# Coding club meet-up: Practical steps of implementing snakemake into ATAC-seq analysis

*Lingzi Li*

*October 14 2019*

## Step 1: Instsall snakemake

Below is the recommendated way to install Snakemake by Miniconda Python3 distribution. For other installation methods, please see: https://snakemake.readthedocs.io/en/stable/getting_started/installation.html

```
# Check and make sure you have Python 3
python --version

# Use Miniconda Python3 to install Snakemake
conda install -c bioconda -c conda-forge snakemake
```

## Step 2: List all the procedures between raw data to final peak profiles, and define workflow in terms of rules

**Attention: Snakemake determines the rule dependencies by matching file names.**

1) Adaptive quality and adapter trimming with **Trim Galore** https://github.com/FelixKrueger/TrimGalore/blob/master/Docs/Trim_Galore_User_Guide.md

```
# Specify fastq file location
vds2Dir = "/data/home/lingzili/ATAC_analysis/19032019/fastqFile"

# Import module os (miscellaneous operating system interfaces)
import os

# Create directory
os.system("mkdir -p trimOutput")

rule trim:
    input:
        vds2Dir + "/{sample}_R1.fastq.gz",
        vds2Dir + "/{sample}_R2.fastq.gz"
    output:
        "trimOutput/{sample}_R1_val_1.fq.gz",
        "trimOutput/{sample}_R2_val_2.fq.gz",
        "trimOutput/{sample}_R1.fastq.gz_trimming_report.txt",
        "trimOutput/{sample}_R2.fastq.gz_trimming_report.txt"
    message:"Trimming reads on sample {wildcards.sample}"
    threads: 6
    shell:
        "trim_galore --cores {threads} --paired -o trimOutput {input}"
```

2) Quality control of trimmed FastQ files with **FastQC** http://www.bioinformatics.babraham.ac.uk/projects/fastqc/

