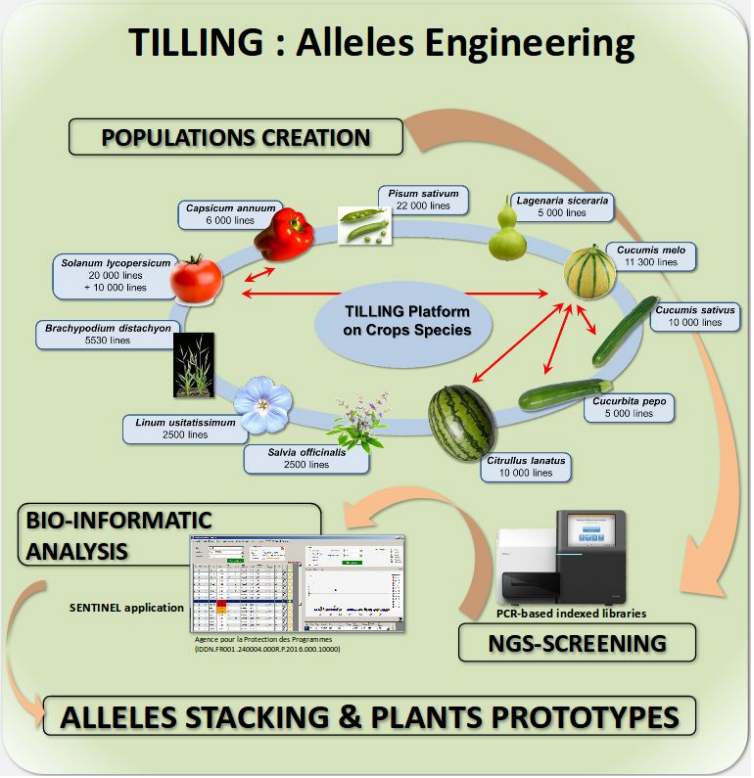
Use of Snakemake on EPITRANS platform / IPS2 Paris-Saclay

