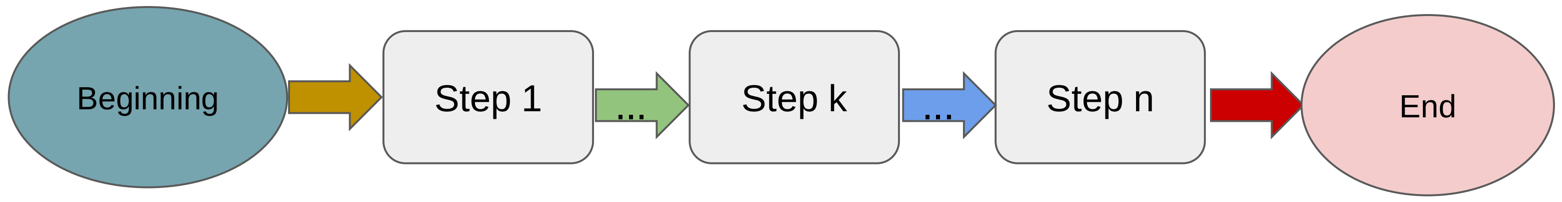
Creating *reproducible, reusable, and scalable* bioinformatics workflows using Snakemake

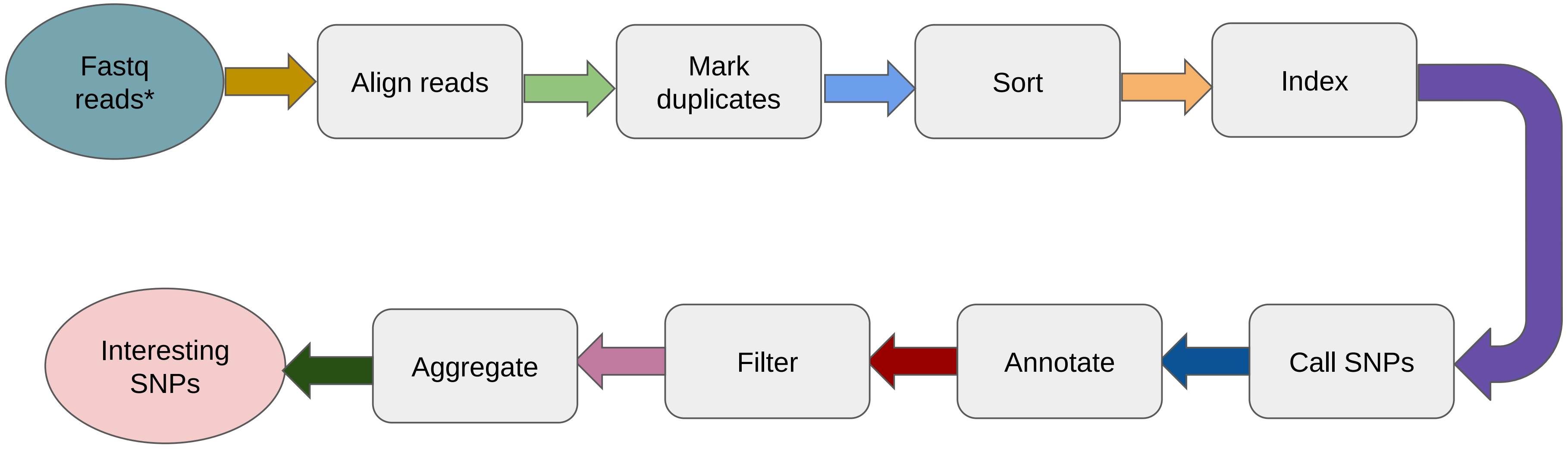
Ebrahim Afyounian
Tampere - 09.09.19

What is a workflow?

Workflow [\ ˈwərk-ˌflō]. the **sequence of steps** involved in moving from the **beginning** to the **end** of a working process