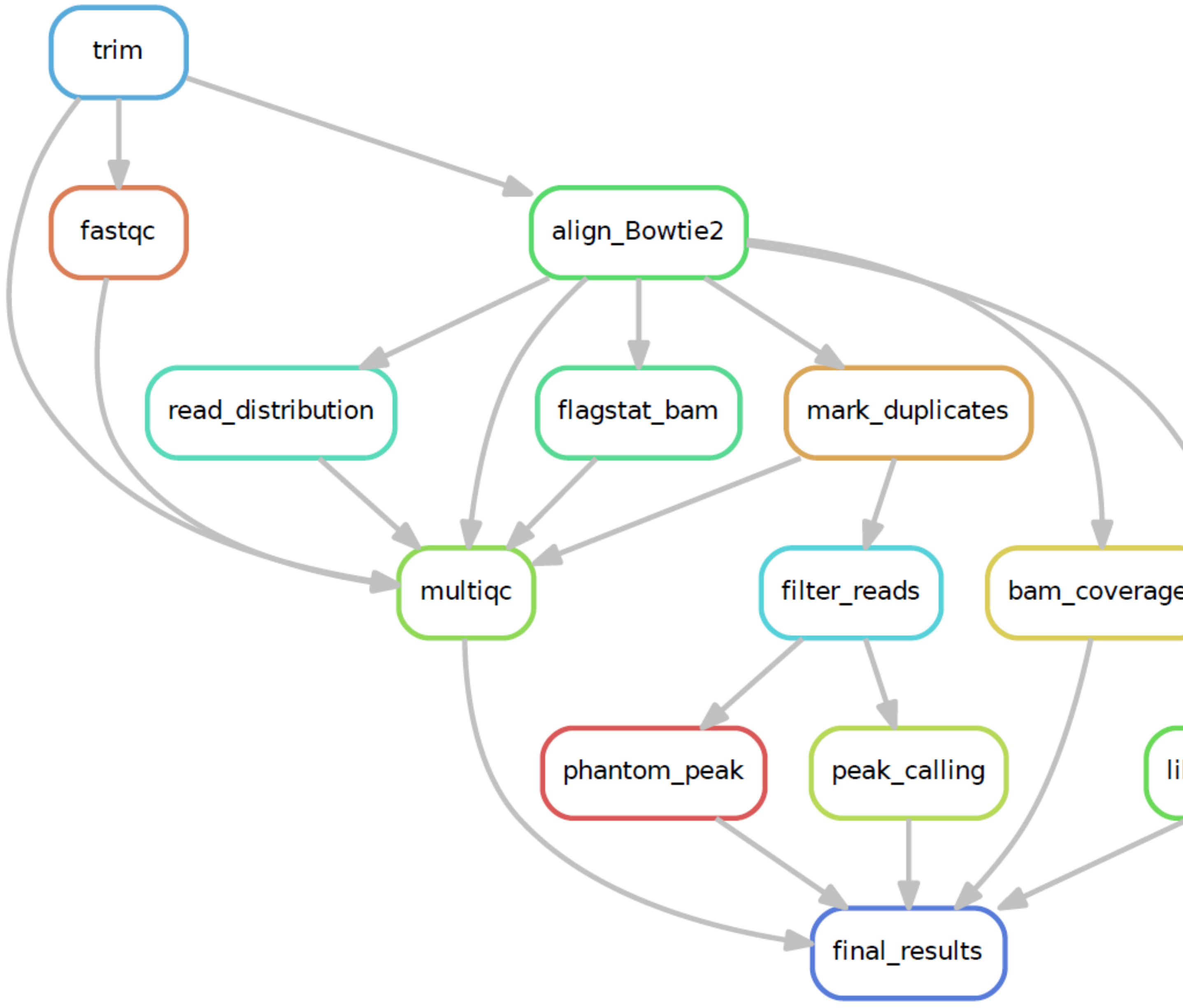


Figure 1: Rule graph

```
# Create directory
os.system("mkdir -p fastqcOutput")

rule fastqc:
    input:
        "trimOutput/{sample}_R1_val_1.fq.gz",
        "trimOutput/{sample}_R2_val_2.fq.gz"
    output:
        "fastqcOutput/{sample}_R1_val_1_fastqc.zip",
        "fastqcOutput/{sample}_R2_val_2_fastqc.zip"
    message: "FastQC"
    threads: 10
    shell:
        "fastqc -t {threads} --quiet -o fastqcOutput {input}"
```

3) Alignment to reference genome with **Bowtie 2** http://bowtie-bio.sourceforge.net/bowtie2/index.shtml

```
# Create directory
os.system("mkdir -p Bowtie2Output")

rule align_Bowtie2:
    input:
        read1 = "trimOutput/{sample}_R1_val_1.fq.gz",
        read2 = "trimOutput/{sample}_R2_val_2.fq.gz"
    output:
        "Bowtie2Output/{sample}.bam"
    params:
        Bowtie2index = "/data/home/lingzili/mm10_genome/Mus_musculus/UCSC/mm10/Sequence/Bowtie2Index/ge
    log:
        "Bowtie2Output/{sample}.bam.log"
    message: "Running Bowtie2 alignment on {wildcards.sample} and sort by coordinate"
    threads: 10
    shell:
        "bowtie2 -p {threads} --very-sensitive -X 2000 -x {params.Bowtie2index} -1 {input.read1} -2 {in
```

4) Locating and tagging duplicate reads in a BAM file with **MarkDuplicates (Picard)** https://software.broadinstitute.org/gatk/documentation/tooldocs/4.0.4.0/picard__sam__markduplicates__MarkDuplicates.php

```
rule mark_duplicates:
    input:
        "Bowtie2Output/{sample}.bam"
    output:
        bam = "Bowtie2Output/{sample}.markDup.bam",
        dupMetrics = "Bowtie2Output/{sample}.dupMetrics.txt"
    message:
        "Mark duplicates on {wildcards.sample} BAM file"
    shell:
        "java -jar /data/home/lingzili/genomeTools/picard.jar MarkDuplicates \
        I={input} O={output.bam} M={output.dupMetrics} ASSUME_SORTED=true REMOVE_DUPLICATES=false QUIET
```

