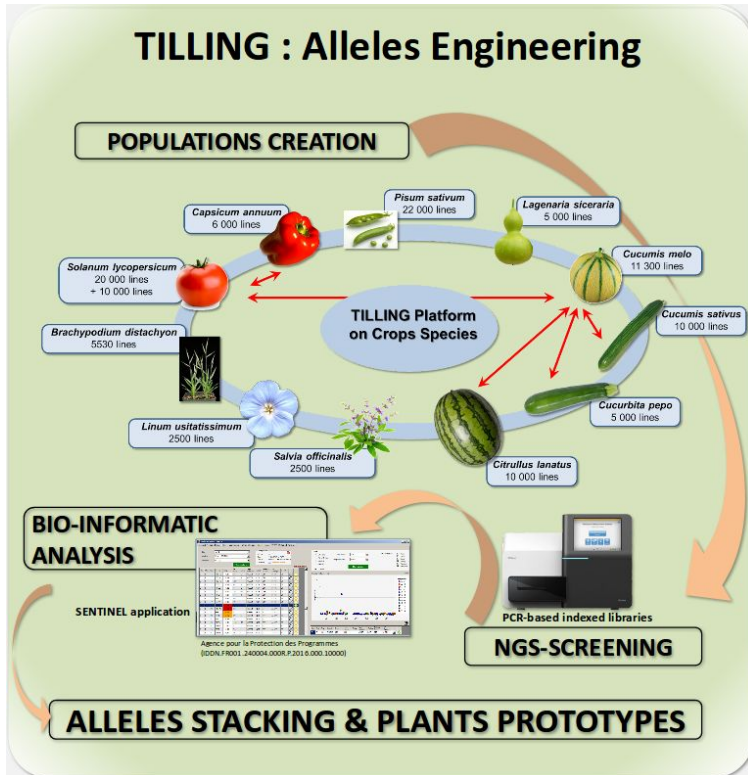
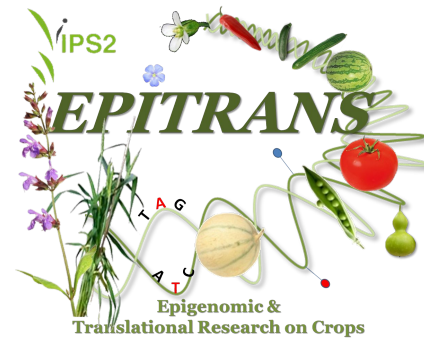


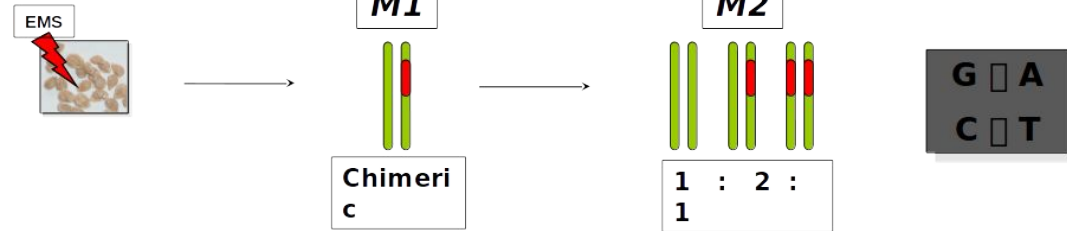
Use of Snakemake on EPITRANS platform / IPS2 Paris-Saclay

Joseph Tran @ EPITRANS/FLOCAD/IPS2
Journées PEPI IBIS / 2019-06-06

EPITRANS platform: TILLING activity



Collection creation



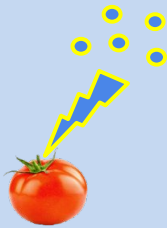
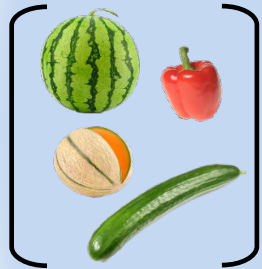
Fast neutron TILLING project



