News about Snakemake

Johannes Köster

February 5, 2013

Motivation

gatk

bwa

samtools

```
refs = defaultdict(list)
with FastaReader(input.fasta) as fr:
    for record in fr:
        name = record.name.split(None, 1)[0]
        namelist = refs[record.sequence]
        if name not in namelist:
            namelist.append(name)
with open(output.out, 'wt') as out:
    for seq, names in refs.items():
        seq = seq.replace('U', 'T')
        print('>{}\n{}'.format(
            '___'.join(names), seq), file=out)
```

Motivation

bwa

↓

samtools

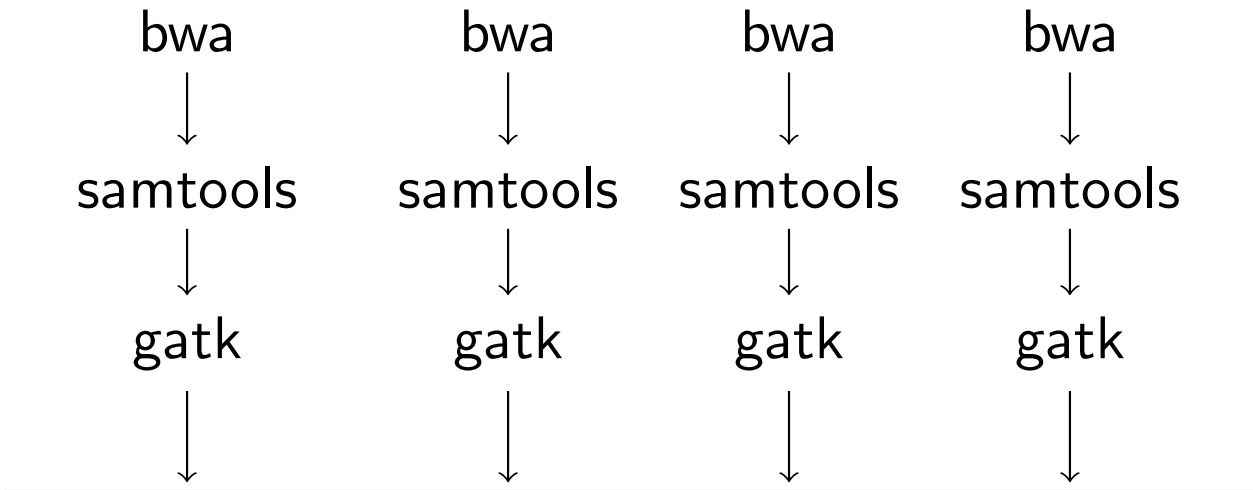
↓

gatk

↓

```
refs = defaultdict(list)
with FastaReader(input.fasta) as fr:
    for record in fr:
        name = record.name.split(None, 1)[0]
        namelist = refs[record.sequence]
        if name not in namelist:
            namelist.append(name)
with open(output.out, 'wt') as out:
    for seq, names in refs.items():
        seq = seq.replace('U', 'T')
        print('>{}\n{}'.format(
            '___'.join(names), seq), file=out)
```

Motivation



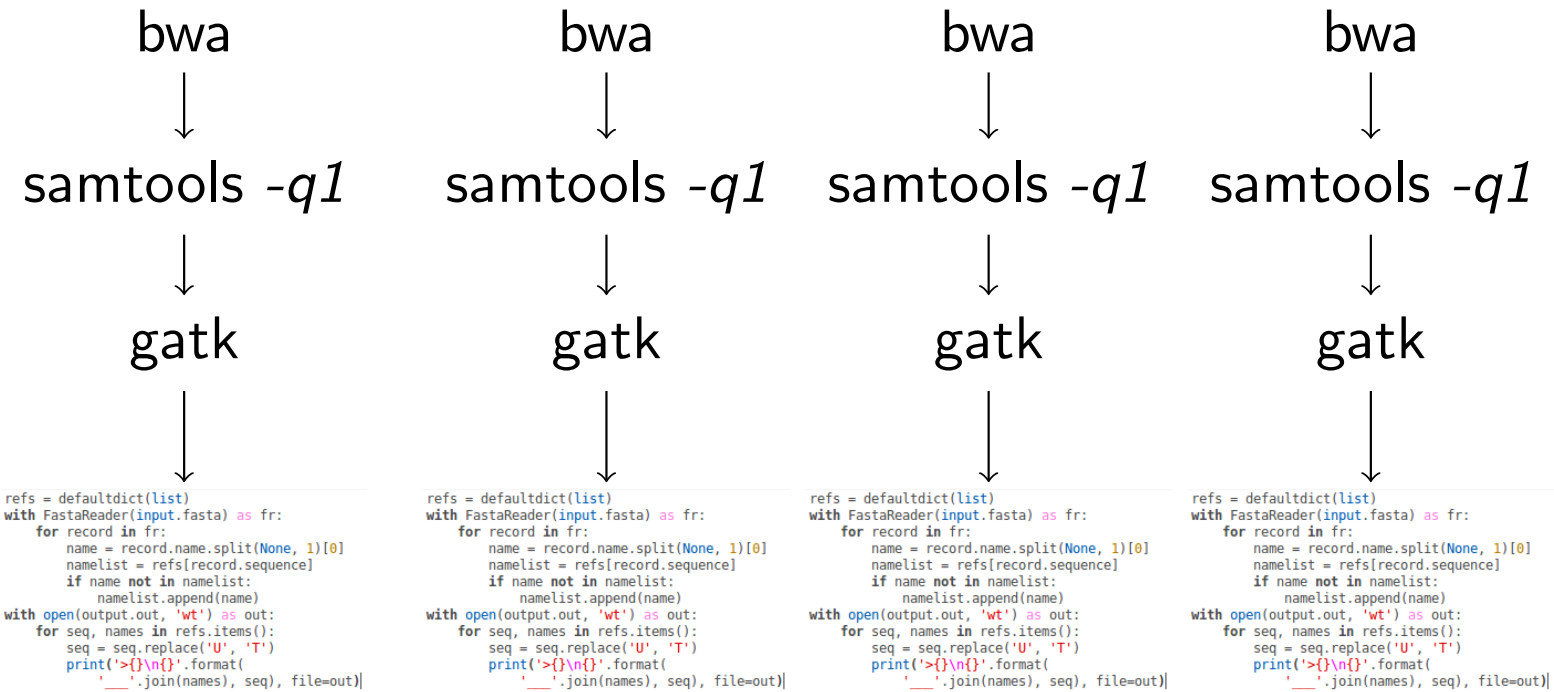

```
refs = defaultdict(list)
with FastaReader(input.fasta) as fr:
    for record in fr:
        name = record.name.split(None, 1)
        namelist = refs[record.sequence]
        if name not in namelist:
            namelist.append(name)
with open(output.out, 'wt') as out:
    for seq, names in refs.items():
        seq = seq.replace('U', 'T')
        print('>{}\n{}'.format(
            '___'.join(names), seq), file
```

```
refs = defaultdict(list)
with FastaReader(input.fasta) as fr:
    for record in fr:
        name = record.name.split(Non
        namelist = refs[record.seque
        if name not in namelist:
            namelist.append(name)
with open(output.out, 'wt') as out:
    for seq, names in refs.items():
        seq = seq.replace('U', 'T')
        print('>{}\n{}'.format(
            '___'.join(names), seq),
```

```
refs = defaultdict(list)
with FastaReader(input.fasta) as fr:
    for record in fr:
        name = record.name.split(Non
        namelist = refs[record.seque
        if name not in namelist:
            namelist.append(name)
with open(output.out, 'wt') as out:
    for seq, names in refs.items():
        seq = seq.replace('U', 'T')
        print('>{}\n{}'.format(
            '___'.join(names), seq),
```

```
refs = defaultdict(list)
with FastaReader(input.fasta) as fr:
    for record in fr:
        name = record.name.split(None, 1)[0]
        namelist = refs[record.sequence]
        if name not in namelist:
            namelist.append(name)
with open(output.out, 'wt') as out:
    for seq, names in refs.items():
        seq = seq.replace('U', 'T')
        print('>{}\n{}'.format(
            '___'.join(names), seq), file=out)
```

Motivation




```
refs = defaultdict(list)
with FastaReader(input.fasta) as fr:
    for record in fr:
        name = record.name.split(None, 1)[0]
        namelist = refs[record.sequence]
        if name not in namelist:
            namelist.append(name)
with open(output.out, 'wt') as out:
    for seq, names in refs.items():
        seq = seq.replace('U', 'T')
        print('>{}\n{}'.format(
            '___'.join(names), seq), file=out)
```

Why Snakemake?

