Workflows for reproducible, replicable, scalable, and portable science

Mark Adams, The University of Edinburgh

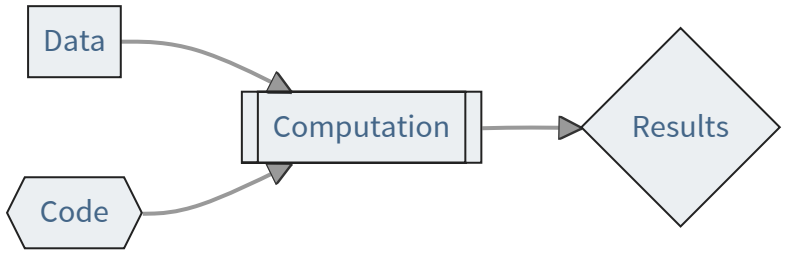
Tech Talk, 13 Aug 2024

✉️ mark.adams@ed.uk
🦣 @markjamesadams@genomic.social
🦋 @markjamesadams.bsky.social
𝕏 @mja

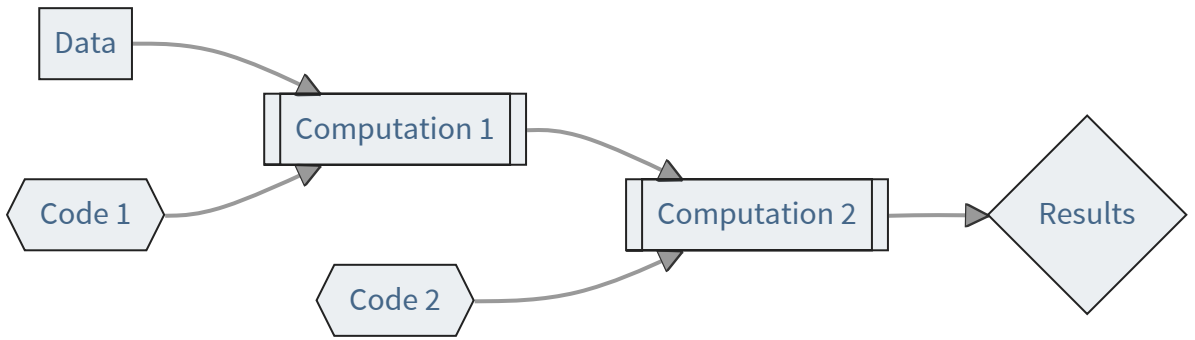
The “-ables” of workflows

- **reproducible**: same data and same code produce the same results
- **replicable**: same code runs with different data
- **scalable**: some code runs with more data and more resources
- **portable**: same code runs in different compute environments

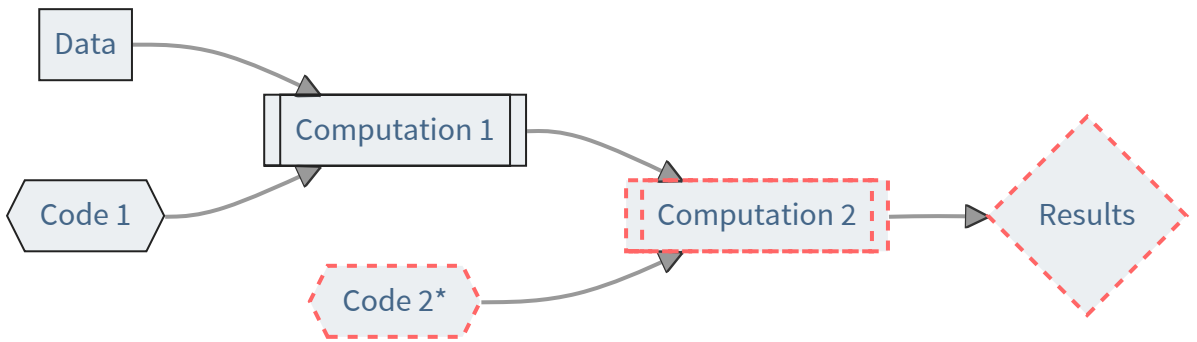
A workflow is a graph



A workflow is a graph



A workflow is a graph



When inputs change, only re-compute descendent outputs.

A workflow is a build system

Collection of notebooks.

```
thesis.qmd    symposium.qmd    workflows.qmd
```

Render them in a loop.

```
$ for QMD in *.qmd; do
    quarto render $QMD
  done
```

Keep it DRY

Only render the notebook if the HTML doesn't exist or the notebook is newer.

```
$ for QMD in *.qmd; do
    PREFIX=$(basename $QMD .qmd)
    if [ ! -e ${PREFIX}.html ] || [ ${PREFIX}.qmd -nt ${PREFIX}.html ]; then
      quarto render $QMD
    fi
  done
```

Makefiles[1]

GNU Make: a tool for building programs from source code.

```
output : inputs
  command
```

Rules encode relationship between inputs (“dependencies”) and outputs (“targets”)

`Makefile` with one rule:

Makefile

```
workflows.html : workflows.qmd
	quarto render workflows.qmd
```

Run `make`

```
$ make workflows.html

quarto render workflows.qmd

processing file: workflows.qmd
1/3
2/3 [unnamed-chunk-1]
3/3
output file: workflows.knit.md
...
Output created: workflows.html
```

```
$ make workflows.html

make: `workflows.html' is up to date.
```

Makefile pattern rules

Pattern rule to render an HTML file from any Quarto notebook.