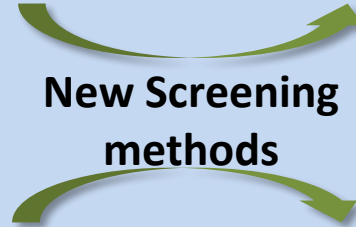
+



Fast Neutron mutagenesis



First FN
population on
tomato



SMALL INDEL

< 500pb

BIG INDEL

> 1 kb



PCR-based
libraries sequencing

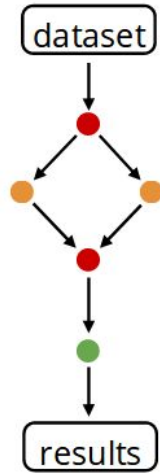


ddPCR screens

Data analysis quality challenges

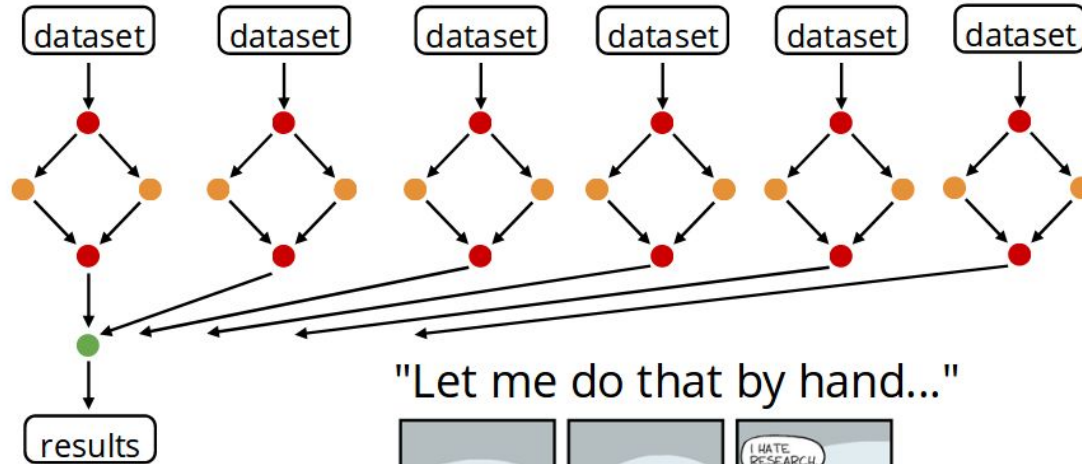
Data analysis usually entail the application of many command line tools/scripts to transform, filter, aggregate or plot data and results.

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



"Let me do that by hand..."

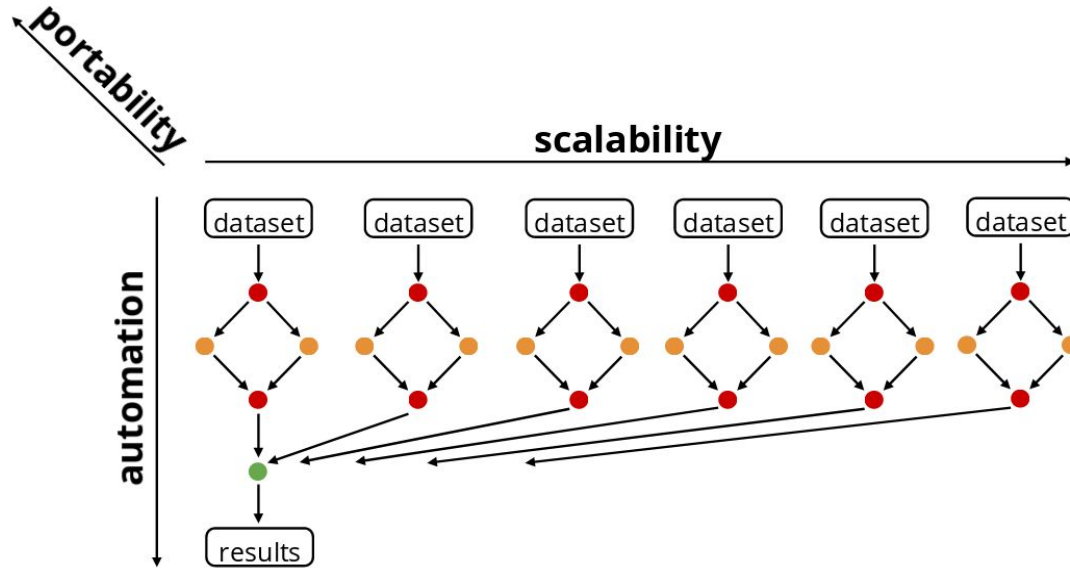




"Let me do that by hand..."



