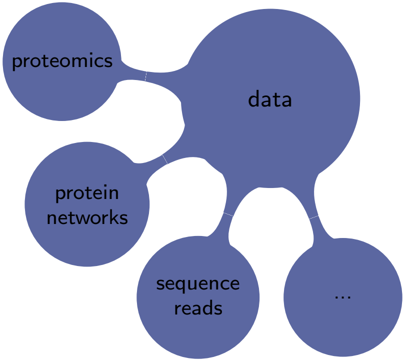
Building and Documenting Bioinformatics Workflows with Python-based Snakemake

Johannes Köster, Sven Rahmann

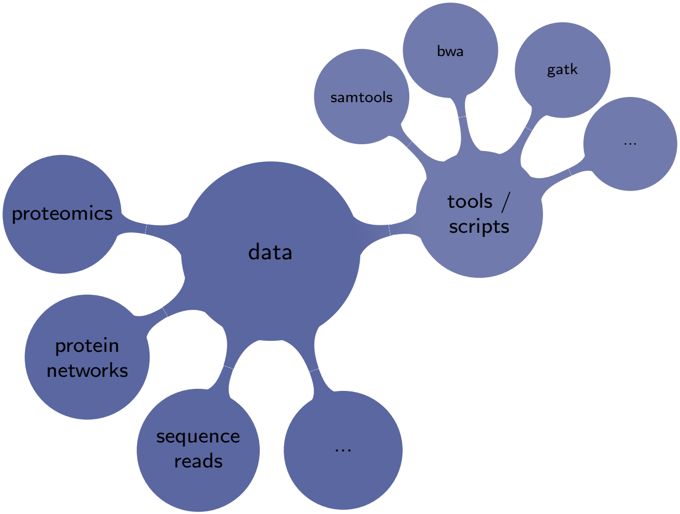
German Conference on Bioinformatics
September 2012

