Snakemake

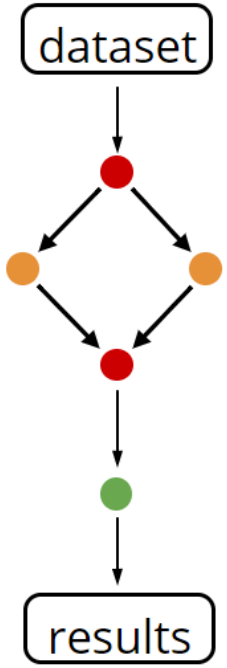
Introductory tutorial

April 5, 2018

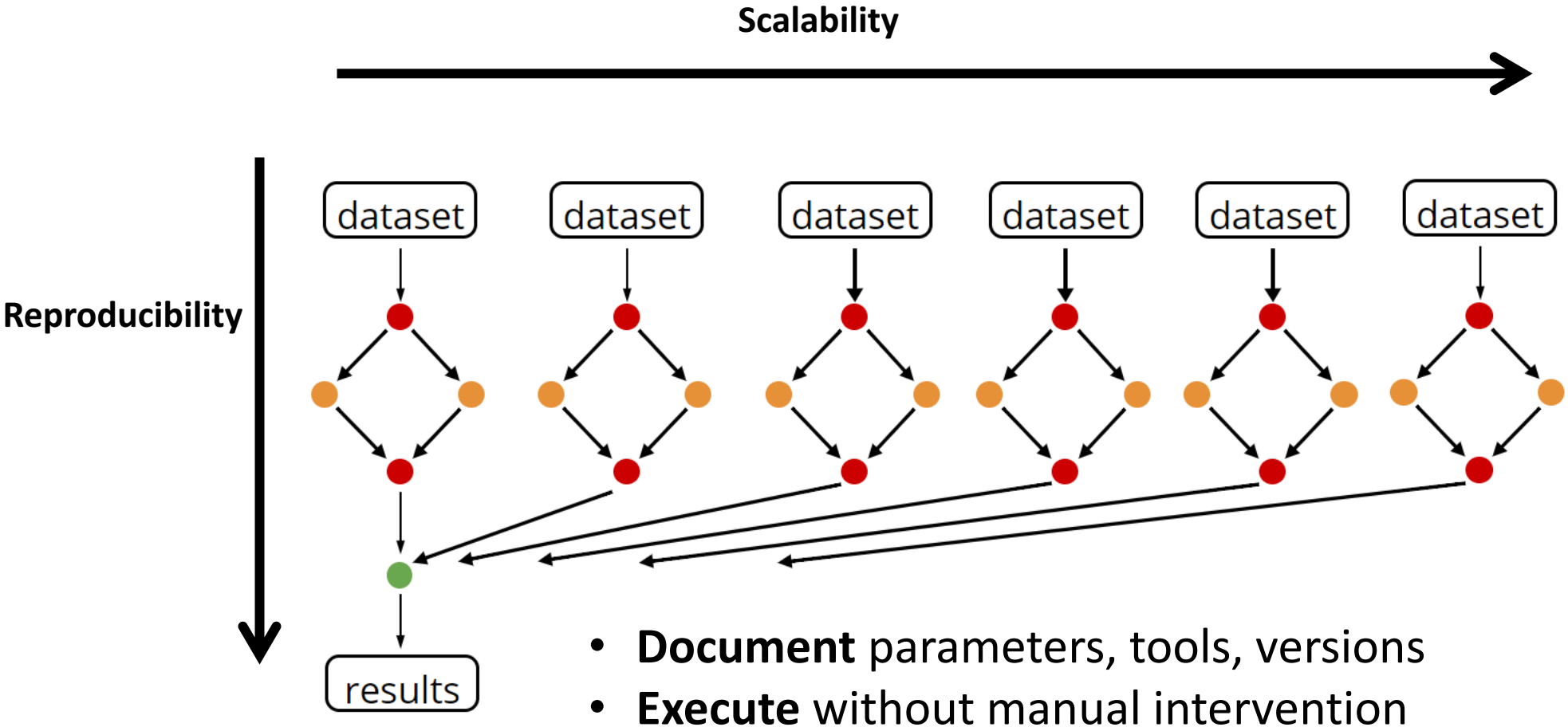
Israa Alqassem

Slides adapted from Johannes Köster

Data Analysis



Data Analysis



- **Document** parameters, tools, versions
- **Execute** without manual intervention
- **Handle parallelization:** execute for tens or thousands of datasets
- **Avoid redundancy**: e.g., in case of failure, reading dataset, etc.