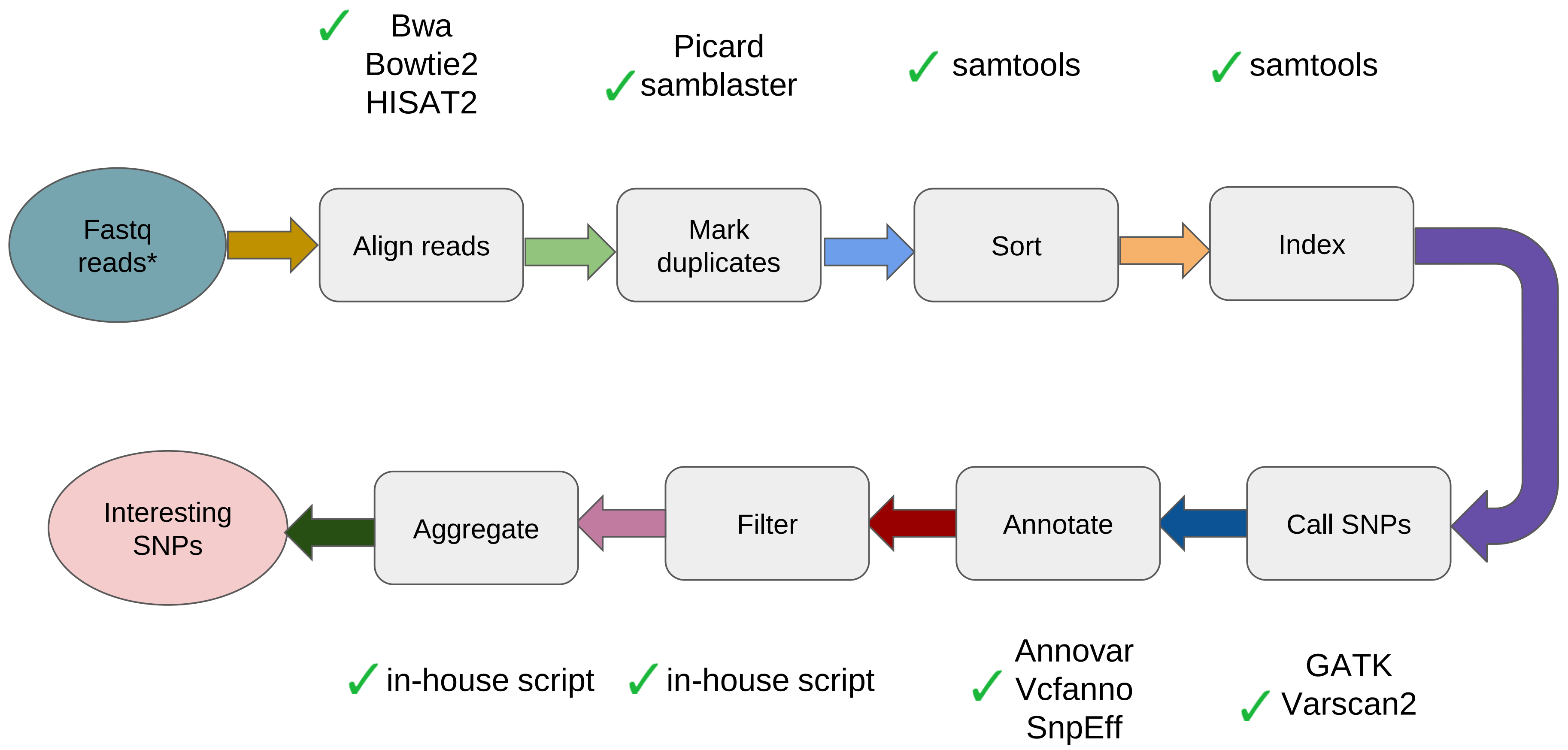


A mock example of a bioinformatics workflow

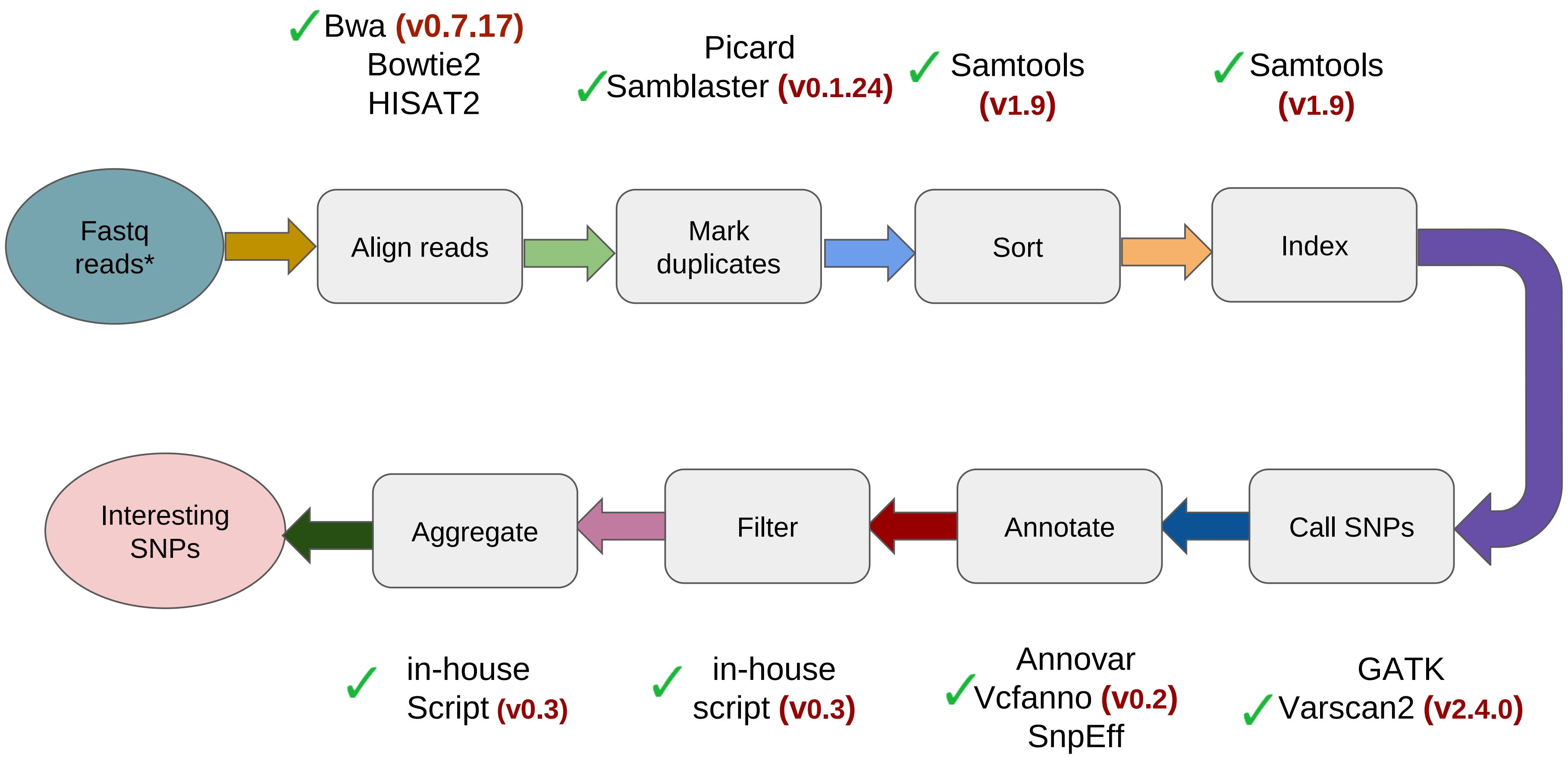


* Let's, for now, assume that we have done quality control (QC) over our fastq files, and we are happy with them.

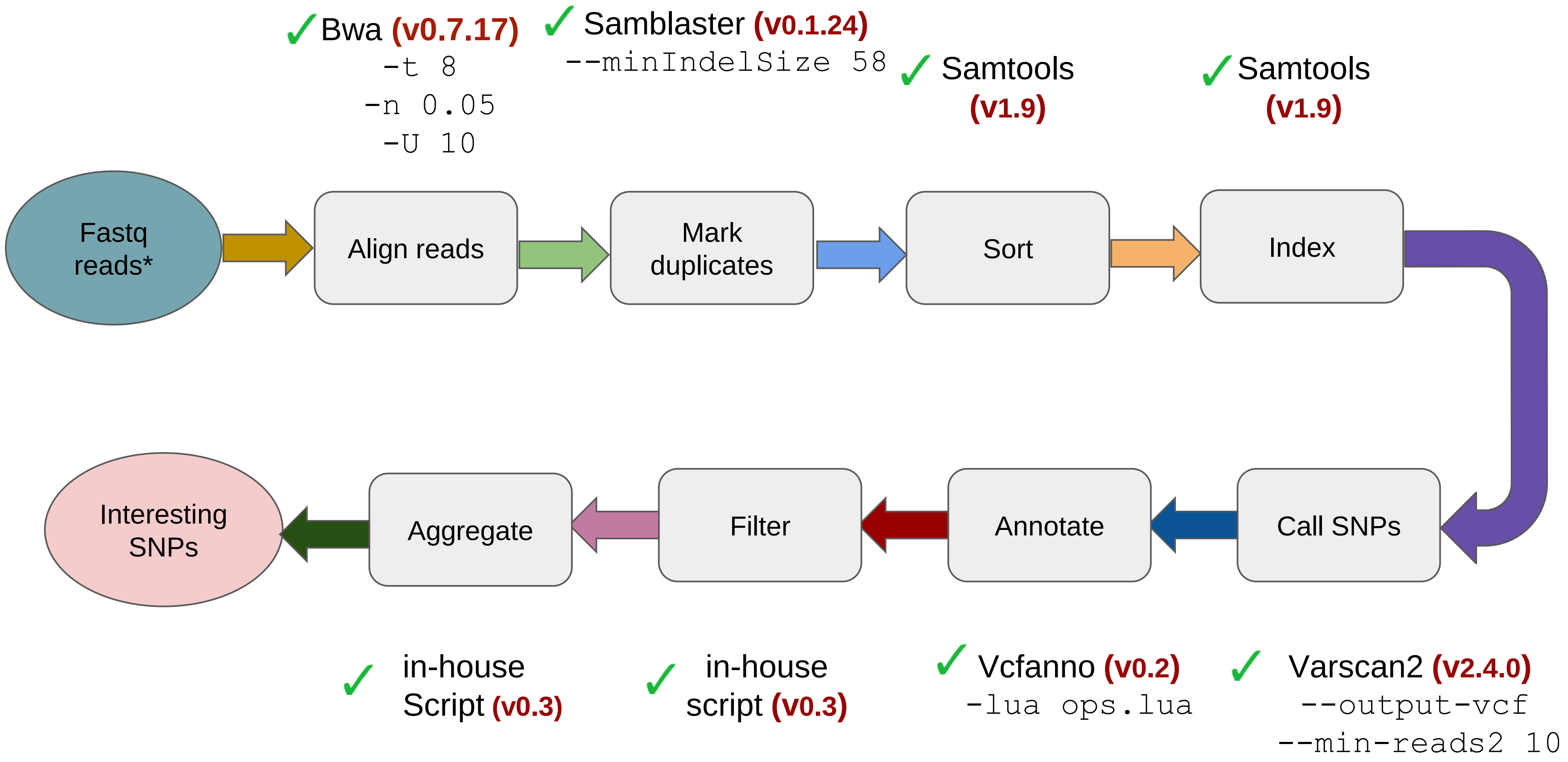
A mock example of a bioinformatics workflow