Makefile

```
%.html : %.qmd
    quarto render $<

all: workflows.html symposium.html thesis.html
```

`all` rule specifies the outputs to render.

Workflows are pipelines

Unix pipes

```
bcftools view --targets-file targets.tsv dbsnp.v153.b37.vcf.gz |\
bcftools query --print-header --format '%CHROM\t%POS\t%ID\t%REF\t%ALT{0}\n' |\
gzip -c > chr_pos_rsid.tsv.gz
```

Scheduler (Sun Grid Engine) dependencies

step1.sh

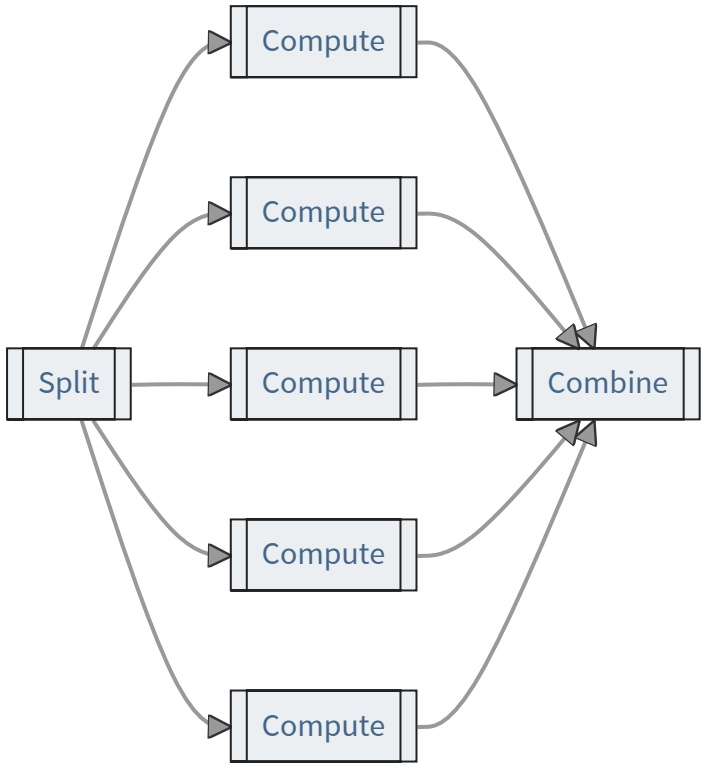
```
1 #$ -cwd
2 command1 --in $1 --out $2
```

step2.sh

```
1 #$ -cwd
2 command2 --in $1 --out $2
```

```
$ qsub -N step1 step1.sh input1 output1
$ qsub -N step2 -hold_jid step1 step2.sh output1 output2
```

Workflows are parallelisable




```
$ make all -j 3

quarto render workflows.qmd
quarto render symposium.qmd
quarto render thesis.qmd
```

Limitations of Makefiles

- No understanding of problem domain (need to run scripts, not just shell commands)
- Not scalable (only executes on local machine, not via HPC or cloud schedulers)
- Limited parallelisation

Limitations of schedulers

- Manually manage connecting outputs to next inputs
- Not portable between HPC environments
- Same workflow can’t be developed and tested on your laptop

Workflow systems

- Provide a language to describe the pipeline
  - Declarative (CWL, WDL, Popper, Remake)
  - General purpose imperative (SciPipe, Dask, targets)
  - Domain specific imperative (bpipe, Snakemake, Nextflow)
- Manage connections and caching of inputs to outputs
- Hook into reproducible software environments (conda, Docker, Singularity)
- Scale to available resources (CPU cores, cluster nodes)

Snakemake and Nextflow

- Domain specific language (DSL)
  - Snakemake: Python
  - Nextflow: Groovy
- Workflow paradigm
  - Snakemake: **rules** determine required inputs from requested outputs
  - Nextflow: **processes** produce outputs from given inputs
- Each workflow step can be a shell command, a script (R, Python, Perl, Julia, etc), or native code [Python/Groovy]

Snakemake

`Snakefile` for rendering a notebook.

Snakefile

```
rule render:
    input: "{notebook}.qmd"
    output: "{notebook}.html"
    shell: "quarto render {input}"
```

- `{notebook}` is an input/output wildcard.
- in the shell command, `{input}` is replaced with the value of `{notebook}.qmd`

Snakemake

Ask Snakemake to produce the file `workflows.html`. Filename is matched to an output pattern, then built from the complementary input file.

```
$ snakemake -j1 workflows.html

Assuming unrestricted shared filesystem usage.
Building DAG of jobs...
Using shell: /bin/bash
Provided cores: 1 (use --cores to define parallelism)
Rules claiming more threads will be scaled down.
Job stats:
job       count
------  -------
render        1
total         1

Select jobs to execute...
Execute 1 jobs...

[Wed Aug  7 07:10:02 2024]
```

```
localrule render:
```

Snakemake

Rerun the workflow

```
snakemake -j1 workflows.html

Assuming unrestricted shared filesystem usage.
Building DAG of jobs...
Nothing to be done (all requested files are present and up to date).
```

