Quadram Institute
Science Health Food Innovation

A quick intro to Nextflow DSL2

Andrea Telatin, PhD
Core Bioinformatics

Programme

- Our typical bioinformatics workflow
- What is Nextflow
- and how is helping us making better workflows
- Overview of the practical
- Hands on

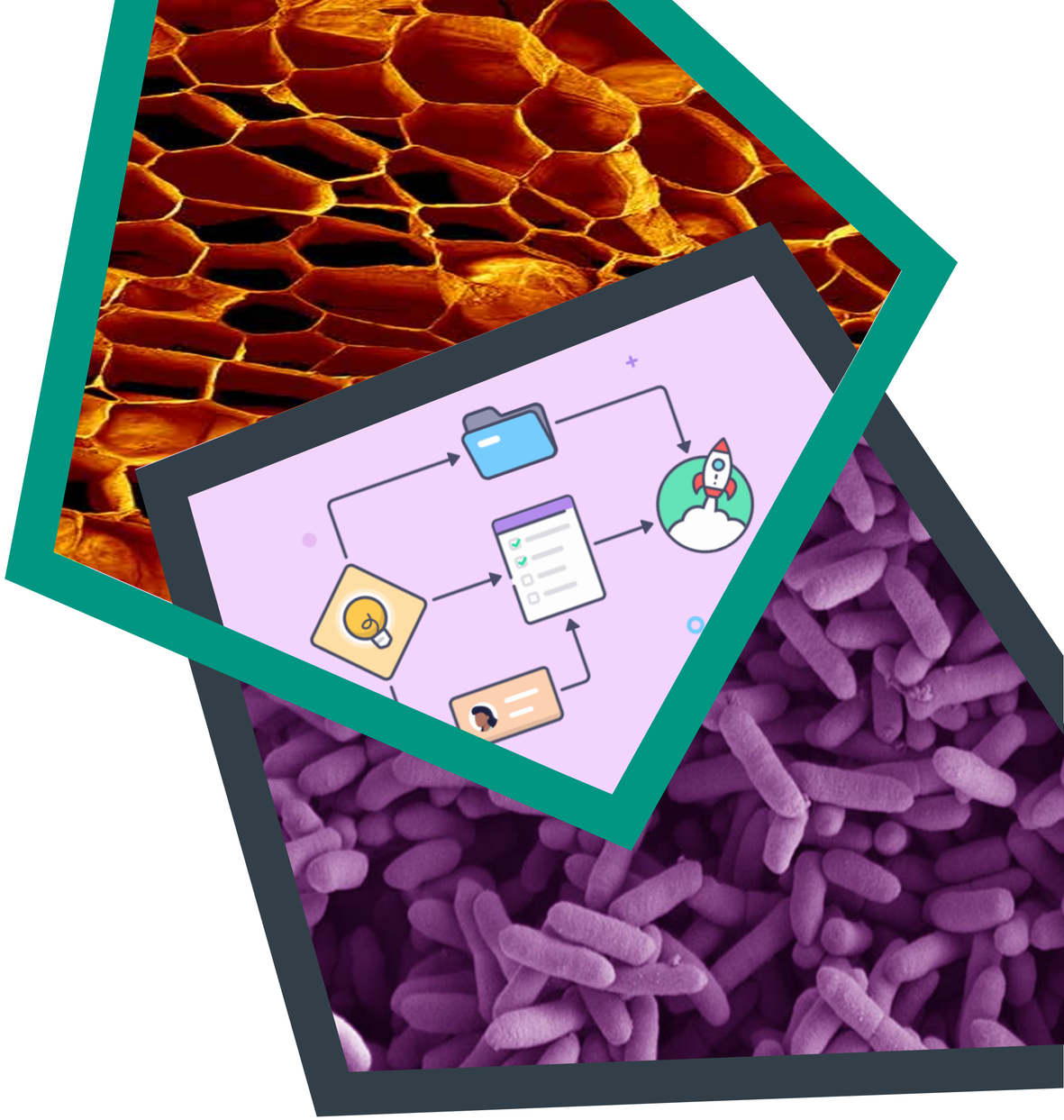
https://github.com/telatin/nextflow-example

Goals

- To raise awareness on the importance of workflow managers
- To spark interest on Nextflow
- To get some of you motivated to try it in the future

https://github.com/telatin/nextflow-example

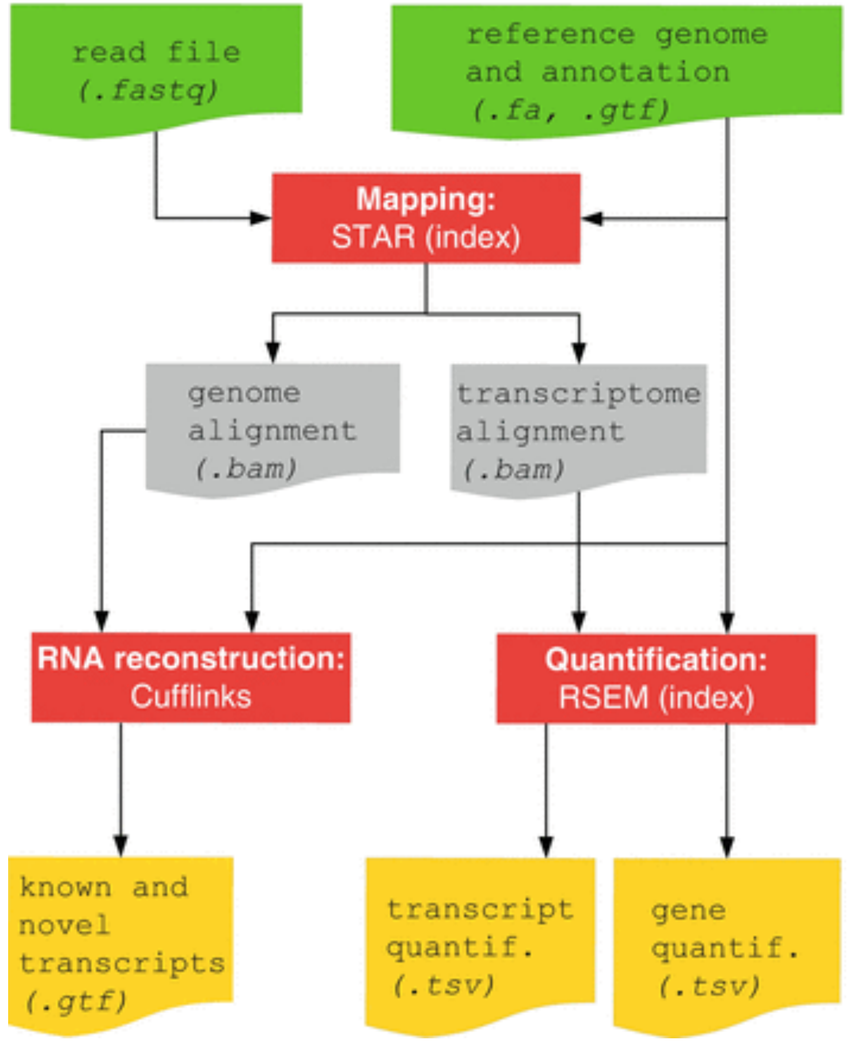
Quadram Institute
Science Health
Food Innovation