A mock example of a bioinformatics workflow



A mock example of a bioinformatics workflow



Scenario 1.

Let's imagine that our workflow works perfectly on one sample and:

- We receive 99 samples (198 fastq files).
- We are told that we have a limited amount of time (preferably ASAP) to find the *interesting SNPs* from these samples. In addition, we are required to provide our results with a report.

Option: Run our workflow in a ***for-loop*** and run one sample at a time.

Problem: what if something happens after processing *x* samples?

- ■ Should we rerun every from scratch?
- ■ Should we modify our for-loop so that it runs over only the unprocessed samples?

Problem: what if it will never finish by the deadline?

Option: Submit 99 jobs (1 per sample) to the job scheduler (e.g. to slurm)

Problem: what if an step in our workflow requires lots of *computational resources*. In such case, we need to request resources that are as big as the maximum amount required for that step, for the whole duration of the workflow's execution → We might end up **queueing for a long time** before our job starts. After, others have to queue until our jobs end!

Scenario 2.

Let's imagine that we survived the 1st scenario ***but****:*

Based on the results, our supervisor(s) decide(s) that: we should use stricter criteria for calling the SNPs (e.g. we need to change the parameter value for `--min-reads2` in varscan2 from `10` to `20`). ***And while we are at it,*** let's also add *gnomAD* annotation to our SNPs. ***By the way,*** we need the results for **tomorrow afternoon's meeting** with the people from the clinic *[... **and pliiis** bring some sweets to the meeting ...]*.

Question: Should we update the criteria and rerun our entire workflow from scratch?

✗ We know that simply our workflow won't finish in less than a day!

Question: Or should we modify our workflow so that it runs only from `call SNPs` step and possibly save time and the resources?

✗ Possible, but then we have broken our workflow into smaller pieces i.e. we have one workflow for scenario 1 and another for scenario 2.

Bioinformatics workflow management systems (WMS)

Snakemake

nextflow

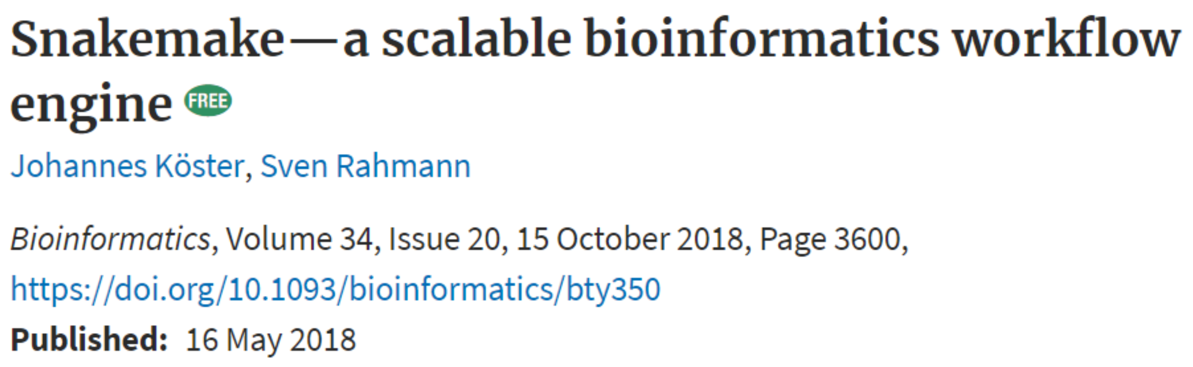
GenePattern

TEHTY SUOMESSA
MADE IN FINLAND

ANDURIL
WORKFLOW PLATFORM

Galaxy
PROJECT

Anduril is from *Systems Biology Laboratory*, University of Helsinki



Snakemake—a scalable bioinformatics workflow engine FREE

Johannes Köster, Sven Rahmann

Bioinformatics, Volume 34, Issue 20, 15 October 2018, Page 3600, https://doi.org/10.1093/bioinformatics/bty350