Snakemake

Ask the workflow for multiple files.

```
$ snakemake -j1 workflows.html symposium.html thesis.html

Assuming unrestricted shared filesystem usage.
Building DAG of jobs...
Using shell: /bin/bash
Provided cores: 1 (use --cores to define parallelism)
Rules claiming more threads will be scaled down.
Job stats:
job       count
------  -------
render        2
total         2

Select jobs to execute...
Execute 2 jobs...
```

Nextflow

Pipeline is defined as channels that flow into and out of processes.

notebooks.nf

```
params.notebook = "*.qmd"

workflow {
  QMD_CH = Channel.fromPath(params.notebook)
  HTML_CH = RENDER(QMD_CH)
}

process RENDER {

  publishDir "."

  input:
  path qmd

  output:
  path("${qmd.baseName}.html")

```

Nextflow

Run workflow process by specifying the input parameters.

```
$ nextflow run notebooks.nf --notebook workflows.qmd -resume

N E X T F L O W  ~  version 22.10.6
Launching `notebooks.nf` [lonely_jang] DSL2 - revision: 02a9f1b9b3
executor >  local (1)
[c8/e651a4] process > RENDER (1) [100%] 1 of 1 ✔
```

Rerun the workflow

```
nextflow run notebooks.nf --notebook workflows.qmd -resume

N E X T F L O W  ~  version 22.10.6
Launching `notebooks.nf` [reverent_venter] DSL2 - revision: 02a9f1b9b3
[c8/e651a4] process > RENDER (1) [100%] 1 of 1, cached: 1 ✔
```

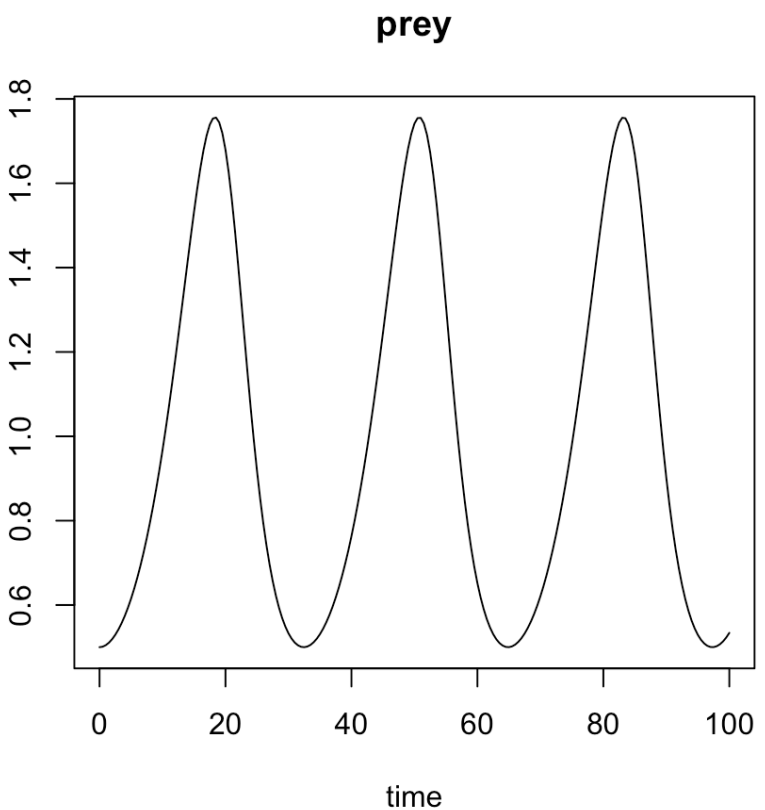
Nextflow

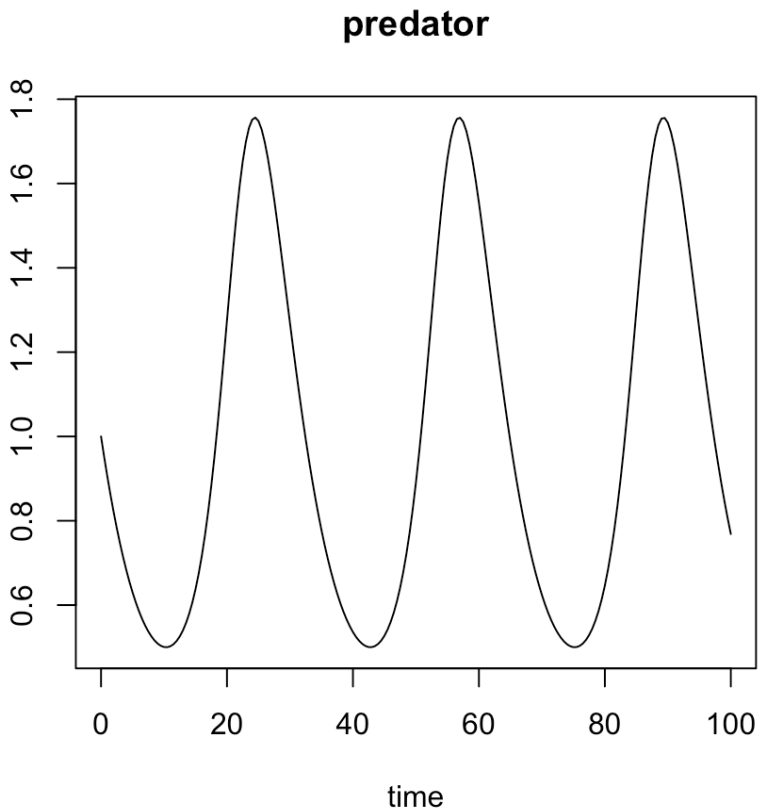
Run workflow by specifying multiple files for the input parameter.

```
$ nextflow run -resume notebooks.nf --notebook "{workflows,symposium,thesis}.qm

N E X T F L O W  ~  version 22.10.6
Launching `notebooks.nf` [pedantic_heisenberg] DSL2 - revision: 02a9f1b9b3
executor >  local (2)
[9e/fcbb56] process > RENDER (1) [100%] 3 of 3, cached: 1 ✔
```

