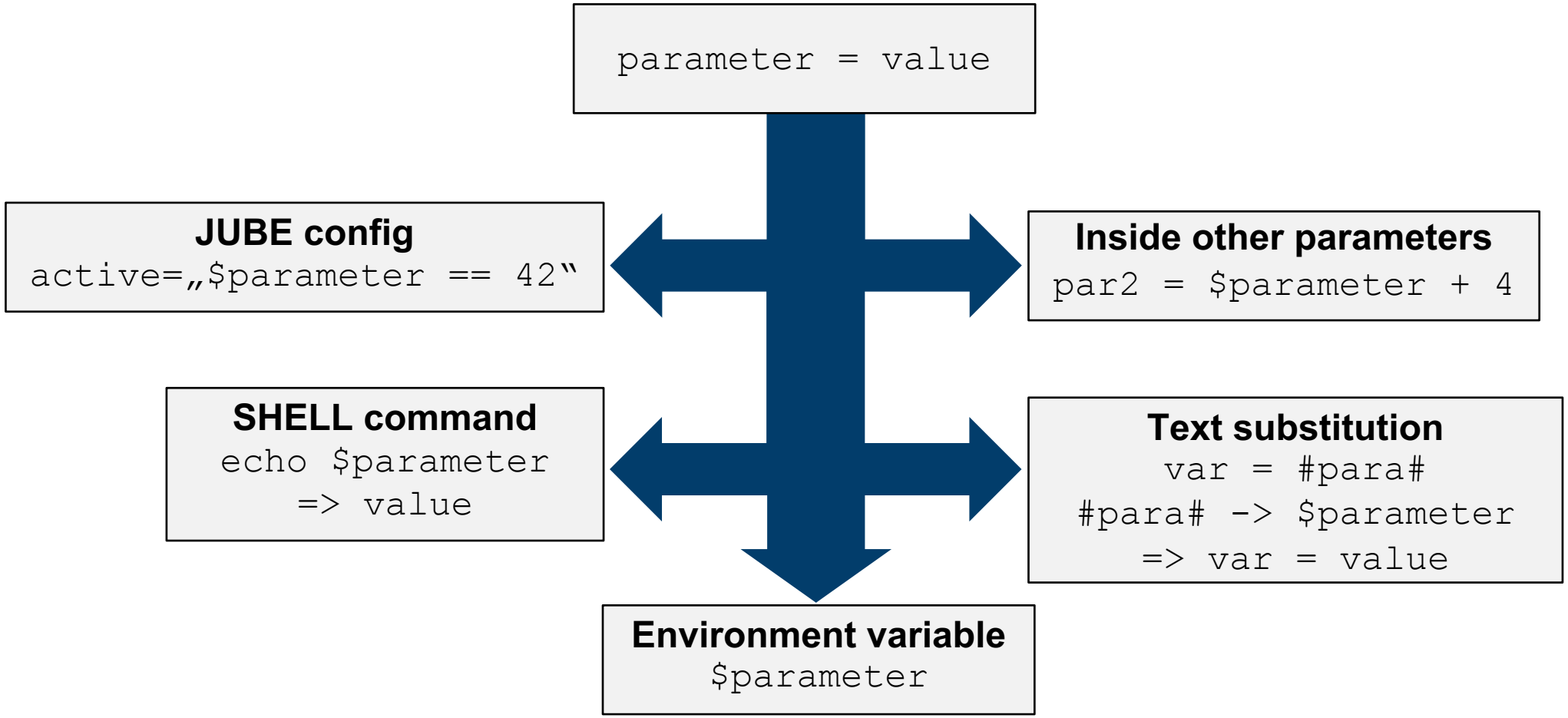
Not an environment variable

```
> jube run ior.xml

################################################################################
# benchmark: ior
# id: 1
#
#
################################################################################

Running workpackages (#=done, 0=wait, E=error):
################################################################ (  3/  3)

  stepname | all | open | wait | error | done
  ---------+-----+------+------+-------+-----
   execute |   3 |    0 |    0 |     0 |    3

>>>> Benchmark information and further useful commands:
>>>>       id: 1
>>>>   handle: runs
>>>>      dir: runs/000001
>>>>  analyse: jube analyse runs --id 1
>>>>   result: jube result runs --id 1
>>>>     info: jube info runs --id 1
>>>>      log: jube log runs --id 1
################################################################################
```

Three individual runs

Parameter Substitution

A configured parameter can be used in different places:



```
<?xml version="1.0" encoding="UTF-8"?>
<jube>
  <benchmark name="ior" outpath="./runs">
    <parameterset name="ior_parameter">
      <parameter name="blocksize">100m</parameter>
      <parameter name="transfersize">
        1m,10m,100m
      </parameter>
    </parameterset>

    <fileset name="source">
      <copy>ior</copy>
    </fileset>

    <step name="compile">
      <use>source</use>
      <do work_dir="ior">
        ./bootstrap &amp;&amp; ./configure
      </do>
      <do work_dir="ior">make</do>
    </step>

    <step name="execute" depend="compile">
      <use>ior_parameter</use>
      <do>compile/ior/src/ior -b $blocksize -t $transfersize</do>
    </step>
  </benchmark>
</jube>
```

Folder/files needed for compilation

In YAML it looks nicer

Dependent step reference

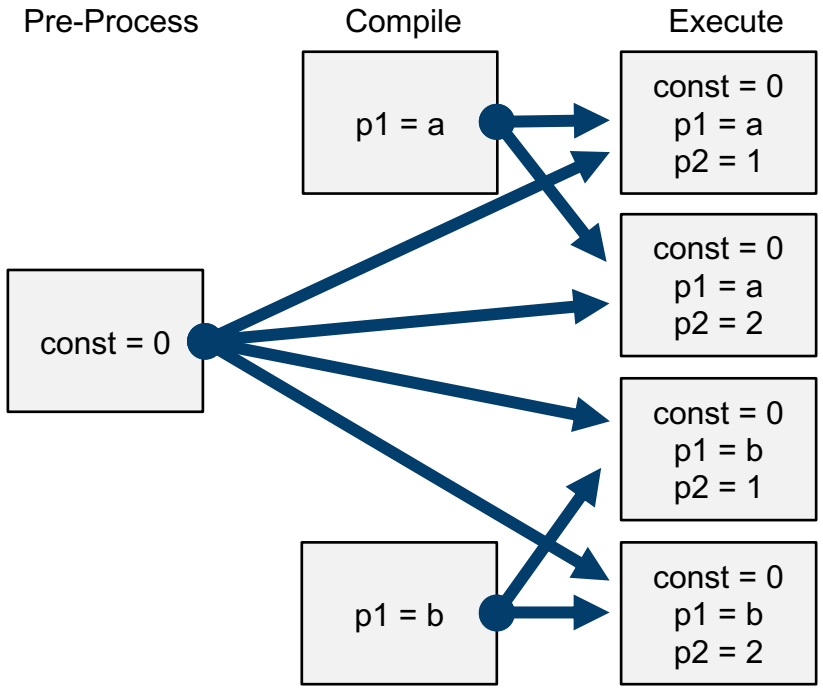
Link to access depend step

```
> jube run ior.xml

################################################################
# benchmark: ior
# id: 2
#
################################################################

Running workpackages (#=done, 0=wait, E=error):
############################################################ (  4/  4)

  stepname | all | open | wait | error | done
  ---------+-----+------+------+-------+-----
   compile |   1 |    0 |    0 |     0 |    1
   execute |   3 |    0 |    0 |     0 |    3

>>>> Benchmark information and further useful commands:
>>>>       id: 2
>>>>   handle: runs
>>>>      dir: runs/000002
>>>>  analyse: jube analyse runs --id 2
>>>>   result: jube result runs --id 2
>>>>     info: jube info runs --id 2
>>>>      log: jube log runs --id 2
################################################################
```

Two steps

Workflow creation

- Dependency-driven step structure
- Parameter-based expansion of steps




```
<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,combile">
  <use>execset</use>
</step>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<jube>
  <benchmark name="ior" outpath="./runs">
    <parameterset name="ior_parameter">
      <parameter name="blocksize">100m</parameter>
      <parameter name="transfersize">
        1m,10m,100m
      </parameter>
    </parameterset>
    <fileset name="source">
      <copy>ior</copy>
    </fileset>
    <step name="compile">
      <use>source</use>
      <do work_dir="ior">
        ./bootstrap &amp;&amp; ./configure
      </do>
      <do work_dir="ior">make</do>
    </step>
    <step name="execute" depend="compile">
      <use>ior_parameter</use>
      <do>compile/ior/src/ior -b $blocksize -t $transfersize</do>
    </step>

    <patternset name="ior_pattern">
      <pattern name="write_bandwidth">
        write\s+$jube_pat_fp
      </pattern>
    </patternset>
    <analyser name="ior_analyser">
      <use>ior_pattern</use>
      <analyse step="execute">
        <file>stdout</file>
      </analyse>
    </analyser>
    <result>
      <use>ior_analyser</use>
      <table name="result_table" style="pretty">
        <column>transfersize</column>
        <column>write_bandwidth</column>
      </table>
    </result>

  </benchmark>
</jube>
```

Regex magic

File to be scanned

Patterns and parameter can be mixed in the result

```
> jube run ior.xml -r

################################################################################
# benchmark: ior
# id: 3
#
#
################################################################################

Running workpackages (#=done, 0=wait, E=error):
############################################################ (  4/  4)

  stepname | all | open | wait | error | done
  ---------+-----+------+------+-------+-----
   compile |   1 |    0 |    0 |     0 |    1
   execute |   3 |    0 |    0 |     0 |    3

>>>> Benchmark information and further useful commands:
>>>>       id: 3
>>>>   handle: runs
>>>>      dir: runs/000003
>>>>  analyse: jube analyse runs --id 3
>>>>   result: jube result runs --id 3
>>>>     info: jube info runs --id 3
>>>>      log: jube log runs --id 3
################################################################################
>>> Start analyse
>>> Analyse finished
result_table:
transfersize | write_bandwidth
-------------+----------------
          1m |         1828.65
         10m |         2058.20
        100m |         2489.41
```

Run implicit result generation

Finally some output

PATTERN Extraction

- A set of regular expressions can be configured to extract relevant output data
- https://regex101.com (Python mode) good place to learn

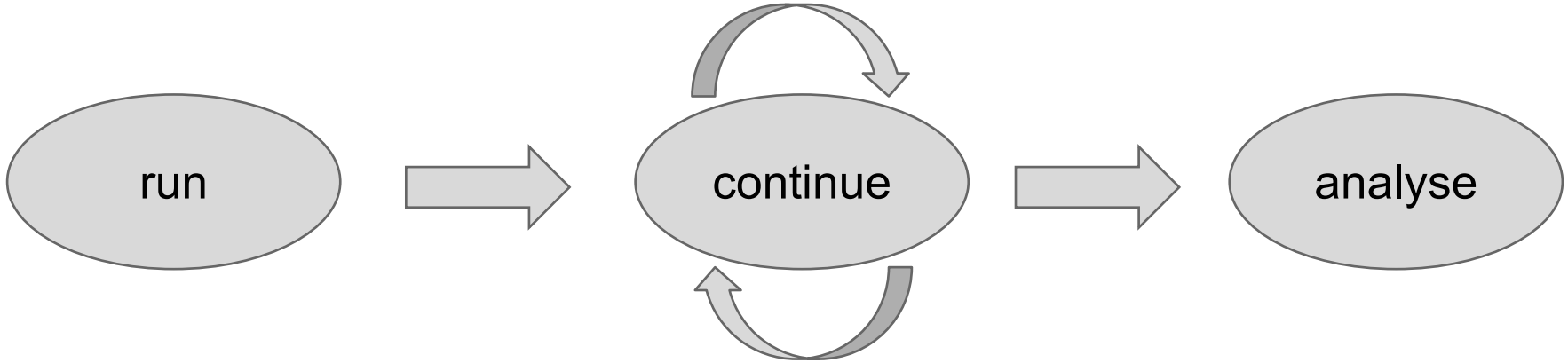
read_pattern = `read\s+(\d*?\.\d*)\s+`

```
access    bw(MiB/s)  block(KiB) xfer(KiB)  open(s)    wr/rd(s)   close(s)
------    ---------  ---------- ---------  --------   --------   --------
Commencing read performance test: Wed Nov 28 16:52:48 2018
read      89.52      1191936    2.27       0.004406   1248.26    0.001339
remove    -          -          -          -          -          -
```