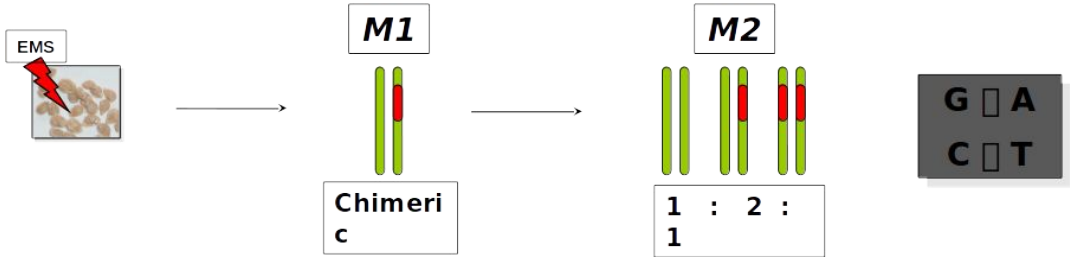
Joseph Tran @ EPITRANS/FLOCAD/IPS2

Journées PEPI IBIS / 2019-06-06

EPITRANS platform: TILLING activity



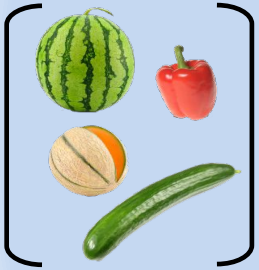
Collection creation

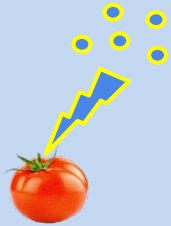


Fast neutron TILLING project

RIJK ZWAAN + FloCaD Flower & Carpel Developement

Fast Neutron mutagenesis





First FN population on tomato

New Screening methods

SMALL INDEL

< 500pb

BIG INDEL

> 1 kb

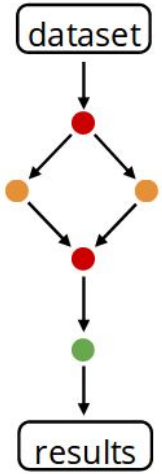
PCR-based libraries sequencing

ddPCR screens

Data analysis quality challenges

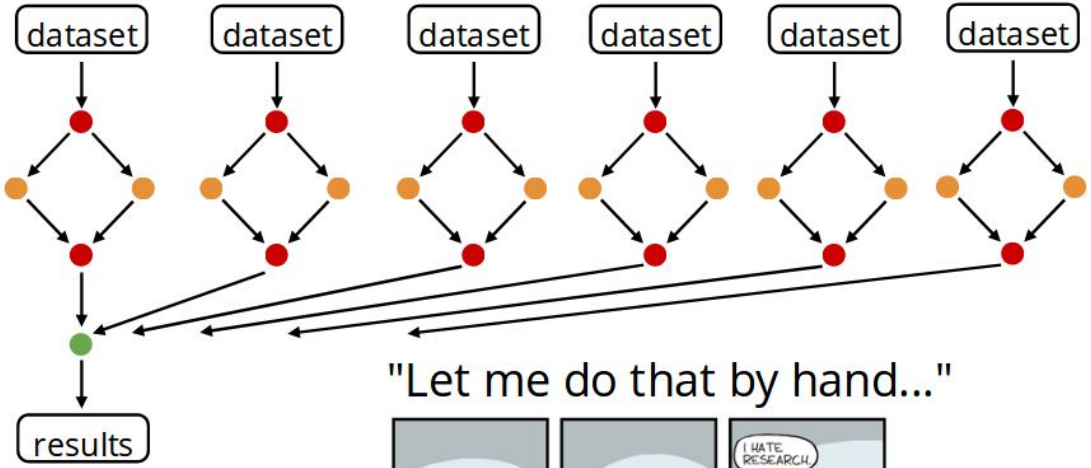
Data analysis usually entail the application of many command line tools/scripts to transform, filter, aggregate or plot data and results.

With Increasing amounts of data being collected in science, reproducible and scalable automatic workflow management becomes increasingly important.



"Let me do that by hand..."

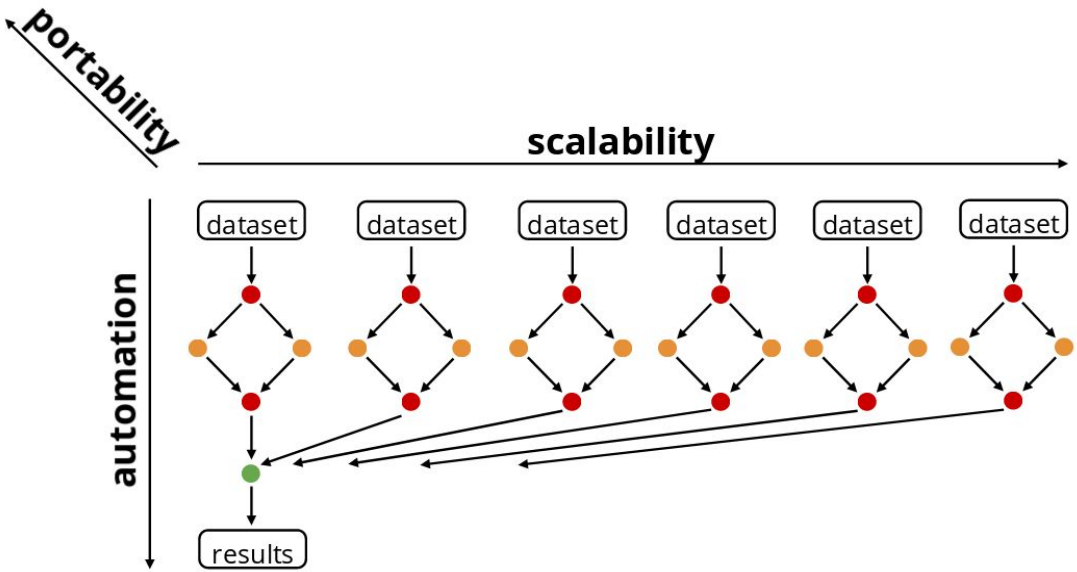
credits: Johannes Köster



"Let me do that by hand..."

credits: Johannes Köster

Reproducibility challenges



1. **Automation**

   From raw data to final figures:
   - **document** parameters, tools, versions
   - **execute** without manual intervention