Reproducibility challenges



1. Automation

From raw data to final figures:

- **document** parameters, tools, versions
- **execute** without manual intervention

2. Scalability

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

Snakemake for reproducible data analysis

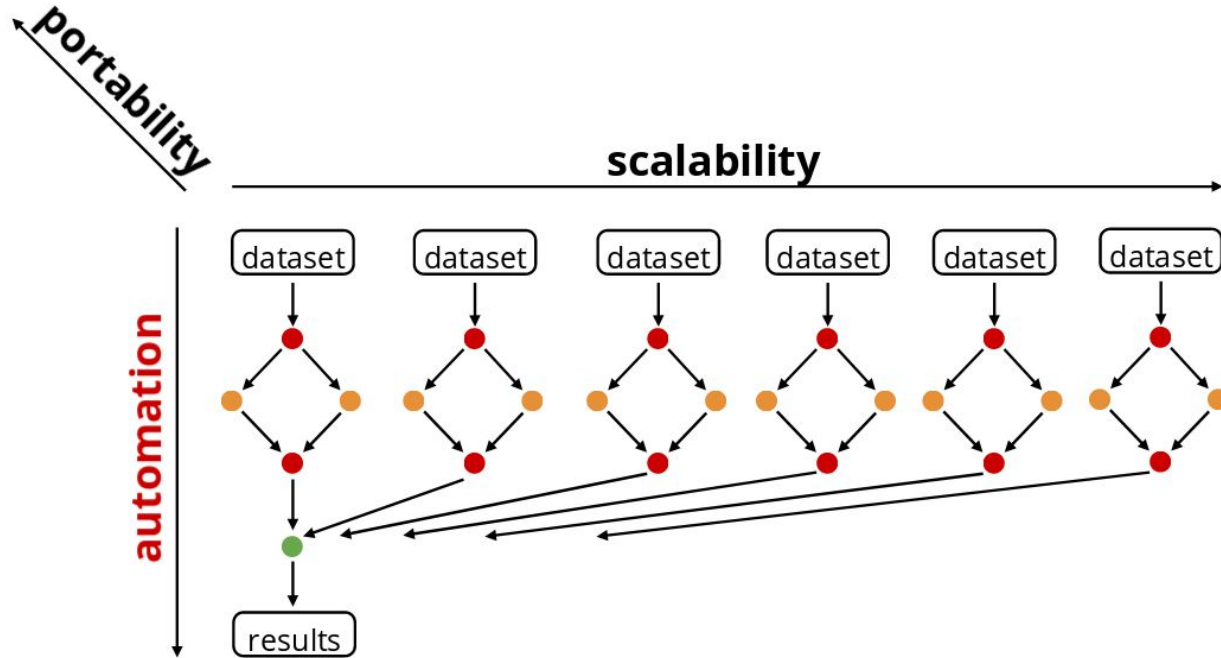
- Decompose analysis into rules, written in a Python dialect.
- Rules define how to obtain output files from input files.
- Snakemake determines dependencies and execution order in the form of a directed acyclic graph (DAG) of jobs.




snakes are very flexible ya know!