5) Removal of duplicates, bad quality reads and mitochondrial reads with **SAM flag** https://broadinstitute.github.io/picard/explain-flags.html

http://www.htslib.org/doc/samtools.html

```
# -F 1804: Remove read unmapped(0x4), mate unmapped(0x8), not primary alignment(0x100), read fails qual
# -f 0x2: Only output alignments with read mapped in proper pair(0x2)
# -q 30: Skip alignments with MAPQ<30
# grep -v chrM: Print reads that do not contain chrM
rule filter_reads:
    input:
        "Bowtie2Output/{sample}.markDup.bam"
    output:
        "Bowtie2Output/{sample}.flt.sortByName.bam"
    message: "Filter reads on {wildcards.sample} and sort by name"
    threads: 10
    shell:
        "samtools view -@ {threads} -F 1804 -f 0x2 -q 30 -h {input} | grep -v chrM | samtools sort -@ {
```

6) Calling peaks of significant enrichment with **Genrich** https://github.com/jsh58/Genrich

```
# Create directory
os.system("mkdir -p peakOutput")

rule peak_calling:
    input:
        "Bowtie2Output/{sample}.flt.sortByName.bam"
    output:
        np = "peakOutput/{sample}.narrowPeak",
        log = "peakOutput/{sample}.peak.log"
    log:
        "peakOutput/{sample}.Genrich.log"
    message: "Peak calling on {wildcards.sample}"
    shell:
        "Genrich -t {input} -j -e chrY -y -r -v -o {output.np} -f {output.log} 2>{log}"
```

7) Convert BAM file to bedGraph (or bigWig) with **bamCoverage** https://deeptools.readthedocs.io/en/develop/content/tools/bamCoverage.html

```
rule bam_coverage:
    input:
        "Bowtie2Output/{sample}.bam"
    output:
        "peakOutput/{sample}.bedgraph"
    message: "Create bedGraph of {wildcards.sample}"
    threads: 10
    shell:
        "bamCoverage -p {threads} -b {input} -of bedgraph -o {output} \
        -bs 10 --extendReads --ignoreDuplicates --normalizeUsing RPKM --samFlagExclude 1804"
```

### Additional quality controls

8) Statistics on SAM flags with **samtools flagstat** http://www.htslib.org/doc/samtools.html

```
rule flagstat_bam:
    input:
        "Bowtie2Output/{sample}.bam"
    output:
        "Bowtie2Output/{sample}.bam.flagstat.txt"
    message: "Flagstat on {wildcards.sample}"
    threads: 10
    shell:
        "samtools flagstat -@ {threads} {input} 1>{output}"
```