2. **Scability**

   Handle parallelization:
   - execute for tens to thousands of datasets
   - efficiently use any computing platform

3. **Portability**

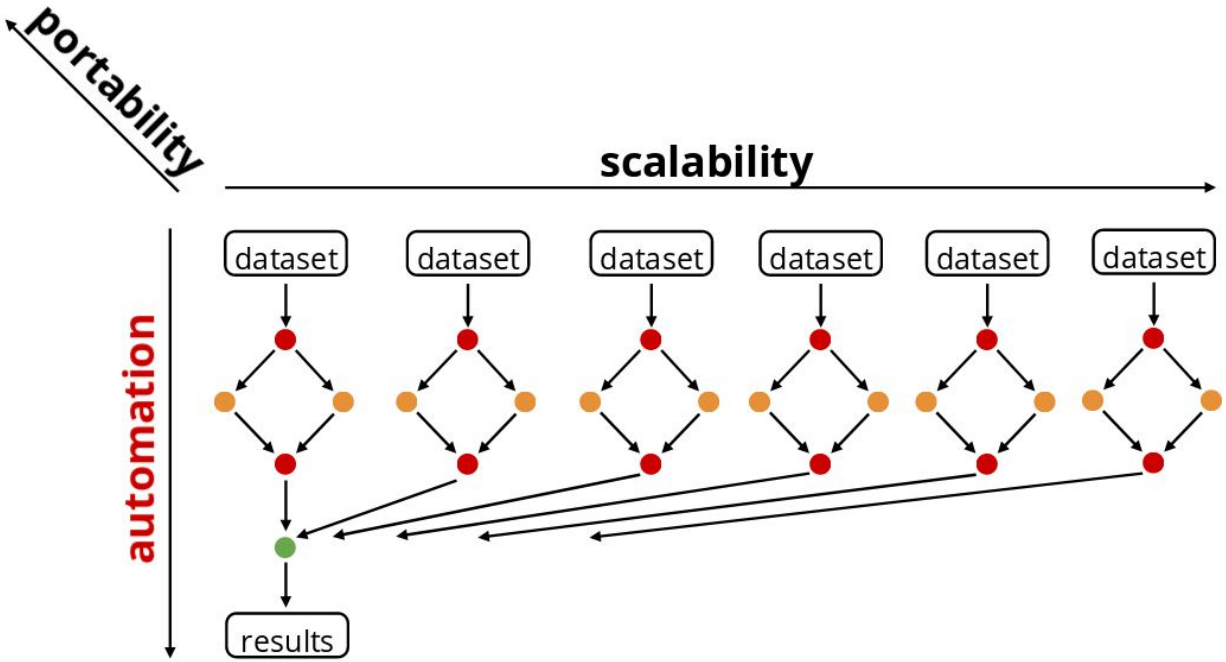
   Handle deployment:
   be able to easily execute analyses on a different system/platform/infrastructure

credits: Johannes Köster

Snakemake for reproducible data analysis

- Decompose analysis into rules, written in a Python dialect.
- Rules define how to obtain output files from input files.
- Snakemake determines dependencies and execution order in the form of a directed acyclic graph (DAG) of jobs.

Automation: define workflows in terms of rules



credits: Johannes Köster

Automation: define workflows in terms of rules

```
rule mytask:
    input:
        "path/to/{dataset}.txt"
    output:
        "result/{dataset}.txt"
    script:
        "scripts/myscript.R"

rule myfiltration:
     input:
        "result/{dataset}.txt"
     output:
        "result/{dataset}.filtered.txt"
     shell:
        "mycommand {input} > {output}"

rule aggregate:
    input:
        "results/dataset1.filtered.txt",
        "results/dataset2.filtered.txt"
    output:
        "plots/myplot.pdf"
    script:
        "scripts/myplot.R"
```

credits: Johannes Köster

Automation: define workflows in terms of rules

rule name

```
rule mytask:
    input:
        "data/{sample}.txt"
    output:
        "result/{sample}.txt"
    shell:
        "some-tool {input} > {output}"
```