Snakemake: Python-based workflow engine

BIOINFORMATICS APPLICATION NOTE

Vol. 28 no. 19 2012, pages 2520–2522
doi:10.1093/bioinformatics/bts480

Genome analysis

Advance Access publication August 20, 2012

Snakemake—a scalable ~~bioinformatics~~ workflow engine

Johannes Köster[1,2,*] and Sven Rahmann[1]

[1]Genome Informatics, Institute of Human Genetics, University of Duisburg-Essen and [2]Paediatric Oncology, University Childrens Hospital, 45147 Essen, Germany

Associate Editor: Alfonso Valencia

ABSTRACT

Summary: Snakemake is a workflow engine that provides a readable Python-based workflow definition language and a powerful execution environment that scales from single-core workstations to compute clusters without modifying the workflow. It is the first system to support the use of automatically inferred multiple named wildcards (or variables) in input and output filenames.

Availability: http://snakemake.googlecode.com.

Contact: johannes.koester@uni-due.de

custom server processes running on the cluster nodes. Finally, Snakemake is the first system to support file name inference with multiple named wildcards in rules.

2 SNAKEMAKE LANGUAGE

A workflow is defined in a 'Snakefile' through a domain-specific language that is close to standard Python syntax. It consists of

Reproducibility with Snakemake

Genome of the Netherlands:
GoNL consortium. **Nature Genetics** 2014.

Cancer:
Townsend et al. **Cancer Cell** 2016.
Schramm et al. **Nature Genetics** 2015.
Martin et al. **Nature Genetics** 2013.

Ebola:
Park et al. **Cell** 2015

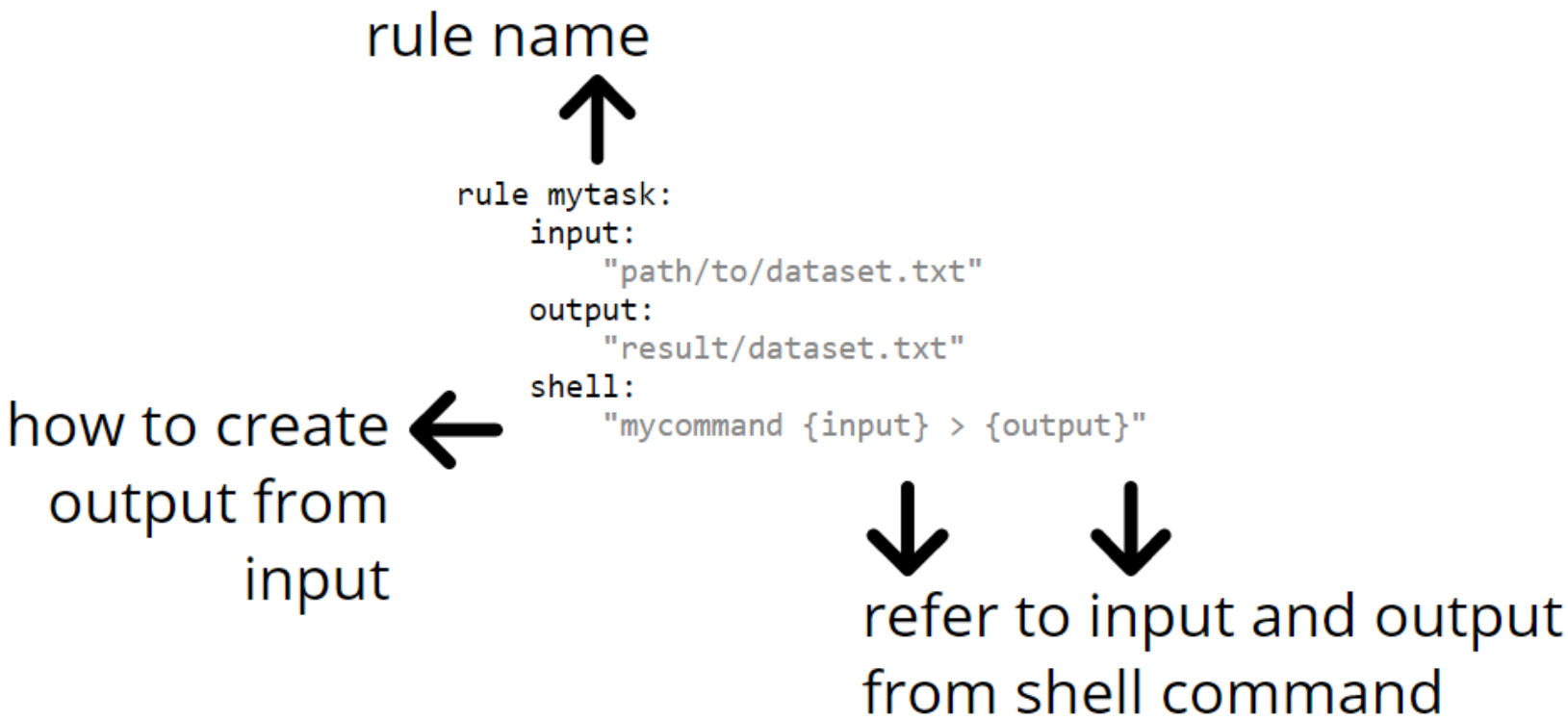
iPSC:
Burrows et al. **PLOS Genetics** 2016.