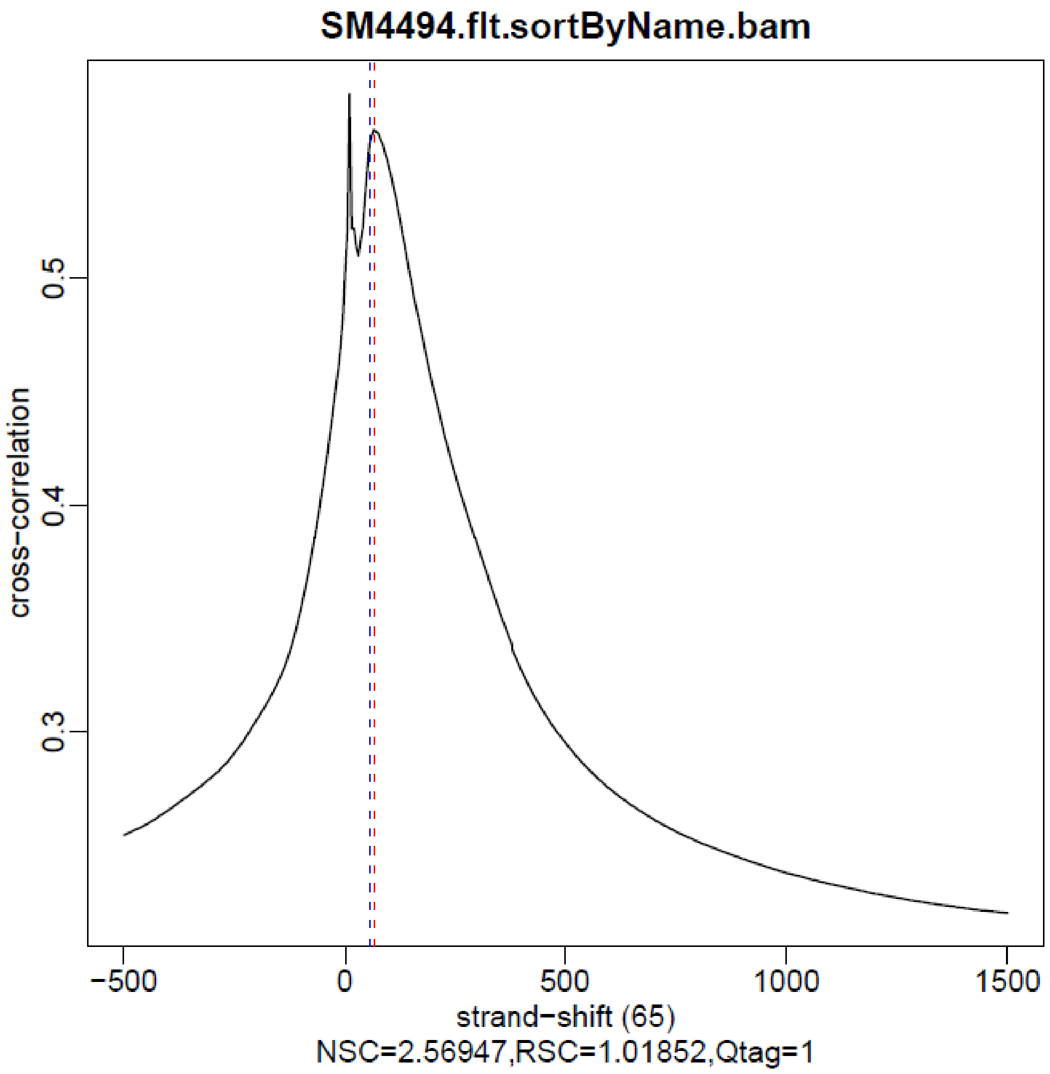
9) Estimate the number of unique DNA fragments in a library with **EstimateLibraryComplexity (Picard)** https://software.broadinstitute.org/gatk/documentation/tooldocs/4.0.6.0/picard_sam_markduplicates_EstimateLibraryComplexity.php

```
# Create directory
os.system("mkdir -p ATACqcOutput")

rule library_complexity:
    input:
        "Bowtie2Output/{sample}.bam"
    output:
        "ATACqcOutput/{sample}.lib.metrics.txt"
    message: "Library complexity for {wildcards.sample}"
    shell:
        "java -jar /data/home/lingzili/genomeTools/picard.jar EstimateLibraryComplexity I={input} O={ou
```

10) Calculating strand cross-correlation statistics (NSC and RSC) with **phantompeakqualtools** https://github.com/kundajelab/phantompeakqualtools

- Successful ChIP-seq experiment: NSC > 1.05 and RSC > 0.8
- Quality tag is based on thresholded RSC (-2: Very low, -1: Low, 0: Medium, 1: High, 2: Very high)



```
rule phantom_peak:
    input:
        "Bowtie2Output/{sample}.flt.sortByName.bam"
    output:
        phantomTxt = "ATACqcOutput/{sample}.phantom.txt",
        plot = "ATACqcOutput/{sample}.phantom.pdf"
    log:
        "ATACqcOutput/{sample}.phantom.log"
    params:
        run_spp = "/data/home/lingzili/genomeTools/phantompeakqualtools/run_spp.R"
    threads: 10
    message: "Phantom peaks for {wildcards.sample}"
    shell: """
        /data/home/lingzili/.conda/envs/lingzili/bin/Rscript {params.run_spp} -p={threads} -c={input} -
        """
```