Working with scripts

Example script that simulates a Lotka-Volterra model





Snakemake

Snakefile

```
rule lv:
    output: "lv/prey{prey}-predator{pred}.png"
    script: "scripts/lv.R"
```

Snakemake passes a special `snakemake` object to the script.

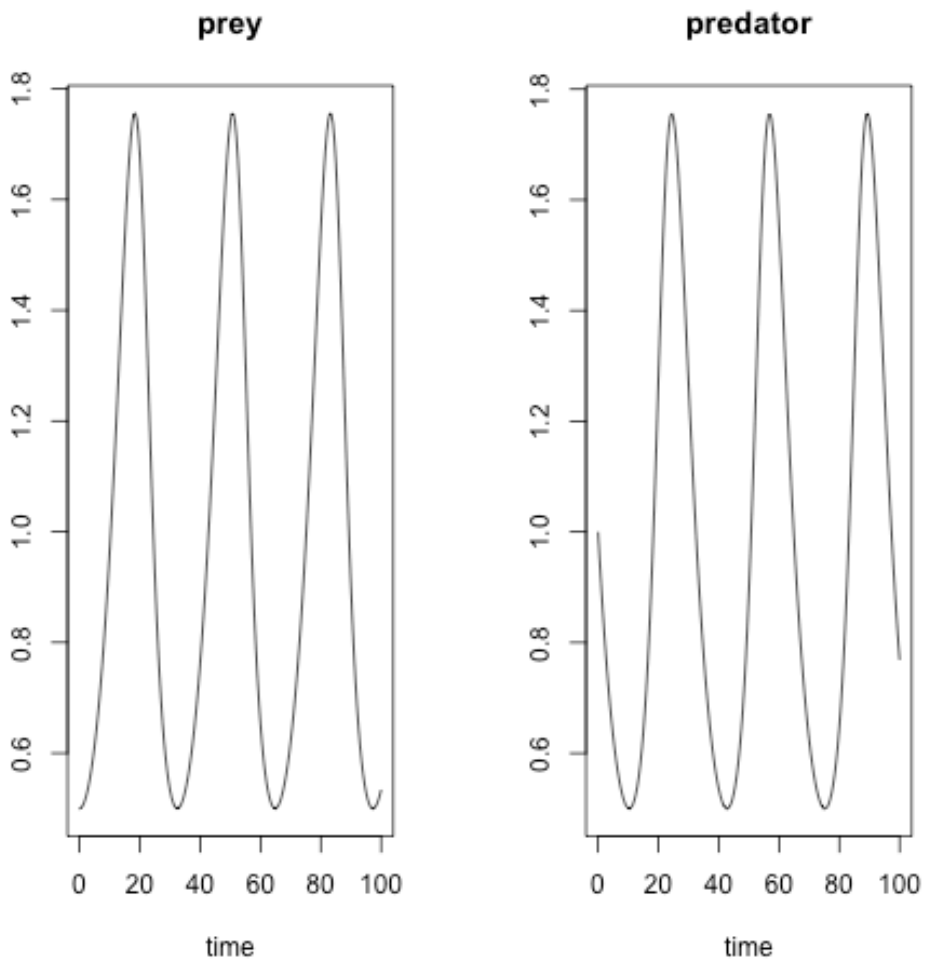
scripts/lv.R

```
library(simecol)

# parameterise simulation
# get parameter values from file wildcards
# prey and predator initial values
data(lv)
init(lv) <- c(prey = as.numeric(snakemake@wildcards$prey),
              predator = as.numeric(snakemake@wildcards$pred))

# run simulation
simObj <- sim(lv)

# make and save plot to specified output
png(snakemake@output[[1]])
```

```
plot(simObj)
dev.off()
```

Snakemake

```
$ snakemake -j1 lv/prey0.5-predator1.0.png
```



Nextflow

lv.nf

```
workflow {
  PREY_CH = Channel.of(params.prey)
  PRED_CH = Channel.of(params.pred)
  LV(PREY_CH, PRED_CH)
}

process LV {

  publishDir 'lv', mode: 'copy'

  input:
  val prey
  val pred

  output:
  path("prey${prey}-pred${pred}.png")

  script:
  """
```