Parameter	read_pattern
value	89.52

Job submission

- The JUBE kernel has no direct knowledge about HPC scheduling systems
- A job template and the substitution system can be used to generalize the job submission process
- JUBE supports asynchronous command execution
- A marker file is used to mark the end of the job; it must be generated by the job script after the parallel part was executed
- *jube continue* triggers JUBE to check all available marker files



```
<?xml version="1.0" encoding="UTF-8"?>
<jube>
  <benchmark name="ior" outpath="./runs">
    <parameterset name="ior_parameter">
      <parameter name="blocksize">100m</parameter>
      <parameter name="transfersize">
        1m,10m,100m
      </parameter>
    </parameterset>
    <fileset name="source">
      <copy>ior</copy>
    </fileset>
    <step name="compile">
      <use>source</use>
      <do work_dir="ior">
        ./bootstrap &amp;&amp; ./configure
      </do>
      <do work_dir="ior">make</do>
    </step>

    <parameterset name="systemParameter" init_with="platform.xml">
      <parameter name="nodes" type="int">1,2,4</parameter>
      <parameter name="taskspernode" type="int">8</parameter>
      <parameter name="executable">compile/ior/src/ior</parameter>
      <parameter name="args_exec">-b $blocksize -t $transfersize</parameter>
    </parameterset>

    <step name="execute" depend="compile">
      <use>ior_parameter</use>
      <use>systemParameter</use>
      <use from="platform.xml">executeset,executesub,jobfiles</use>
      <do done_file="$done_file">$submit $submit_script</do>
    </step>

    <patternset name="ior_pattern">
      <pattern name="write_bandwidth">
        write\s+$jube_pat_fp
      </pattern>
    </patternset>
    <analyser name="ior_analyser">
      <use>ior_pattern</use>
      <analyse step="execute">
        <file>job.out</file>
      </analyse>
    </analyser>
    <result>
      <use>ior_analyser</use>
      <table name="result_table" style="pretty">
        <column>nodes</column>
        <column>transfersize</column>
        <column format=".1f">write_bandwidth</column>
      </table>
    </result>
  </benchmark>
</jube>
```

Use default platform configuration

Overwrite defaults where necessary

Wait until done_file is created

Use predefined job templates and parameter substitutions

Scan job output instead of stdout

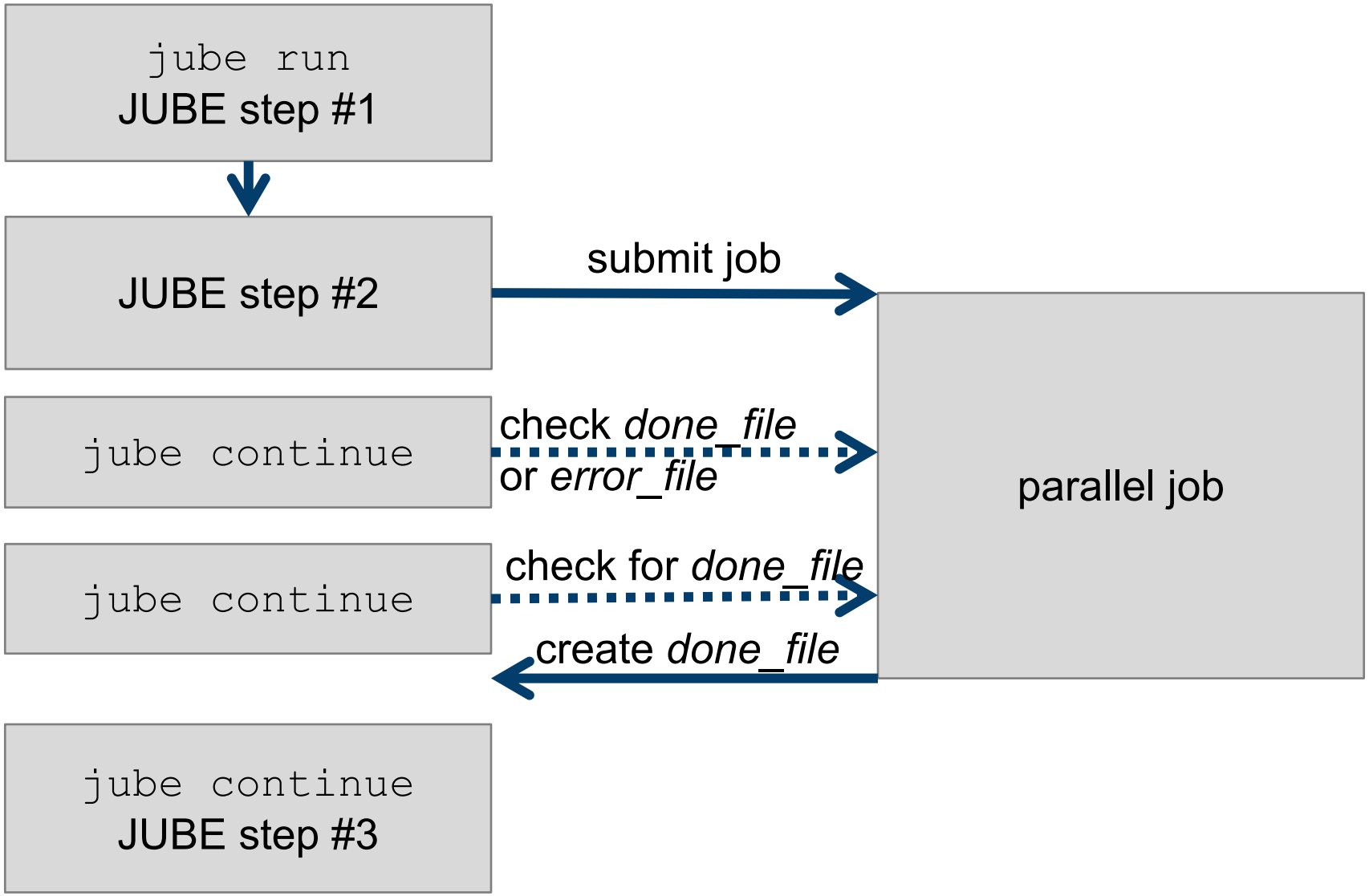
```
> jube run ior.xml

################################################################################
# benchmark: ior
# id: 4
#
#
################################################################################

Running workpackages (#=done, 0=wait, E=error):
######0000000000000000000000000000000000000000000000000000 (  1/ 10)

  stepname | all | open | wait | error | done
  ---------+-----+------+------+-------+-----
   compile |   1 |    0 |    0 |     0 |    1
   execute |   9 |    0 |    9 |     0 |    0

>>>> Benchmark information and further useful commands:
>>>>       id: 4
>>>>   handle: runs
>>>>      dir: runs/000004
>>>> continue: jube continue runs --id 4
>>>>  analyse: jube analyse runs --id 4
>>>>   result: jube result runs --id 4
>>>>     info: jube info runs --id 4
>>>>      log: jube log runs --id 4
################################################################################>

> jube continue runs -r
...
result_table:
nodes | transfersize | write_bandwidth
------+--------------+----------------
    1 |           1m |          5780.0
    1 |          10m |          3193.3
    1 |         100m |          3064.4
    2 |           1m |          5866.0
    2 |          10m |          6462.0
    2 |         100m |          6227.0
    4 |           1m |          6946.0
    4 |          10m |         10535.0
    4 |         100m |          8609.0
```

Jobs were queued

Nine jobs in total

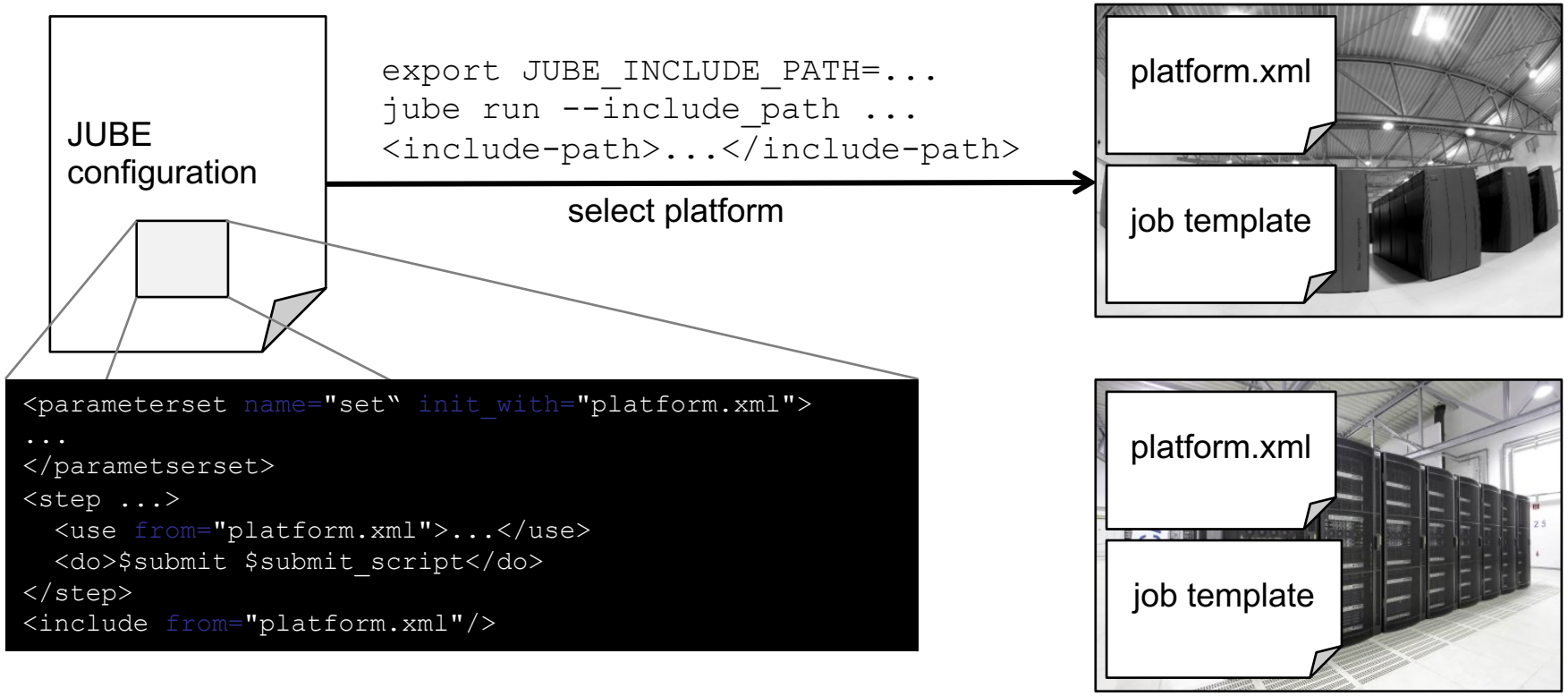
Check job state and create result

Job submission



CONFIGURATION REUSAGE

- E.g. separation of platform-dependent and -independent configuration options




```
<parameterset name="set" init_with="platform.xml">
...
</parametserset>
<step ...>
  <use from="platform.xml">...</use>
  <do>$submit $submit_script</do>
</step>
<include from="platform.xml"/>
```