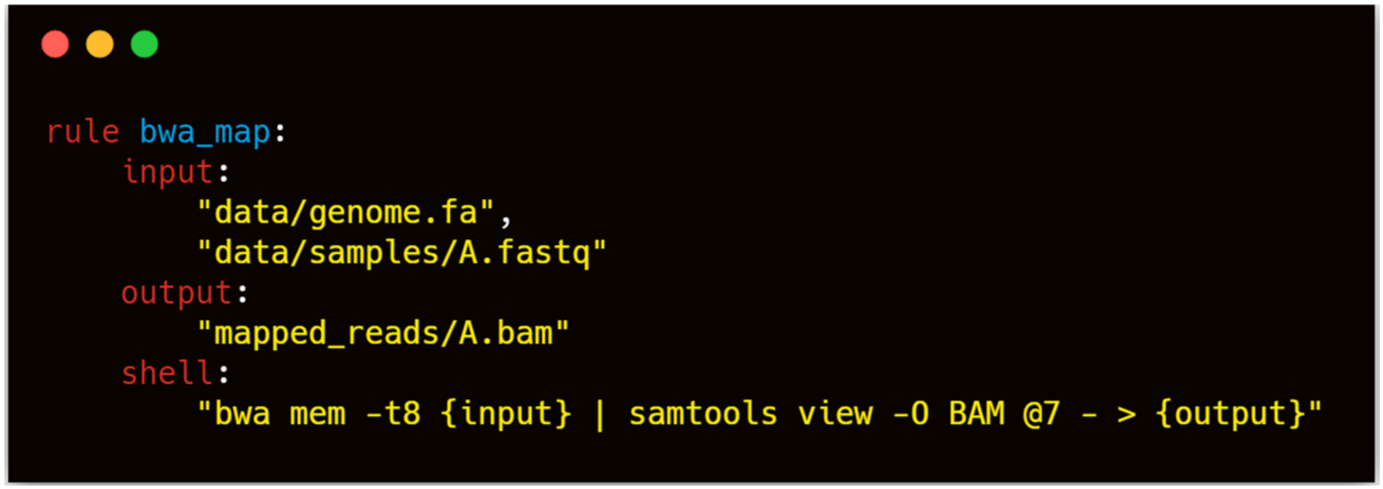
Published: 16 May 2018

- A *text-based* workflow management system introduced in 2012
- Inherits many of its features from `GNU make`.
- Implements a domain-specific language (an extension of Python) with which we define/formalize the steps in our data analysis as a workflow

- A workflow is composed of **rules** i.e. Each rule is one step in the data analysis / workflow
- Each rule has ***input***, ***output***, and ***the computation that turns the input to output***
- Having rules makes the workflow **modular**.
  - ➕ It can be **reused** in another workflow
  - ➕ We can request computational resources for each rule **separately**

- Checks the dependencies among the rules (e.g. ***rule 2*** needs ***rule 1***'s result as input) and infers a dependency graph and execution order
- Decides what steps need to be done in the analysis ensuring that we do/re-do only what is needed
- If possible, rules can be executed parallelly.

Basic anatomy of a rule

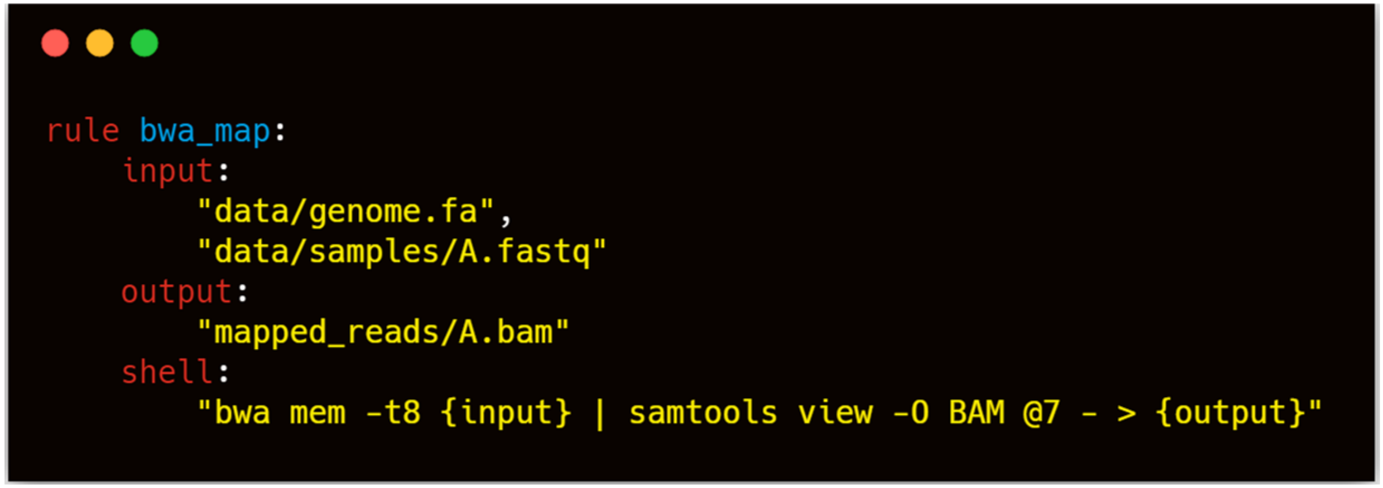


```
rule bwa_map:
    input:
        "data/genome.fa",
        "data/samples/A.fastq"
    output:
        "mapped_reads/A.bam"
    shell:
        "bwa mem -t8 {input} | samtools view -O BAM @7 - > {output}"
```

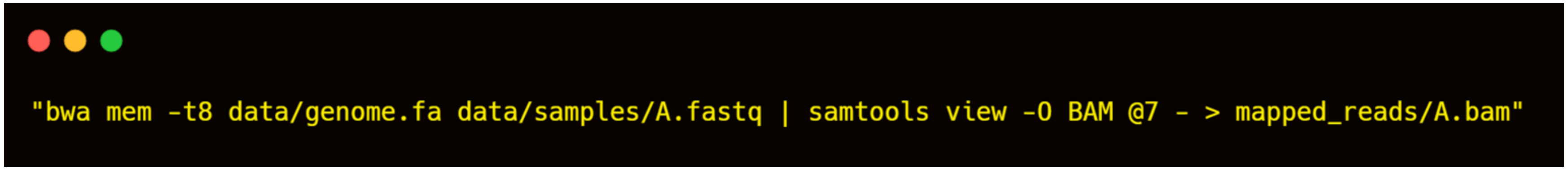
- `input`, `output`, and `shell` are called **directives**

- There are few more very useful directive:
  - `script:` used to run scripts such as Python and R
  - `conda:` used to manage environments and packages
  - `thread:` used to specify number of threads
  - `log:` used to specify where the log info is written

What happens under the hood?



```
rule bwa_map:
    input:
        "data/genome.fa",
        "data/samples/A.fastq"
    output:
        "mapped_reads/A.bam"
    shell:
        "bwa mem -t8 {input} | samtools view -O BAM @7 - > {output}"
```



```
"bwa mem -t8 data/genome.fa data/samples/A.fastq | samtools view -O BAM @7 - > mapped_reads/A.bam"
```

Some issues to be addressed:

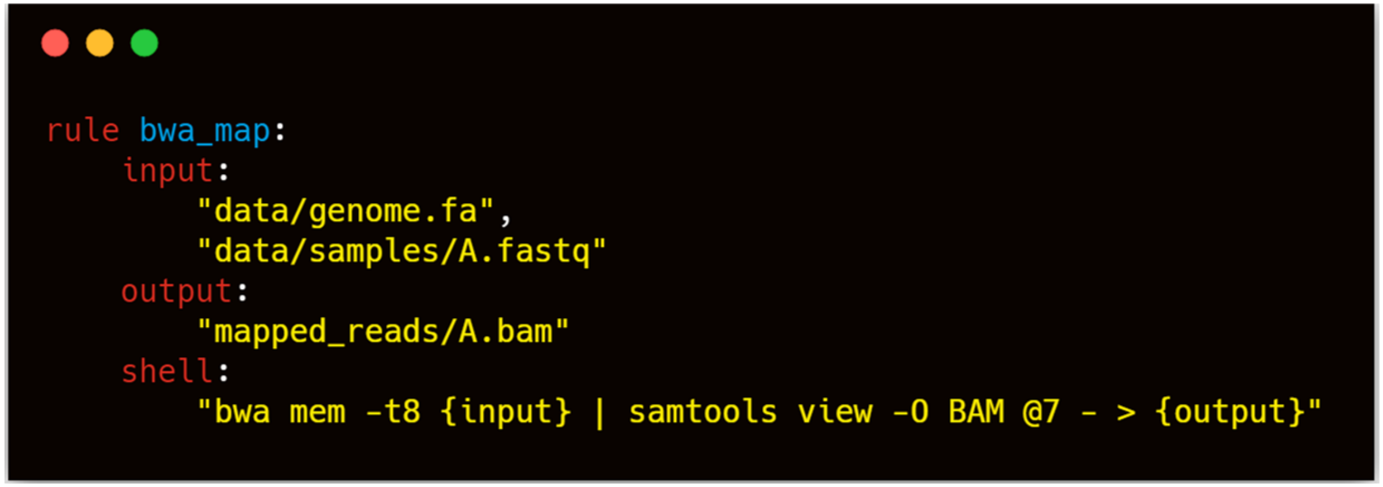
- Not scalable

  Hard-coding the sample name(s)

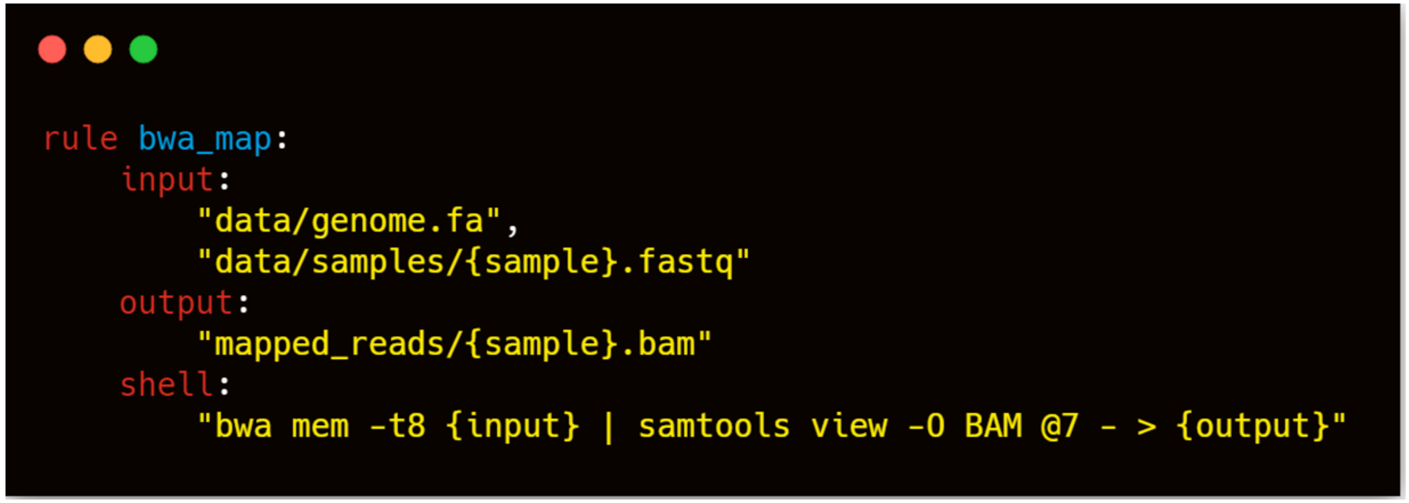
- Not necessarily reusable:

  It assumes that `bwa` and `samtools` are installed on the system (which versions we do not know)?

Addressing some of the issues (1)



```
rule bwa_map:
    input:
        "data/genome.fa",
        "data/samples/A.fastq"
    output:
        "mapped_reads/A.bam"
    shell:
        "bwa mem -t8 {input} | samtools view -O BAM @7 - > {output}"
```



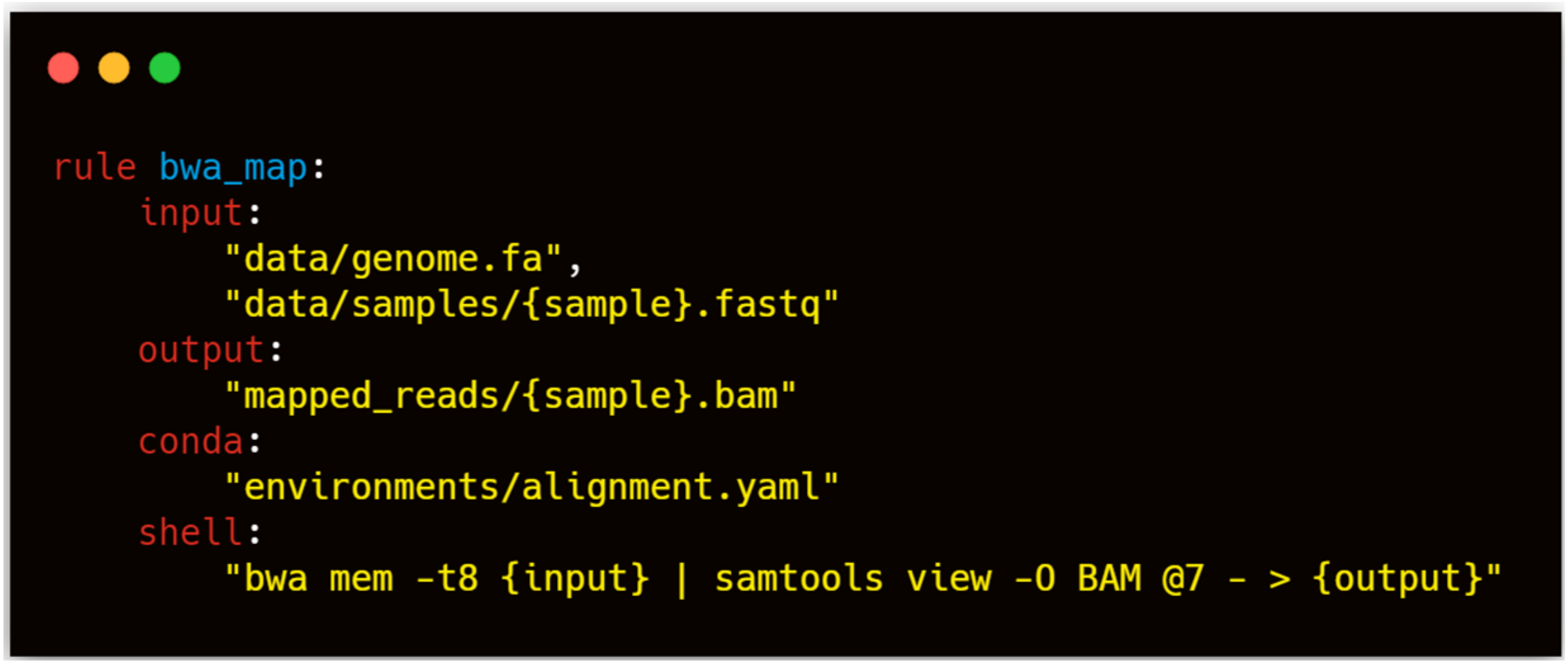
```
rule bwa_map:
    input:
        "data/genome.fa",
        "data/samples/{sample}.fastq"
    output:
        "mapped_reads/{sample}.bam"
    shell:
        "bwa mem -t8 {input} | samtools view -O BAM @7 - > {output}"
```

Better! But not quite!