Automation: define workflows in terms of rules




Automation: define workflows in terms of rules



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

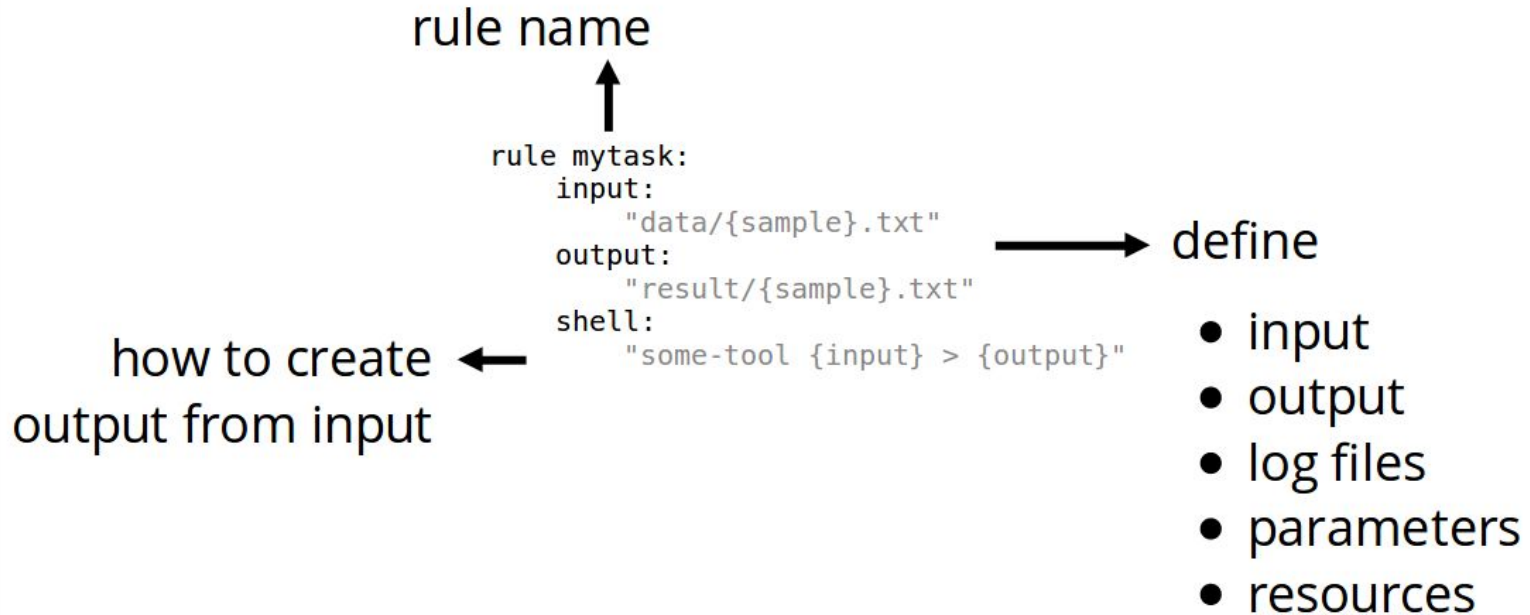


```
rule myfiltration:  
  input:  
    "result/{dataset}.txt"  
  output:  
    "result/{dataset}.filtered.txt"  
  shell:  
    "mycommand {input} > {output}"
```



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

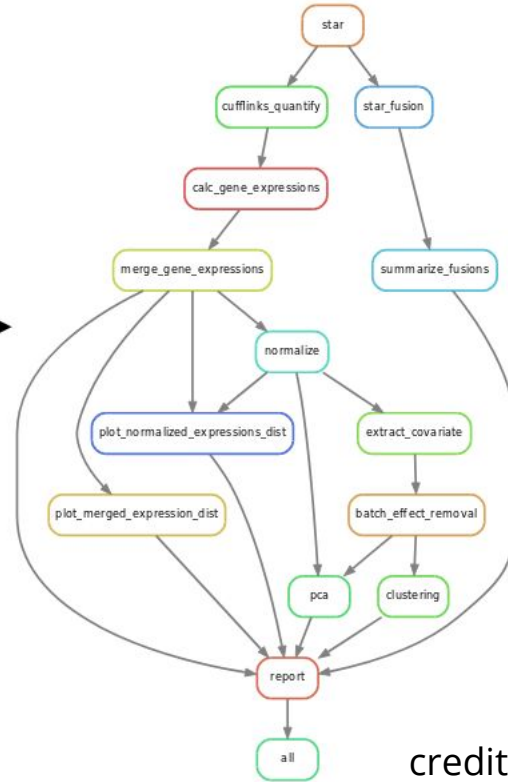
Automation: define workflows in terms of rules