ONE more THING

Not a friend of XML?
Use **YAML**!

```yaml
name: ior
outpath: ./runs
parameterset:
  - name: ior_parameter
    parameter: [{name: blocksize, _: 100m}, {name: transfersize, _: "1m,10m,100m"}]
  - name: systemParameter
    init_with: platform.xml
    parameter:
    - {name: nodes, type: int, _: "1,2,4"}
    - {name: taskspernode, type: int, _: 8}
    - {name: executable, _: compile/ior/src/ior},{name: "args_exec", _: -b $blocksize -t $transfersize}
fileset: {name: source, copy: ior }
step:
  - name: compile
    do: {work_dir: ior, _: [./bootstrap && ./configure,make]}
  - name: execute
    use: [ior_parameter,systemParameter,{from: platform.xml, _: [executeset,executesub,jobfiles]}]
    do: {done_file: $done_file, _: $submit $submit_script}
patternset:
  name: ior_pattern
  pattern: {name: write_bandwidth, _: write\s+$jube_pat_fp}
analyser:
  name: ior_analyser
  use: ior_pattern
  analyse:
    step: execute
    file: job.out
result:
  use: ior_analyser
  table:
    name: result_table
    style: pretty
    column: [nodes,transfersize,{format: .1f, _: write_bandwidth}]
```

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>]
```

- **All in one**

```
jube-autorun benchmark.xml
```

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 …
  ```

JUBE availability

- Download, Tutorials and Documentation:
  www.fz-juelich.de/jsc/jube
- Prerequisites:
  - *OS: Linux*
  - *Python 3.2+ (+pyyaml if you use YAML)*
- Contact:
  jube.jsc@fz-juelich.de

JUBE availability on JSC Systems

- `module load JUBE`
- `(module load PyYAML)`

- Platform setup and job template are loaded automatically and can be used in personal JUBE configurations:
  - `cat $JUBE_INCLUDE_PATH/platform.xml`
  - `cat $JUBE_INCLUDE_PATH/submit.job.in`

ADVANCED JUBE

Advanced JUBE

- JUBE tutorial
  - Installation
  - Configuration
  - Input format
  - Hello World
  - Parameter space creation
  - Step dependencies
  - Loading files and substitution
  - Creating a result table
  - Help

- Advanced tutorial
  - Schema validation
  - Scripting parameter
  - Scripting pattern
  - Statistic pattern values
  - Jobsystem
  - Include external data
  - Tagging
  - Platform-independent benchmarking
  - Multiple benchmarks
  - Shared operations
  - Environment handling
  - Parameter dependencies
  - Parameter update
  - Step iteration
  - Step cycle
  - Parallel workpackages
  - Result database
  - Creating a do log
  - The duplicate option

Advanced JUBE

- JUBE tutorial
  - - Installation
  - - Configuration
  - + Input format
  - ++ Hello World
  - + Parameter space creation
  - + Step dependencies
  - (+) Loading files and substitution
  - + Creating a result table
  - + Help

- Advanced tutorial
  - + Schema validation
  - ++ Scripting parameter
  - - Scripting pattern
  - - Statistic pattern values
  - ++ Jobsystem
  - - Include external data
  - - Tagging
  - ++ Platform-independent benchmarking
  - - Multiple benchmarks
  - - Shared operations
  - - Environment handling
  - - Parameter dependencies
  - - Parameter update
  - - Step iteration
  - - Step cycle
  - - Parallel workpackages
  - - Result database
  - - Creating a do log
  - - The duplicate option

Advanced JUBE

- Not documented
  - Detailed XML <-> YAML Guide
  - Active clause

Schema Validation

- Schema validation for XML and YAML
  - Name your files .jube.xml / .jube.yaml
    - *Editor can apply language rules and highlighting to .xml /.yaml*
    - *Editor can apply schema checking to jube scripts only*
  - Easy to setup e.g. VScode or emacs (ask us if you need help)
    - *auto completion*
    - *static error checking*
  - Find the schemas in contrib/schema

Advanced JUBE

SCRIPTING PARAMETER

```
<parameter name="System_Name" mode="shell">cat /etc/FZJ/systemname | tr -d '\n'</parameter>
<parameter name="CuArch" mode="python">
    {
        "juwels":   "sm_70",
        "jurecadc": "sm_80",
        "deep":     "sm_70",
    }[ "${System_Name}" ]
</parameter>
<parameter name="DefineCuArch" mode="python">
    {
        "CUDA": "CUARCH=${CuArch}",
        "":     ""
    }[ "${CUDA}" ]
</parameter>
<parameter name="Make">make -j24 ${Enable_Layer} ${DefineCuArch}</parameter>
<parameter name="sizeMessages" type="int" mode="python">
  ",".join([str(2**x) for x in range(10,25)])
</parameter>
```

Shell gives easy acces to OS

Python is readable esp. dictionaries

Use intermediate variables. KISS!

Expands to
`1024,2048,4096,…,33554432`

Advanced JUBE

USING PLATFORM TEMPLATES

submit.job.in

```
#!/bin/bash -x
#SBATCH --job-name=#BENCHNAME#
#SBATCH --mail-user=#NOTIFY_EMAIL#
#SBATCH --mail-type=#NOTIFICATION_TYPE#
#SBATCH --nodes=#NODES#
#SBATCH --ntasks=#TASKS#
#SBATCH --cpus-per-task=#NTHREADS#
#SBATCH --time=#TIME_LIMIT#
#SBATCH --output=#STDOUTLOGFILE#
#SBATCH --error=#STDERRLOGFILE#
#SBATCH --partition=#QUEUE#
#SBATCH --gres=#GRES#
#ACCOUNT_CONFIG#
#ADDITIONAL_JOB_CONFIG#

#ENV#

#PREPROCESS#
```

```
JUBE_ERR_CODE=$?
if [ $JUBE_ERR_CODE -ne 0 ]; then
    #FLAG_ERROR#
    exit $JUBE_ERR_CODE
fi
#MEASUREMENT# #STARTER# #ARGS_STARTER#
#EXECUTABLE# #ARGS_EXECUTABLE#
JUBE_ERR_CODE=$?
if [ $JUBE_ERR_CODE -ne 0 ]; then
    #FLAG_ERROR#
    exit $JUBE_ERR_CODE
fi
#POSTPROCESS#
JUBE_ERR_CODE=$?
if [ $JUBE_ERR_CODE -ne 0 ]; then
    #FLAG_ERROR#
    exit $JUBE_ERR_CODE
fi
#FLAG#
```

platform.xml

```
<parameterset name="executeset">
  <!-- Jobscript handling -->
  <!-- Chainjob handling -->
</parameterset>
<parameterset name="systemParameter">
  <!-- Default jobscript parameter nodes, tasks, queue... -->
  <parameter name="submit_script">submit.job</parameter>
</parameterset>
<substituteset name="executesub">
  <!-- Default jobscript substitution -->
  <iofile in="${submit_script}.in" out="$submit_script" />
  <sub source="#BENCHNAME#“ dest="${jube_benchmark_name}_${jube_step_name}_${jube_wp_id}“ />
  <sub source="#NODES#" dest="$nodes" />
  <!-- ... -->
</substituteset>
<fileset name="jobfiles">
  <!-- Default jobscript access -->
  <copy>${submit_script}.in</copy>
</fileset>
```

Using the templates

```
<?xml version="1.0" encoding="UTF-8"?>
<jube>
  <benchmark name="Get Started" outpath="runs">
    <parameterset name="systemParameter" init_with="platform.xml">
      <parameter name="account">iosea</parameter>
      <parameter name="queue">dp-cn</parameter>
      <parameter name="executable">hostname</parameter>
    </parameterset>
    <step name="execute">
      <use>systemParameter</use>
      <use from="platform.xml">executeset,executesub,jobfiles</use>
      <do done_file="$done_file" error_file="$error_file">
        $submit $submit_script
       </do>
    </step>
  </benchmark>
</jube>
```

Advanced JUBE

XML ⇔ YAML

XML <=> YAML

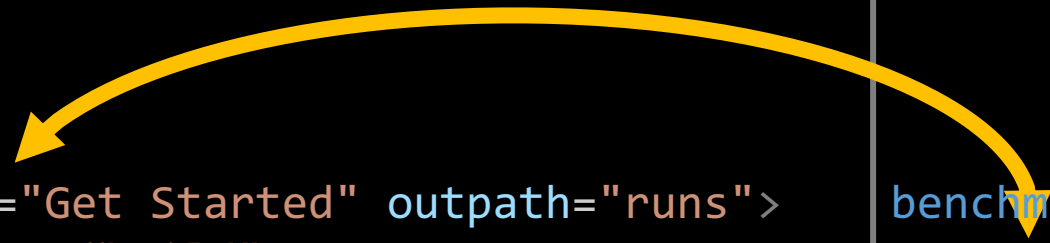
Unique Tag Mapping (Key = Tag Name)

```xml
<benchmark name="Get Started" outpath="runs">
        <step name="build">
                …
    </step>
        <step name="execute">
                …
    </step>
</benchmark>
```

```yaml
benchmark:
  name: Get Started
  outpath: runs
  step:
  - name: build
    …
  - name: execute
    …
```

XML <=> YAML

Attributes Key/Value Pair

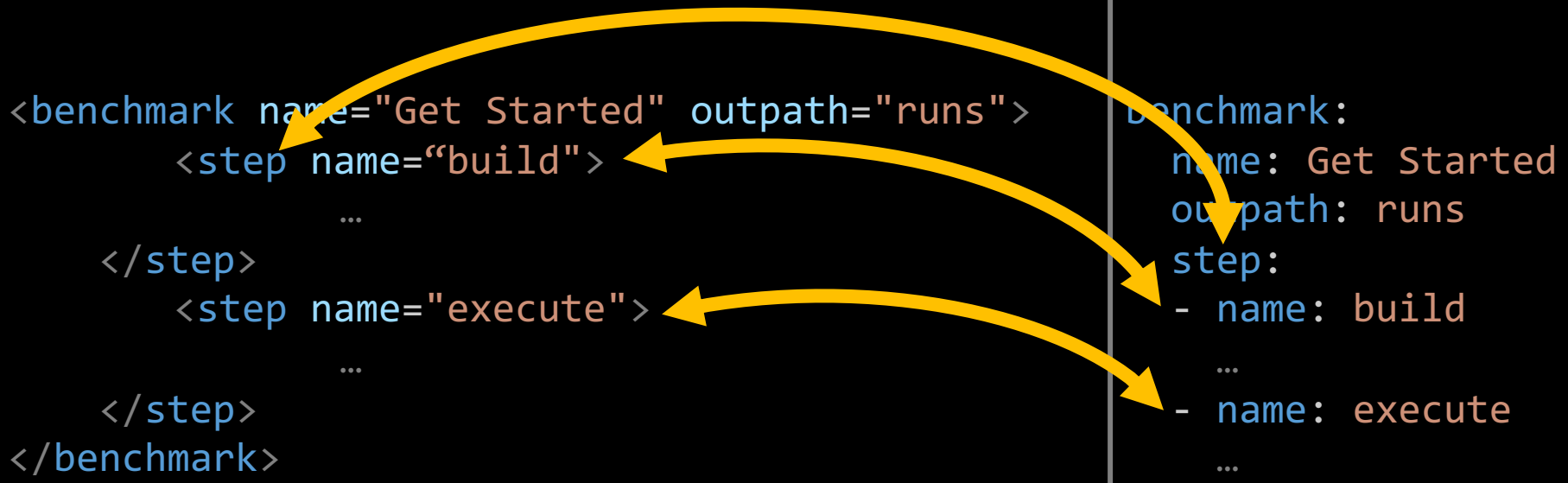


```xml
<benchmark name="Get Started" outpath="runs">
        <step name="build">
                …
    </step>
        <step name="execute">
                …
    </step>
</benchmark>
```

```yaml
benchmark:
  name: Get Started
  outpath: runs
  step:
  - name: build
    …
  - name: execute
    …
```