define

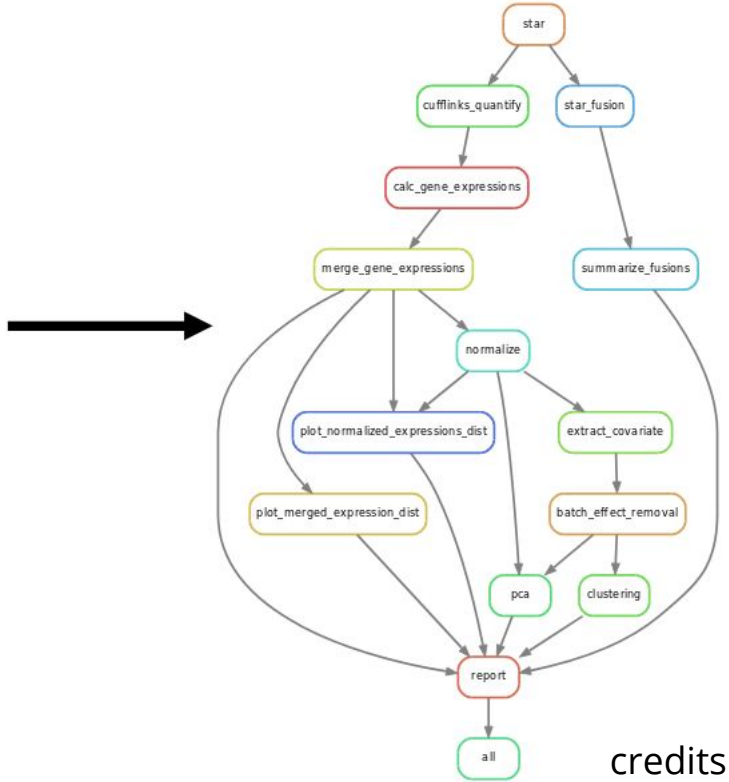
- input
- output
- log files
- parameters
- resources

how to create output from input

credits: Johannes Köster

Automation: define workflows in terms of rules

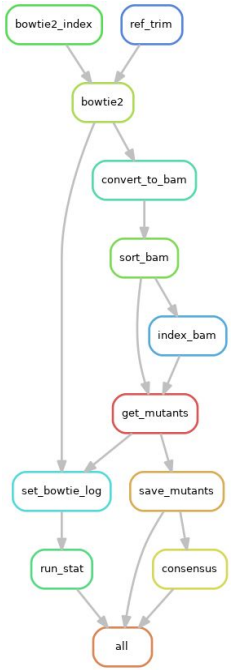
```
rule mytask:
    input:
        "path/to/{dataset}.txt"
    output:
        "result/{dataset}.txt"
    script:
        "scripts/myscript.R"

rule myfiltration:
    input:
        "result/{dataset}.txt"
    output:
        "result/{dataset}.filtered.txt"
    shell:
        "mycommand {input} > {output}"

rule aggregate:
    input:
        "results/dataset1.filtered.txt",
        "results/dataset2.filtered.txt"
    output:
        "plots/myplot.pdf"
    script:
        "scripts/myplot.R"
```