Structure

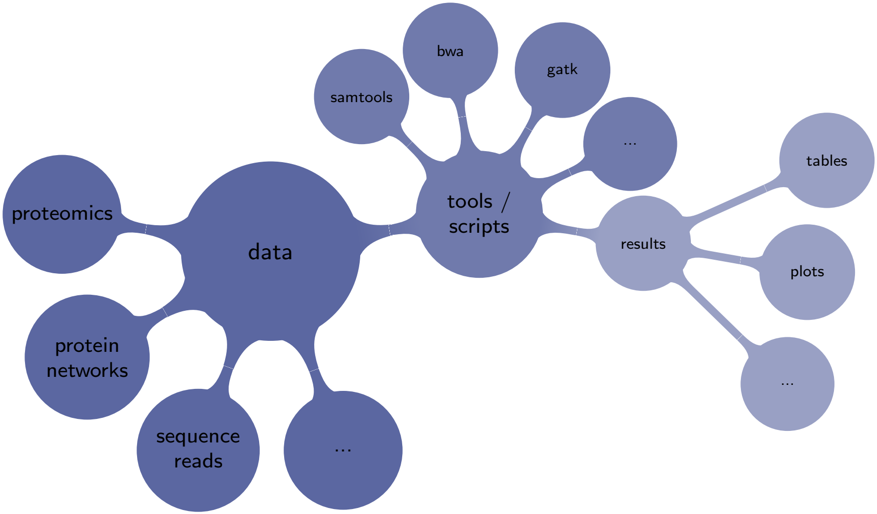
Motivation



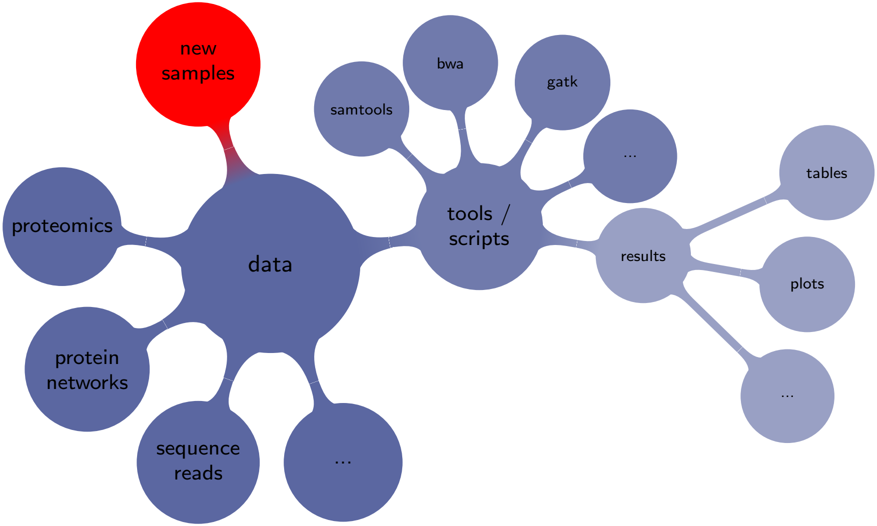
Motivation


data
proteomics
protein networks
sequence reads
...
tools / scripts
samtools
bwa
gatk
...

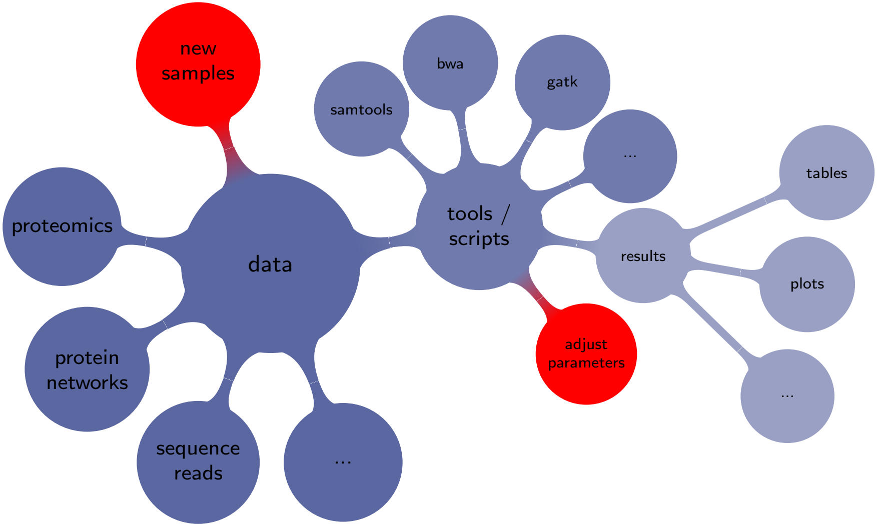
Motivation


data
proteomics
protein networks
sequence reads
...
tools / scripts
samtools
bwa
gatk
...
results
tables
plots
...