Computational methods:
Ziller et al. **Nature Methods** 2015.
Schmied et al. **Bioinformatics** 2015.
Břinda et al. **Bioinformatics** 2015
Chang et al. **Molecular Cell** 2014.
Marschall et al. **Bioinformatics** 2012.

Snakemake

- Python + domain specific syntax
- Decompose workflow into rules
- Rules define how to obtain output files from input files
- Snakemake infers dependencies and execution order

Define workflows in terms of rules



Define workflows in terms of rules

generalize rules with named wildcards

```
rule mytask:
    input:
        "path/to/{dataset}.txt"
    output:
        "result/{dataset}.txt"
    shell:
        "mycommand {input} > {output}"
```

Define workflows in terms of rules

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

refer to Python script

Dependencies are determined automatically

```
rule mytask:
    input:
        "path/to/dataset1.txt"
    output:
        "result/dataset1.txt"
    script:
        "scripts/myscript.R"
```

```
rule mytask:
    input:
        "path/to/dataset2.txt"
    output:
        "result/dataset2.txt"
    script:
        "scripts/myscript.R"
```

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

Dependencies are determined top-down

```
1  DATASETS = ["D1", "D2", "D3"]
2
3  rule all:
4      input: expand("{dataset}.sorted.txt",dataset=DATASETS)
5
6
7  rule sort:
8      input:
9          "{dataset}.txt"
10     output:
11         "{dataset}.sorted.txt"
12     shell:
13         "sort -n {input} > {output}"
14
```

Python code

Job #1: Target rule to collect all results

Job #i: Create i-th input for job #1

A job is executed if and only if…

- Output file is target and does not exist
- Output file needed by another executed job and does not exist
- Input file newer than output file
- Input file will be updated by other job
- Execution is enforced
- Determined via breadth-first-search on DAG of jobs

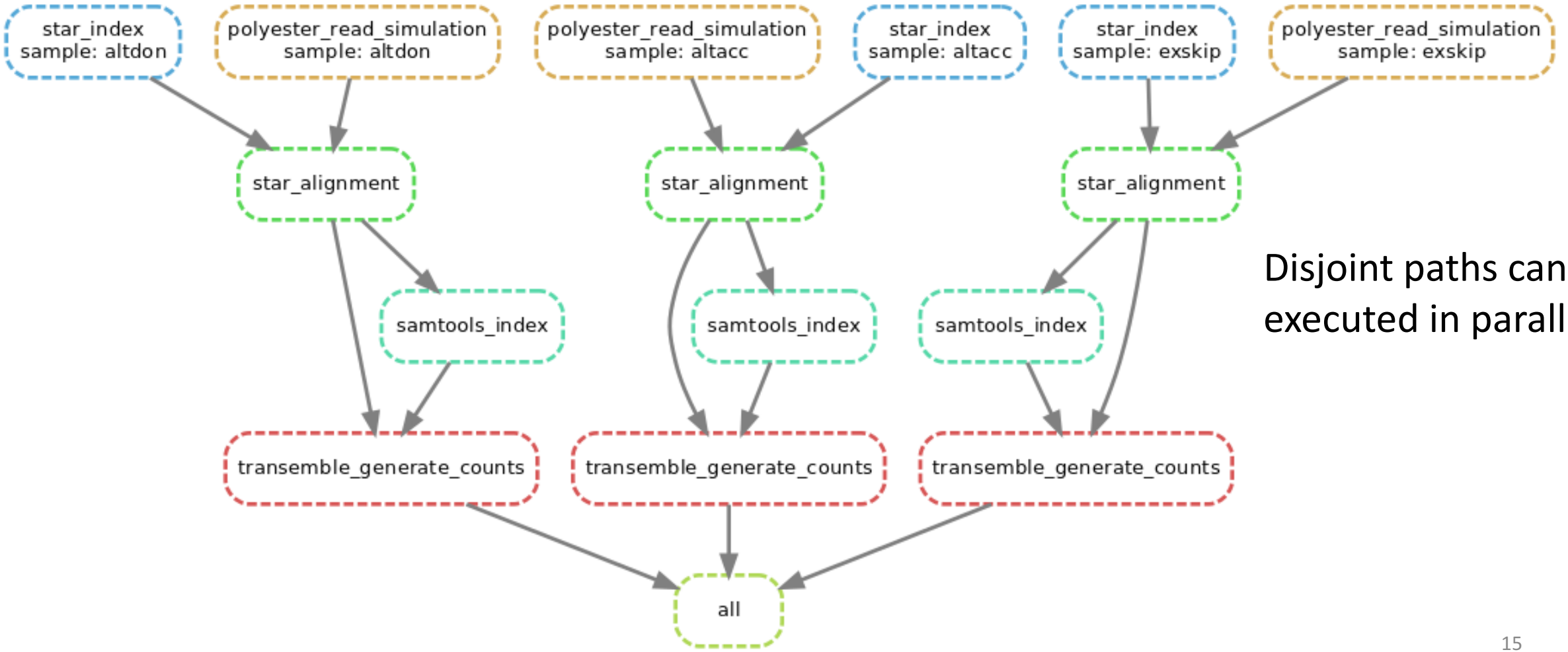
→ Determined via breadth-first-search on DAG of jobs