GNU Make provided us with...

- a language to write rules to create each output file from input files
- wildcards for generalization
- implicit dependency resolution
- implicit parallelization
- fast and collaborative development on text files

Why Snakemake?

GNU Make provided us with...

- a language to write rules to create each output file from input files
- wildcards for generalization
- implicit dependency resolution
- implicit parallelization
- fast and collaborative development on text files

but we missed...

- easy to read syntax
- simple scripting inside the workflow
- creating more than one output file with a rule
- multiple wildcards in filenames

Structure

Snakemake Idea

Example: for samples $\{500, \ldots, 503\}$ map reads to hg19.

Snakemake Idea

Example: for samples $\{500, \ldots, 503\}$ map reads to hg19.

```
rule map_reads:
    input:  "hg19.fasta", "{sample}.fastq"
    output: "{sample}.sai"
    shell:  "bwa aln {input} > {output}"
```

Snakemake Idea

Example: for samples $\{500, \ldots, 503\}$ map reads to hg19.

```
rule sai_to_bam:
  input:  "hg19.fasta", "{sample}.sai", "{sample}.fastq"
  output: "{sample}.bam"
  shell:
    "bwa samse {input} | samtools view -Sbh - > {output}"

rule map_reads:
  input:  "hg19.fasta", "{sample}.fastq"
  output: "{sample}.sai"
  shell:  "bwa aln {input} > {output}"
```

Snakemake Idea

Example: for samples $\{500, \ldots, 503\}$ map reads to hg19.

```
SAMPLES = ["500", "501", "502", "503"]
rule all:
  input: expand("{sample}.bam", sample=SAMPLES)


rule sai_to_bam:
  input:  "hg19.fasta", "{sample}.sai", "{sample}.fastq"
  output: "{sample}.bam"
  shell:
    "bwa samse {input} | samtools view -Sbh - > {output}"

rule map_reads:
  input:  "hg19.fasta", "{sample}.fastq"
  output: "{sample}.sai"
  shell:  "bwa aln {input} > {output}"
```

Snakemake Idea

Example: for samples $\{500, \ldots, 503\}$ map reads to hg19.

```
SAMPLES = ["500", "501", "502", "503"]
rule all:
  input: expand("{sample}.bam", sample=SAMPLES)


rule sai_to_bam:
  input:  "hg19.fasta", "{sample}.sai", "{sample}.fastq"
  output: protected("{sample}.bam")
  shell:
    "bwa samse {input} | samtools view -Sbh - > {output}"

rule map_reads:
  input:  "hg19.fasta", "{sample}.fastq"
  output: "{sample}.sai"
  shell:  "bwa aln {input} > {output}"
```

Snakemake Idea

Example: for samples $\{500, \ldots, 503\}$ map reads to hg19.

```
SAMPLES = ["500", "501", "502", "503"]
rule all:
  input: expand("{sample}.bam", sample=SAMPLES)


rule sai_to_bam:
  input:  "hg19.fasta", "{sample}.sai", "{sample}.fastq"
  output: protected("{sample}.bam")
  shell:
    "bwa samse {input} | samtools view -Sbh - > {output}"

rule map_reads:
  input:  "hg19.fasta", "{sample}.fastq"
  output: temp("{sample}.sai")
  shell:  "bwa aln {input} > {output}"
```

Snakemake Idea

Example: for samples $\{500, \ldots, 503\}$ map reads to hg19.

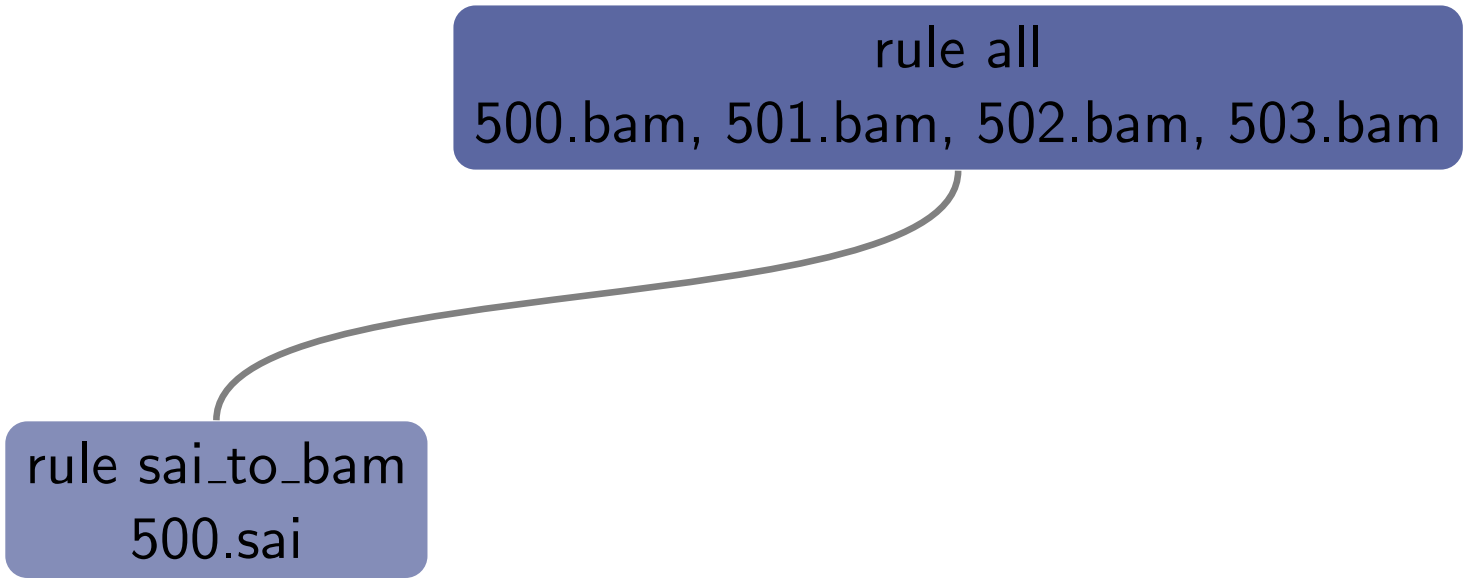
rule all
500.bam, 501.bam, 502.bam, 503.bam

```
rule sai_to_bam:
  input:  "hg19.fasta", "{sample}.sai", "{sample}.fastq"
  output: protected("{sample}.bam")
  shell:
    "bwa samse {input} | samtools view -Sbh - > {output}"

rule map_reads:
  input:  "hg19.fasta", "{sample}.fastq"
  output: temp("{sample}.sai")
  shell:  "bwa aln {input} > {output}"
```