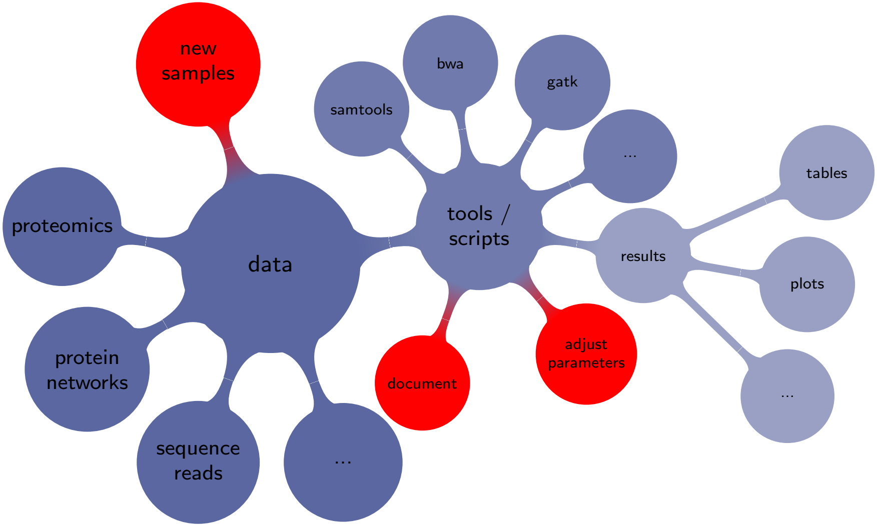
Motivation


new samples
proteomics
data
protein networks
sequence reads
...
tools / scripts
samtools
bwa
gatk
...
results
tables
plots
...

Motivation

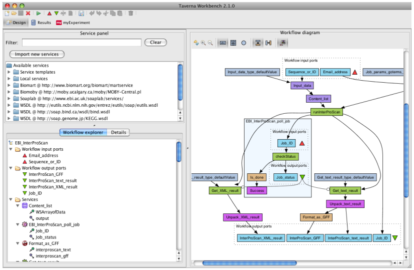

new samples
samtools
bwa
gatk
...
tables
proteomics
data
tools / scripts
results
plots
adjust parameters
protein networks
...
sequence reads
...

Motivation


new samples
proteomics
data
protein networks
sequence reads
...
samtools
bwa
gatk
...
tools / scripts
tables
results
plots
...
document
adjust parameters

Workflow Descriptions

```
IDIR=../include
ODIR=obj
LDIR=../lib

LIBS=-lm

CC=gcc
CFLAGS=-I$(IDIR)

_HEADERS = hello.h
HEADERS = $(patsubst %,$(IDIR)/%,$(_HEADERS))

_OBJS = hello.o hellofunc.o
OBJS = $(patsubst %,$(ODIR)/%,$(_OBJS))

# build the executable from the object files
hello: $(OBJS)
        $(CC) -o $@ $^ $(CFLAGS)

# compile a single .c file to an .o file
$(ODIR)/%.o: %.c $(HEADERS)
        $(CC) -c -o $@ $< $(CFLAGS)


# clean up temporary files
.PHONY: clean
clean:
        rm -f $(ODIR)/*.o *~ core $(IDIR)/*~
```



http://www.cs.colby.edu/maxwell/courses/tutorials/maketutor

http://www.taverna.org.uk

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...

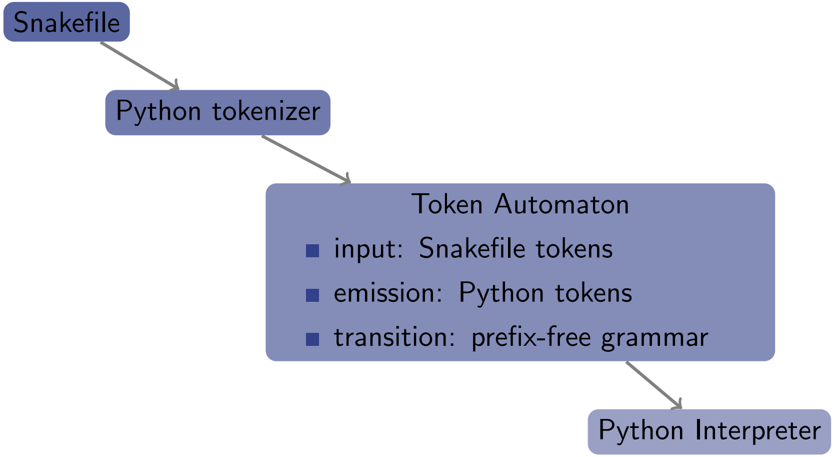
- a language to write rules to create each output file from input files
- wildcards for generalization
- implicit dependency resolution
- implicit parallelization
- fast and collaborative development on text files