Working with resources

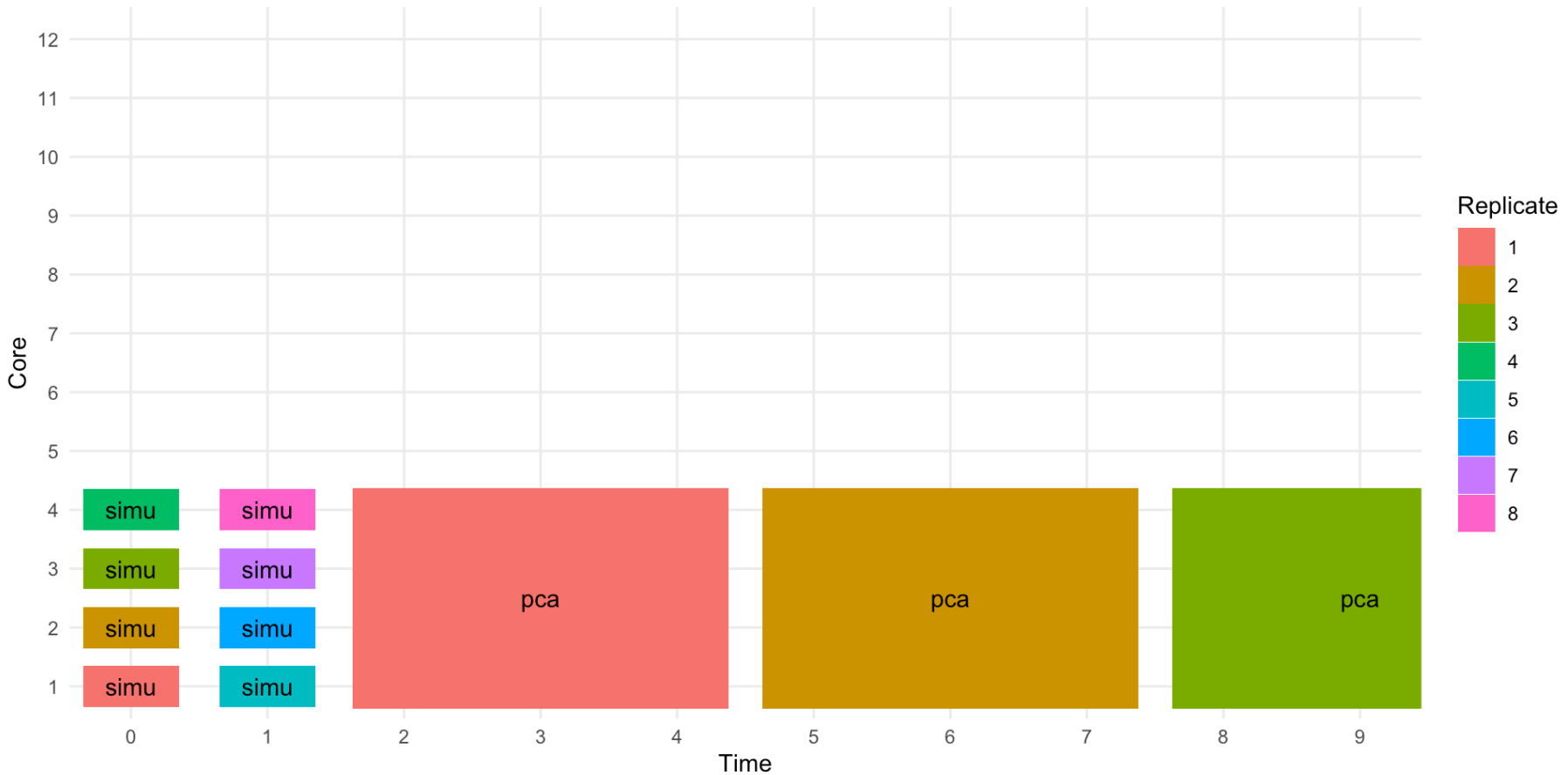
Snakefile

```
# Simulate 100k genotypes for 2k samples
rule simu:
  output: multiext("sim/{bfile}", ".pgen", ".psam", ".pvar")
  threads: 1
  resources:
    mem_mb=1000
  shell: """
  plink2 --dummy 2000 100000 --out sim/{wildcards.bfile} \
  --threads {threads} --memory {resources.mem_mb}
  """

# run principal components analysis on genetic data
rule pca:
  input: multiext("sim/{bfile}", ".pgen", ".psam", ".pvar")
  output: multiext("pca/{bfile}", ".eigenval", ".eigenvec")
  threads: 4
  resources:
    mem_mb=8000
  shell: """
```

Generate and analyze 8 simulated datasets, utilising **4** cores

```
$ snakemake -c4 pca/sim-{1,2,3,4,5,6,7,8}.{eigenvec,eigenval}
```

