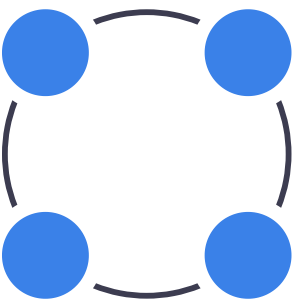


NEXTFLOW

Crash course



<https://github.com/glebus-sasha/nextflow-course/>

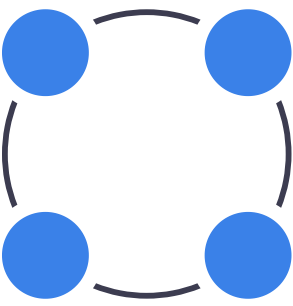


Key Benefits

- Automation
- Reproducibility

more details on
[link ->](#)

Comparison of workflow systems



👎 More difficult
👍 More features

👍 Easier
👎 Fewer features

 **nextflow**

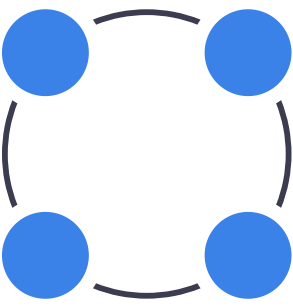
VS

 **snakemake**

- Groovy
- Integration with containerization systems, clusters, clouds

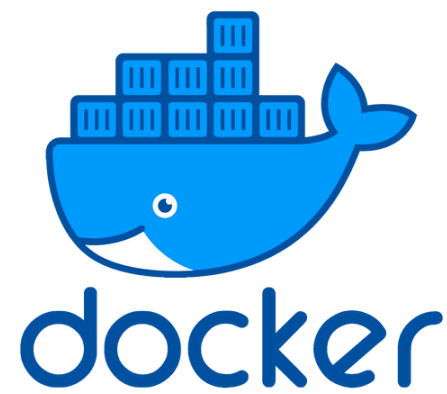
- Python
- Limited support for containerization, clusters, clouds

more details on
[link ->](#)



Advantages of Nextflow

- “Out of the box” integrated with many systems

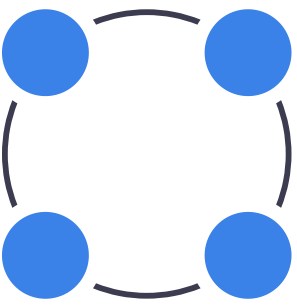
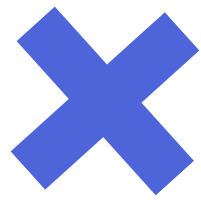


kubernetes



more details on
[link ->](#)

Requirmments



Requirements



Nextflow

Nextflow is a workflow system for creating scalable, portable, and reproducible workflows.

- [Install Nextflow](#)



Mamba/Conda

Conda and Mamba are tools for managing packages and virtual environments, used for installing and updating dependencies in projects. Mamba is faster but fully compatible with Conda.

- [Install Mamba](#)
- [Install Conda](#)



Git

Git is a version control system used for tracking changes in code and collaborating on software

- [Install git](#)



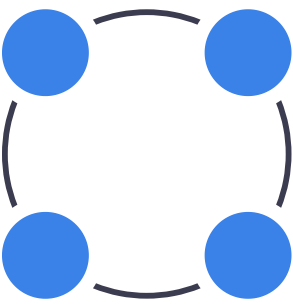
Apptainer/Singularity/ Docker

Docker and Singularity (Apptainer) are containerization tools used to package applications and their dependencies into isolated environments, ensuring consistent execution across different systems.