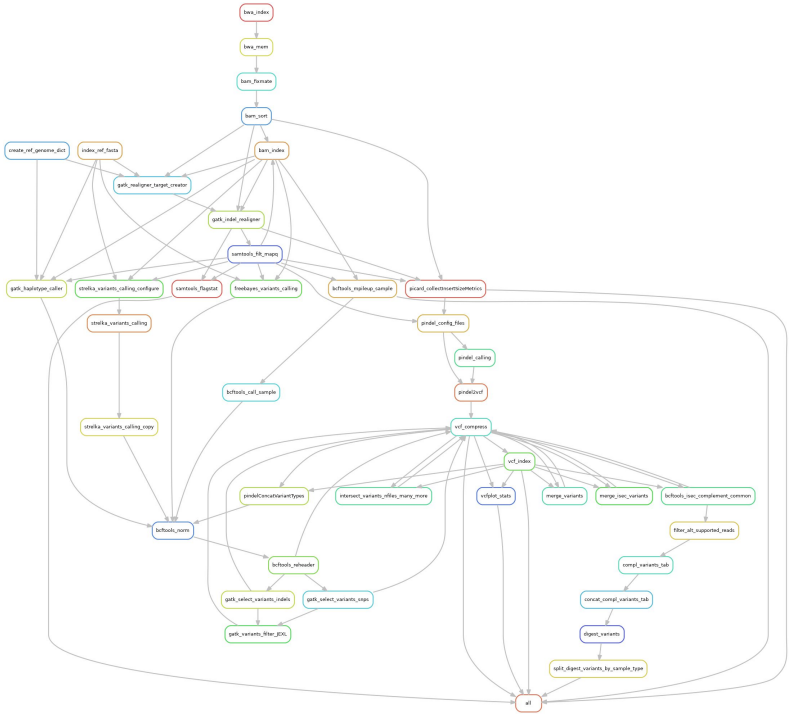
credits: Johannes Köster

Simple to complex workflows

EMS variants detection



Fast neutron variants detection



Automation: test

```
# performs the rulegraph generation
snakemake --rulegraph | dot -Tpng >rulegraph.png

# performs the dag generation
snakemake --dag | dot -Tpng >dag.png

# performs dry-run
snakemake -n

# prints shell commands to be executed
snakemake -p

# performs on a subset of rules
snakemake --omit-from rule2 --until rule5

# performs timestamp logs
snakemake --timestamp
```

Automation: modularity

```
rule mytask:
    input:
        "data/{sample}.txt"
    output:
        "result/{sample}.txt"
    script:
        "scripts/myscript.py"
```

reusable Python/R scripts

```
rule sort_reads:
    input:
        "{sample}.bam"
    output:
        "{sample}.sorted.bam"
    wrapper:
        "0.22.0/bio/samtools/sort"
```

reuseable wrappers from central repository

```
rule sort_reads:
    input:
        "{sample}.bam"
    output:
        "{sample}.sorted.bam"
    cwl:
        "https://github.com/common-workflow-language/"
        "workflows/blob/fb406c95/tools/samtools-sort.cwl"
```

use CWL tool definitions

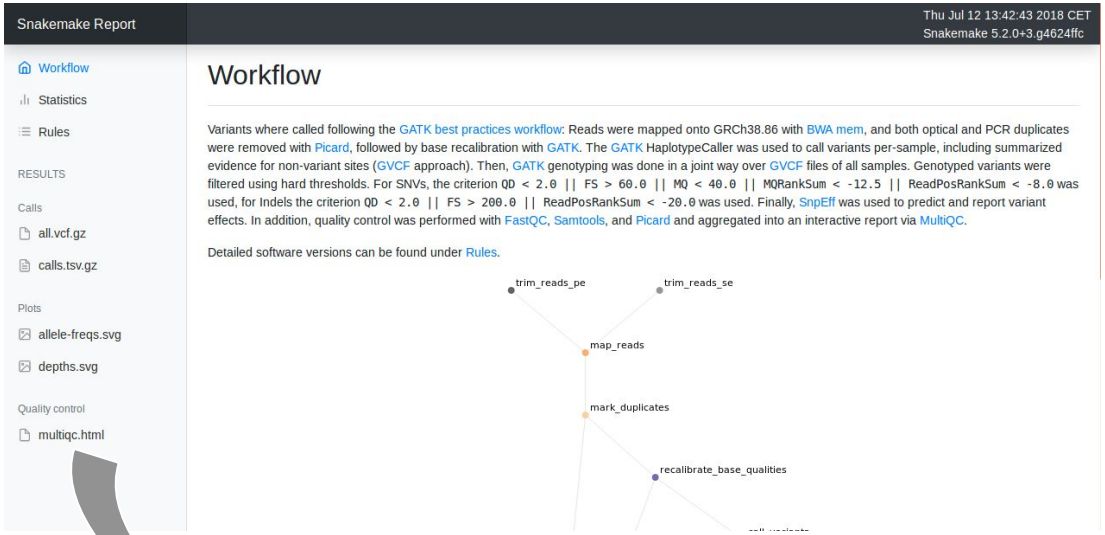
credits: Johannes Köster

Documentation

- parameters: config.yaml
- rules as documentation (parameters, tools, version)
- automatic reports:
  - report function to annotate output files for inclusion.
  - Define categories and (jinja-templated) captions.
  - Obtain self-contained HTML5 document including all files, workflow description, runtime statistics, and provenance information.