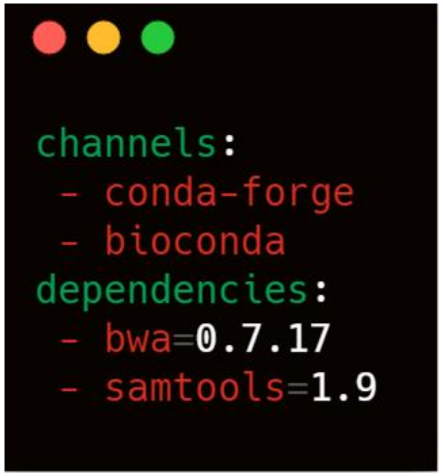
No hard-coded samples

Reference is hard-coded

Addressing some of the issues (2)



```
rule bwa_map:
    input:
        "data/genome.fa",
        "data/samples/{sample}.fastq"
    output:
        "mapped_reads/{sample}.bam"
    conda:
        "environments/alignment.yaml"
    shell:
        "bwa mem -t8 {input} | samtools view -O BAM @7 - > {output}"
```



```
channels:
 - conda-forge
 - bioconda
dependencies:
 - bwa=0.7.17
 - samtools=1.9
```

- No hard-coded samples
- Uses `conda*` to create an environment for the rule
- Installs required tools in the environment.

⚠ Information about where the sample(s) are still missing.

* Conda is an open source **environment management** and **package management** system

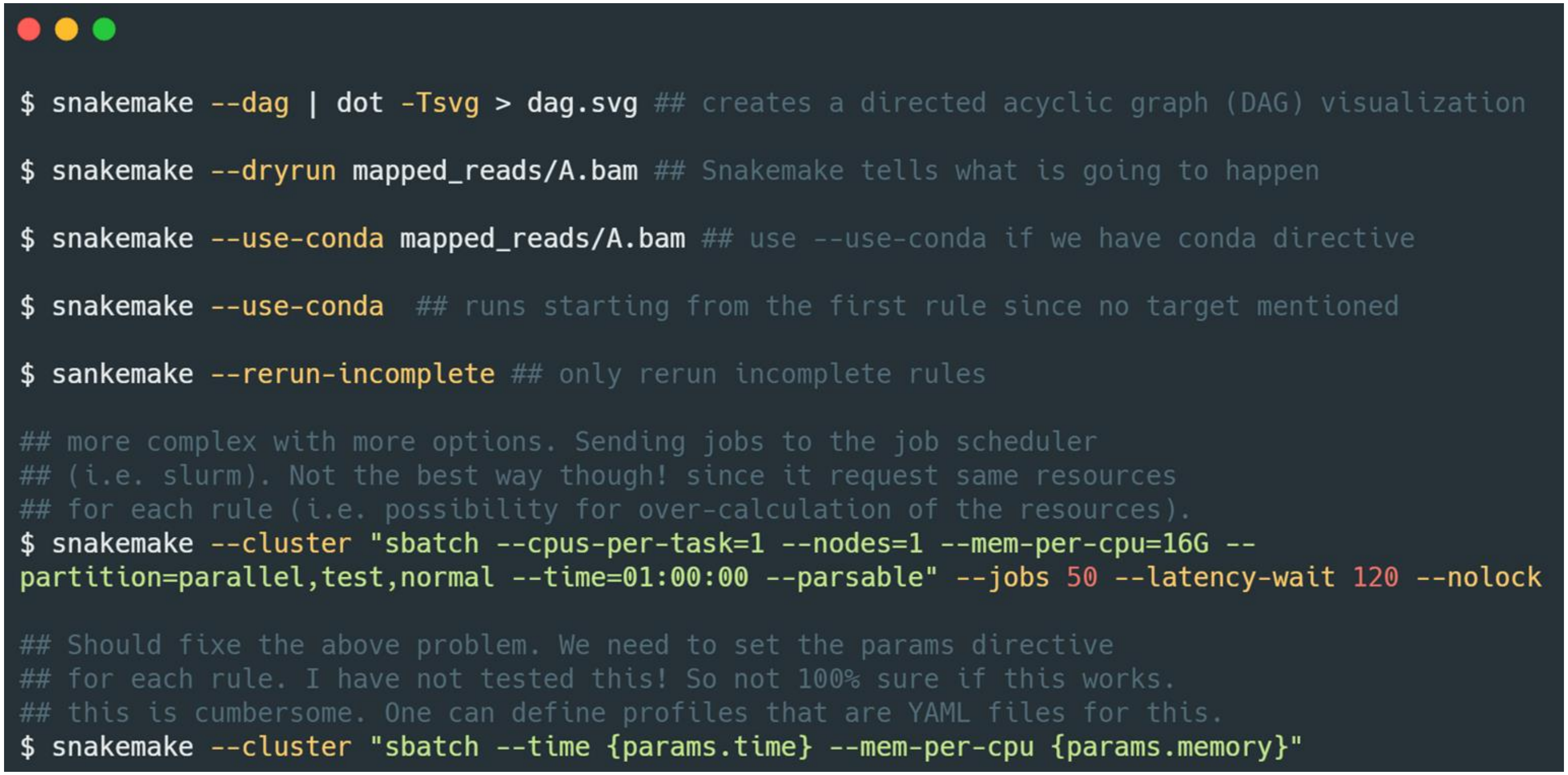
Running a Snakemake workflow

- Snakemake workflow can be run on:
  Desktop machine; Server / Cluster; Cloud

- When running a workflow, we need to specify a **target** i.e. the *output(s)* we want to have.