Snakemake Idea

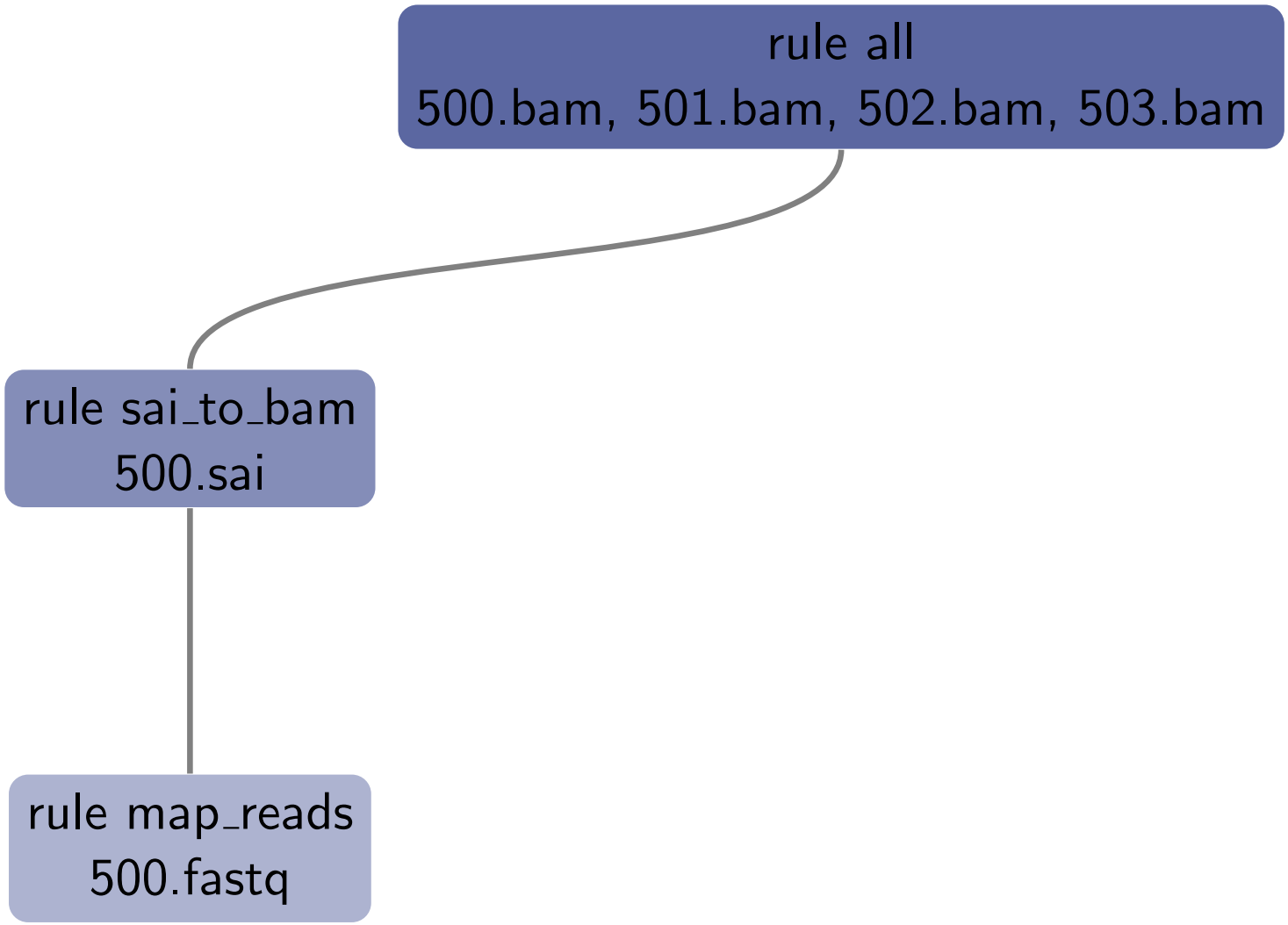
Example: for samples $\{500, \ldots, 503\}$ map reads to hg19.




```
rule map_reads:
    input:  "hg19.fasta", "{sample}.fastq"
    output: temp("{sample}.sai")
    shell:  "bwa aln {input} > {output}"
```

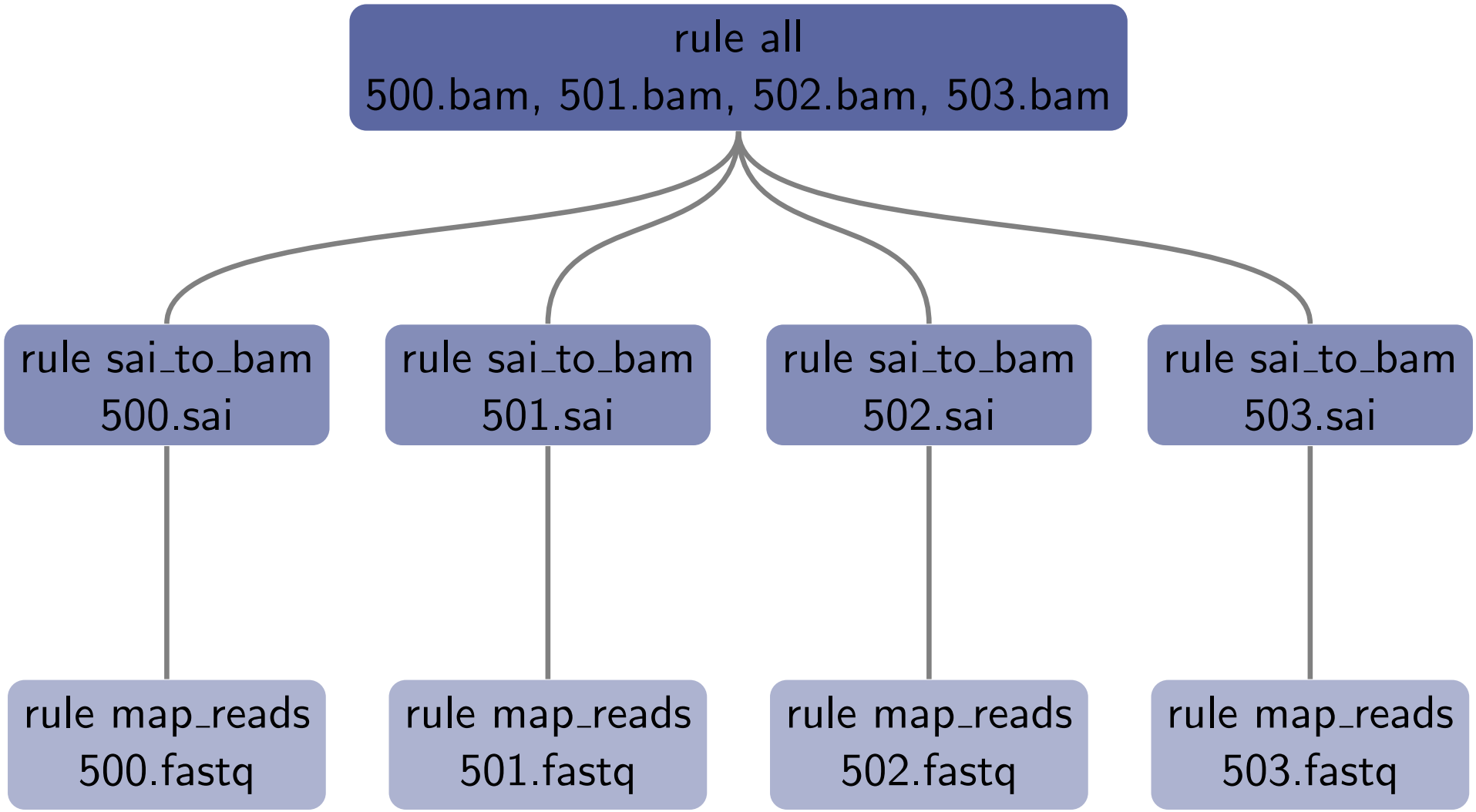
Snakemake Idea

Example: for samples $\{500, \ldots, 503\}$ map reads to hg19.