11) Distribution of exons, intron, intergenic regions etc. with **read_distribution.py** from RSeQC package http://rseqc.sourceforge.net/#read-distribution-py

```
rule read_distribution:
    input:
        "Bowtie2Output/{sample}.bam"
    output:
        "ATACqcOutput/{sample}.readDist.txt"
    params:
        read_distribution = "/data/home/lingzili/genomeTools/RSeQC-3.0.0/scripts/read_distribution.py",
        RefBED = "/data/home/lingzili/genomeTools/RSeQC-3.0.0/genomeBED/mm10_RefSeq.bed"
    message:
        "Running read distribution on {wildcards.sample}"
    shell:
        "python {params.read_distribution} -i {input} -r {params.RefBED} 1>{output}"
```

12) Summarising the output from analysis logs with **MultiQC** https://multiqc.info/

```
rule multiqc:
    input:
        expand(["trimOutput/{sample}_R1.fastq.gz_trimming_report.txt", "trimOutput/{sample}_R2.fastq.gz
    output:
        "fastqcOutput/multiqc_report.html"
    message: "MultiQC"
    shell:
        "/usr/local/bin/multiqc -o fastqcOutput {input}"
```

## Step 3: Define wildcards

### Define mannually

```
ID = ["SM4492", "SM4493", "SM4494", "SM4495"]
```

### Define by Python scripts

```
extractList = []

for fileName in os.listdir(vds2Dir):
    extractList.append(fileName.split('_')[0])

# Get unique sample ID
ID = list(set(extractList))
```

## Step 4: Define the desired outputs in the first rule

By default snakemake executes the first rule in the snakefile, therefore the beginning of the snakefile can be used to define the desired outputs from the workflow. Snakemake will figure out how to create them by using dependency graph generated from the rules.

```
rule final_results:
    input:
        expand("peakOutput/{sample}.narrowPeak", sample = ID),
        expand("peakOutput/{sample}.bedgraph", sample = ID),
        expand("ATACqcOutput/{sample}.phantom.pdf", sample = ID),
        expand("ATACqcOutput/{sample}.lib.metrics.txt", sample = ID),
        "fastqcOutput/multiqc_report.html"
```