XML <=> YAML

Non-Unique Tag Sequence of Mappings (Key = Tag Name)



```xml
<benchmark name="Get Started" outpath="runs">
    <step name="build">
        …
    </step>
    <step name="execute">
        …
    </step>
</benchmark>
```

```yaml
benchmark:
  name: Get Started
  outpath: runs
  step:
  - name: build
    …
  - name: execute
    …
```

XML <=> YAML

```xml
<do done_file="$done_file“>
  $submit $submit_script
</do>
```

```yaml
do:
  done_file: $done_file
  _: $submit $submit_script
```

Text Content Key/Value Pair with Key = „_“

XML <=> YAML

```xml
<do>
  $submit $submit_script
</do>
```

```yaml
do: $submit $submit_script
```

May be simplified to scalar if „_“ is the only Key

JUBE XML vs JUBE YAML

XML

- Text Fields 😐
  - always plain (CDATA)
- Comments <!-- --> 😐
- Debugging ☺
  - Native xml errors or jube errors

YAML

- Scalars ☺
  - allow the **script writer** to **chose** between plain and interpreted
  - very handy for special symbols and whitespace
- Comments # ☺
- Debugging ☹
  - YAML converted to XML
    - *may throw XML errors*
    - *may cite lines that you didn't write*

XML <=> YAML

```xml
<parameterset name="systemParameter">
  <parameter name="bad_symbols">
    &lt; &gt; &amp; &apos; &quot
  </parameter>
</parameterset>
```

```yaml
parameterset:
- name: systemParameter
  parameter:
  - name: bad_symbols
    _: ' < > & '' " '
```

Encodes a single quote

XML <=> YAML

```xml
<parameterset name="systemParameter">
  <parameter name="bad_symbols">
    &lt; &gt; &amp; &apos; &quot
  </parameter>
</parameterset>
```

```yaml
parameterset:
- name: systemParameter
  parameter:
  - name: bad_symbols
    _: |
      < > & ' "
```

XML <=> YAML

```xml
<parameterset name="systemParameter">
  <!-- An XML comment -->
</parameterset>
```

```yaml
parameterset:
- name: systemParameter
  # A YAML comment
```

XML <=> YAML

```xml
<parameterset name="systemParameter">
  <parameter name="postprocess">
    source linktest-venv/bin/activate
    linktest-report -i linktest-results.sion
  </parameter>
</parameterset>
```

```yaml
parameterset:
- name: systemParameter
  parameter:
  - name: postprocess
    _: |
      source linktest-venv/bin/activate
      linktest-report -i linktest-results.sion
```

```
#!/bin/bash
#SBATCH …

    source linktest-venv/bin/activate
    linktest-report -i linktest-results.sion
```

```
#!/bin/bash
#SBATCH …

source linktest-venv/bin/activate
linktest-report -i linktest-results.sion
```

Using the templates

```
<?xml version="1.0" encoding="UTF-8"?>
<jube>
  <benchmark name="Get Started" outpath="runs">
    <parameterset name="systemParameter" init_with="platform.xml">
      <parameter name="account">iosea</parameter>
      <parameter name="queue">dp-cn</parameter>
      <parameter name="executable">hostname</parameter>
    </parameterset>
    <step name="execute">
      <use>systemParameter</use>
      <use from="platform.xml">executeset,executesub,jobfiles</use>
      <do done_file="$done_file" error_file="$error_file">
        $submit $submit_script
       </do>
    </step>
  </benchmark>
</jube>
```

Using the templates

```
name: Get Started with Templates
outpath: runs
parameterset:
- name: systemParameter
  init_with: platform.xml
  parameter:
  - { name: account,    _: iosea }
  - { name: queue,      _: dp-cn }
  - { name: executable, _: hostname }
step:
- name: execute
  use:
  - systemParameter
  - {from: platform.xml, _: 'executeset, executesub, jobfiles' }
  do:
    done_file: $done_file
    error_file: $error_file
    _: $submit $submit_script
```

Advanced JUBE

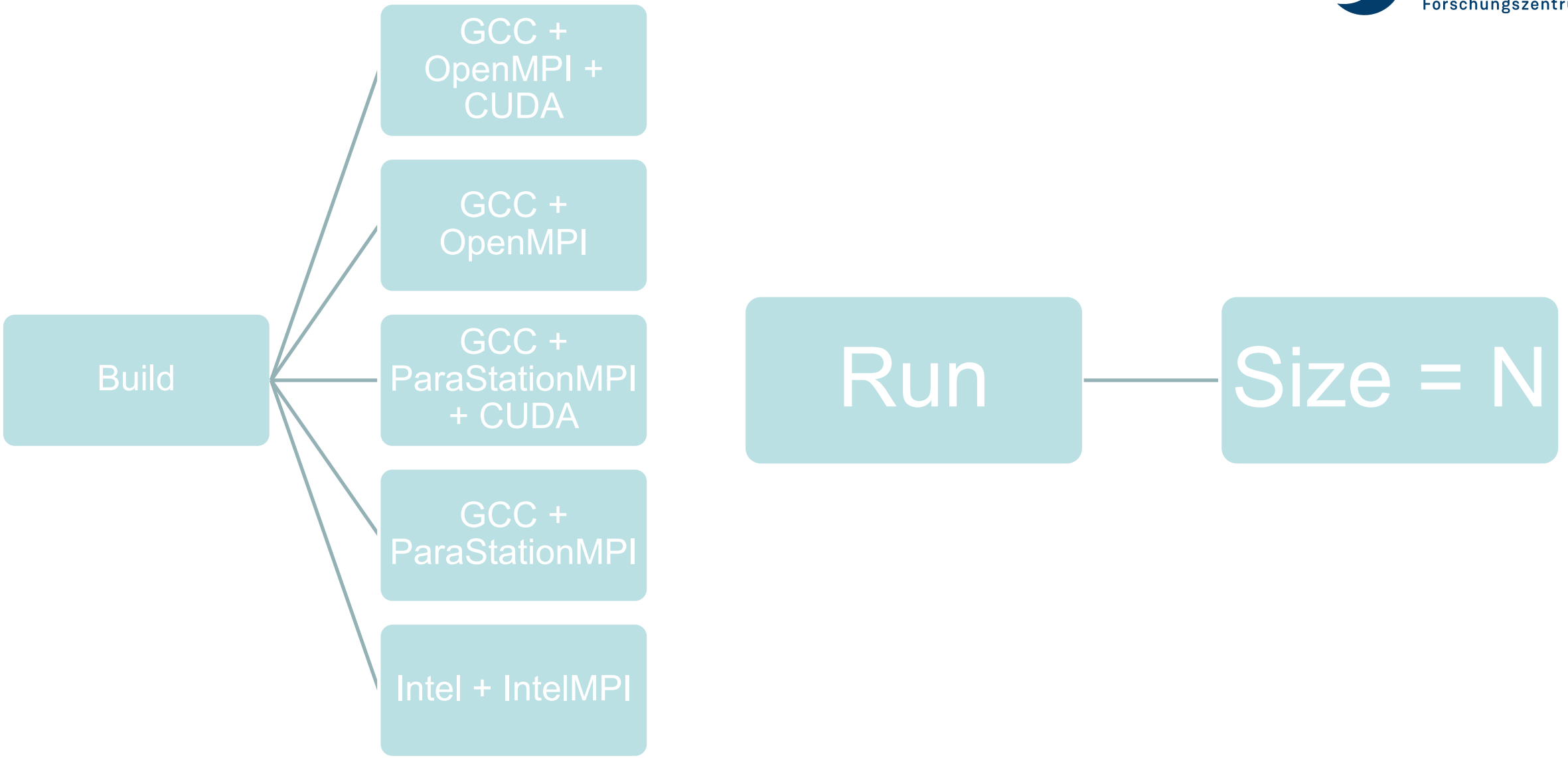
NARROWING THE PARAMETER TREE

XML <=> YAML

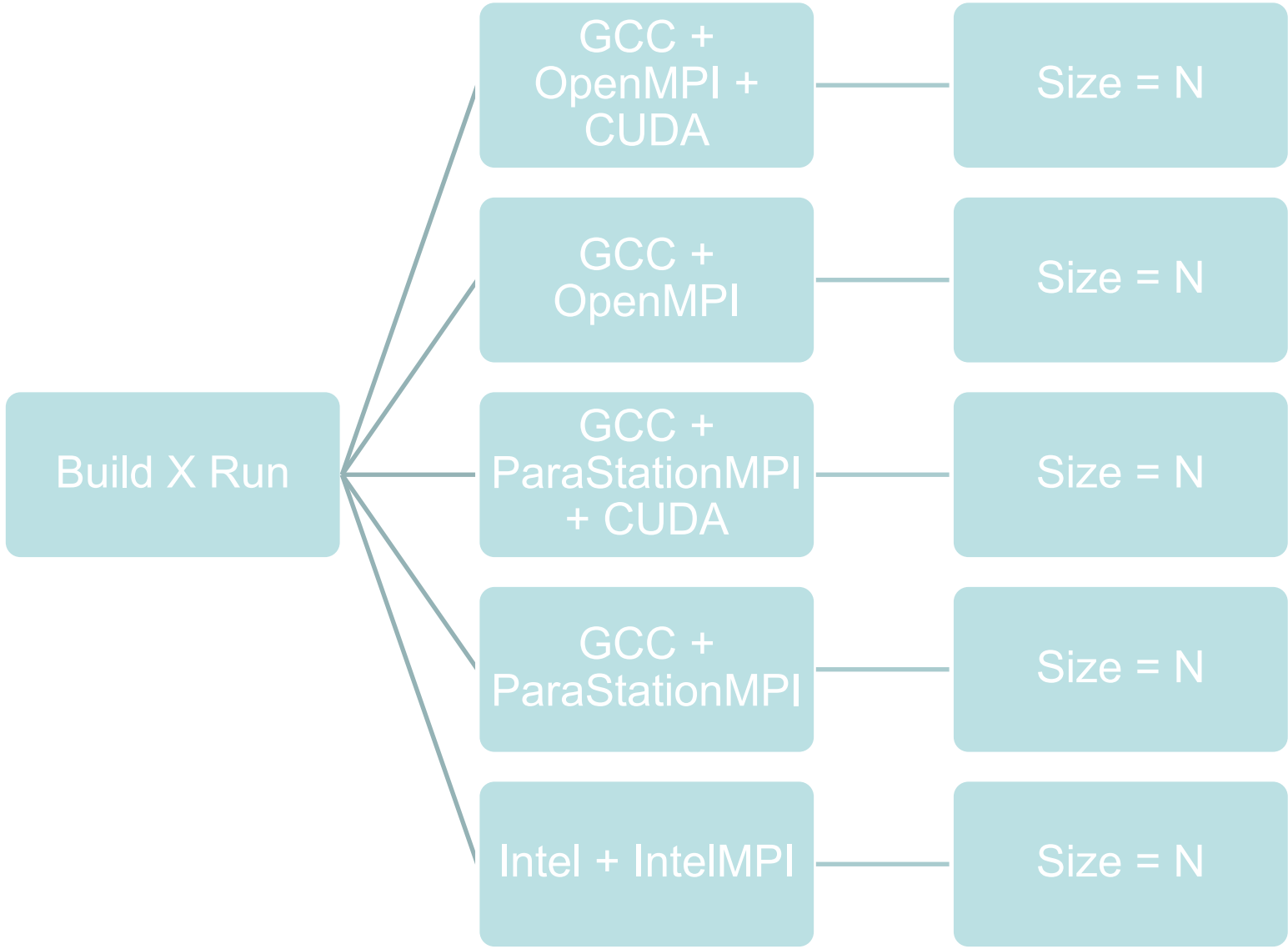
```xml
<parameterset name="params">
  <parameter name="numCores">1,2,4</parameter>
</parameterset>
…
<step name="single" active="$number == 1">
  <!-- … -->
</step>
```

```yaml
parameterset:
- name: params
  parameter:
  - { name: number, _: 1,2,4 }
step:
- name: uneven
  active: $number == 1
```