our typical bioinformatics workflow

What do we do
typical workflow

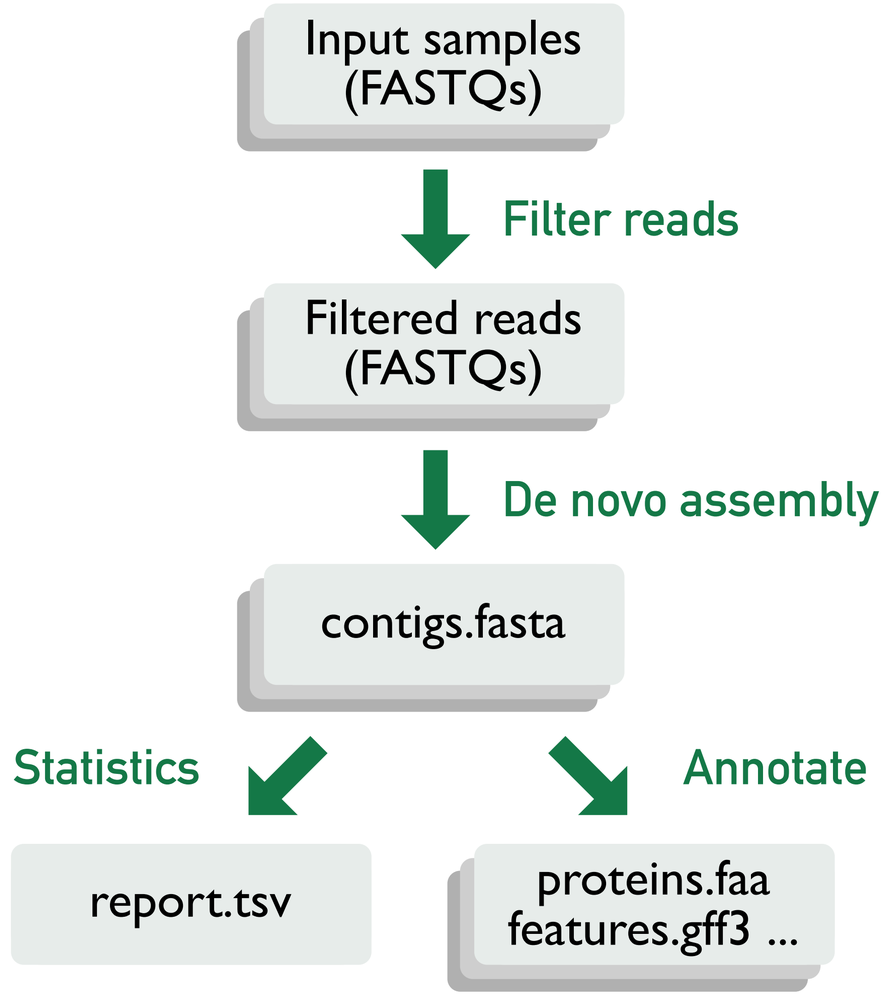
- We use multiple tools
  (like bwa, spades, prokka, …)
- We process multiple input files
  (tipically a set of FASTQ files)
- We often have databases
  (reference genome, kraken …)
- We have specific computation requirements
  (memory, cpus…)



Bioinformatics Pipeline for Transcriptome Sequencing Analysis
https://doi.org/10.1007/978-1-4939-4035-6_14

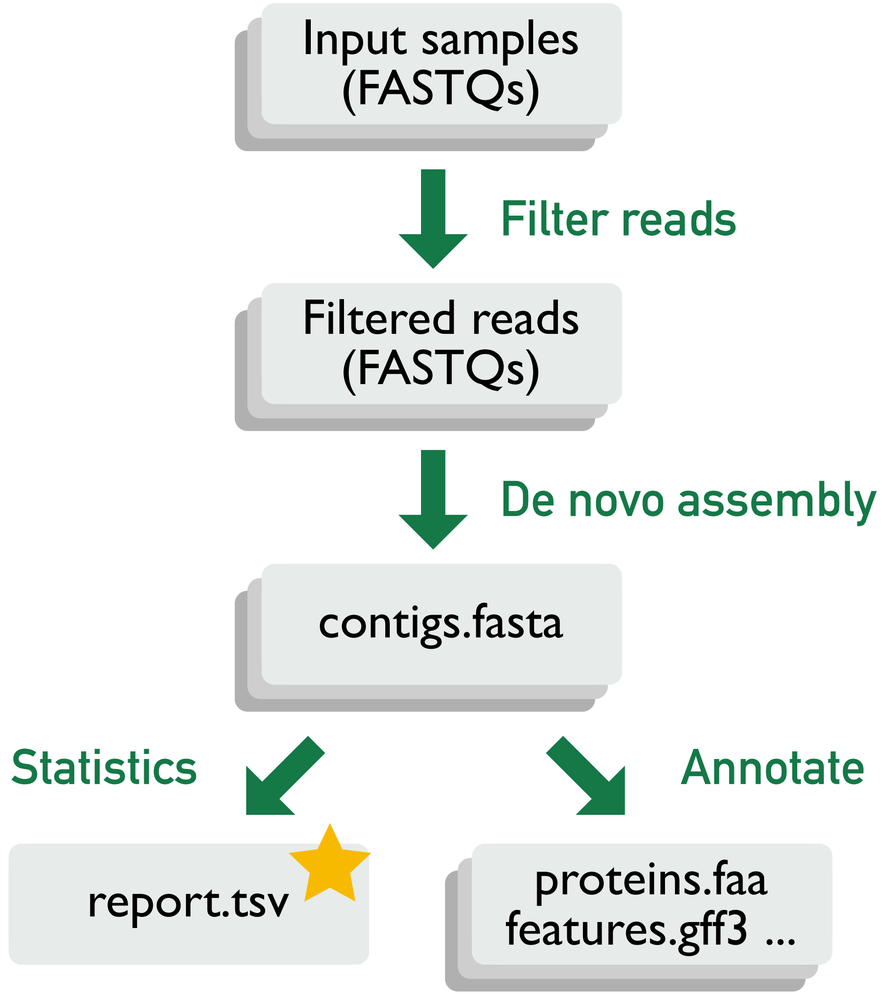
Writing a script: limitations

- **Dependencies:** No automatic management (we can use containers)
- **Fault tolerance:** What happens if a process fails?
- **Portability**: how can we use our script in a different environment? (HPC with slurm, VM, …)
- **Readability**: Will my code concisely describe the workflow structure?



Our "*de novo*" pipeline

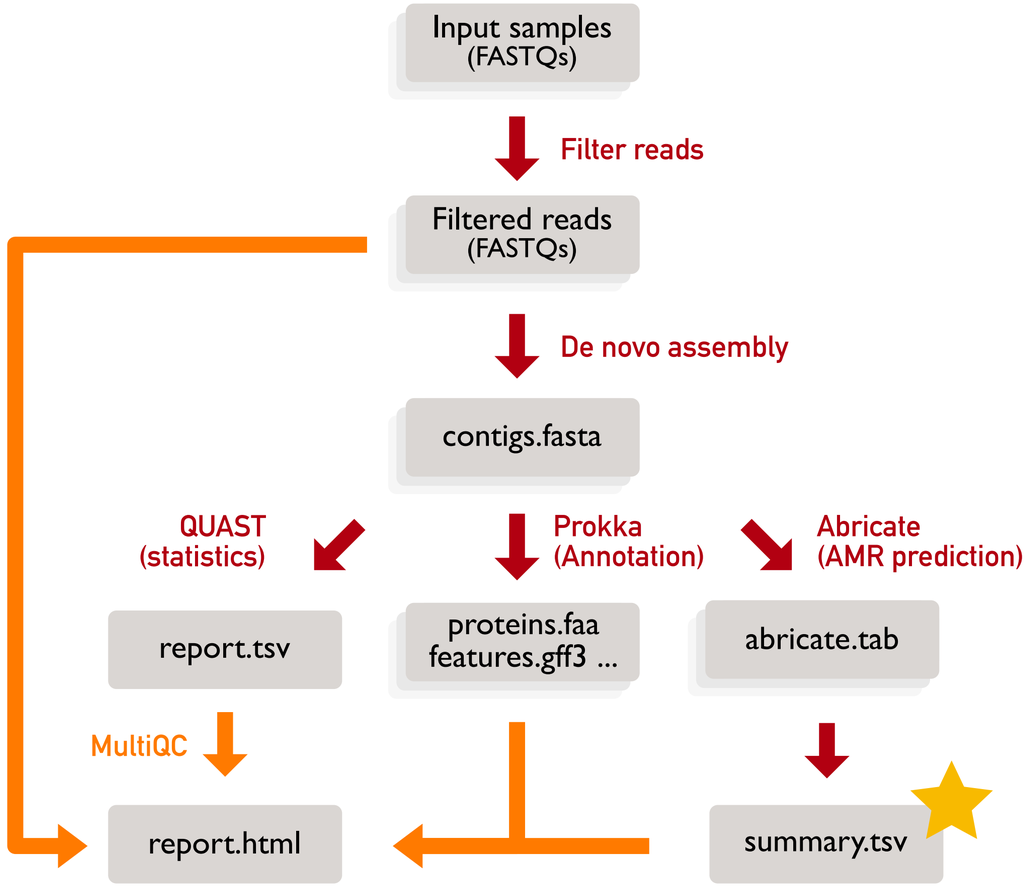
- **Dependencies:** a set of tools available from BioConda (fastp, skesa, prokka …)
- **Input:** a set of paired-end FASTQ files, from a WGS of isolates/plasmids.
- For each sample (pair of files): filter reads, assemble, annotate.
  Finally gather all the assemblies to produce a summary of statistics (Quast)⭐



Bioinformatics Pipeline for Transcriptome Sequencing Analysis
https://doi.org/10.1007/978-1-4939-4035-6_14