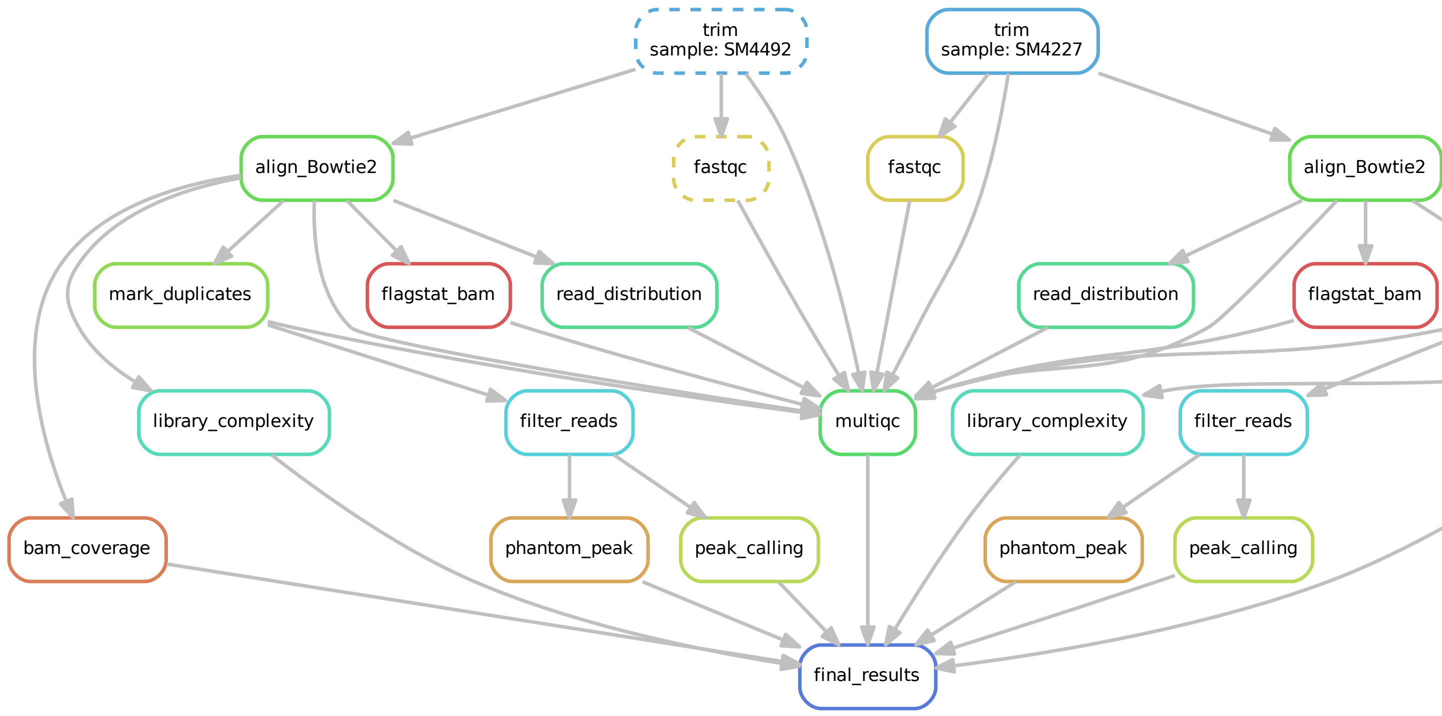


Figure 2: Directed acyclic graph (DAG)

## Step 5: Rehearse snakemake before the real execution (dry-run)

### A dry-run is useful to test if the workflow is defined properly

```
snakemake --snakefile mm10.ATAC.snakefile -n

# To view what is going on:
## -r: print the reason for each executed rule
snakemake --snakefile mm10.ATAC.snakefile -n -r

## -p: print out the shell commands that will be executed
snakemake --snakefile mm10.ATAC.snakefile -n -p

## --quite:
snakemake --snakefile mm10.ATAC.snakefile -n --quiet
```

### Visualization

```
# Print directed acyclic graph
snakemake --snakefile mm10.ATAC.snakefile --dag | dot -Tpdf > dag.pdf

# Print rule graph
snakemake --snakefile mm10.ATAC.snakefile --rulegraph | dot -Tpdf > ruleGraph.pdf
```

In the directed acyclic graph (DAG) below, dashed line indicates completed rule, whereas solid line indicates uncompleted rule.