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: 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"
```

reusable ←

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

Snakemake Report Thu Jul 12 13:42:43 2018 CET
Snakemake 5.2.0+3.g4624ffc

[Workflow](#)

Statistics
Rules

RESULTS

Calls

- all.vcf.gz
- calls.tsv.gz

Plots

- allele-freqs.svg
- depths.svg

Quality control

- multiqc.html

Workflow

Variants were called following the [GATK best practices workflow](#): Reads were mapped onto GRCh38.86 with [BWA mem](#), and both optical and PCR duplicates were removed with [Picard](#), followed by base recalibration with [GATK](#). The [GATK HaplotypeCaller](#) was used to call variants per-sample, including summarized evidence for non-variant sites ([GVCF](#) approach). Then, [GATK](#) genotyping was done in a joint way over [GVCF](#) files of all samples. Genotyped variants were filtered using hard thresholds. For SNVs, the criterion $QD < 2.0 \ || \ FS > 60.0 \ || \ MQ < 40.0 \ || \ MQRankSum < -12.5 \ || \ ReadPosRankSum < -8.0$ was used, for Indels the criterion $QD < 2.0 \ || \ FS > 200.0 \ || \ ReadPosRankSum < -20.0$ was used. Finally, [SnPEff](#) was used to predict and report variant effects. In addition, quality control was performed with [FastQC](#), [Samtools](#), and [Picard](#) and aggregated into an interactive report via [MultiQC](#).

Detailed software versions can be found under [Rules](#).