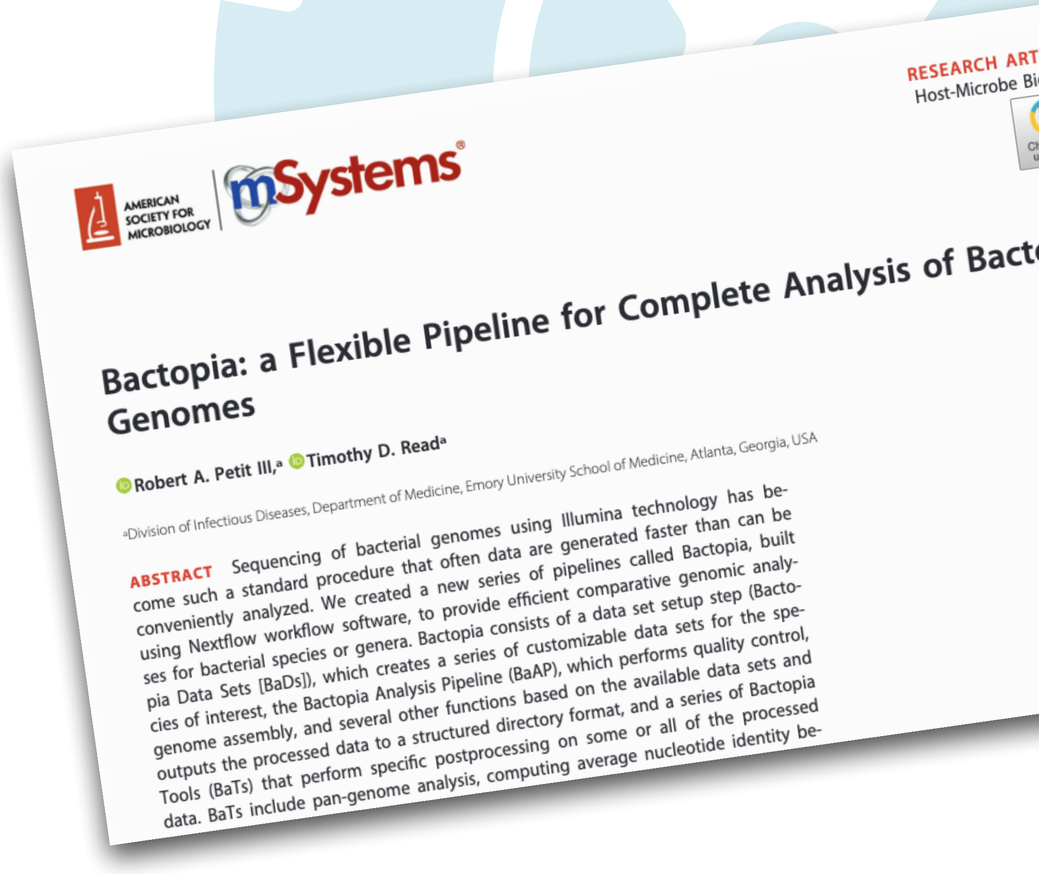
Extending the “De novo” pipeline

- **AMR Prediction:** Abricate on every assembly, and a summary step collecting all the abricate tables
- **MultiQC:** aggregate all informations and produce an HTML report. MultiQC supports a lot tools natively (including Quast, Prokka and Fastp).
- We can convert the abricate summary in a “MultiQC compatible” table, using a **custom script**.



We are not going to reinvent the wheel

- Our example pipeline is a good example of a typical workflow, and definitely an important task to address
- But… there are good pipelines, and one is great, constantly improved and **based on Nextflow, now DSL2**
- Check bactopia.github.io



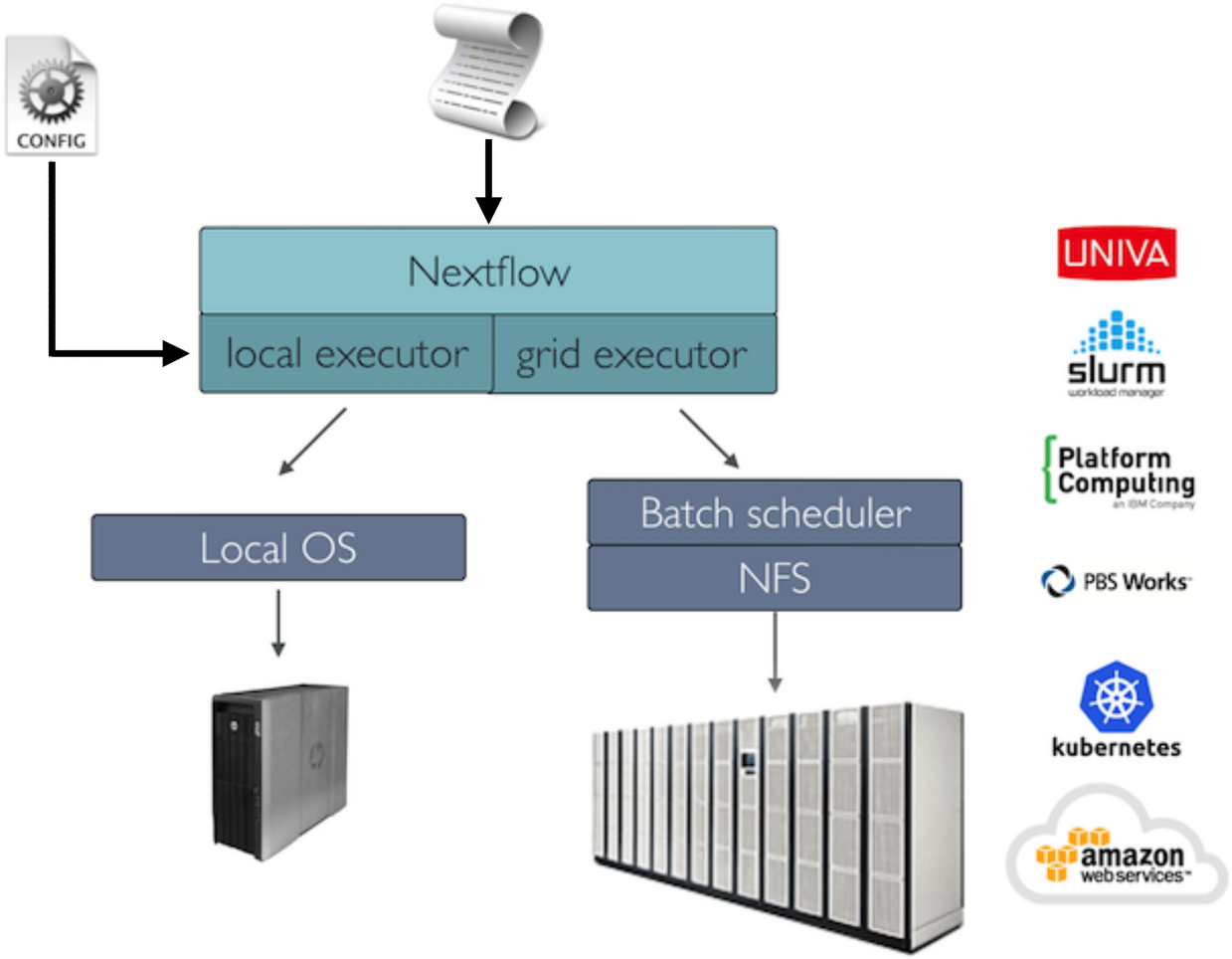
Quadram Institute

Science Health
Food Innovation

Nextflow:
what is it?