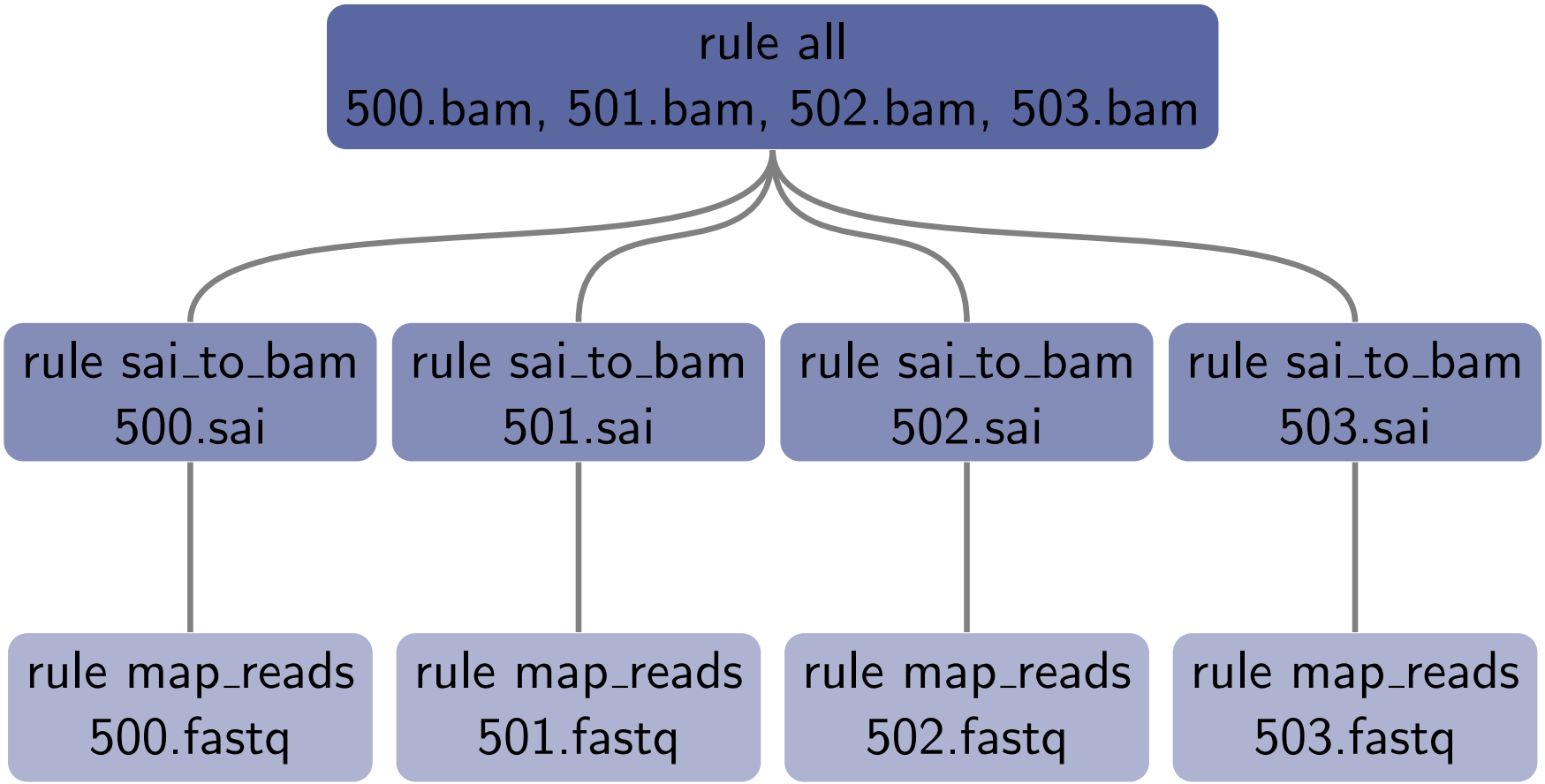


Snakemake Idea

Example: for samples $\{500, \ldots, 503\}$ map reads to hg19.



Snakemake Idea



- DAG of jobs
- each path needs to be executed serially
- two disjoint paths can be executed in parallel

Building the DAG

an edge between two jobs A,B if input of A is matched by output of B, e.g.

$$\texttt{"500.bam"} \text{ matches } \texttt{"\{sample\}.bam"}$$
$$\Leftrightarrow$$
$$\texttt{"500.bam"} \in L(\texttt{".+\textbackslash.bam"})$$

In case of ambiguity:

- Constrain wildcards: `"{sample,[0-9]+}.bam"`
- Order rules: `ruleorder: sai_to_bam > sort_bam`

Scheduling

execute the set of jobs E^* among all $E \subseteq J$ that maximizes under lexicographical order

$$\sum_{j \in E} (p_j, i_j)$$

such that

$$\sum_{j \in E} t_j \leq I$$

J	set of jobs ready to execute
T	provided cores
I	idle cores
t_j	threads of job j
p_j	priority of job j
i_j	input size of job j

Python Rules


```
rule plot_coverage_histogram:
  input: "{sample}.bam"
  output: hist="{sample}.coverage.pdf"
  run:
    plt.hist(np.fromiter(
          shell("samtools mpileup {input} | cut -f4",
                iterable=True),
          dtype=int))
    plt.savefig(output.hist)
```

R Rules


```
rule plot_coverage_histogram:
    input: ...
    output: ...
    run:
        R("""
        # some R code
        """)
```

Dynamic Output

In case of unknown number of output files...
dynamically update the DAG to process these

Dynamic Output

In case of unknown number of output files...
dynamically update the DAG to process these

```
rule all:
  input: dynamic("{cluster}.pdf")

rule plot:
  input: "{cluster}.csv"
  output: "{cluster}.pdf"
  shell: "gnuplot ..."

rule cluster:
  input: ...
  output: dynamic("{cluster}.csv")
  shell: "cluster ..."
```

Dynamic Output

In case of unknown number of output files...
dynamically update the DAG to process these

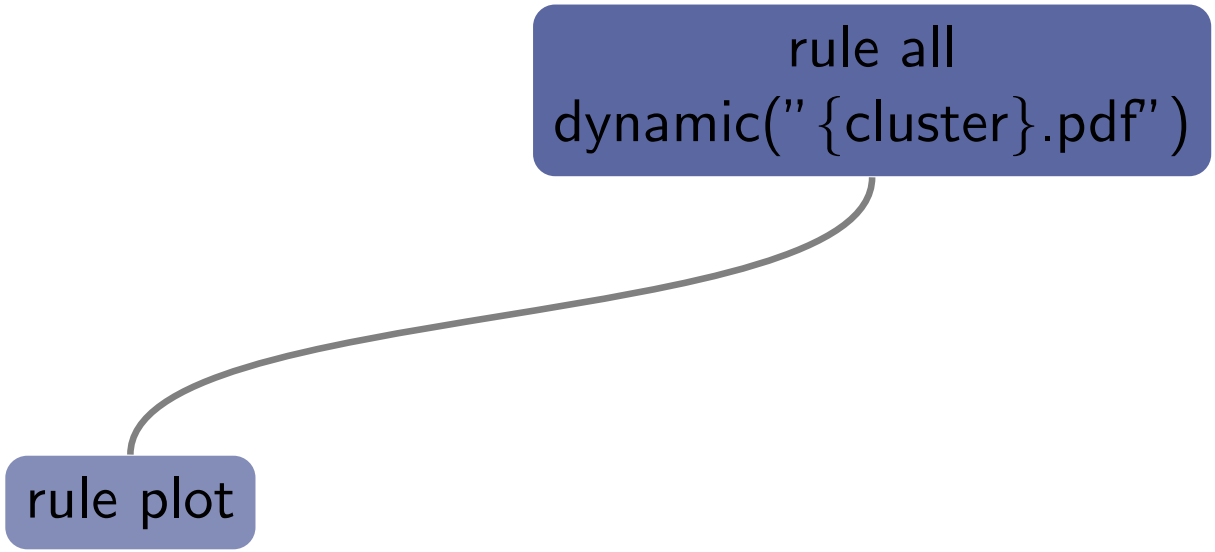
rule all
dynamic("{cluster}.pdf")

```
rule plot:
    input: "{cluster}.csv"
    output: "{cluster}.pdf"
    shell: "gnuplot ..."

rule cluster:
    input: ...
    output: dynamic("{cluster}.csv")
    shell: "cluster ..."
```