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

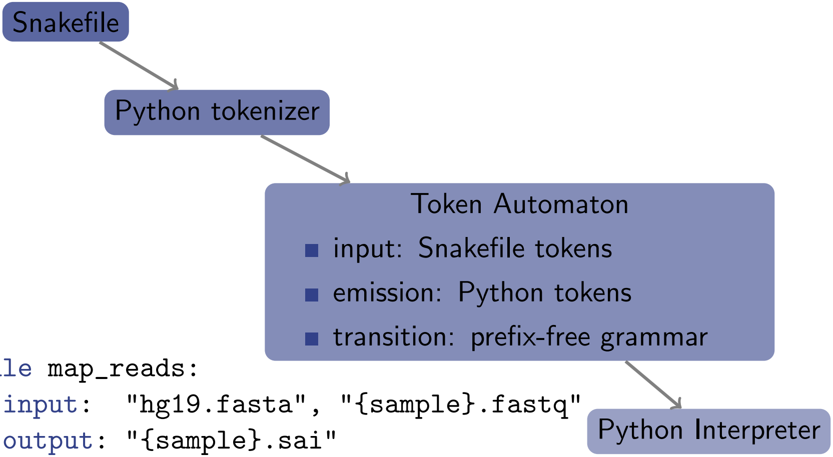
Snakemake Language

Idea: extend the Python syntax but avoid to write a full parser



Snakemake Language

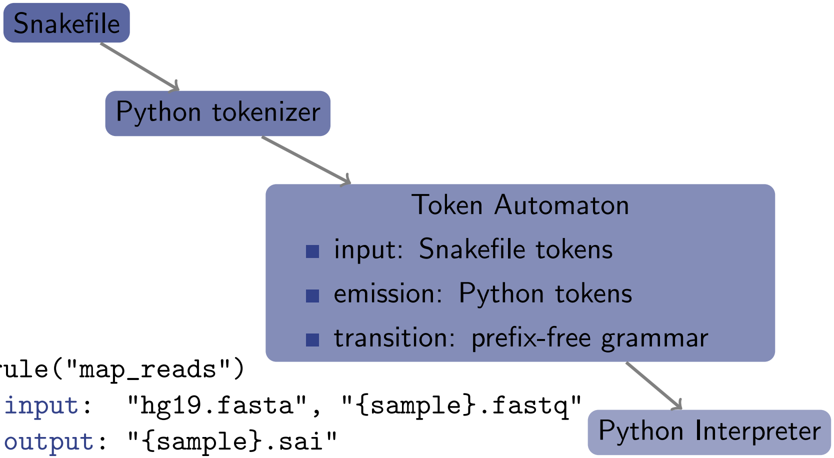
Idea: extend the Python syntax but avoid to write a full parser



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

Snakemake Language

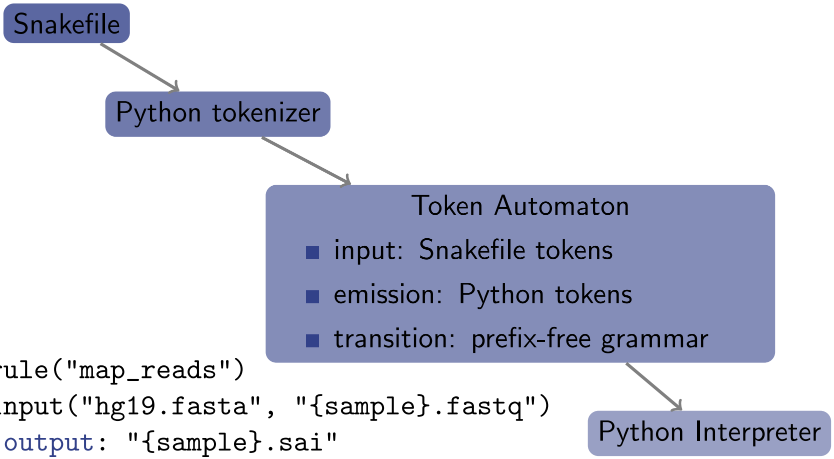
Idea: extend the Python syntax but avoid to write a full parser