In short

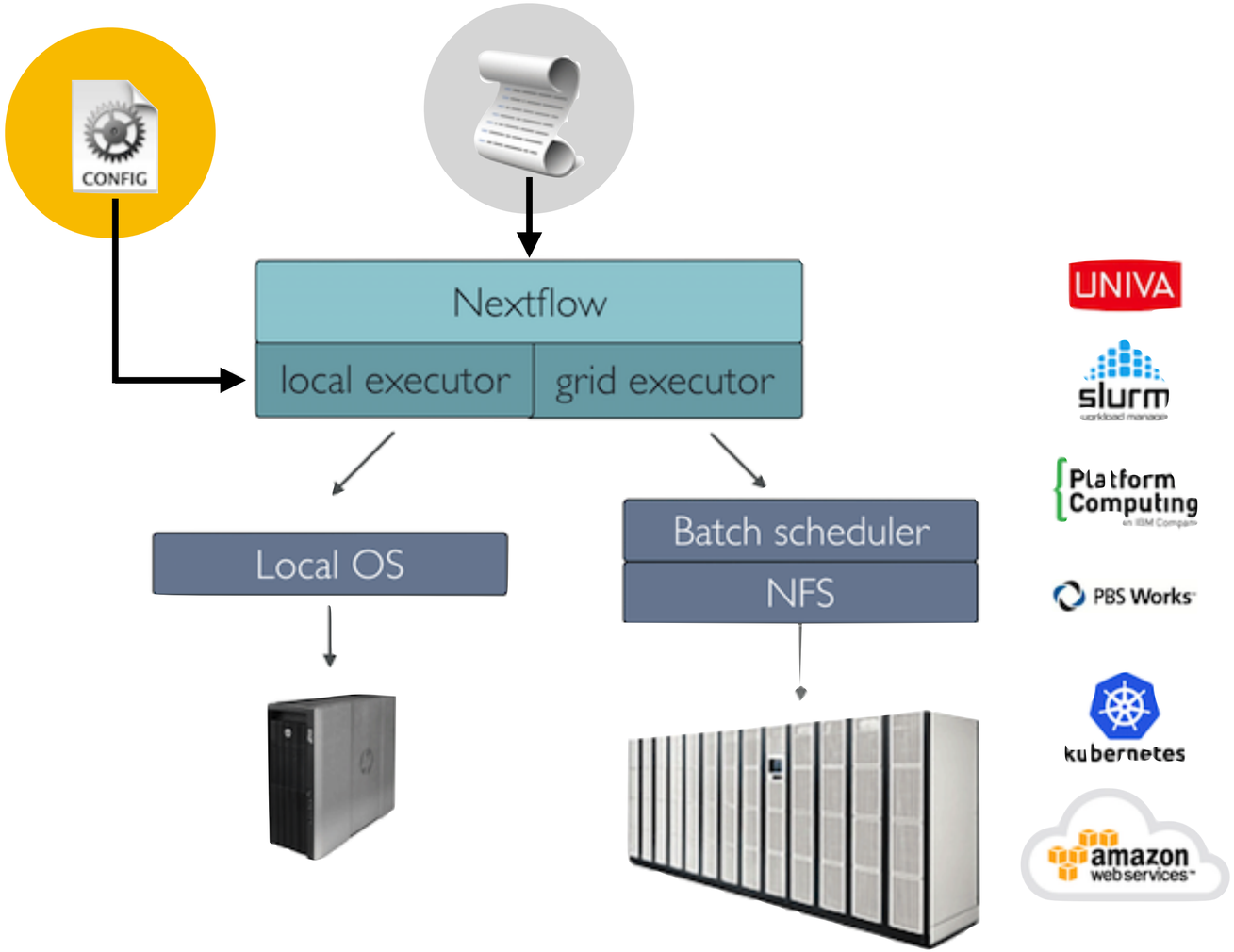
- A **workflow language** (DSL) built upon Groovy
- A **runtime environment** acting as task orchestrator
- Enable separation of workflow logic (what to do) and configuration (how to run)



In short

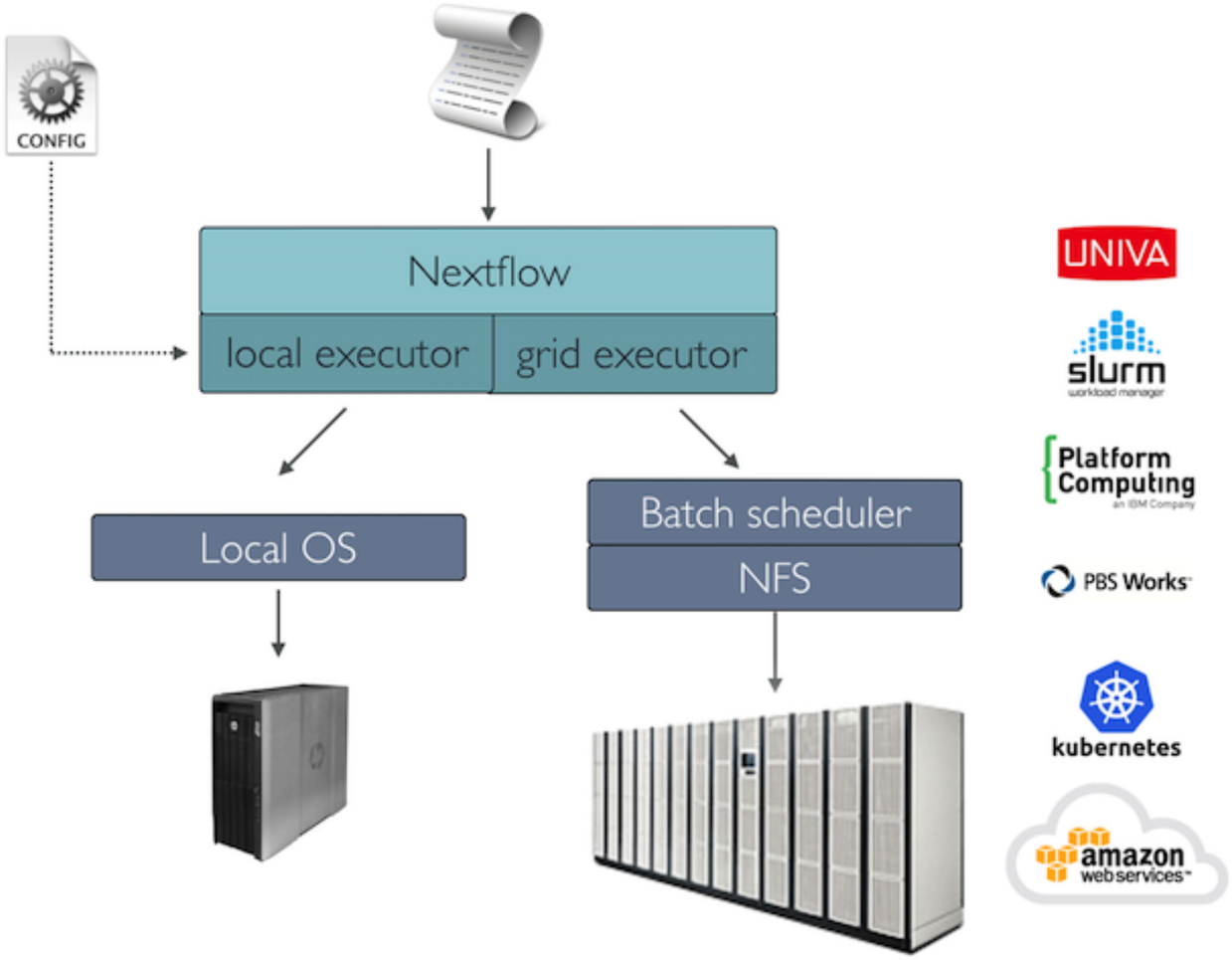
- In practice we have to learn how to write:
  - the workflow logic
  - the configuration

But the configuration will be the same for most of our scripts



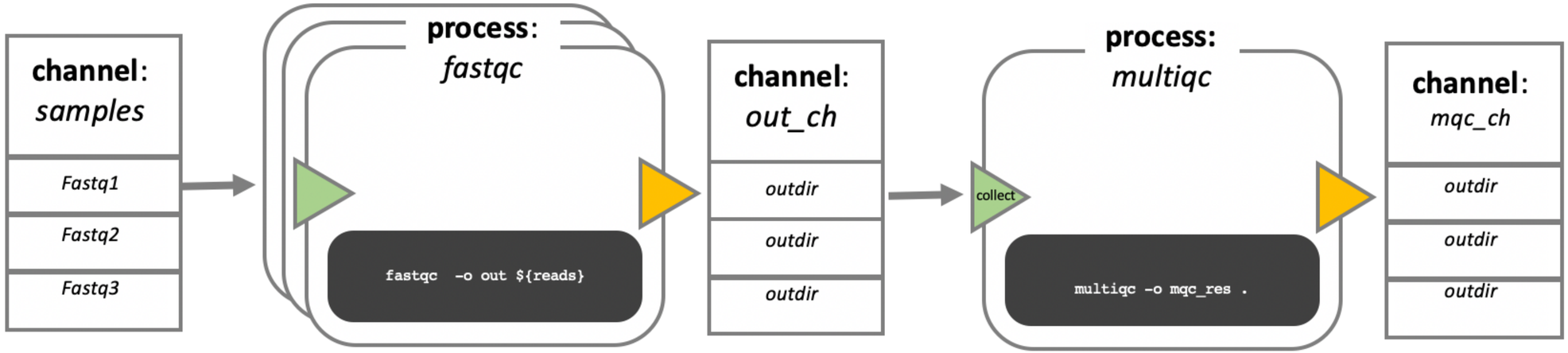
A big caveat

- Nextflow has been around for years with a syntax (DSL1)
- A completely redesigned language (DSL2) is out and totally worth embracing:
    - modularity
    - code reuse



The core concepts

- **process:** the description of a task to be run (*e.g.* assembly with spades)
- **channel:** asynchronous queues (*e.g.* the samples to be analyzed)
- **workflows:** a set of processes and channels



https://carpentries-incubator.github.io/workflows-nextflow/figures/index.html

nextflow *pipeline*

Write code
in any language

Orchestrate tasks with
dataflow programming

Define software
dependencies
via containers

Built-in version
control with Git

Git GitHub GitLab Bitbucket

nextflow *runtime*

Task orchestration
and execution

Supported Platforms

aws Azure GRID ENGINE slurm HTCondor Platform Computing kubernetes PBS Works

https://seqera.io/training/#_execution_abstraction

Some benefits

- Built-in support of **containers** (docker, singularity…)
- Built-in support of **HPC schedulers** (Slurm, PBS…)
- **Scalability** (Cloud providers like Amazon, Google Cloud…)
- Easier **reproducibility**, **distribution**, built-in **versioning** of pipelines
- **Community**