Dynamic Output

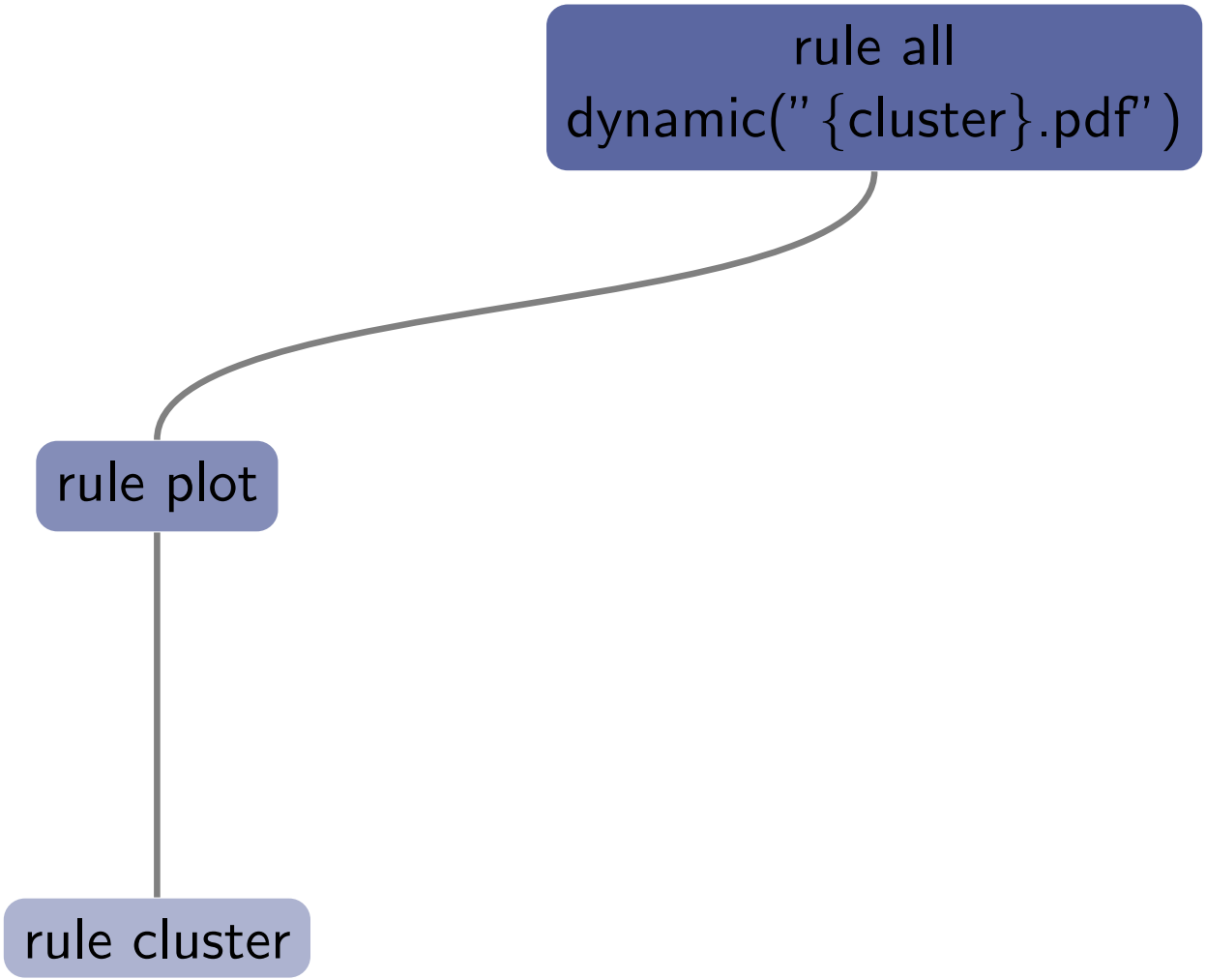
In case of unknown number of output files...
dynamically update the DAG to process these




```
rule cluster:
    input: ...
    output: dynamic("{cluster}.csv")
    shell: "cluster ..."
```

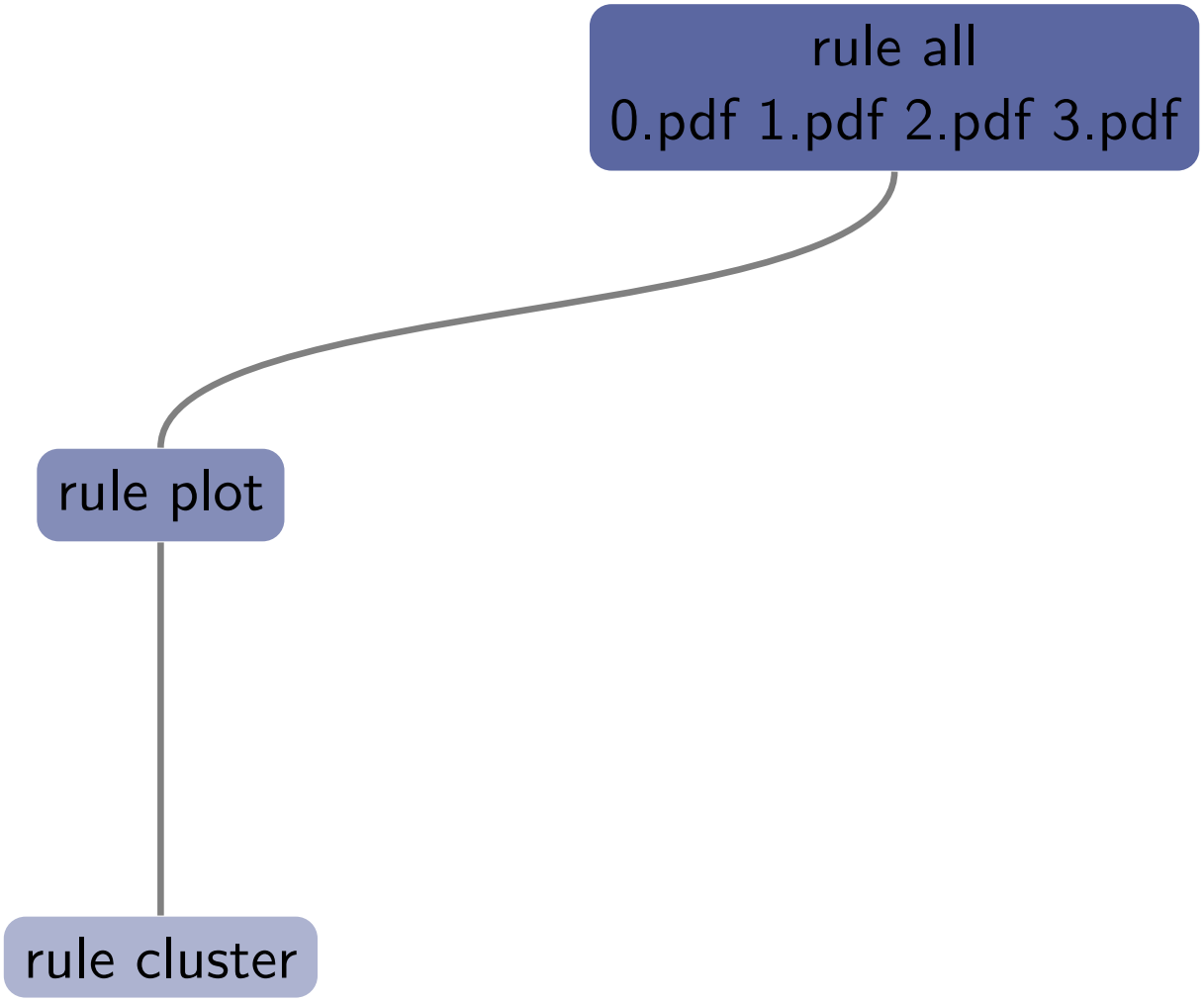
Dynamic Output

In case of unknown number of output files...
dynamically update the DAG to process these