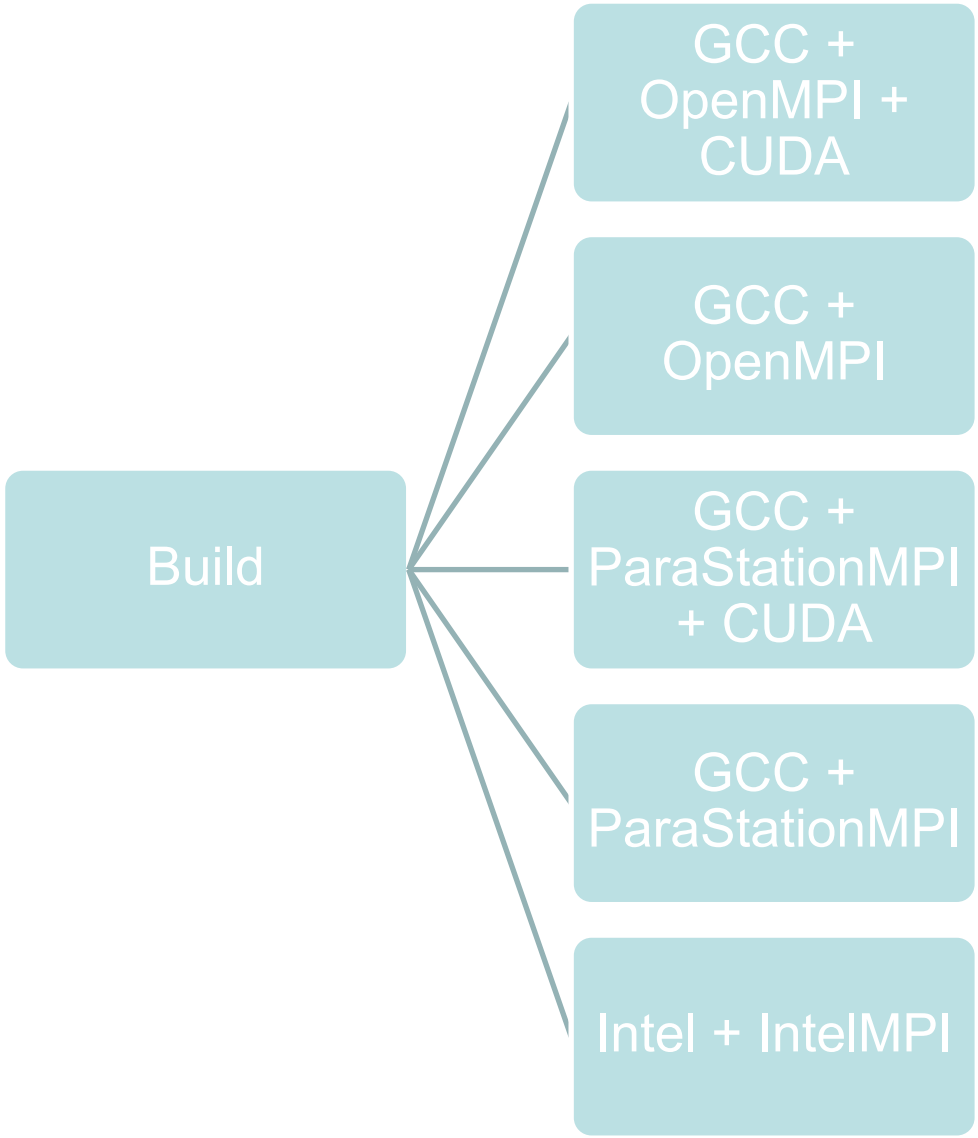
Narrowing the Parameter Tree

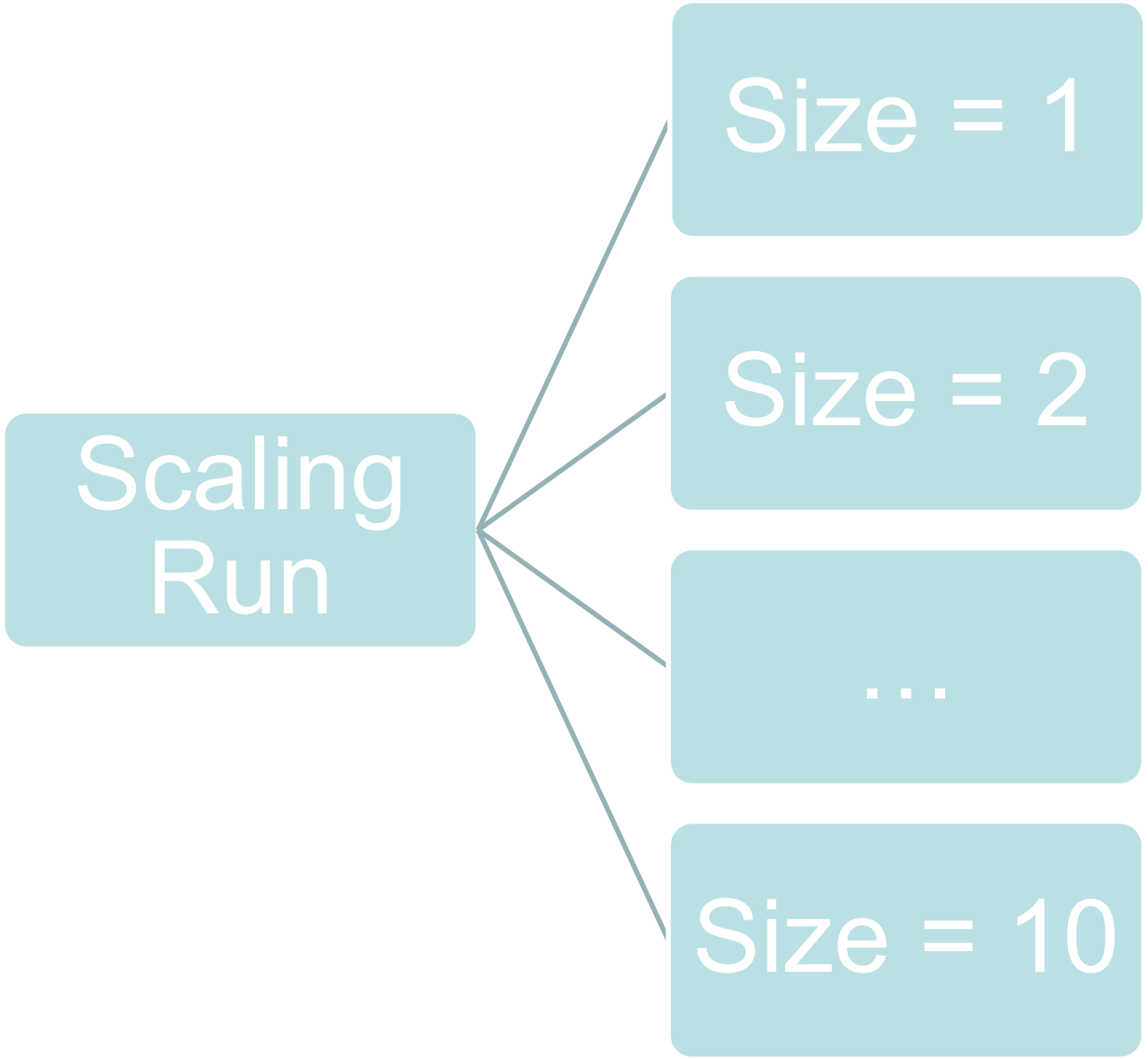


Narrowing the Parameter Tree

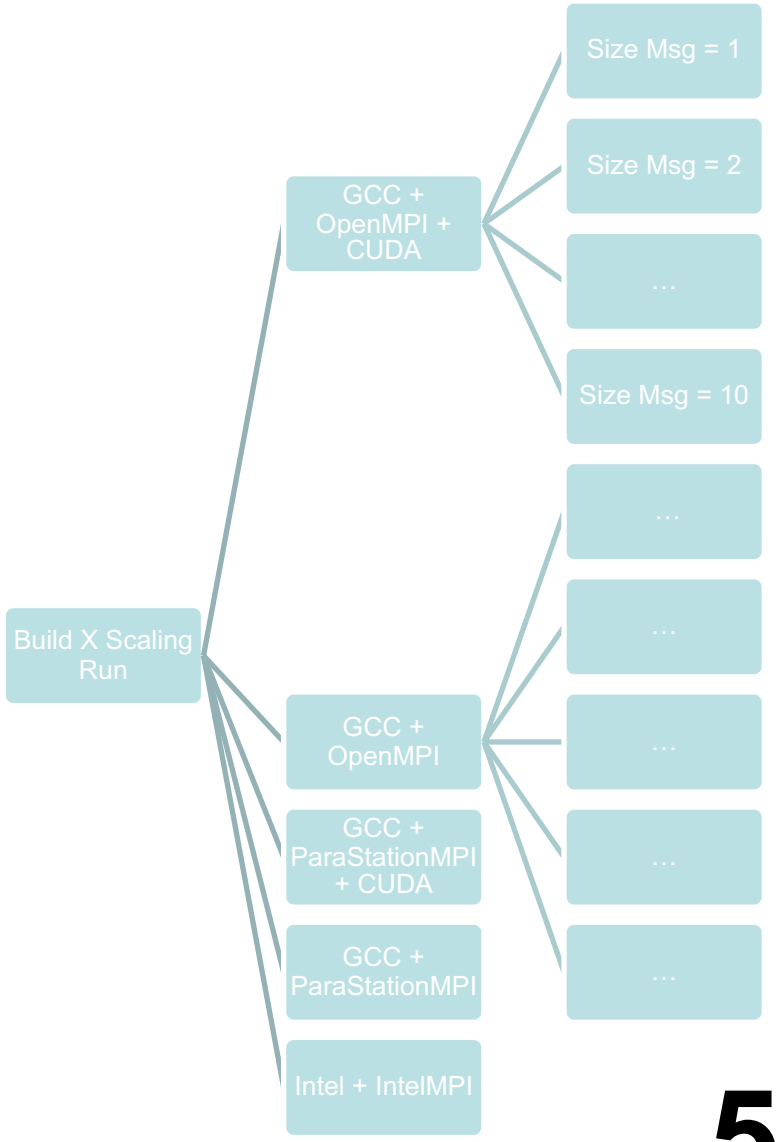