Powerful configuration

- Configuration files can be bundled with the pipeline (workflow-specific)
- User settings can be saved in the home
- “Institute” settings to describe the premises (e.g. HPC configuration)
- Any configuration file can be specified at runtime with -c FILE

nextflow.config

Default ’base’ config (always loaded)

Core profiles (e.g. docker, conda, test)

Server profiles (nf-core/config)

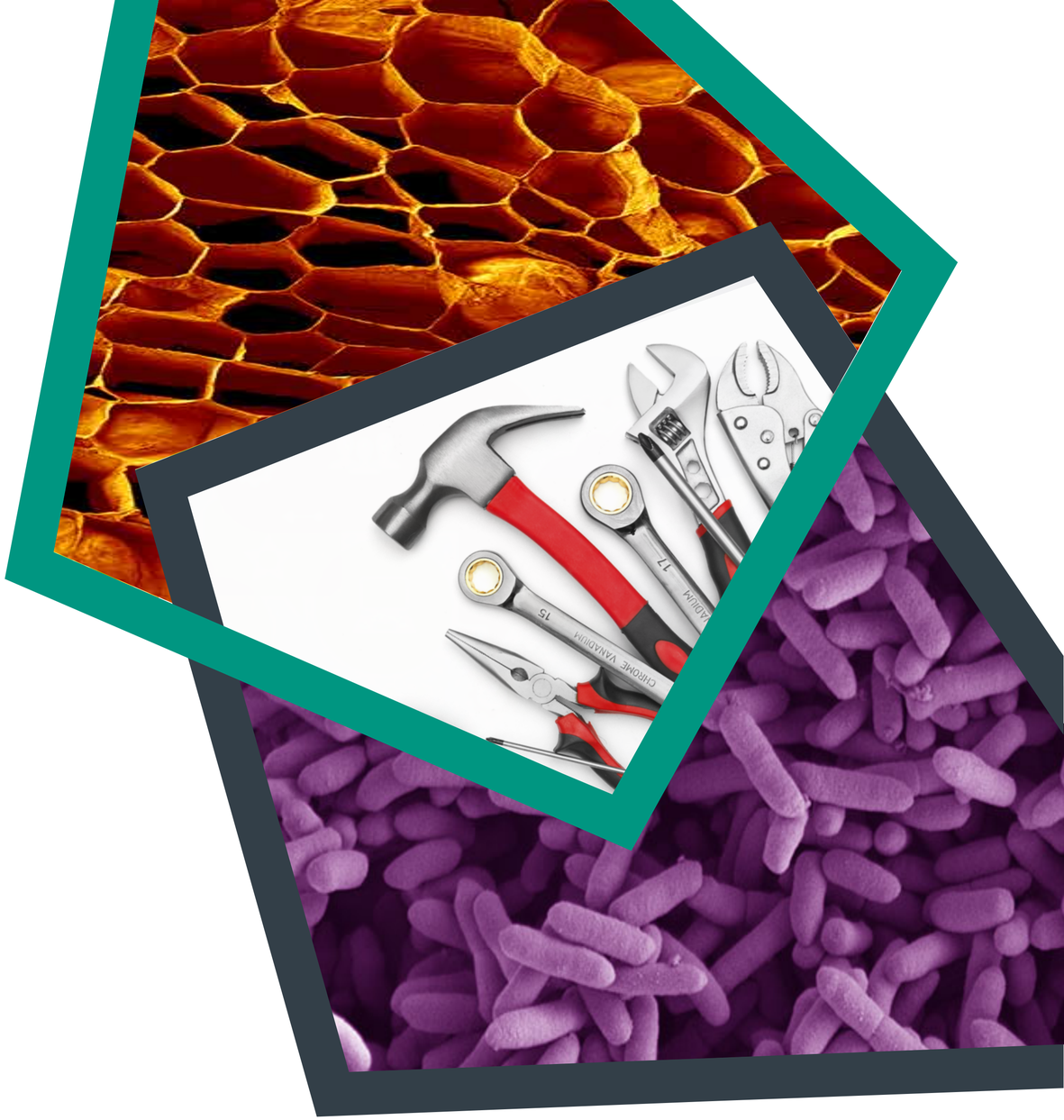
\+ Your local config files

- $HOME/.nextflow.config
- -c custom.config

Quadram Institute

Science Health
Food Innovation

Nextflow:
an overview



Crammed bash script

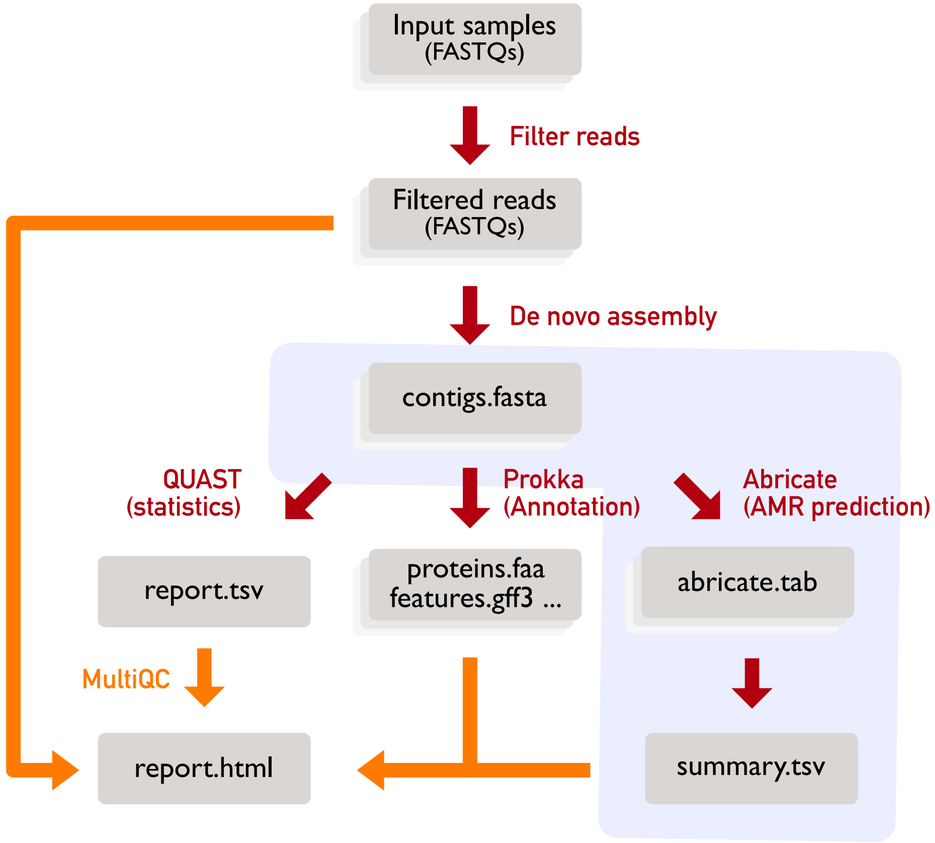
denovo.sh
Edited

Shell Script

#!/bin/bash

```
1  #!/bin/bash
2
3  set -euxo pipefail
4  THREADS=12
5
6  for R1 in reads/*_R1.fastq.gz;
7  do
8      R2=${R1/_R1/_R2}
9      SAMPLE=$(basename $R1 | cut -f1 -d_)
10     spades.py -1 $R1 -2 $R2 -o assembly-$SAMPLE/ -t $THREADS
11     abricate assembly-$SAMPLE/contigs.fa > $SAMPLE.tab
12 done
13
14 abricate --summary *.tab > abricate-summary.tsv
```