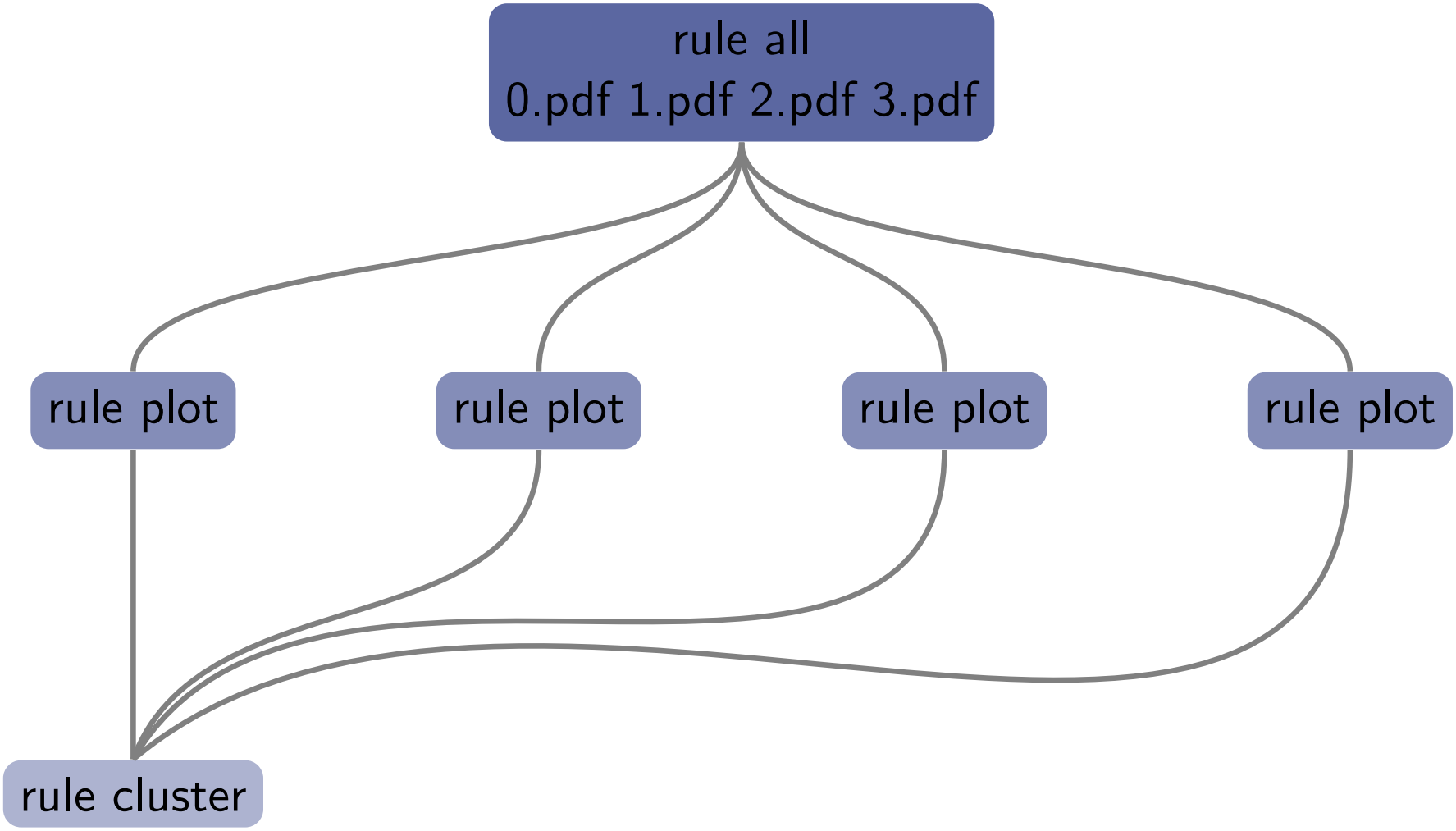
Dynamic Output

In case of unknown number of output files...
dynamically update the DAG to process these



Dynamic Output

In case of unknown number of output files...
dynamically update the DAG to process these