```
graph TD; trim_reads_pe --> map_reads; trim_reads_se --> map_reads; map_reads --> mark_duplicates; mark_duplicates --> recalibrate_base_qualities;
```

multiqc.html

MultiQC
v1.2 (4624816)

General Stats

SnPEff

- Variants by Genomic Region
- Variant Effects by Impact
- Variant Effects by Class
- Variant Qualities

Picard

Samtools

Percent Mapped

Alignment metrics

MultiQC

A modular tool to aggregate results from bioinformatics analyses across many samples into a single report.

Report generated on 2018-07-11, 13:19 based on data in:

- /home/johannes/scs/snake/variant-calling/germline/test/gp/samtools-status
- /home/johannes/scs/snake/variant-calling/germline/test/snpeff

Welcome! Not sure where to start? [Watch a tutorial video](#) (8:06) [Start here again](#)

General Statistics

Copy table | Configure Columns | Plot | Showing % rows and % columns.

Sample Name	Change rate	Ts/Tv	M Variants	% Dups	Error rate	M Non-Primary	M Reads Mapped	% Mapped	M Totl
A-1			3.6%	0.81%	0.0	0.0	95.6%	0.0	
B-1			3.0%	0.81%	0.0	0.0	95.7%	0.0	
B-2			7.1%	0.59%	0.0	0.0	97.4%	0.0	
all	99.925	1.578	0.00						

SnPEff

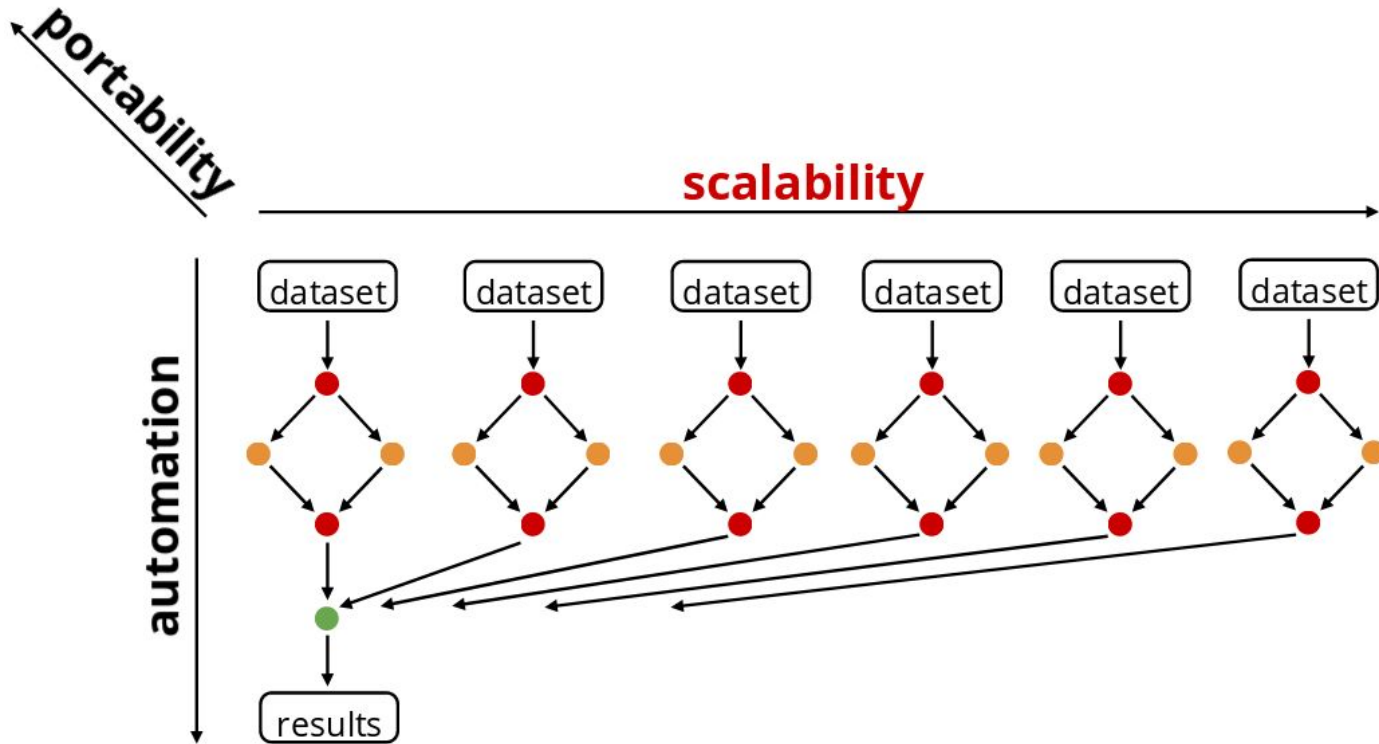
SnPEff is a genetic variant annotation and effect prediction toolbox. It annotates and predicts the effects of variants on genes (such as amino acid changes).

Variants by Genomic Region Help

Counts | Percentages | Log10

SnPEff: Counts by Genomic Region

Scalability: maximize parallelization



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



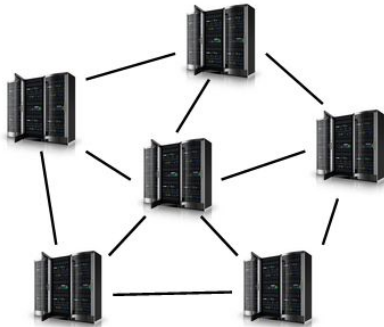
compute server



cluster



grid computing



cloud computing



Google Cloud Platform



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:

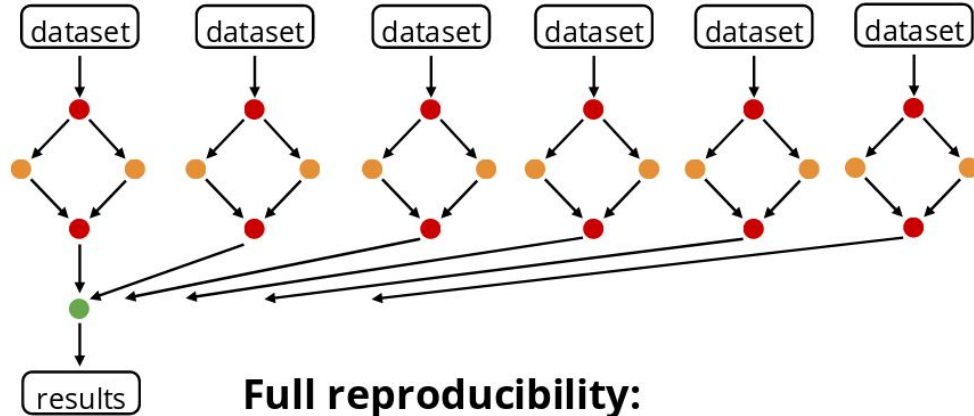
- Output files can be marked as pipes.
- Consuming jobs will be assigned to the same group.
- Output will not be written to disk but streamed between the jobs.

Portability

portability

scalability

automation



Full reproducibility:

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

Portability: use package management with



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
```



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.

Portability: Singularity + Conda

```
define OS ← singularity: "docker://continuumio/miniconda3:4.4.1"

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

define tools/libs ←
```

Snakemake evaluation on EPITRANS platform

- Very very positive
- python
- reusable (modularization capabilities)
- great doc
- great test/reporting capabilities
- quick and 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