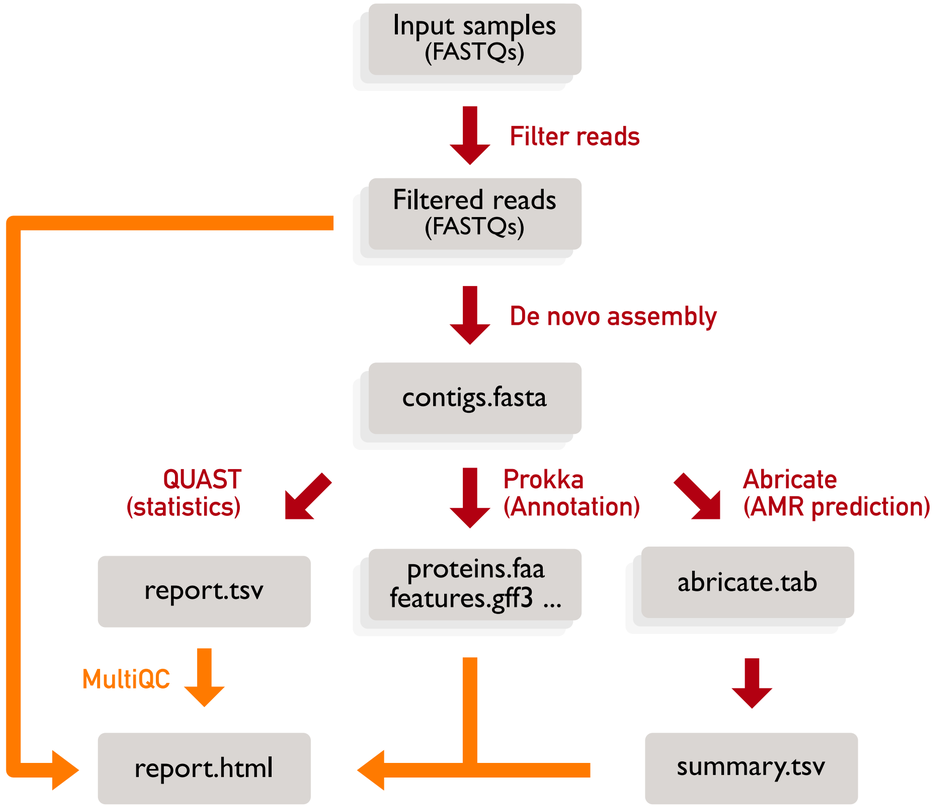
Lines: 14 Characters: 310 Location: 13 Line: 2 310 bytes Unicode (UTF-8) LF



few steps, already a mess

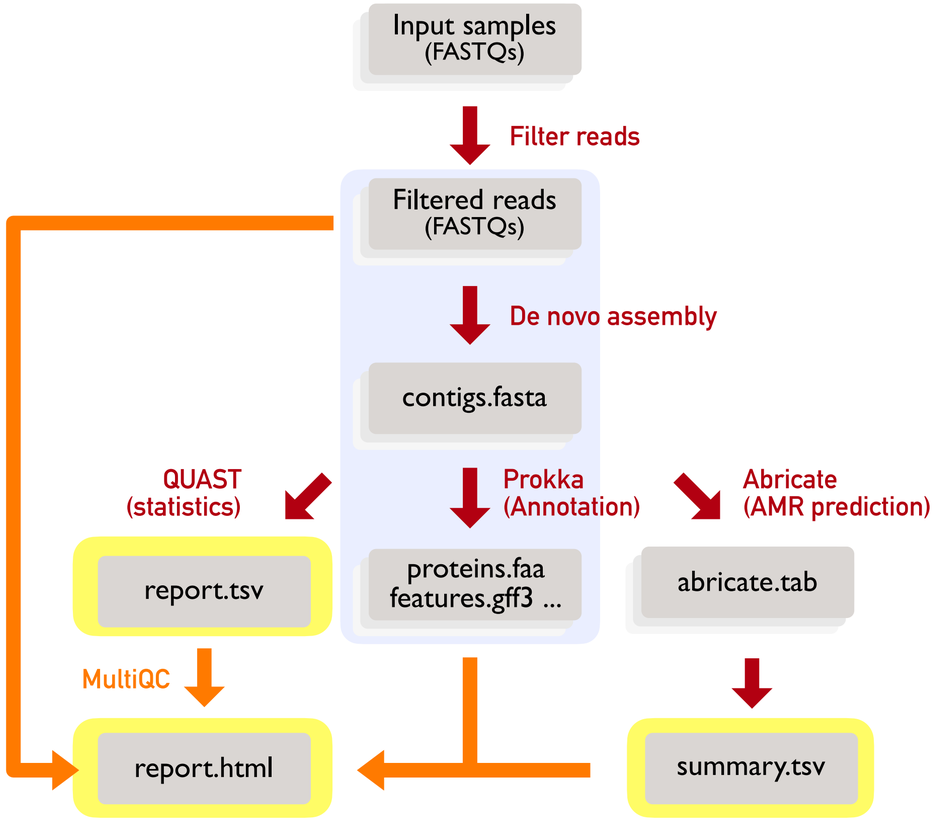
What we need to know

- **Get input** from the user
- Use **file channels** to propagate data
- Describe each **process**
- Describe the **workflow**



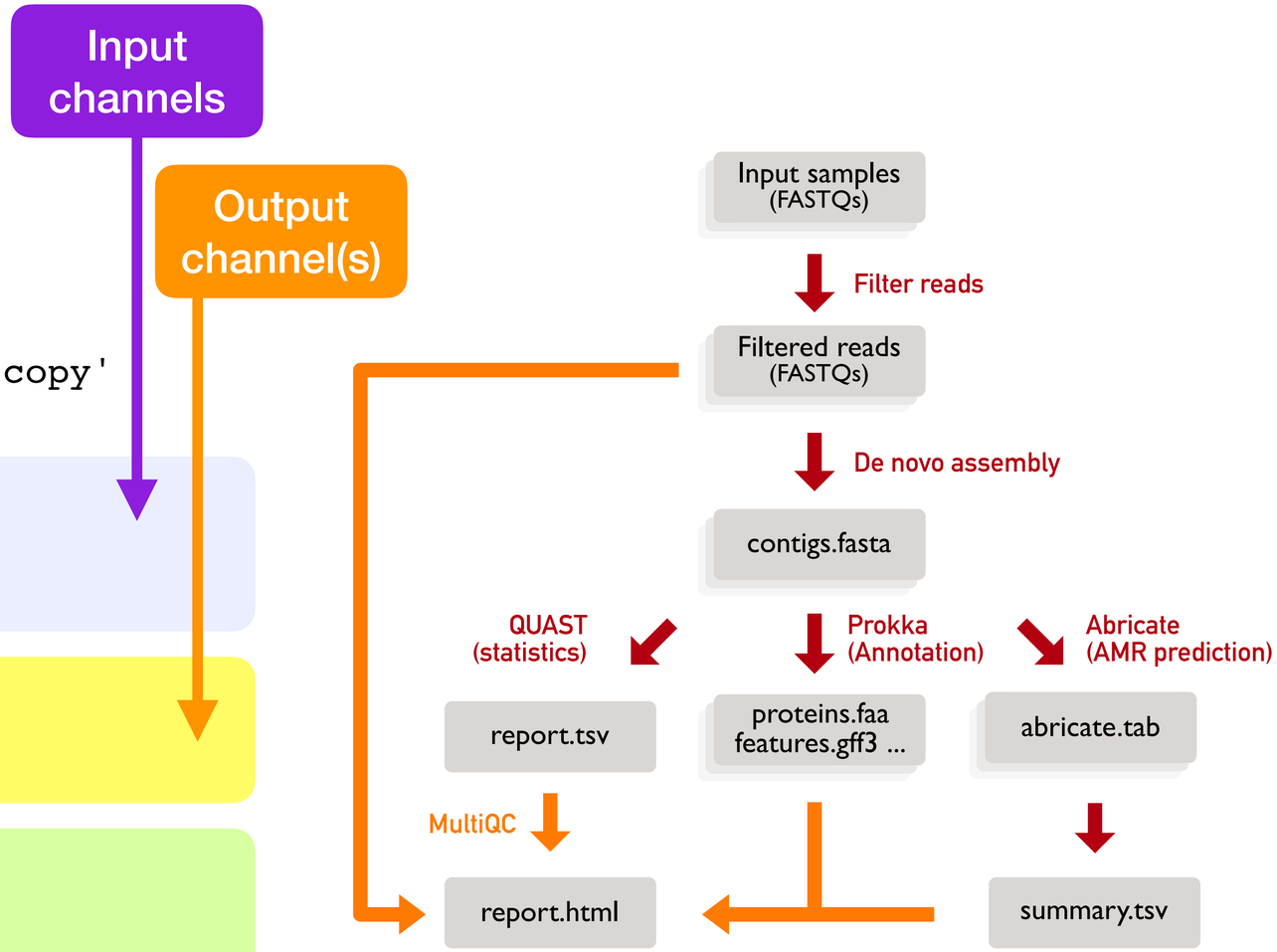
Expressive syntax for the workflow

```
workflow {
    fastp( reads )
    assembly( fastp.out.reads )
    amr( assembly.out )
    prokka( assembly.out )
    // collect assemblies for QUAST
    quast( assembly.out.map{it -> it[1]}.collect() )
    // Prepare the summary of Abricate
    abricate(amr.out.map{it -> it[1]}.collect() )
    // Collect all the relevant files for MultiQC
    multiqc(
      fastp.out.json.mix(quast.out, prokka.out, abricate.out.mqc)
      .collect()
    )
}
```