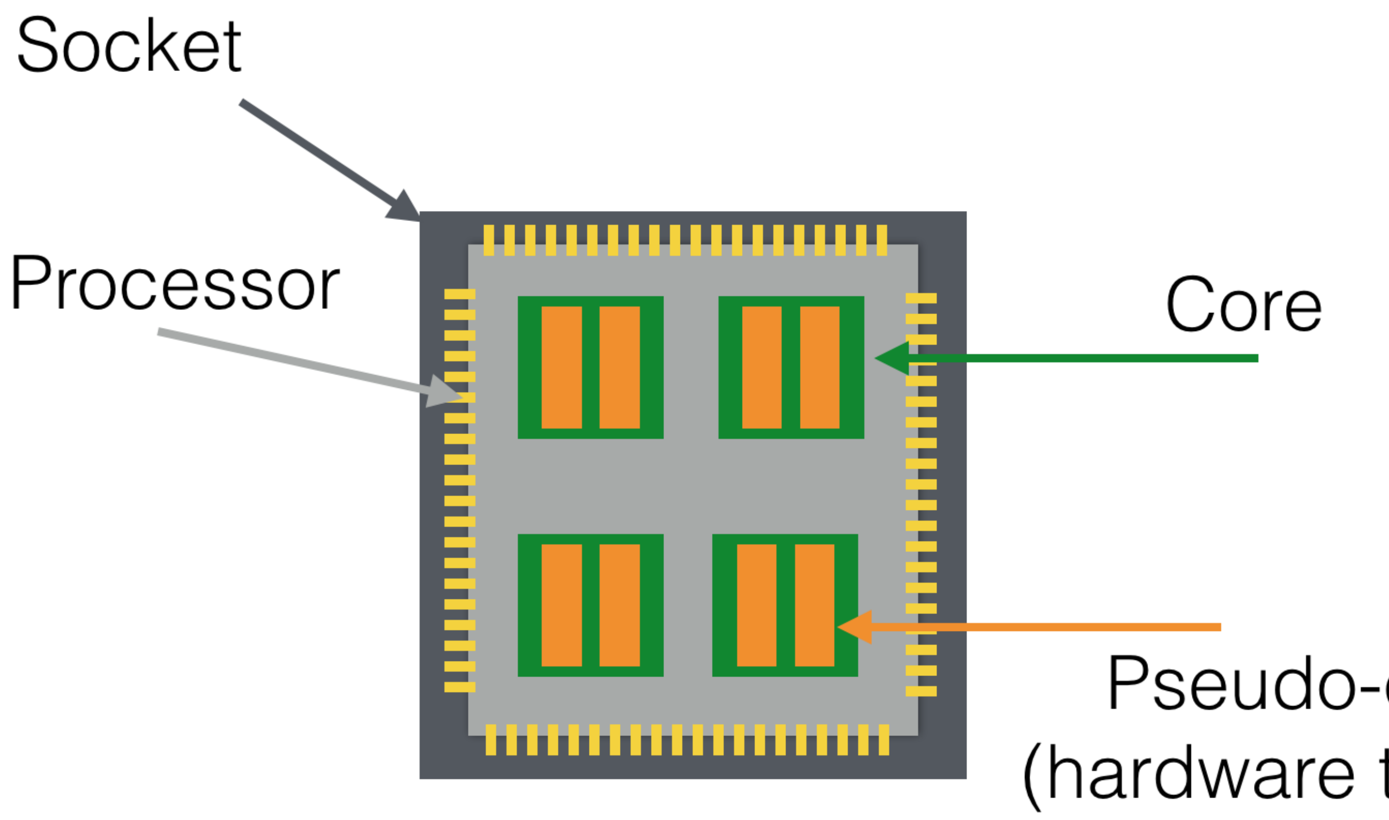


Figure 3: Downloaded from https://github.com/ljdursi/beyond-single-core-R

## Step 6: Execute snakefile

**Parallelism**

Processor (aka socket): Physical unit containing one or more cores that do the computation

Core: Smallest computation unit of the processor, capable of running a single program/task

Thread: Independent computations can occur nearly simultaneously within a single core

```
# Check available CPU cores
lscpu
htop

# To execute workflows in a snakefile in the current working directory
snakemake --snakefile mm10.ATAC.snakefile -j {number of CPU cores allocated to this workflow}
```

**Attention: Rules claiming more threads will be scaled down**

When snakemake doesn't have enough cores to run a rule (as defined by {threads}), Snakemake will run that rule with the maximum available number of cores as defined by snakemake -j.

## Step 7: Reports

Snakemake can generate detailed self-contained HTML reports that encompass runtime statistics, workflow topology and results (additional annotation is needed to include the results).

```
snakemake --snakefile mm10.ATAC.snakefile --report report.html
```

To include results into the report: https://snakemake.readthedocs.io/en/stable/snakefiles/reporting.html

## References

Snakemake-Workflows https://github.com/snakemake-workflows

The Snakemake Wrappers repository https://snakemake-wrappers.readthedocs.io/en/stable/index.html

Snakemake https://snakemake.readthedocs.io/en/stable/index.html

Snakemake—a scalable bioinformatics workflow engine https://academic.oup.com/bioinformatics/article/28/19/2520/290322

Analysis pipelines with Python https://hpc-carpentry.github.io/hpc-python/