Narrowing the Parameter Tree





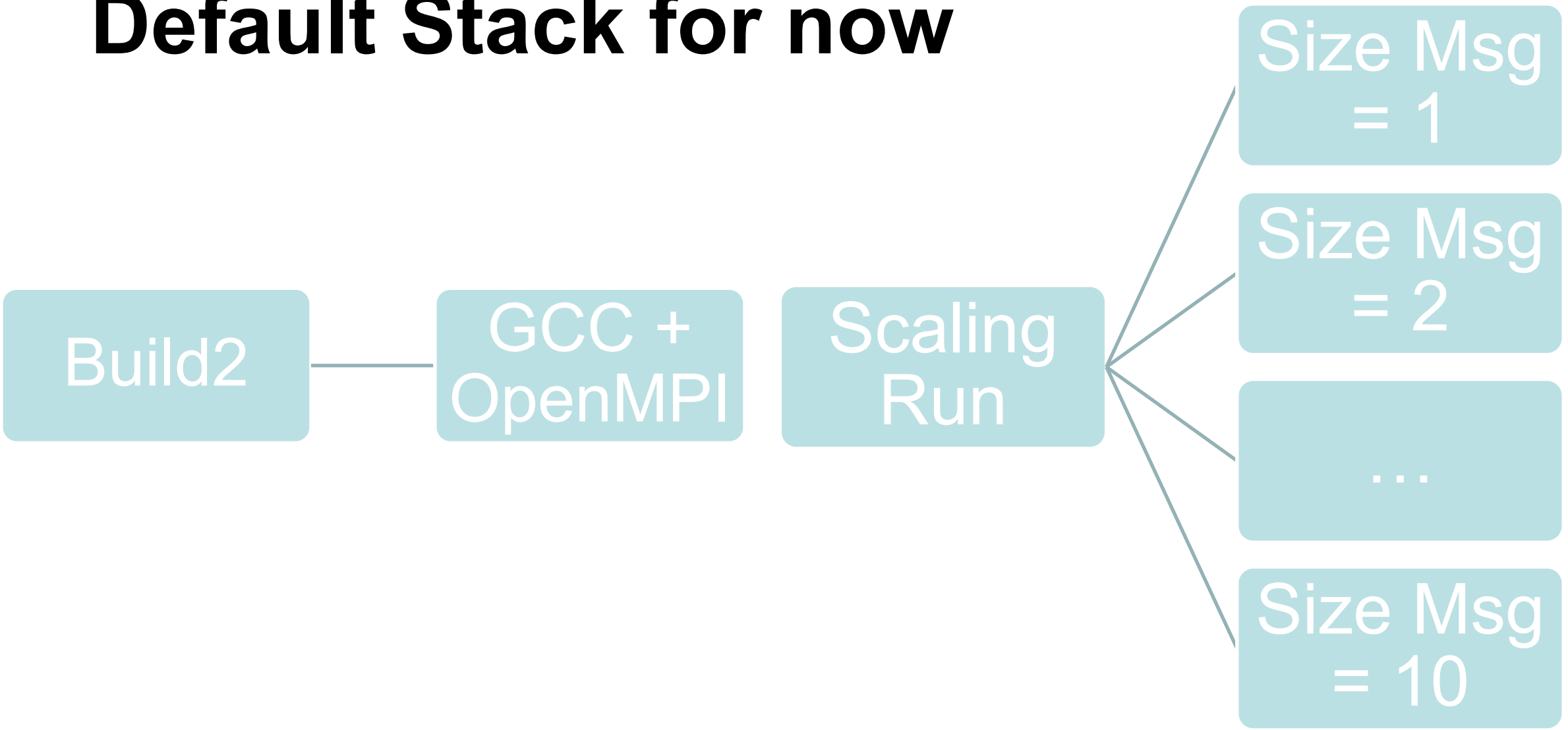
Narrowing the Parameter Tree



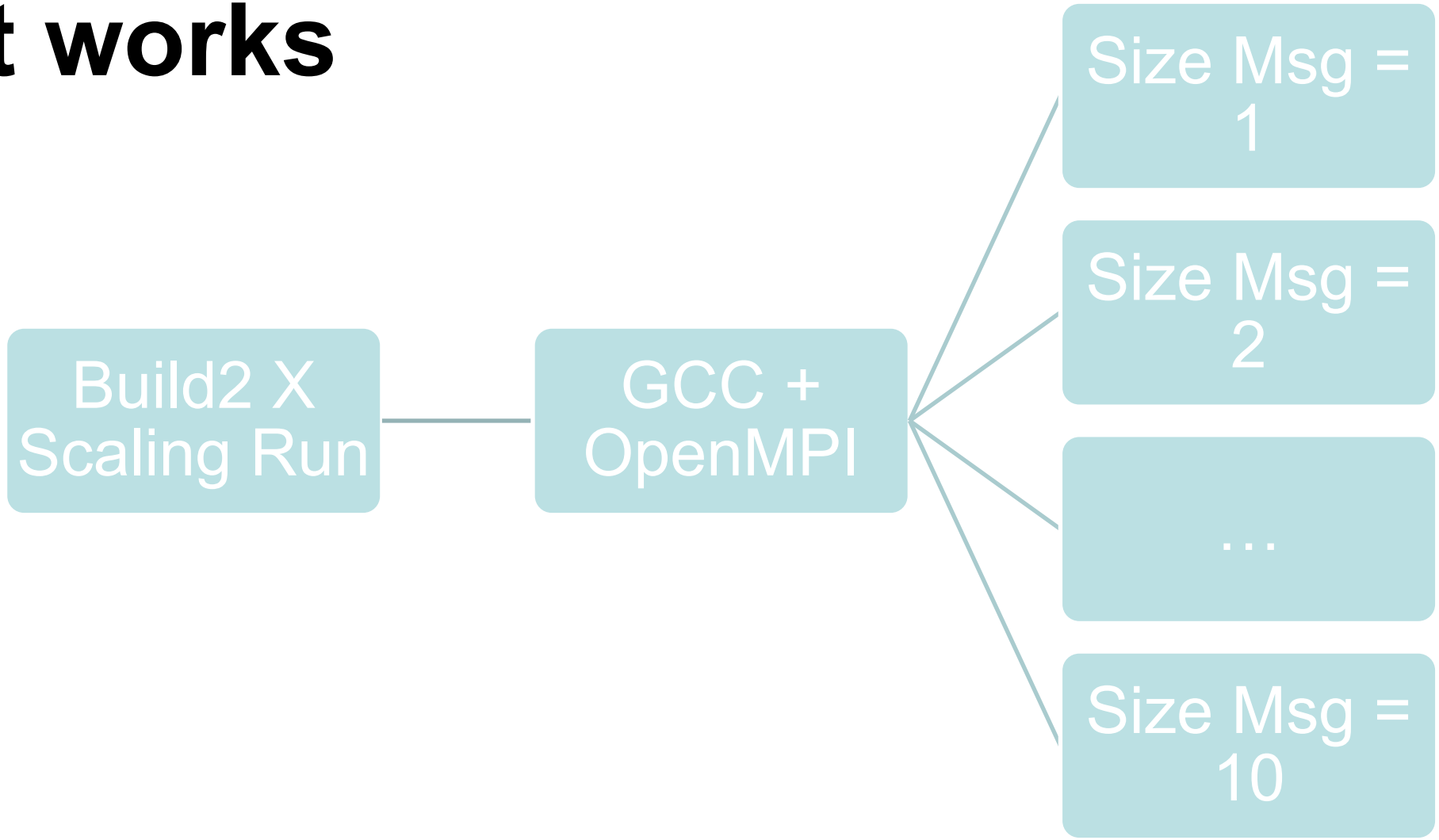
…50 runs!