The process

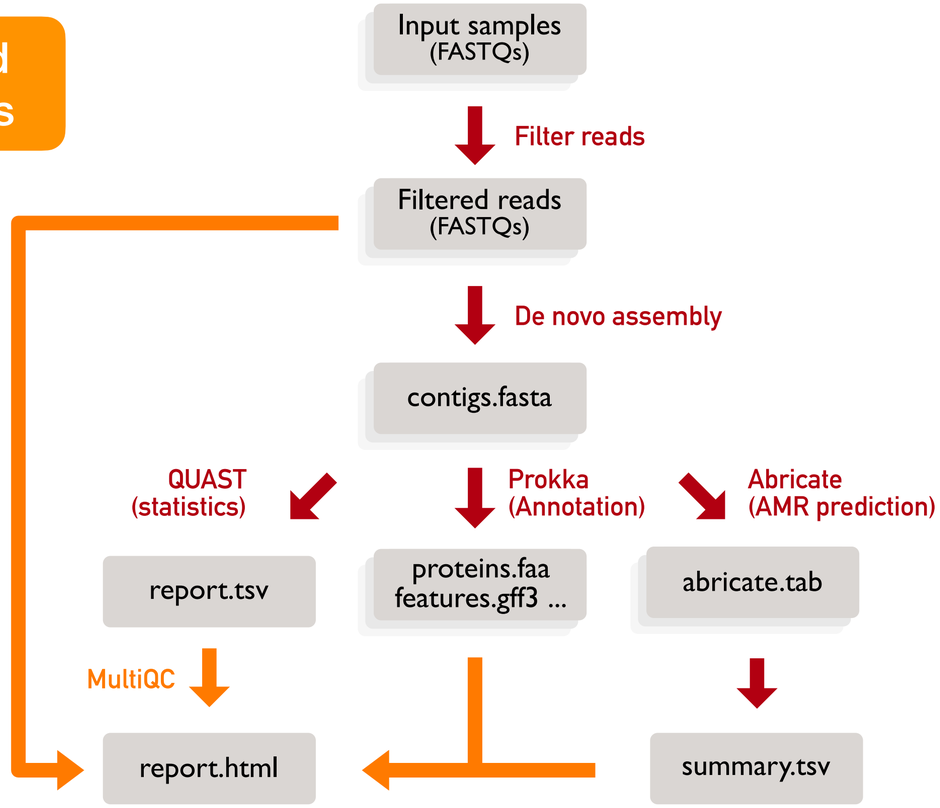
```
process assembly {
    tag { sample_id }
    publishDir "$params.outdir/assemblies/", mode: 'copy'

    input:
    tuple val(sample_id), path(reads)

    output:
    tuple val(sample_id), path("${sample_id}.fa")

    script:
    """
    shovill --R1 ${reads[0]} --R2 ${reads[1]} \\
        --cpus ${task.cpus} --outdir ctgs
    mv ctgs/contigs.fa ${sample_id}.fa
    """
}
```



The process

```
process assembly {
    tag { sample_id }
    publishDir "$params.outdir/assemblies/", mode: 'copy'

    input:
    tuple val(sample_id), path(reads)

    output:
    tuple val(sample_id), path("${sample_id}.fa"), emit: ctgs
    tuple val(sample_id), path("${sample_id}.log"), emit: log

    script:
    """
    shovill --R1 ${reads[0]} --R2 ${reads[1]} \\
        --cpus ${task.cpus} --outdir ctgs 2> ${sample_id}.log
    mv ctgs/contigs.fa ${sample_id}.fa
    """
}
```

Named outputs



The “input” channel

```
// input parameters are readily available

params.reads = "*_R{1,2}.fastq.gz"


// We can create a special “Paired end” channel
reads = Channel.fromFilePairs(params.reads,
                              checkIfExists: true)


// to “view” a channel for debugging:

reads.view()
```

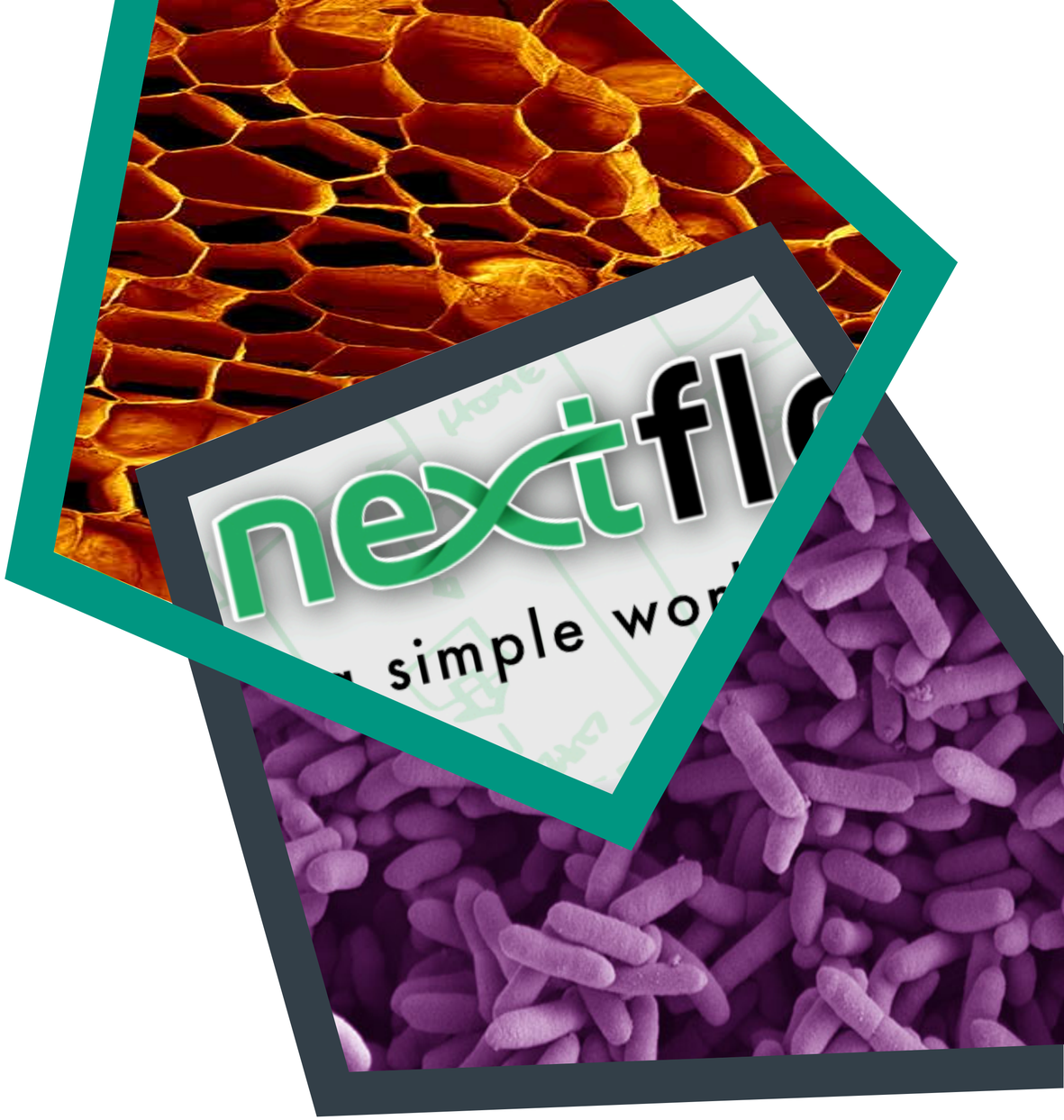
Further topics

- Storing processes in separate files (modules)
- Creating subworkflows that acts as processes

How to answer most questions?

- Try with a minimal example!

Quadram Institute
Science Health
Food Innovation



A worked example

https://telatin.github.io/microbiome-bioinformatics/Nextflow-start/

1 What we will achieve

- We will start running our final product to see that a shared pipeline can be invoked from any client thanks to github support:

```
nextflow run telatin/nextflow-example -r main -profile docker \
--reads “data/*_R{1,2}.fastq.gz"
```

- You can run local files (pipeline.nf) or remote files, in this case from github.com
- You must specify a release, a branch or a commit
- The workflow need to find the dependencies. Options: docker/singularity/conda

https://telatin.github.io/microbiome-bioinformatics/Nextflow-start/

1 What we will achieve

- We will start running our final product to see that a shared pipeline can be invoked fro

```
nextflow ru
--reads “da
```

- You can run
- You must s
- The workflo

```
(base) ubuntu@parallel:~/test$ nextflow run telatin/nextflow-example -r main -profil
N E X T F L O W  ~  version 21.10.6
Launching `telatin/nextflow-example` [modest_miescher] - revision: 9332bda5f2 [main]

Denovo Pipeline (version 5)
==================================
reads        : data/*_R{1,2}.fastq.gz
outdir       : ./denovo

executor >  local (3)
[7a/a11aaf] process > FASTP (filter SRR12825099) [  0%] 0 of 3
[-          ] process > SHOVILL                      -
[-          ] process > ABRICATE                     -
[-          ] process > PROKKA                       -
[-          ] process > QUAST                        -
[-          ] process > ABRICATE_SUMMARY             -
[-          ] process > MULTIQC                      -
```