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

Snakemake Language

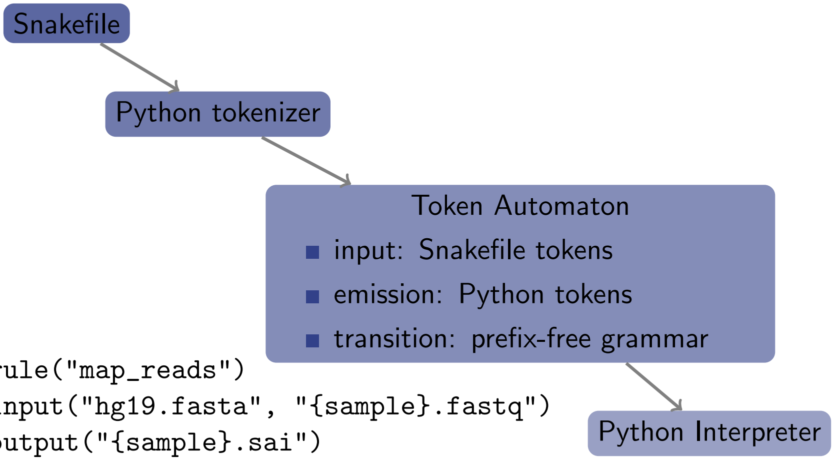
Idea: extend the Python syntax but avoid to write a full parser



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

Snakemake Language

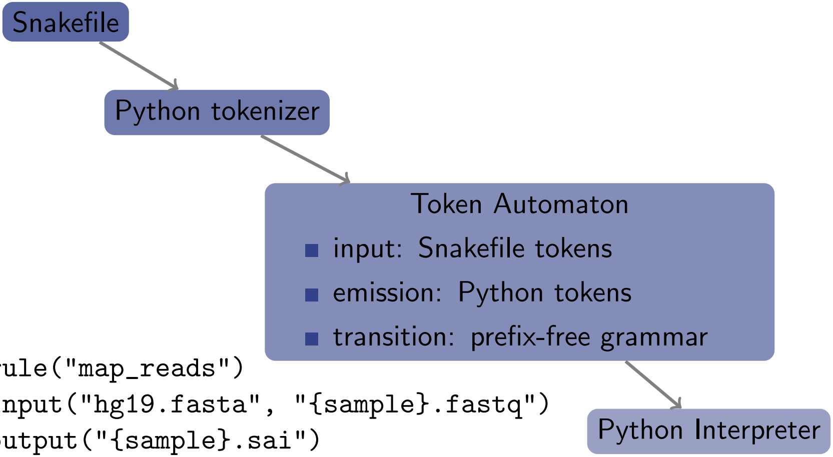
Idea: extend the Python syntax but avoid to write a full parser



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

Snakemake Language

Idea: extend the Python syntax but avoid to write a full parser



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

Example Workflow

For samples $\{500, \ldots, 503\}$ map reads to hg19.

Example Workflow

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}"
```

Example Workflow

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}"
```

Example Workflow

For samples $\{500, \ldots, 503\}$ map reads to hg19.

```
SAMPLES = "500 501 502 503".split()
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}"
```

Example Workflow

For samples $\{500, \ldots, 503\}$ map reads to hg19.

```
SAMPLES = "500 501 502 503".split()
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}"
```

Example Workflow

For samples $\{500, \dots, 503\}$ map reads to hg19.

```
SAMPLES = "500 501 502 503".split()
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}"
```

Example Workflow

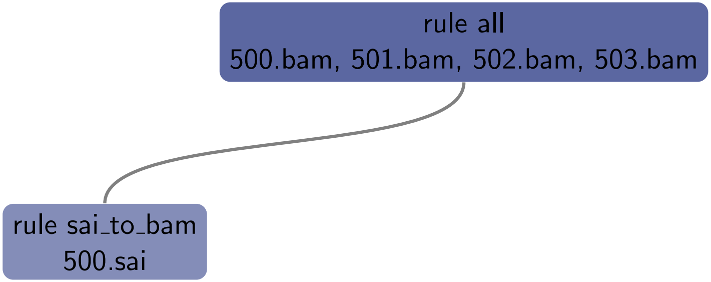
For samples $\{500, \ldots, 503\}$ map reads to hg19.