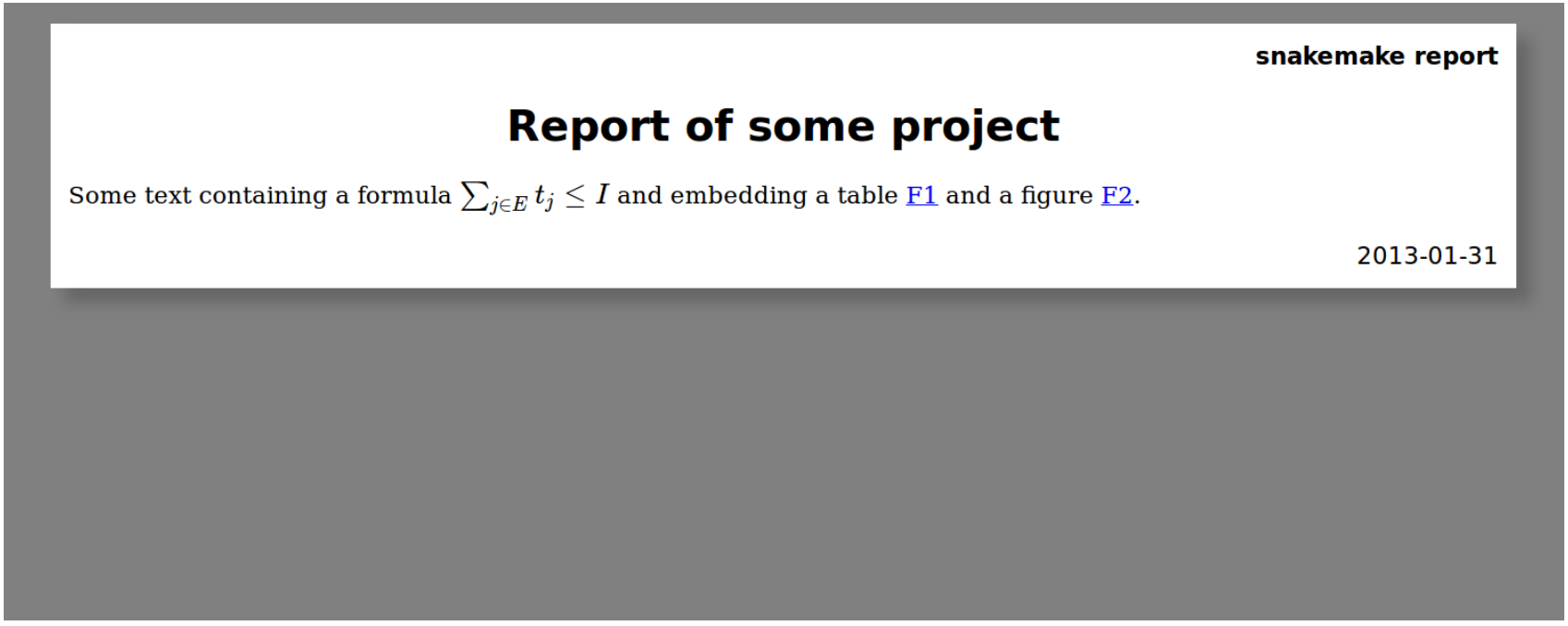


HTML Reports

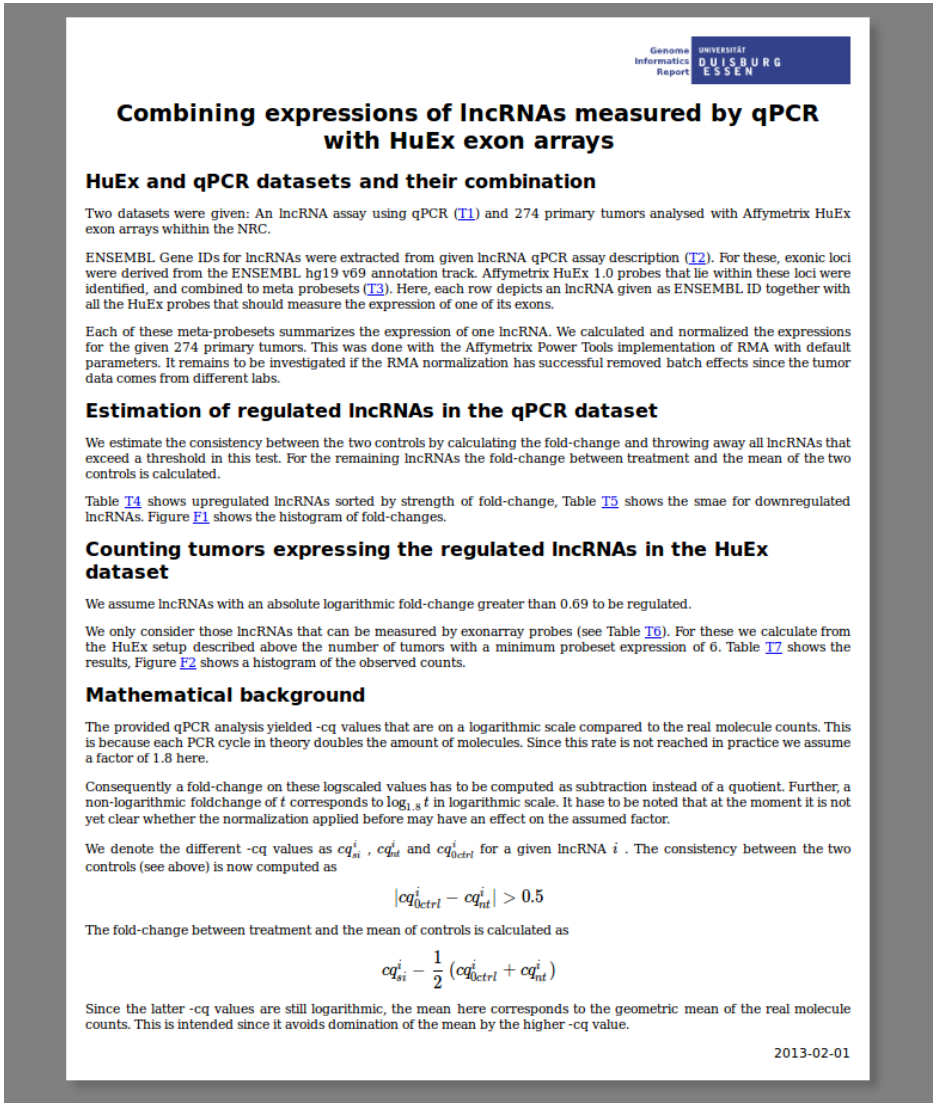

```
rule report:
    input: "table.csv", "plot.pdf"
    output: "report.html"
    run:
        report("""
        ======================
        Report of some project
        ======================

        Some text containing a formula
        :math:`\sum_j \in E t_j \leq I`
        and embedding a table F1_ and a figure F2_.
        """, output[0], F1=input[0], F2=input[1])
```

HTML Reports



HTML Reports



Genome Informatics Report
UNIVERSITÄT DUISBURG ESSEN

Combining expressions of lncRNAs measured by qPCR with HuEx exon arrays

HuEx and qPCR datasets and their combination

Two datasets were given: An lncRNA assay using qPCR (T1) and 274 primary tumors analysed with Affymetrix HuEx exon arrays whithin the NRC.

ENSEMBL Gene IDs for lncRNAs were extracted from given lncRNA qPCR assay description (T2). For these, exonic loci were derived from the ENSEMBL hg19 v69 annotation track. Affymetrix HuEx 1.0 probes that lie within these loci were identified, and combined to meta probesets (T3). Here, each row depicts an lncRNA given as ENSEMBL ID together with all the HuEx probes that should measure the expression of one of its exons.

Each of these meta-probesets summarizes the expression of one lncRNA. We calculated and normalized the expressions for the given 274 primary tumors. This was done with the Affymetrix Power Tools implementation of RMA with default parameters. It remains to be investigated if the RMA normalization has successful removed batch effects since the tumor data comes from different labs.

Estimation of regulated lncRNAs in the qPCR dataset

We estimate the consistency between the two controls by calculating the fold-change and throwing away all lncRNAs that exceed a threshold in this test. For the remaining lncRNAs the fold-change between treatment and the mean of the two controls is calculated.

Table T4 shows upregulated lncRNAs sorted by strength of fold-change, Table T5 shows the smae for downregulated lncRNAs. Figure F1 shows the histogram of fold-changes.

Counting tumors expressing the regulated lncRNAs in the HuEx dataset

We assume lncRNAs with an absolute logarithmic fold-change greater than 0.69 to be regulated.

We only consider those lncRNAs that can be measured by exonarray probes (see Table T6). For these we calculate from the HuEx setup described above the number of tumors with a minimum probeset expression of 6. Table T7 shows the results, Figure F2 shows a histogram of the observed counts.

Mathematical background

The provided qPCR analysis yielded -cq values that are on a logarithmic scale compared to the real molecule counts. This is because each PCR cycle in theory doubles the amount of molecules. Since this rate is not reached in practice we assume a factor of 1.8 here.

Consequently a fold-change on these logscaled values has to be computed as subtraction instead of a quotient. Further, a non-logarithmic foldchange of t corresponds to $\log_{1.8} t$ in logarithmic scale. It hase to be noted that at the moment it is not yet clear whether the normalization applied before may have an effect on the assumed factor.

We denote the different -cq values as cq^i_{si} , cq^i_{nt} and cq^i_{0ctrl} for a given lncRNA i . The consistency between the two controls (see above) is now computed as

$$|cq^i_{0ctrl} - cq^i_{nt}| > 0.5$$

The fold-change between treatment and the mean of controls is calculated as

$$cq^i_{si} - \frac{1}{2}\left(cq^i_{0ctrl} + cq^i_{nt}\right)$$

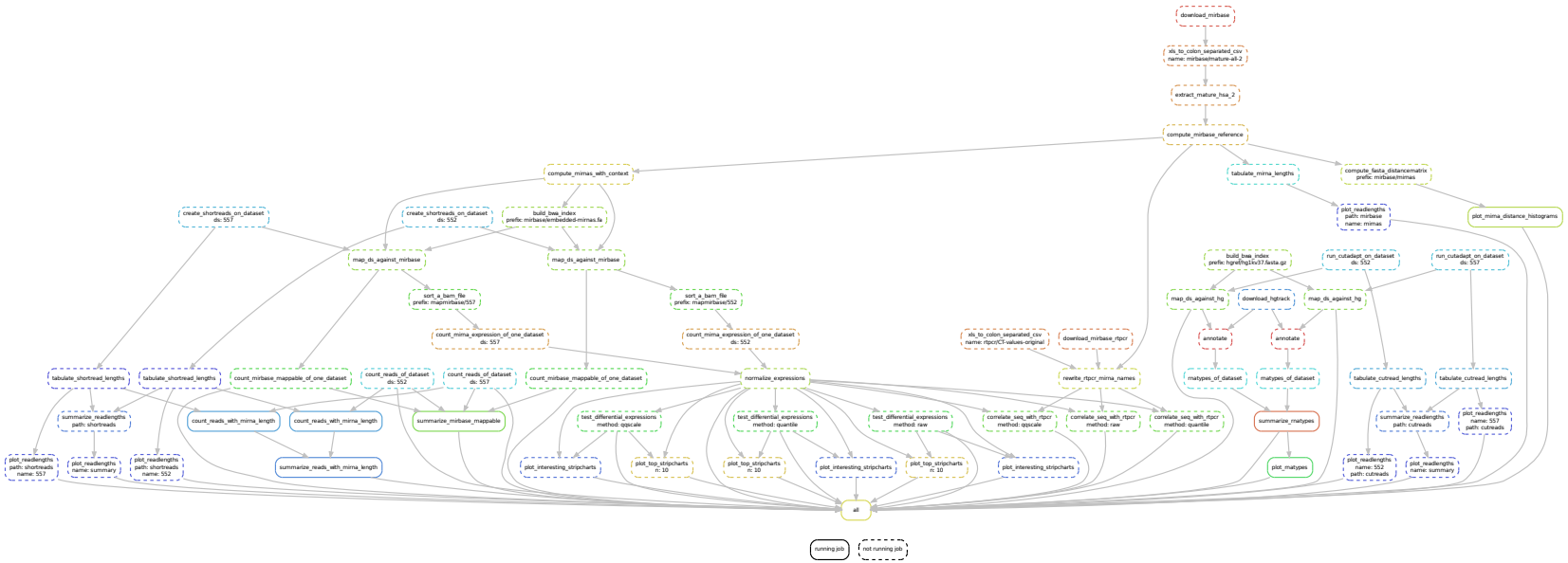
Since the latter -cq values are still logarithmic, the mean here corresponds to the geometric mean of the real molecule counts. This is intended since it avoids domination of the mean by the higher -cq value.