Startup

https://telatin.github.io/microbiome-bioinformatics/Nextflow-start/

1 What we will achieve

- We will start running our final product to see that a shared pipeline can be invoked from

```
nextflow ru
--reads “da
```

- You can run
- You must sp
- The workflo

```
(base) ubuntu@parallel:~/test$ nextflow run telatin/nextflow-example -r main -profile docker
N E X T F L O W  ~  version 21.10.6
Launching `telatin/nextflow-example` [modest_miescher] - revision: 9332bda5f2 [main]

Denovo Pipeline (version 5)
==================================
reads        : data/*_R{1,2}.fastq.gz
outdir       : ./denovo

executor >  local (15)
[b3/e61980] process > FASTP (filter T7)       [100%] 3 of 3 ✔
[30/143953] process > SHOVILL (T7)            [100%] 3 of 3 ✔
[94/6300eb] process > ABRICATE (T7)           [100%] 3 of 3 ✔
[ba/de80aa] process > PROKKA (T7)             [100%] 3 of 3 ✔
[d5/edf0a9] process > QUAST (quast)           [100%] 1 of 1 ✔
[8b/b3b76f] process > ABRICATE_SUMMARY (null) [100%] 1 of 1 ✔
[4e/95b026] process > MULTIQC                 [100%] 1 of 1 ✔
WARN: To render the execution DAG in the required format it is r
Completed at: 31-Jan-2022 10:46:44
Duration    : 3m 4s
CPU hours   : 0.2
Succeeded   : 15
```

Completion

https://telatin.github.io/microbiome-bioinformatics/Nextflow-start/

1 What we will achieve

- We will start running our final product to see that a shared pipeline can be invoked from

```
nextflow run
--reads “dat
```

- You can run I
- You must spe
- The workflow

```
(base) ubuntu@parallel:~/test$ ls -lha work/b3/e619801 0c6ac4996a4f9dac68fc07/
total 16M
drwxrwxr-x 2 ubuntu docker 4.0K Jan 31 10:45 .
drwxrwxr-x 3 ubuntu docker 4.0K Jan 31 10:43 ..
-rw-rw-r-- 1 ubuntu docker    0 Jan 31 10:43 .command.begin
-rw-rw-r-- 1 ubuntu docker 2.0K Jan 31 10:45 .command.err
-rw-rw-r-- 1 ubuntu docker 2.0K Jan 31 10:45 .command.log
-rw-rw-r-- 1 ubuntu docker    0 Jan 31 10:43 .command.out
-rw-rw-r-- 1 ubuntu docker 9.9K Jan 31 10:43 .command.run
-rw-rw-r-- 1 ubuntu docker  201 Jan 31 10:43 .command.sh
-rw-r--r-- 1 ubuntu root    219 Jan 31 10:45 .command.trace
-rw-rw-r-- 1 ubuntu docker    1 Jan 31 10:45 .exitcode
-rw-r--r-- 1 ubuntu root   467K Jan 31 10:45 fastp.html
-rw-r--r-- 1 ubuntu root   124K Jan 31 10:45 report.json
-rw-r--r-- 1 ubuntu root   124K Jan 31 10:45 T7.fastp.json
-rw-r--r-- 1 ubuntu root   7.1M Jan 31 10:45 T7_filt_R1.fastq.gz
-rw-r--r-- 1 ubuntu root   7.5M Jan 31 10:45 T7_filt_R2.fastq.gz
lrwxrwxrwx 1 ubuntu docker   37 Jan 31 10:43 T7_R1.fastq.gz -> /home/ubuntu/test/data/T7_R1.fastq.gz
lrwxrwxrwx 1 ubuntu docker   37 Jan 31 10:43 T7_R2.fastq.gz -> /home/ubuntu/test/data/T7_R2.fastq.gz
```

Debugging

Log files (stderr, all, stdout)

The script being executed

Input files symlinked

https://telatin.github.io/microbiome-bioinformatics/Nextflow-start/

2 Getting the parameters

- The user will need to supply some parameters. A very basic way (although not elegant) of receiving the input is asking for the very pattern to create the channel
- Parameters can be initialized with defaults in the script, and should at least be declared as “param.something = false”.

https://telatin.github.io/microbiome-bioinformatics/Nextflow-start/

3 Our first process

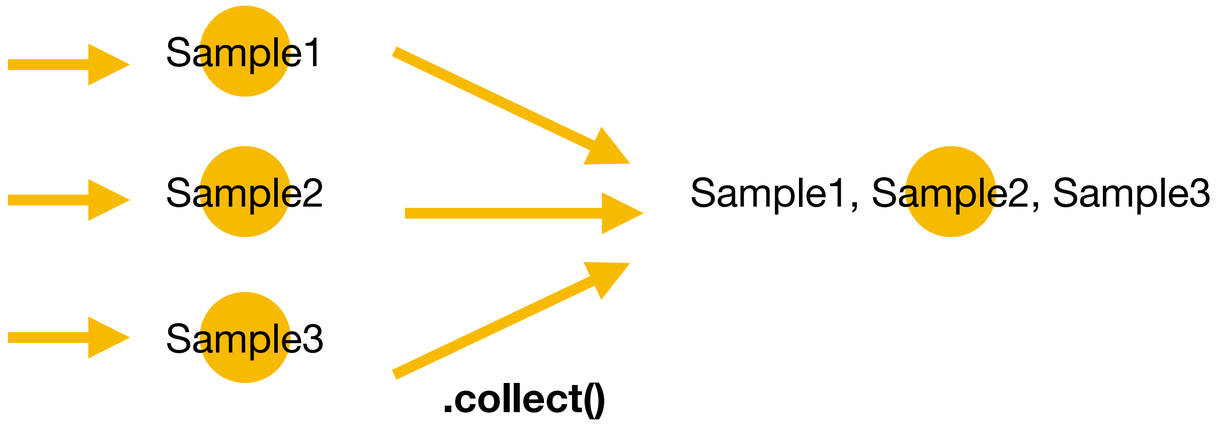
- The user will need to supply some parameters. A very basic way (although not elegant) of receiving the input is asking for the very pattern to create the channel
- Parameters can be initialized with defaults in the script, and should at least be declared as “param.something = false”.
- This already can be embedded in a wf

```
process fastp {
    /*
       fastp process to remove adapters and low quality sequences
    */
    tag "filter $sample_id"

    input:
    tuple val(sample_id), path(reads)

    output:
    tuple val(sample_id), path("${sample_id}_filt_R*.fastq.gz"), emit: reads
    path("${sample_id}.fastp.json"), emit: json


    script:
    """
    fastp -i ${reads[0]} -I ${reads[1]} \\
      -o ${sample_id}_filt_R1.fastq.gz -O ${sample_id}_filt_R2.fastq.gz \\
      --detect_adapter_for_pe -w ${task.cpus} -j ${sample_id}.fastp.json

    """
}
```

https://telatin.github.io/microbiome-bioinformatics/Nextflow-start/

4 More than linear channels

- There are operators to manipulate channels.
- We will use `.collect()` and `.mix(…)`
- Check the documentation and experiment creating simple channels and using `.view()` to see how they get arranged



https://telatin.github.io/microbiome-bioinformatics/Nextflow-start/

5 A taste of modularity

- In the last script we will refactor the code to put some processes in separate files, that are imported in the main script.
- This will simplify reusing modules in multiple workflows
- What next? Subworkflows

https://telatin.github.io/microbiome-bioinformatics/Nextflow-start/

nextflow

can be hard,

...it feels good

but when it runs...

```
U S E A R C H - N F   P I P E L I N E
  Rev X.22
 =====================================
 ref          : /home/ubuntu/Dropbox/nextflow/amplicon/db/rdp_16s_v16_sp.udb
 reads        : 16S/*_R{1,2}.fastq
 outdir       : ./ampliflow
 cpus         : 12
Monitor the execution with Nextflow Tower using this url https://tower.nf/wat
executor >  local (25)
[29/9563e5] process > mergepairs (NHP11)  [100%] 19 of 19 ✔
[09/3f1c8f] process > collect_merged      [100%] 1 of 1 ✔
[53/1fdc8c] process > uniq_asv            [100%] 1 of 1 ✔
[a4/8ed771] process > otutab_raw          [  0%] 0 of 1
[-        ] process > otutab              -
[00/b5efb1] process > taxonomy (1)        [100%] 1 of 1 ✔
[3b/9d8155] process > refine_taxonomy (1) [100%] 1 of 1 ✔
[ed/13848f] process > taxonomy_tab (1)    [100%] 1 of 1 ✔
[-        ] process > taxonomy_ranks      -
```