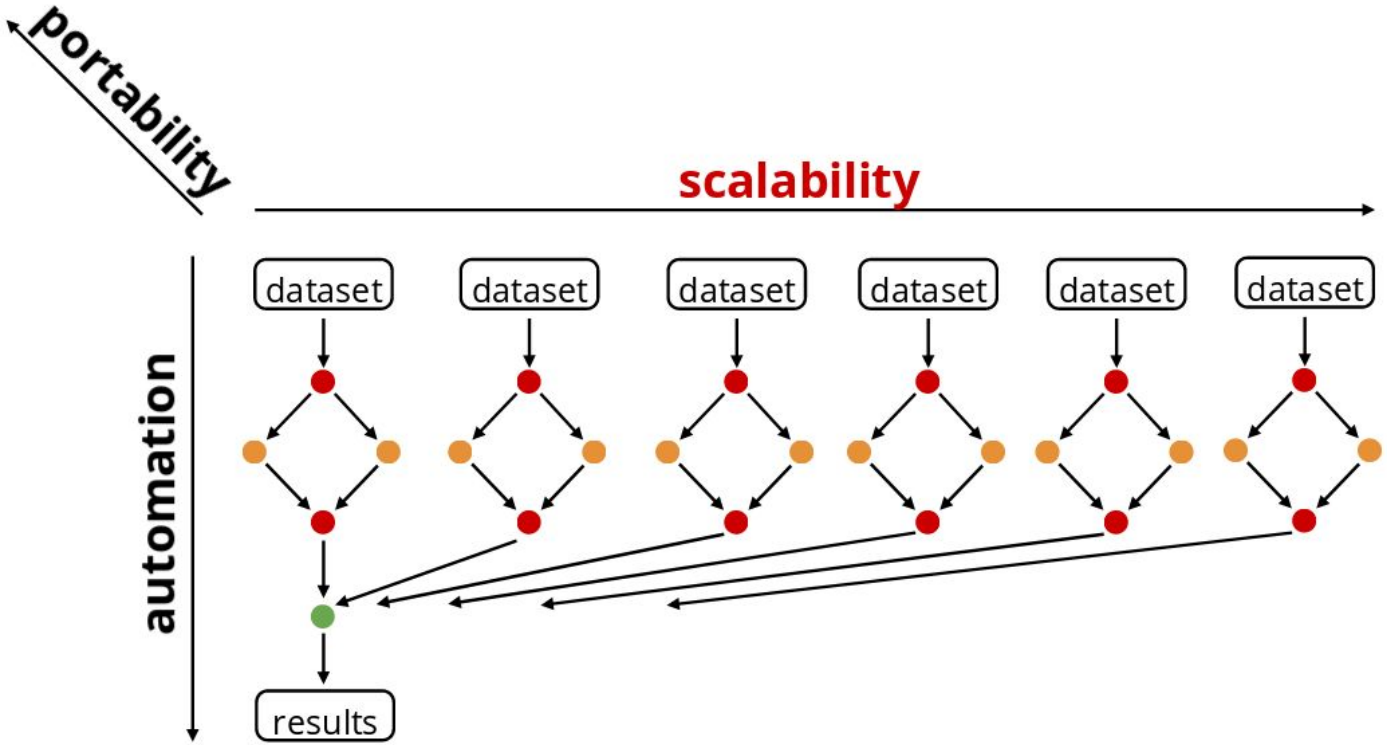
```
# performs report generation
snakemake --report report.html
```

report.html



multiqc.html

Scalability: maximize parallelization



credits: Johannes Köster

Scalability: maximize parallelism

Workflow definition shall be independent of computing platform and available resources.

Rules define resource usage (threads, memory, etc.)

And the scheduler

- schedules independent jobs in parallel
- passes resource requirements to any backend.