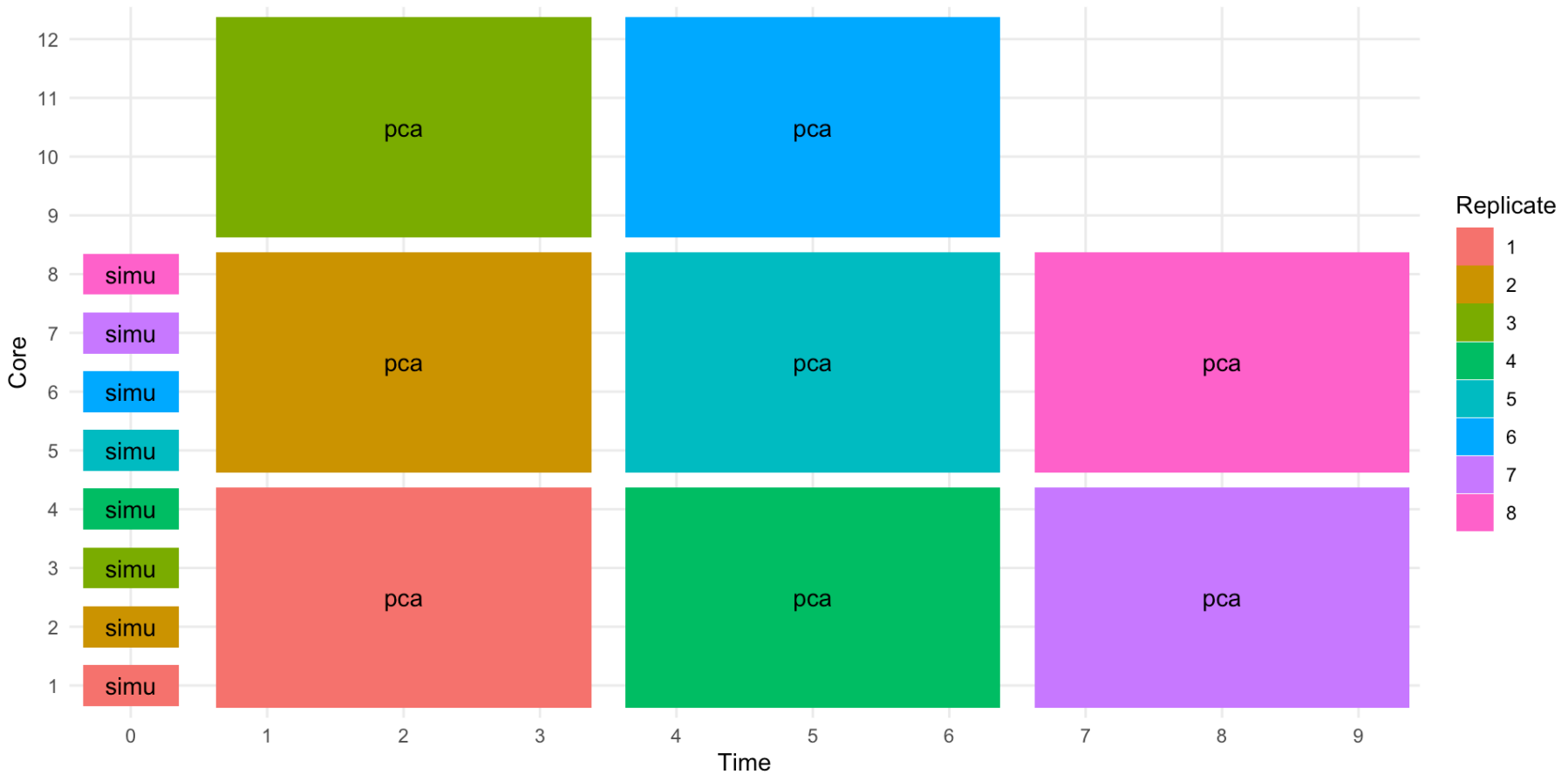
Core occupancy over time



Generate and analyze 8 simulated datasets, utilising **12** cores

```
$ snakemake -c12 pca/sim-{1,2,3,4,5,6,7,8}.{eigenvec,eigenval}
```

Core occupancy over time



Schedulers

Amazon Web Services, Azure, Google Cloud, HTCondor, Kubernetes, LSF, Torque, SGE, SLURM, etc

- Snakemake:
  - Profiles: github.com/Snakemake-Profiles
  - Plugins: snakemake.github.io/snakemake-plugin-catalog/
- Nextflow
  - Executors: nextflow.io/docs/latest/executor.html
  - Institution cluster configs: github.com/nf-core/configs

Working with schedulers

```
$ snakemake -j1 --profile sge pca/sim-{1,2,3,4,5,6,7,8}.{eigenvec,eigenval}

Building DAG of jobs...
Using shell: /usr/bin/bash
Provided cluster nodes: 100
Job stats:
job      count
-----  -------
pca          8
simu         8
total       16

Select jobs to execute...
```