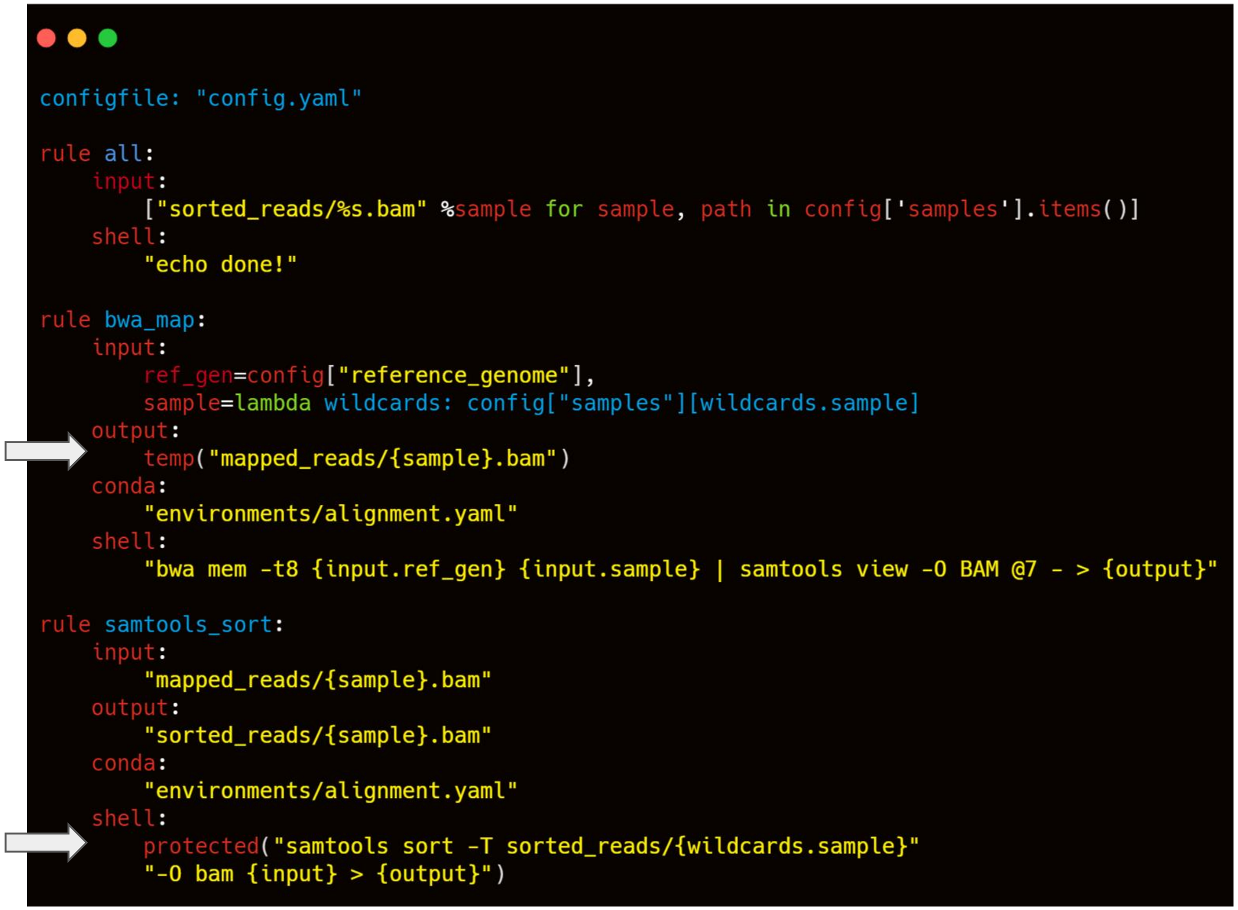
- If a **target** is not provided, Snakemake runs the **first rule** by default as well as any other rule needed for successful completion of the first rule.

- In practice, the first rule is the “rule all”.



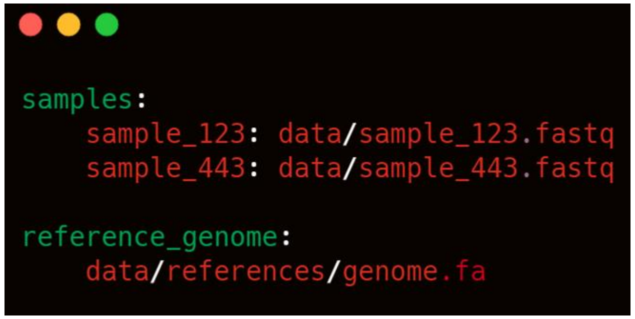
```
$ snakemake --dag | dot -Tsvg > dag.svg ## creates a directed acyclic graph (DAG) visualization

$ snakemake --dryrun mapped_reads/A.bam ## Snakemake tells what is going to happen

$ snakemake --use-conda mapped_reads/A.bam ## use --use-conda if we have conda directive

$ snakemake --use-conda  ## runs starting from the first rule since no target mentioned

$ sankemake --rerun-incomplete ## only rerun incomplete rules

## more complex with more options. Sending jobs to the job scheduler
## (i.e. slurm). Not the best way though! since it request same resources
## for each rule (i.e. possibility for over-calculation of the resources).
$ snakemake --cluster "sbatch --cpus-per-task=1 --nodes=1 --mem-per-cpu=16G --
partition=parallel,test,normal --time=01:00:00 --parsable" --jobs 50 --latency-wait 120 --nolock

## Should fixe the above problem. We need to set the params directive
## for each rule. I have not tested this! So not 100% sure if this works.
## this is cumbersome. One can define profiles that are YAML files for this.
$ snakemake --cluster "sbatch --time {params.time} --mem-per-cpu {params.memory}"
```

An example workflow



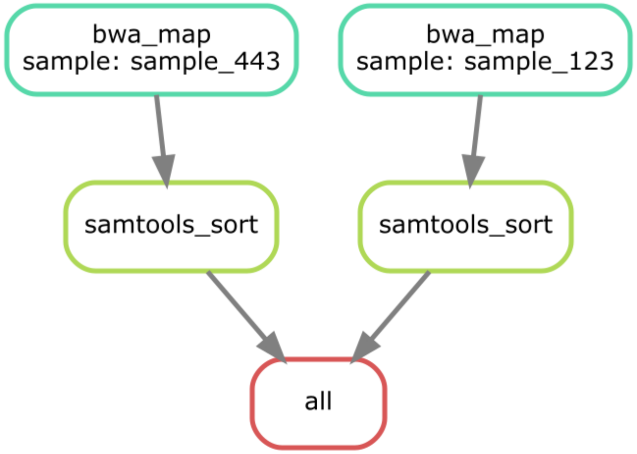
```
configfile: "config.yaml"

rule all:
    input:
        ["sorted_reads/%s.bam" %sample for sample, path in config['samples'].items()]
    shell:
        "echo done!"

rule bwa_map:
    input:
        ref_gen=config["reference_genome"],
        sample=lambda wildcards: config["samples"][wildcards.sample]
    output:
        temp("mapped_reads/{sample}.bam")
    conda:
        "environments/alignment.yaml"
    shell:
        "bwa mem -t8 {input.ref_gen} {input.sample} | samtools view -O BAM @7 - > {output}"

rule samtools_sort:
    input:
        "mapped_reads/{sample}.bam"
    output:
        "sorted_reads/{sample}.bam"
    conda:
        "environments/alignment.yaml"
    shell:
        protected("samtools sort -T sorted_reads/{wildcards.sample}"
        "-O bam {input} > {output}")
```