Scalability in Snakemake

- Independent parts of the DAG of jobs can be executed in parallel.
- Snakemake maximizes parallelism while respecting given resources.
- Without modification of the workflow definition, Snakemake can scale to any number of cores, compute clusters, the grid, and the cloud.

```
# execute workflow locally with 16 CPU cores
snakemake --cores 16

# execute on cluster
snakemake --cluster qsub --jobs 100

# execute in the cloud
snakemake --kubernetes --jobs 1000 --default-remote-provider GS --default-remote-prefix mybucket
```

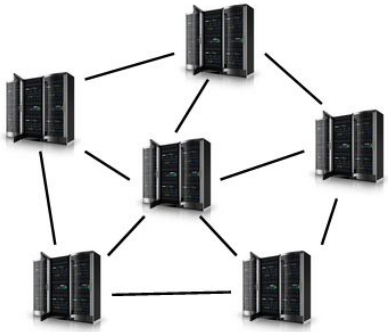
Scalable to any platform

workstation

compute server

cluster

grid computing



cloud computing

kubernetes

Google Cloud Platform

amazon web services

credits: Johannes Köster

Snakemake with Slurm Workload Manager

```
# executes workflow on slurm cluster
CONFIG=config.yaml

CLUSTER="sbatch --mem={cluster.mem} -p {cluster.partition}
-c {cluster.c} -J {cluster.jobname} -o
log/{cluster.jobname}.%N.%j.out -e
log/{cluster.jobname}.%N.%j.err"

CLUSTER_CONFIG=cluster.json

RULES=snakemake-workflows/ngs-qc-filter-trim/Snakefile

MAX_JOBS=999

snakemake --configfile $CONFIG -s $RULES -p --use-conda -j
$MAX_JOBS --cluster-config $CLUSTER_CONFIG --cluster
"$CLUSTER" --latency-wait 60
```

```
# cluster.json
{
	"__default__" :
	{
		"jobname": "default",
	"c" : 1,
	"partition" : "workq",
		"mem": 2000
	},"bowtie_index" :
	{
	"jobname": "bowtie_index",
	"c": 4,
	"mem": 16000
	},
	"bowtie2_index" :
	{
	"jobname": "bowtie2_index",
	"c": 4,
	"mem": 16000
	},
	"fastqc" :
	{
```

Scalability: optimization

Job groups:

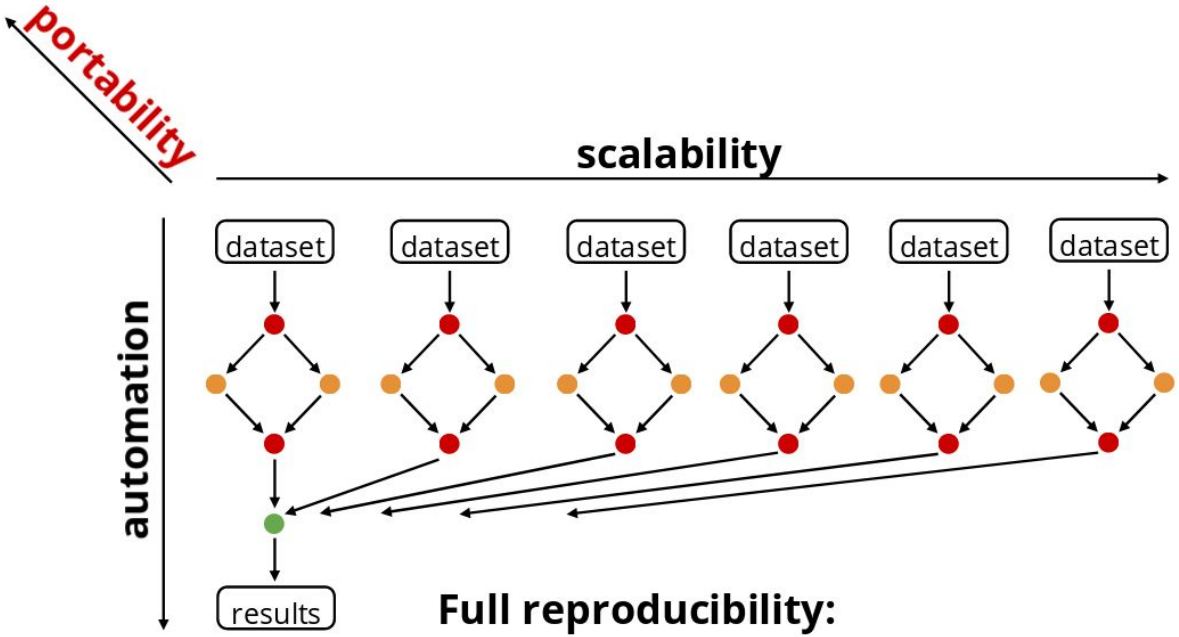
- The DAG of jobs can be partitioned into groups.
- Minimizes queueing and network overhead in cloud and cluster.