Working with software environments

Package managers, containers, and modules

Snakefile

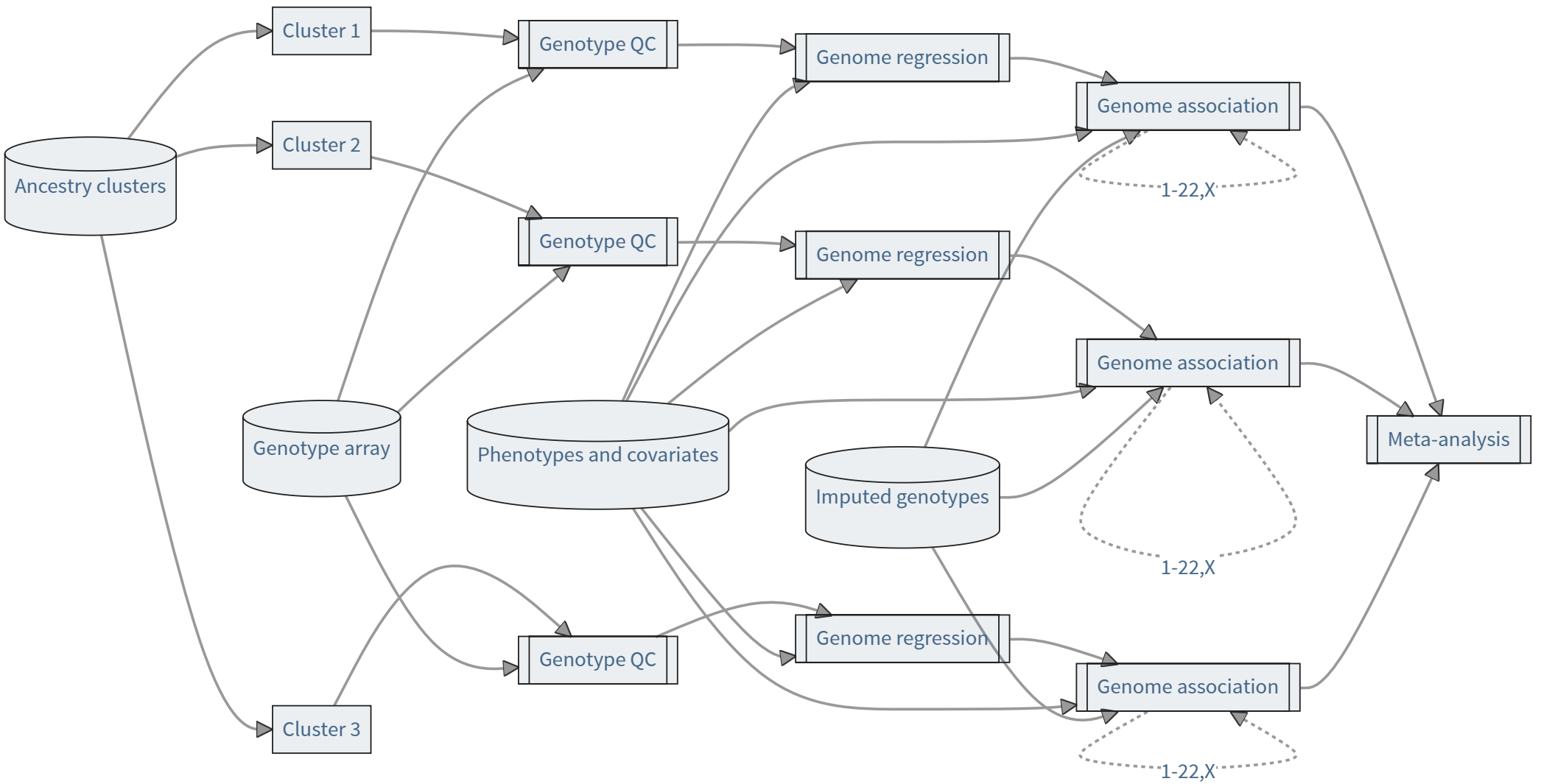
```
rule pca:
  conda:
    "envs/plink.yaml"
  container:
    "quay.io/biocontainers/plink2:2.00a2.3--hf22980b_0"
  envmodule:
    "igmm/apps/plink/2.00a3LM"
```

envs/plink.yaml

```
channels:
  - conda-forge
  - bioconda
dependencies:
  - plink2 =2.00a2.3
```

Examples

Multi-ancestry genetic association study (UK Biobank, All of Us, etc)



Inputs

ukb-regenie-hrc.nf

```
params.bt = null // binary phenotypes file
params.qt = null // quantitative phenotypes file
params.keep = "rf_hgdp1kg_clusters.keep"
params.remove = "PGC.remove"
params.bfile = "autosome.{bed,bim,fam}"
params.pfile = "ukb_imp_v3.qc.{pgen,psam,pvar}"
params.clusters = "ukb_randomforest_clusters.tsv"
params.covar = "ukb_randomforest_clusters.covar"
params.covar_list = "PC1,PC2,PC3,PC4,PC5,PC6"
params.covar_cat_list = "sex,genotyping"
params.min_cases = 80
```

Parse inputs from CSV/TSV file

ukb-regenie-hrc.nf

```
// genetic similarity clusters
CLUSTERS_CH = Channel
  .fromPath(params.clusters, checkIfExists: true)

// parse cluster file to get names of each cluster
CLUSTER_NAMES_CH = CLUSTERS_CH
        .splitCsv(sep: "\t", skip: 1, header: ['fid', 'iid', 'cluster'])
        .map { it -> it.cluster }
        .unique()
```

- `splitCsv()`: parse delimited file. Each row becomes an item.
- `map()`: get value of `cluster` column for each item.
- `unique()` output unique items.