JÜLICH Forschungszentrum

Default Stack for now

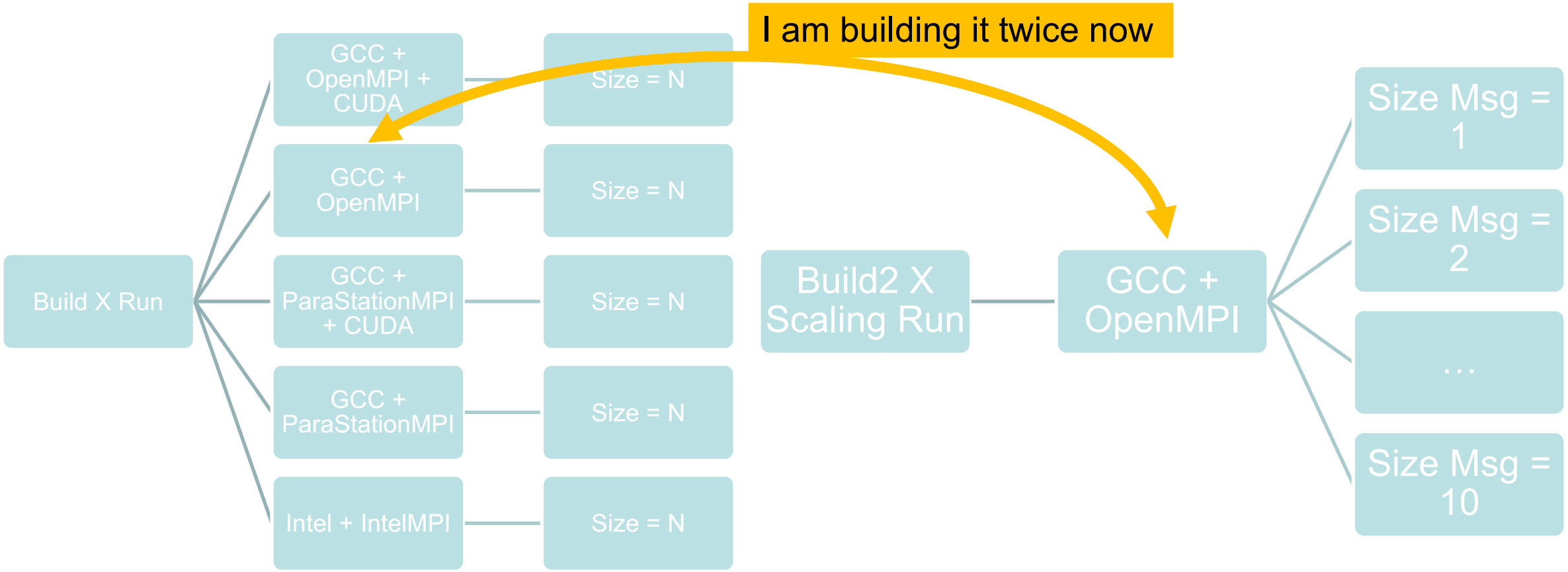


That works


Build2 X Scaling Run
GCC + OpenMPI
Size Msg = 1
Size Msg = 2
...
Size Msg = 10

JÜLICH Forschungszentrum

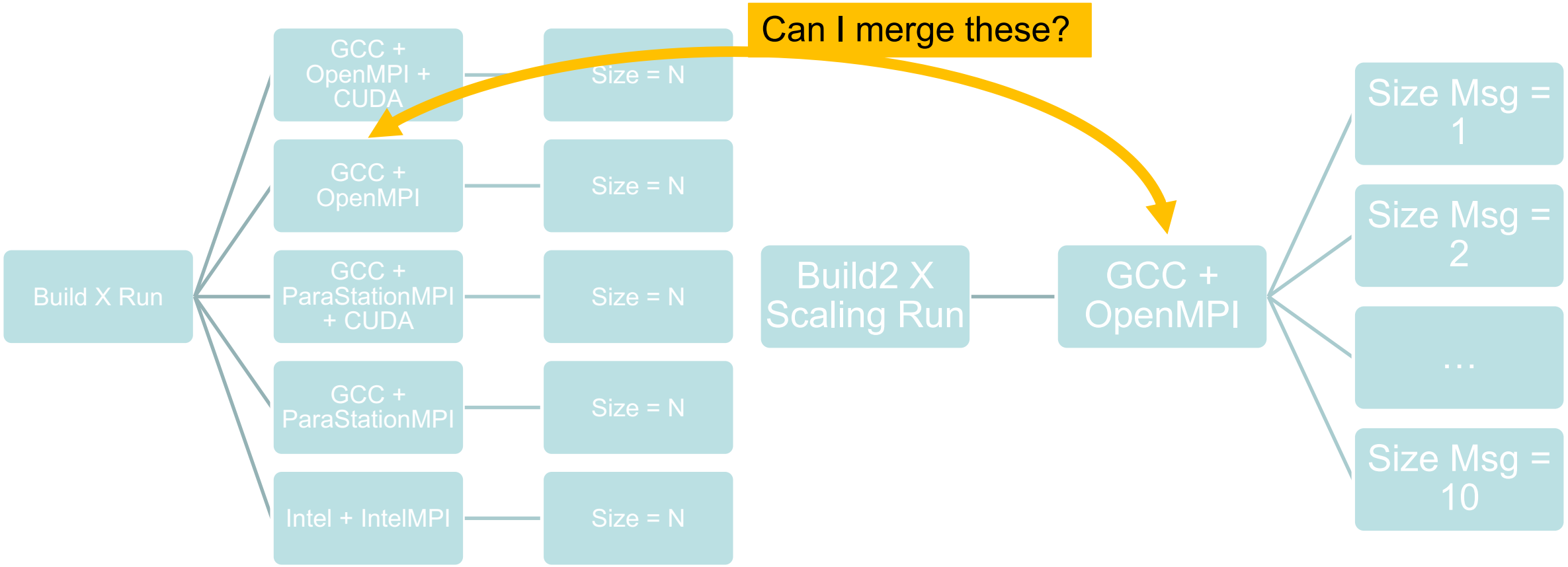
Now Both



Narrowing the Parameter Tree

JÜLICH Forschungszentrum

But I have 7 study sets …



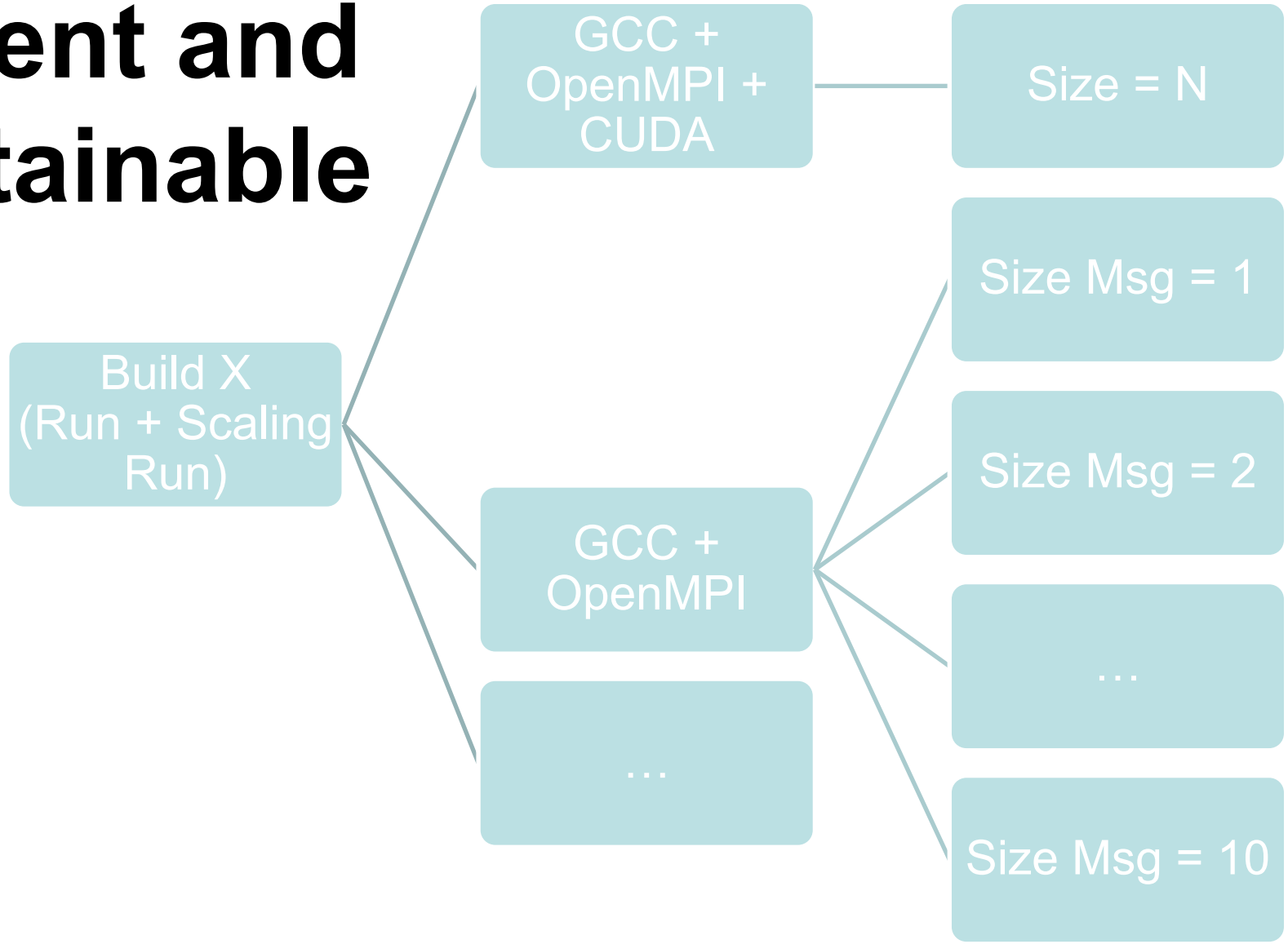
YES

```
<step
  name=“RunTest"
  depend="Build">
    <use from=“RunTest.xml">Run</use>
    <use from="Default.xml">Build</use>
    …
</step>

<step
  name="ScalingTest"
  depend="Build"
  active="'$Compiler_$MPI_CUDA' == 'GCC_ParaStationMPI'">
    <use from=“ScalingTest.xml">Run</use>
    <use from="Default.xml">Build</use>
    …
</step>
```

Allows to restrict N x M parameter space to a subset

JÜLICH Forschungszentrum

Efficient and maintainable



Advanced JUBE

LOOSE COUPLING

Loose Coupling

RUN
(app parameters)

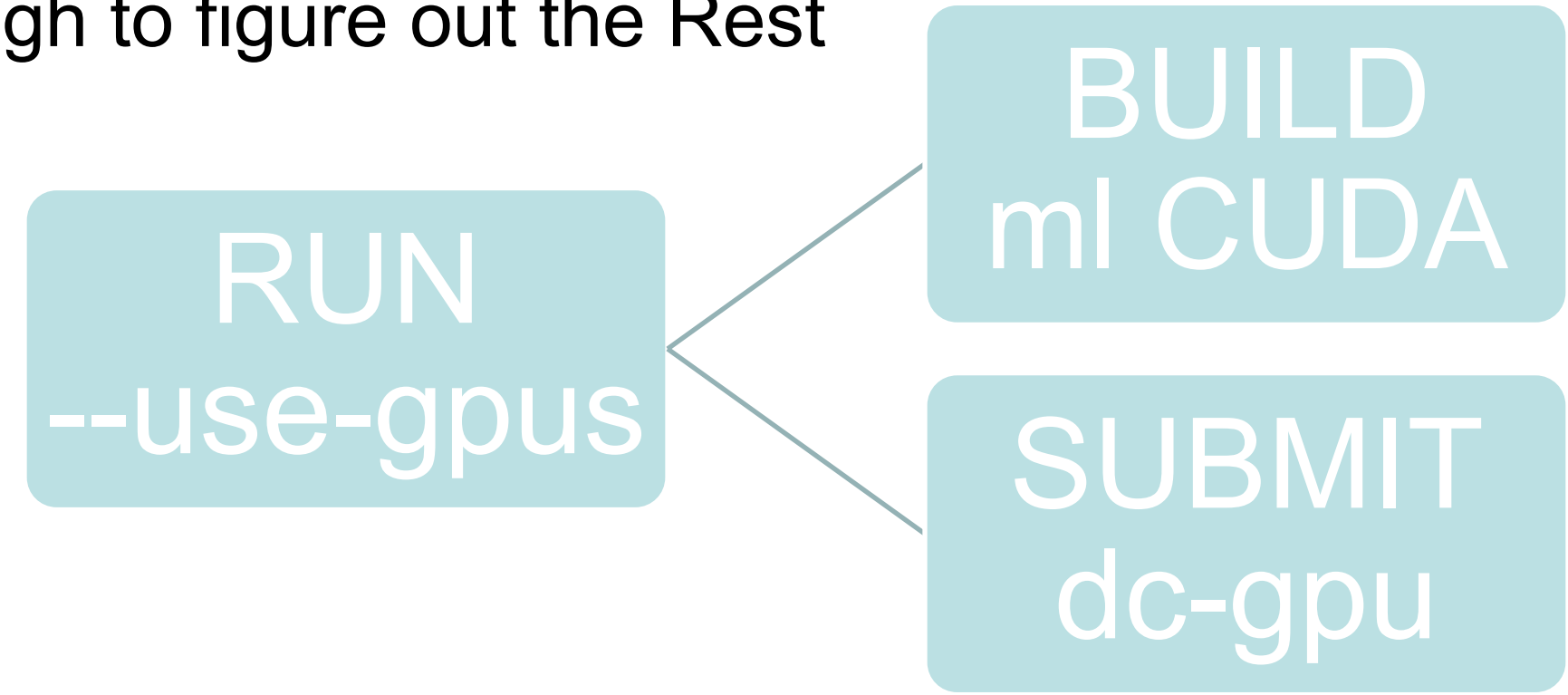
Where to deal with GPU stuff?