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}"
```

Example Workflow

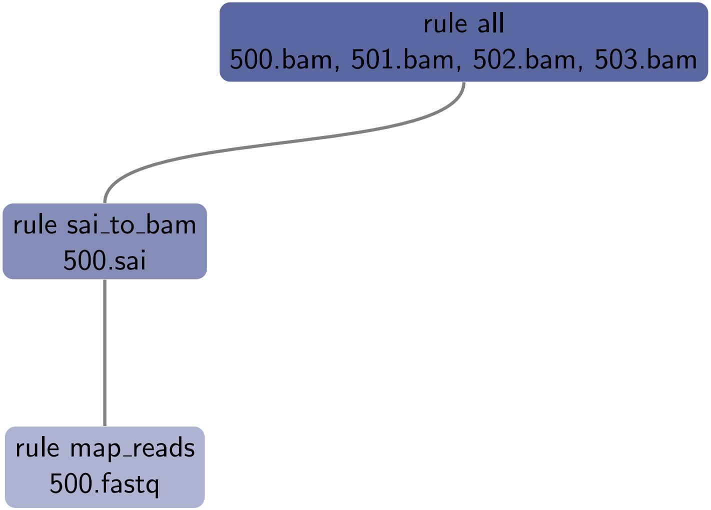
For samples $\{500, \ldots, 503\}$ map reads to hg19.



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

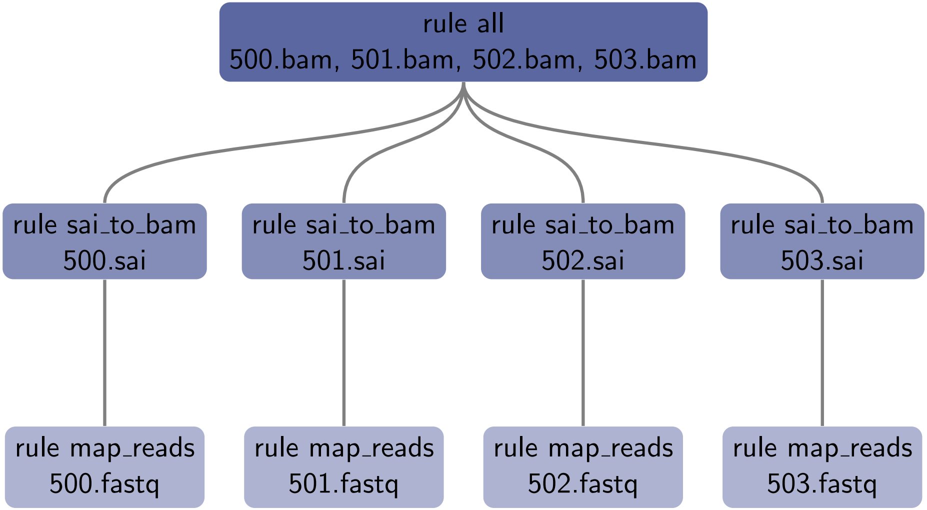
Example Workflow

For samples $\{500, \ldots, 503\}$ map reads to hg19.



Example Workflow

For samples $\{500, \ldots, 503\}$ map reads to hg19.



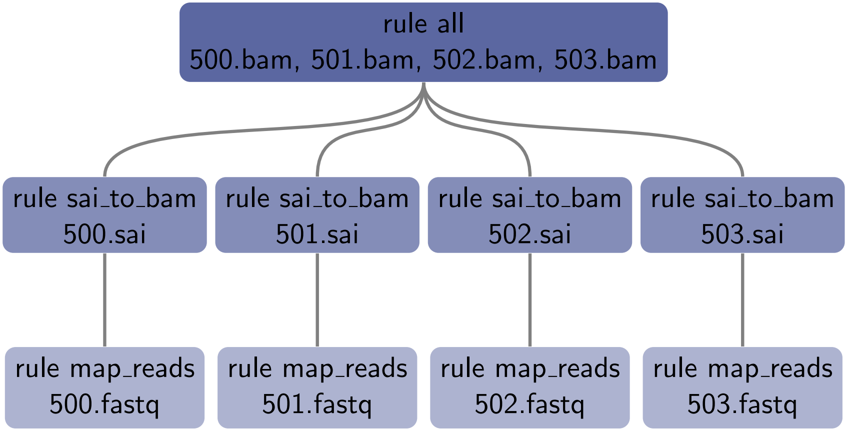
Python Rules