Perform genotype QC for each ancestry cluster

ukb-regenie-hrc.nf

```
// Genotype QC
BFILE_CLUSTERS_CH = BFILE_CH
  .combine(KEEP_CH)
  .combine(REMOVE_CH)
  .combine(CLUSTERS_CH)

QC_CH = QC(BFILE_CLUSTERS_CH, CLUSTER_NAMES_CH)
```

ukb-regenie-hrc.nf

```
process QC {
    tag "QCing genotypes ${cluster}"

    cpus = 1
    memory = 16.GB
    time = '1h'

    input:
    tuple val(bfile), path(bedbimfam), path(keep), path(remove), path(clust
    each cluster

    output:
    tuple val(cluster), path("${cluster}.snplist"), path("${cluster}.id")
```

Prepend key to phenotype channels

ukb-regenie-hrc.nf

```
// binary
if(params.bt != null) {
  BT_CH = Channel
    .fromPath(params.bt, checkIfExists: true)
    .map { it -> ["bt", it] }
} else {
  BT_CH = Channel.empty()
}

// quantitative
if(params.qt != null) {
  QT_CH = Channel
    .fromPath(params.qt, checkIfExists: true)
    .map { it -> ["qt", it] }
} else {
  QT_CH = Channel.empty()
}
```

Specify different option flags for each phenotype

ukb-regenie-hrc.nf

```
STEP1_FLAGS_CH = Channel
  .of(["bt", "--bt --minCaseCount ${params.min_cases}"],
    ["qt", ""]
  )

STEP2_FLAGS_CH = Channel
  .of(["bt", "--bt --af-cc --firth --approx --pThresh 0.01 --minCaseCount $
    ["qt", ""]
  )

FLAGS_CH = STEP1_FLAGS_CH
  .join(STEP2_FLAGS_CH)
  .map { it -> [ it[0], ["step1": it[1], "step2": it[2]] ] }
```

→

```
  [["bt"] ["step1": "--bt --minCaseCount 80", "step2": "--bt --af-cc --firth -
-approx --pThresh 0.01 --minCaseCount 80"]]
  [["qt"] ["step1": "", "step2": ""]]
```

Combine phenotypes, program flags, and other input files

ukb-regenie-hrc.nf

```
    PHENO_FLAGS_BFILE_CH = PHENO_COUNT_CH
        .combine(FLAGS_CH, by: 0)
        .combine(BFILE_CH)
        .combine(QC_CH)
        .combine(COVAR_CH)
        .combine(COVAR_LIST_CH)
        .combine(COVAR_CAT_LIST_CH)
```

Dynamic resource allocation

Genome regression step

ukb-regenie-hrc.nf

```
process STEP1 {
    tag "${cluster}-${traits}-${pheno}"

  // increase time on each attempt
  // allocate 16GB of memory for each phenotype
    cpus = 8
    memory = { 8.GB + n_pheno * 16.GB }
    time = { 6.hour * task.attempt }

    input:
    tuple val(cluster), val(traits), val(pheno), val(n_pheno), path(phenos)
```

Parallelise across chromosomes

Genome association step, then merge

ukb-regenie-hrc.nf

```
CHR_CH = Channel.of(1..23)
STEP2_CH = STEP2(STEP1_FLAGS_PFILE_CH, CHR_CH)
GWAS_CH = MERGE(STEP2_CH)
```

ukb-regenie-hrc.nf

```
process STEP2 {
    tag "${cluster}-${traits}-${pheno}"

    cpus = 8
    memory = 16.GB
    time = '24h'

    input:
    tuple val(cluster), val(traits), val(pheno), path(phenos), val(flags),
    each chr

    output:
    tuple val(cluster), val(traits), val(pheno), path("step2_${cluster}-${t
```

Multiple levels of paralellisation

- Each input file can contain multiple phenotypes
- Workflow accepts multiple phenotype files
- Separate analysis for each ancestry cluster
- Separate analysis for each chromosome
- Regression/association steps run across multiple cores

Process UK Biobank release

Turns UKB release object into a DuckDB database

github.com/mja/ukb-release-nf

```
$ nextflow run duckdb.nf -c custom.config -resume --enc ukb12345.enc --key k123
```

- Decrypts release file
- Downloads data dictionary
- Determines which fields are in the release file
- Converts fields for each category
- Processes fields through R
- Combines tables into SQL database

Major depressive disorder GWAS meta-analysis

From the Psychiatric Genomics Consortium

github.com/psychiatric-genomics-consortium/mdd-wave3-meta

Determine inputs/outputs from configuration file

config.yaml

```yaml
sumstats:
  daner_mdd_GenScot.eur.hg19.SCID_0721a: data/sumstats/daner_mdd_genscot.SC
  text_mdd_FinnGen.eur.hg38.R5_18032020: data/sumstats/Depressio_FinnGen_R5
  text_mdd_UKBB.eur.hg19.MD_glm_202107:  data/sumstats/ukb_mdd.202107.md.eu
```

meta.smk

```
# Copy summary statistics listed in config.yaml under sumstats
# with key FORMAT_COHORT.POP.hgNN.RELEASE
rule stage_sumstats:
    input: lambda wildcards: config["sumstats"][wildcards.cohort]
    output: "resources/sumstats/{cohort}.gz"
    log: "logs/sumstats/stage/{cohort}.log"
    shell: "ln -sv {input} {output} > {log}"

# Harmonize names of all summary statistics listed under sumstats in config
rule sumstats:
    input: expand("resources/sumstats/{sumstats}.gz", sumstats=config["sums
```

Conclusions

- Workflow as a record of analysis steps: **reproducibility**
- Workflow as a re-runnable pipeline: **replicability**
- Workflow as a resource manager: **scalability**
- Workflow as a compute environment: **portable**

Code for this talk: github.com/mja/tech-talk-workflows

List of pipeline frameworks:
github.com/pditommaso/awesome-pipeline