2013-02-01

Visualization

Using the dot language of graphviz:

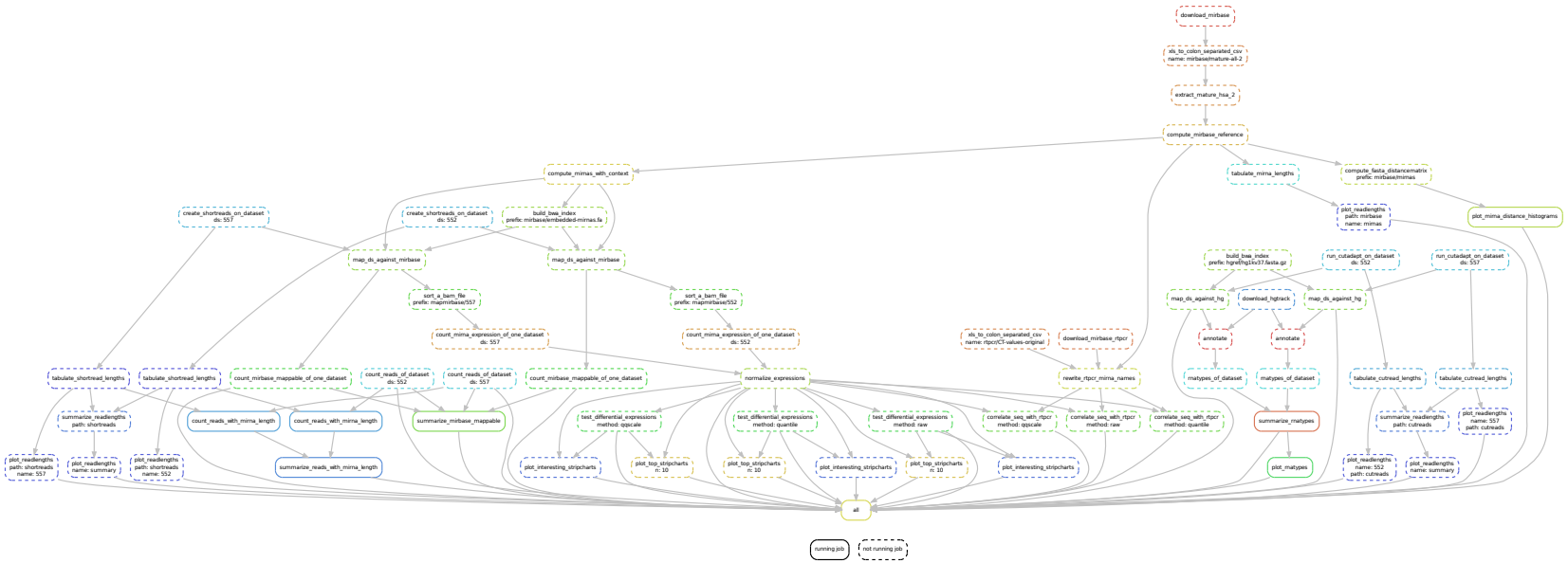
```
$ snakemake --dag | dot | display
```



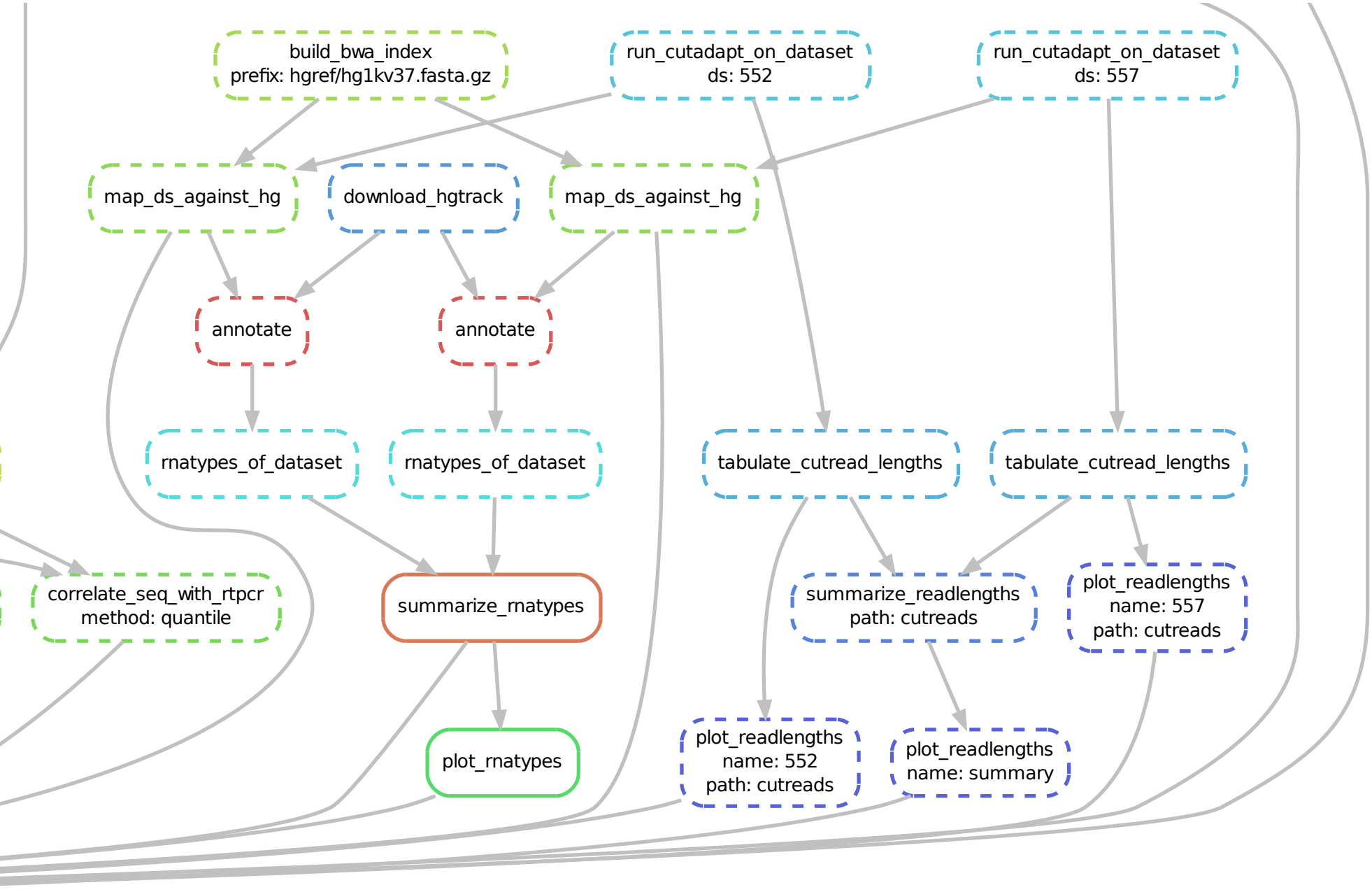
Visualization

Using the dot language of graphviz:

```
$ snakemake --dag | dot | display
```



Visualization



Conclusion

Snakemake is a new workflow system that provides:

- an easy pythonic textual representation
- multiple wildcards in filenames
- dynamic update of job DAG
- implicit parallelization and dependency resolution
- job scheduling considering threads, priorities and input size
- cluster and batch support

https://bitbucket.org/johanneskoester/snakemake

depends on Python ≥ 3.2

Early adopters:
Shirley Liu's lab, Broad Institute
Genome of the Netherlands Project