Config file

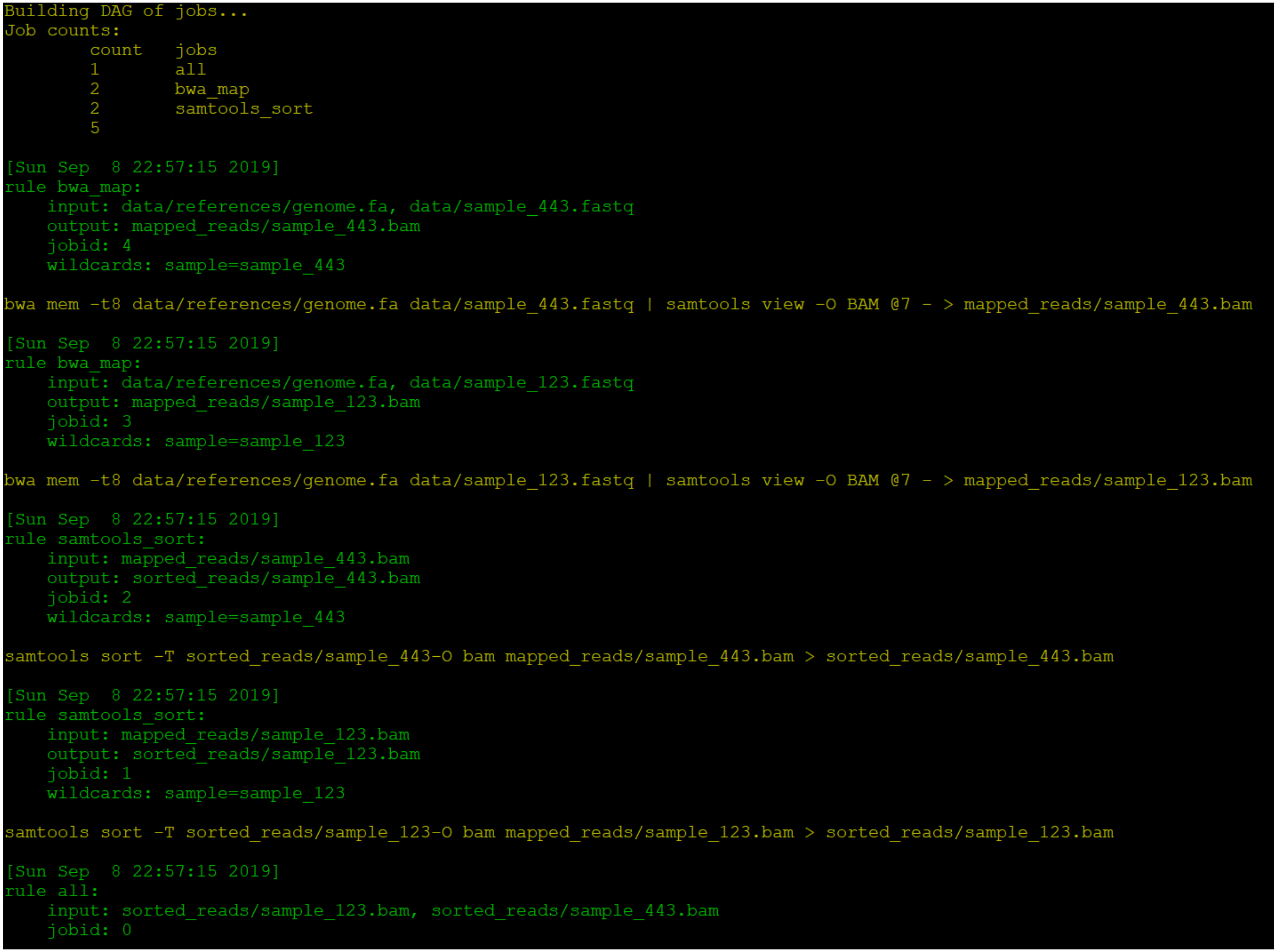


```
samples:
    sample_123: data/sample_123.fastq
    sample_443: data/sample_443.fastq

reference_genome:
    data/references/genome.fa
```

Dependency graph



Dry run output



```
Building DAG of jobs...
Job counts:
        count   jobs
        1       all
        2       bwa_map
        2       samtools_sort
        5

[Sun Sep  8 22:57:15 2019]
rule bwa_map:
    input: data/references/genome.fa, data/sample_443.fastq
    output: mapped_reads/sample_443.bam
    jobid: 4
    wildcards: sample=sample_443

bwa mem -t8 data/references/genome.fa data/sample_443.fastq | samtools view -O BAM @7 - > mapped_reads/sample_443.bam

[Sun Sep  8 22:57:15 2019]
rule bwa_map:
    input: data/references/genome.fa, data/sample_123.fastq
    output: mapped_reads/sample_123.bam
    jobid: 3
    wildcards: sample=sample_123

bwa mem -t8 data/references/genome.fa data/sample_123.fastq | samtools view -O BAM @7 - > mapped_reads/sample_123.bam

[Sun Sep  8 22:57:15 2019]
rule samtools_sort:
    input: mapped_reads/sample_443.bam
    output: sorted_reads/sample_443.bam
    jobid: 2
    wildcards: sample=sample_443

samtools sort -T sorted_reads/sample_443-O bam mapped_reads/sample_443.bam > sorted_reads/sample_443.bam

[Sun Sep  8 22:57:15 2019]
rule samtools_sort:
    input: mapped_reads/sample_123.bam
    output: sorted_reads/sample_123.bam
    jobid: 1
    wildcards: sample=sample_123

samtools sort -T sorted_reads/sample_123-O bam mapped_reads/sample_123.bam > sorted_reads/sample_123.bam

[Sun Sep  8 22:57:15 2019]
rule all:
    input: sorted_reads/sample_123.bam, sorted_reads/sample_443.bam
    jobid: 0
```

Few of the features not presented here

- Can create HTML reports showing e.g. the rules' run time, files creation time, even results / visualizations from the rules…

  `--report`

- Specify that a rule output should be *piped* into another rule preventing writing to disk

  `pipe()`

- `resources` directive can be used to set the resources needed to run a job

- Ability to run in containers (e.g. Docker, Singularity)

- And many more ...

- **Upcoming feature(s):**
  - Integration with ***Jupyter notebook*** (i.e. jupyter directive)

More resources

- Snakemake homepage:
  https://snakemake.readthedocs.io

- Snakemake tutorial:
  https://snakemake.readthedocs.io/en/stable/tutorial/tutorial.html

- Bioinformatics workflows using Snakemake (e.g. DNA-seq GATK variant calling, Single-cell RNA-seq analysis):
  https://github.com/snakemake-workflows/docs

- Collection of reusable wrappers allowing use of popular tools (such as `fatsqc`, `samtools`, `bwa`, etc.) from Snakemake rules and workflows:
  https://snakemake-wrappers.readthedocs.io/en/stable/

- Live demo created by the Johannes Köster:
  https://www.katacoda.com/johanneskoester/scenarios/snakemake-intro

  ⚠ No need to use your real email address to run the live demo. Just type a dummy email address and password in the sign-up form and continue.

- Nice presentation by Johannes Köster:
  https://youtu.be/hPrXcUUp70Yp70Y