```
import matplotlib.pyplot as plt
rule plot_coverage_histogram:
  input: "{sample}.bam"
  output: hist = "{sample}.coverage.pdf"
  run:
    plt.hist(list(map(int,
         shell("samtools mpileup {input} | cut -f4",
               iterable = True))))
    plt.savefig(output.hist)
```

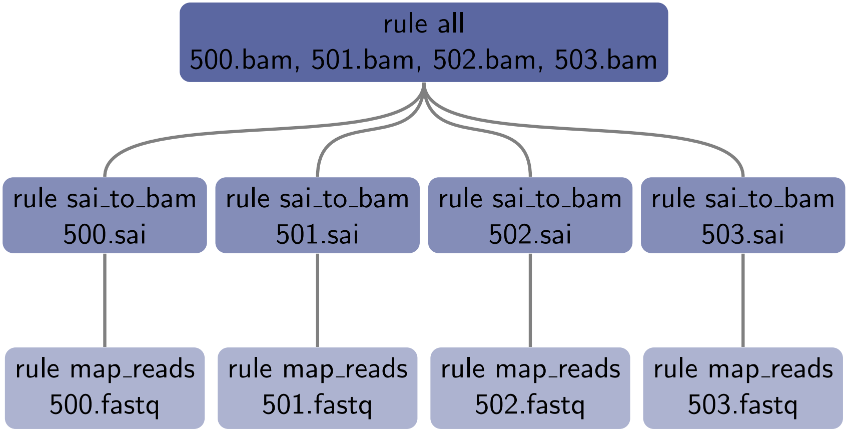
R Rules

```
import rpy2.robjects as robjects
rule plot_coverage_histogram:
  input: "{sample}.fastq"
  output: "{sample}.stats.csv"
  run:
    robjects.r(format("""

      # some R code

    """))
```

Snakemake Engine



Snakemake Engine



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

Building the DAG

File matching

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

In case of ambiguity:

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

Job Scheduling

Goals:

- restrict the number of parallel jobs
- take threads of individual jobs into account

Job Scheduling

Goals:

- restrict the number of parallel jobs
- take threads of individual jobs into account

Job Scheduling Problem

- let J be the set of jobs ready to execute
- let t_j be the number of threads a job j uses (1 by default)
- let T be a given threshold of available cores (I of them being idle)
- then execute the set of jobs E^* among all $E \subseteq J$ that maximizes

$$\sum_{j \in E} \min(t_j, T)$$

  such that the sum remains bounded by I

Conclusion

Snakemake is a new workflow system that provides:

- an easy pythonic textual representation
- multiple wildcards in filenames
- implicit parallelization and dependency resolution
- job scheduling that takes threads into account
- cluster support

http://bitbucket.org/johanneskoester/snakemake

depends on Python ≥ 3.2