Command line interface

```
# execute the workflow with target D1.sorted.txt
snakemake D1.sorted.txt

# execute the workflow without target: first rule defines target
snakemake

# dry-run
snakemake -n

# dry-run, print shell commands
snakemake -n -p

# dry-run, print execution reason for each job
snakemake -n -r

# visualize the DAG of jobs using the Graphviz dot command
snakemake --dag | dot -Tsvg > dag.svg
```

Real-world example: McSplicer Simulation

DAG of jobs



Disjoint paths can be executed in parallel

Advanced: Parallelization

```
rule mytask:
    input:
        "path/to/{dataset}.txt"
    output:
        "result/{dataset}.txt"
    threads: 4
    resources:
        mem_gb=2
    shell:
        "mycommand {input} > {output}"
```

→ schedule according to given resources

execute 8 jobs in parallel?

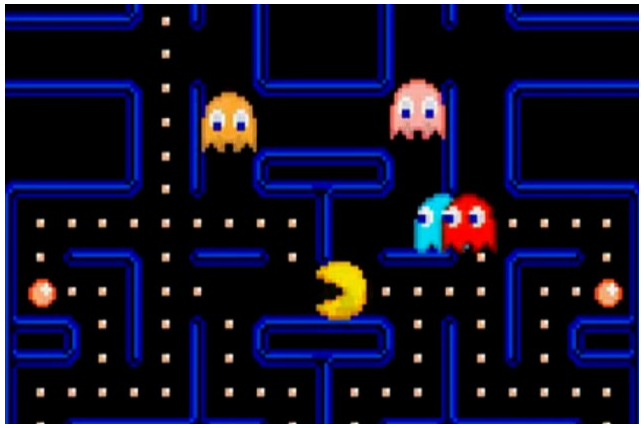
↑

```
# execute the workflow with 8 cores
snakemake --cores 8
```

Many additional features

- Logging
- Handling of temporary and protected files
- HTML5 reports
- Tracking of tool versions and code changes
- Check Snakemake online tutorial:
  http://snakemake.readthedocs.io/en/stable/getting_started/installation.html

Thanks!



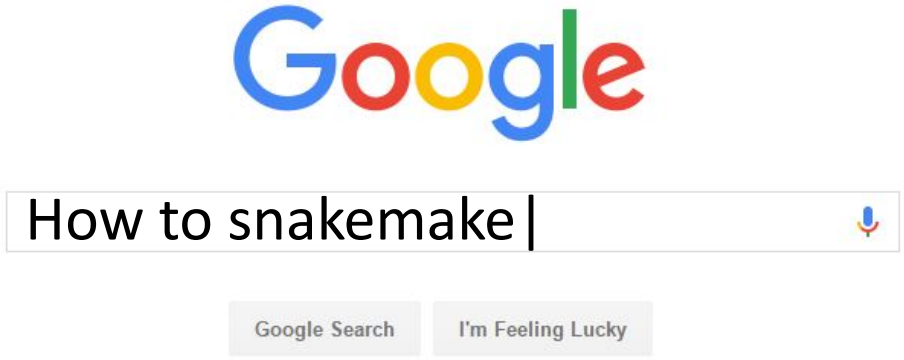
What my friends think I do.

What my family thinks I do.

What society thinks I do.

What my boss thinks I do.

What I think I do.



What I actually do.