BUILD
(build env)

SUBMIT
(job sumission)

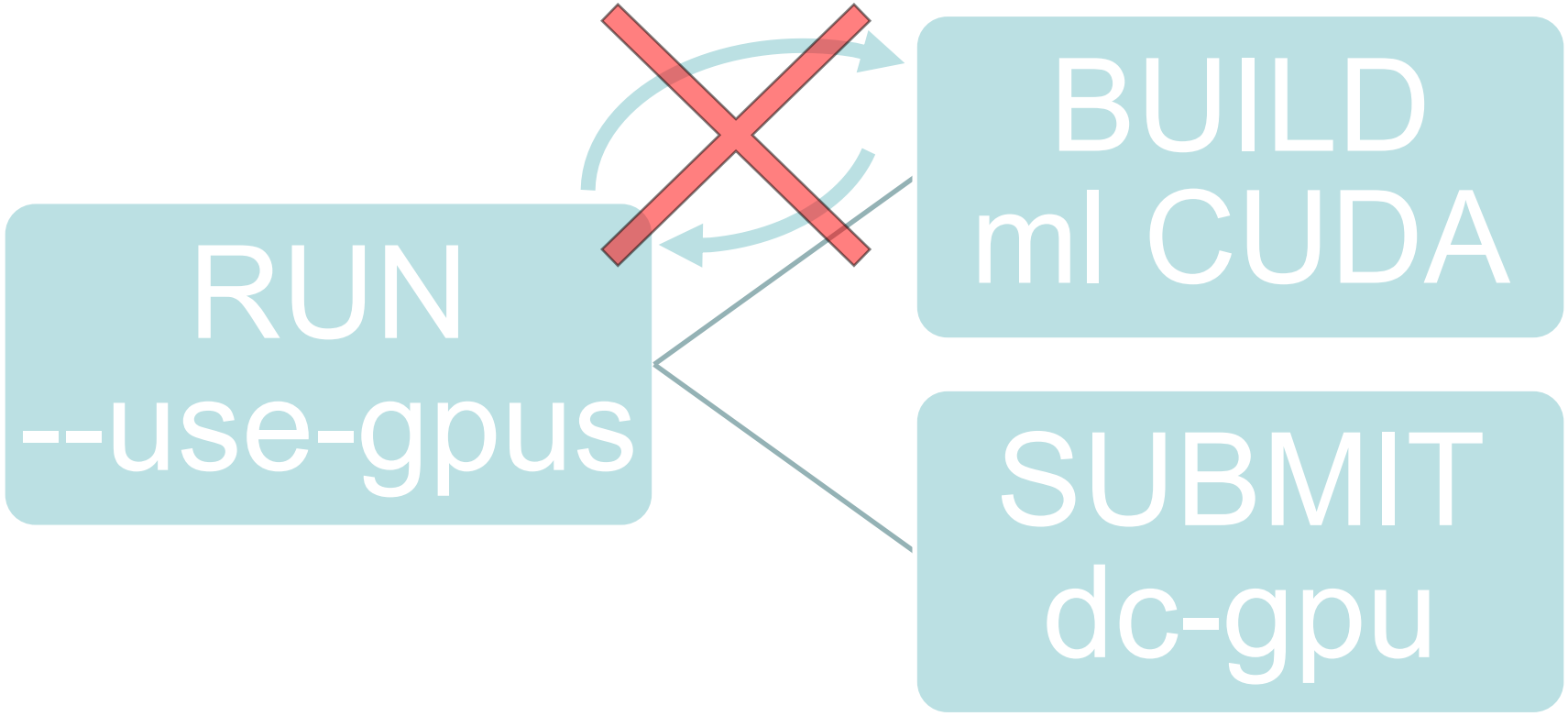
Loose Coupling

In RUN! BUILD and SUBMIT should be smart enough to figure out the Rest



Loose Coupling

But we built RUN with reversed dependencies
Plus we would build once for every run configuration now



Loose Coupling

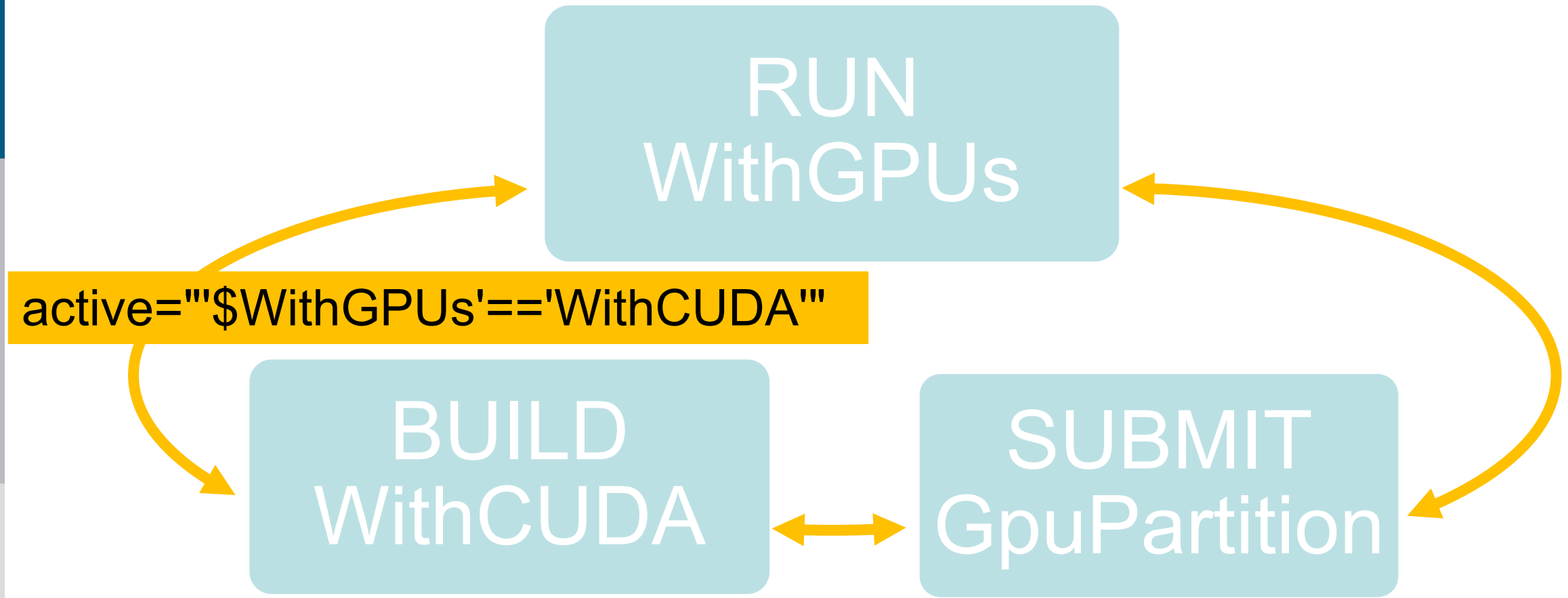
RUN
WithGPUs

Let them all be independent!

BUILD
WithCUDA

SUBMIT
GpuPartition

Loose Coupling

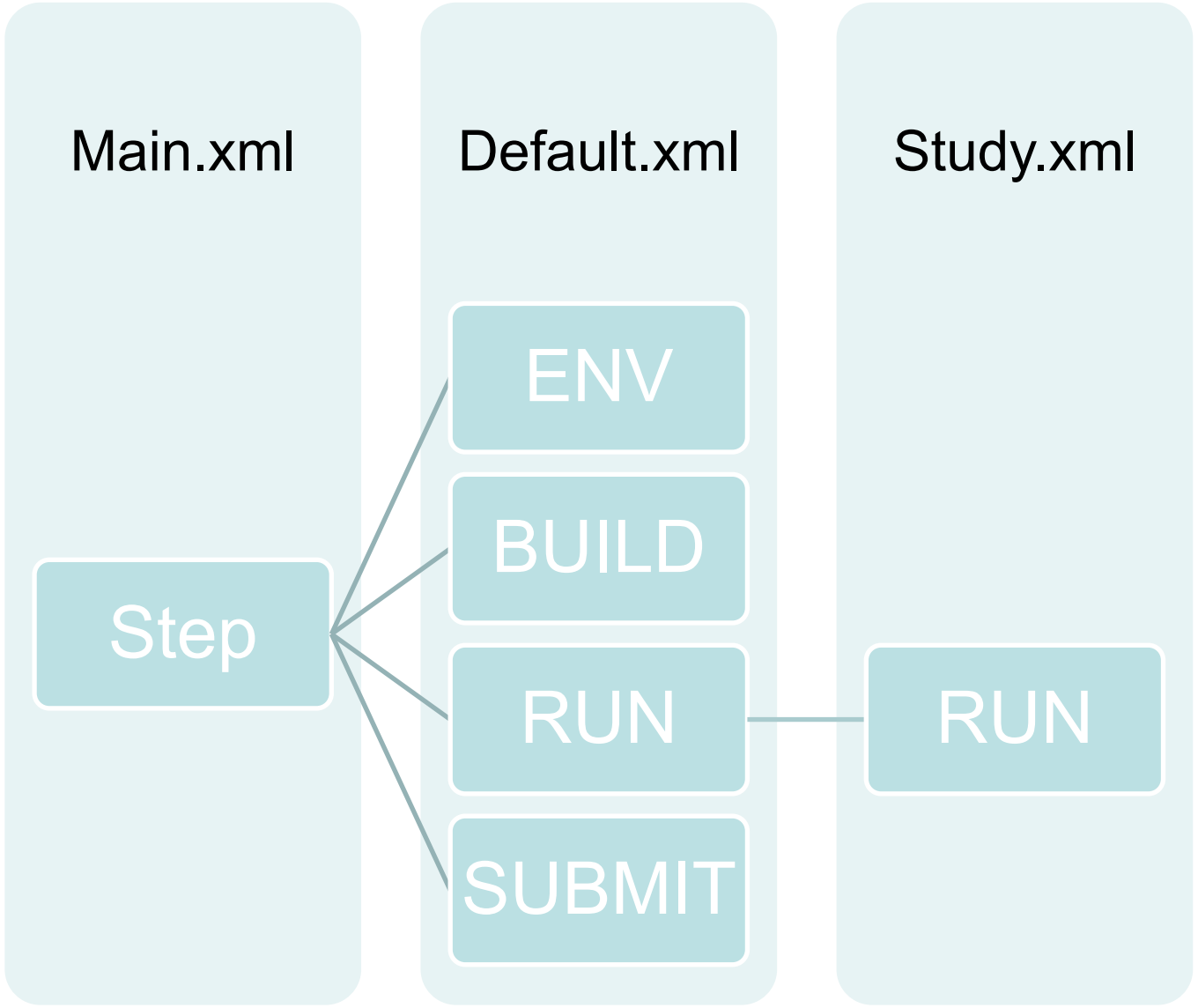


Advanced JUBE

SMART DEFAULTS

Smart Defaults

Study.xml
- no step explosion
- no breaking
- as many as I want
- super concise



ADVANCED JUBE SCRIPT EXAMPLES

LayerTest.xml

```
<parameterset name="Linktest_Args" init_with="Default.xml">
    <parameter name="Messaging_Layer" mode="python">
    {
    "juwels": "ibverbs,ucp,tcp,cuda",
    "jurecadc": "ibverbs,ucp,tcp,cuda",
    "deep": "ibverbs,ucp,tcp,cuda,portals"
    }[ "${System_Name}" ]
    </parameter>
</parameterset>
```

Main.xml

```
<step name="LayerTest" depend="Build" active="'$Stack' == '$Default_Stack' and ${WithCUDA} == ${WithGPUs}">
    <use from="LayerTest.xml">Linktest_Args</use>
    <use from="Default.xml">System, Environment, Slurm, Misc</use>
    <do done_file="ready" error_file="error" tag="!dryRun">sbatch execute.sbatch</do>
</step>
```

Contributors

- Sebastian Lührs
  - Many of the “Basic Features” slides

THANK YOU FOR YOUR ATTENTION!
QUESTIONS?