Pipe output instead of temp:

- Output files can be marked as pipes.
- Consuming jobs will be assigned to the same group.
- Output will not be written to disk but streamed between the jobs.

credits: Johannes Köster

Portability



Full reproducibility:

install required software and **all** dependencies in exact versions

credits: Johannes Köster

Portability: use package management with

CONDA®

Idea:

Normalization installation via recipes

- source or binary
- recipe and build script

```
package:
  name: seqtk
  version: 1.2

source:
  fn: v1.2.tar.gz
  url: https://github.com/lh3/seqtk/archive/v1.2.tar.gz

requirements:
  build:
    - gcc
    - zlib
  run:
    - zlib

about:
  home: https://github.com/lh3/seqtk
  license: MIT License
  summary: Seqtk is a fast and lightweight tool for processing sequences

test:
  commands:
    - seqtk seq
```

```
#!/bin/bash

export C_INCLUDE_PATH=${PREFIX}/include
export LIBRARY_PATH=${PREFIX}/lib

make all
mkdir -p $PREFIX/bin
cp seqtk $PREFIX/bin
```

- package

credits: Johannes Köster

Portability: Conda support in snakemake

- Rules can be annotated with (isolated) Conda environments that define a software stack with particular versions to use.
- Jobs are executed within these environments.

```
rule mytask:
    input:
        "path/to/{dataset}.txt"
    output:
        "result/{dataset}.txt"
    conda:
        "envs/some-tool.yaml"
    shell:
        "some-tool {input} > {output}"
```

```
channels:
  - conda-forge
dependencies:
  - some-tool =2.3.1
  - some-lib =1.1.2
```

credits: Johannes Köster

Portability: Singularity support in snakemake

```
rule mytask:
    input:
        "path/to/{dataset}.txt"
    output:
        "result/{dataset}.txt"
    singularity:
        "docker://biocontainers/some-tool#2.3.1"
    shell:
        "some-tool {input} > {output}"
```

- Rules/workflows can be annotated with container images.
- Jobs are executed within the container.
- Combination with Conda possible: use container image to define OS, use Conda to define the software stack, let Snakemake perform the composition.

credits: Johannes Köster

Portability: Singularity + Conda

define OS ←

```
singularity:
    "docker://continuumio/miniconda3:4.4.1"
```

define tools/libs ←

```
rule mytask:
    input:
        "path/to/{dataset}.txt"
    output:
        "result/{dataset}.txt"
    conda:
        "envs/some-tool.yaml"
    shell:
        "some-tool {input} > {output}"
```

credits: Johannes Köster

Snakemake evaluation on EPITRANS platform

- Very very positive
- python
- reusable (modularization capabilities)
- great doc
- great test/reporting capabilities
- quick and less less dirty to develop workflows
- seamless execution on all platforms without adaptation of the workflow
- integrated package management

Perspectives

- modularity: use/develop wrappers to include into workflows
- profiles: use generic cluster profile
- containerization: use singularity images

Acknowledgements

Thanks to the EPITRANS platform and FLOCAD team

especially

- Eulalie Lefeuvre
- Fabien Marcel
- Brahim Mania
- Marion Dalmais
- and Abdel Bendahmane


ARIGATÔ
RAHMET
MOTASHAKKERAM
MAHALO
SUKRIYA
TAK
MERCI
TACK
BARKA
THANK YOU
DANKE
NAJIS TUKE
OBRIGADO
MANANA
СПАСИБО
谢谢
HVALA
GRACIAS
TODA