- [Install Apptainer](#)
- [Install Singularity](#)
- [Install Docker](#)



Installation



Install Nextflow

```
curl -s https://get.nextflow.io | bash
chmod +x nextflow
mkdir -p $HOME/.local/bin/
mv nextflow $HOME/.local/bin/
echo 'export PATH="$HOME/.local/bin:$PATH"' >> ~/.bashrc && source ~/.bashrc
```

more details on
[link ->](#)

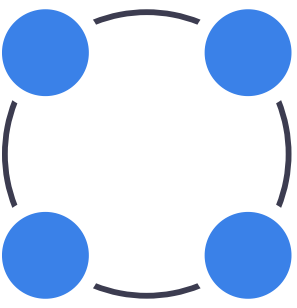
Nextflow programming language

Java -> Groovy -> Nextflow

```
// Java  
String name = "John";
```

```
// Groovy (динамическая типизация)  
def name = "John";
```

more details on
[link ->](#)

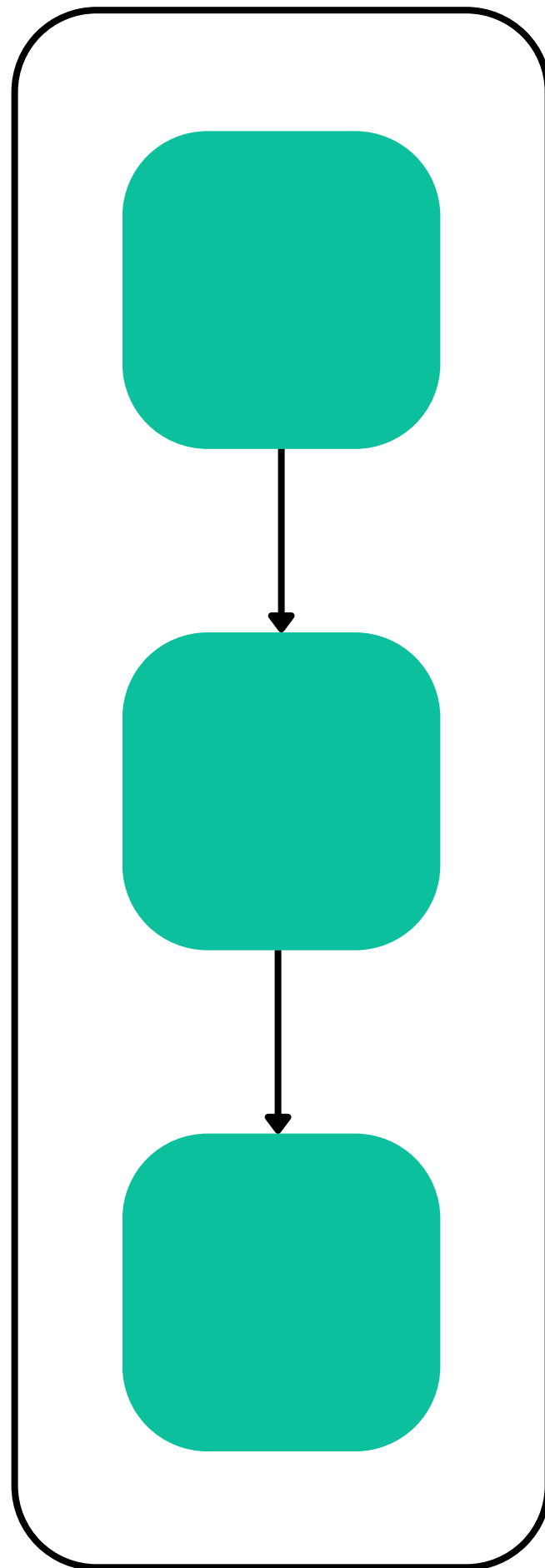


Lifecode

Let's write our own hello world

- 1.hello_world.nf
- 2.conditional_expressions.nf
- 3.functions.nf

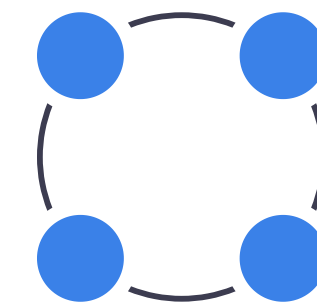
Workflow

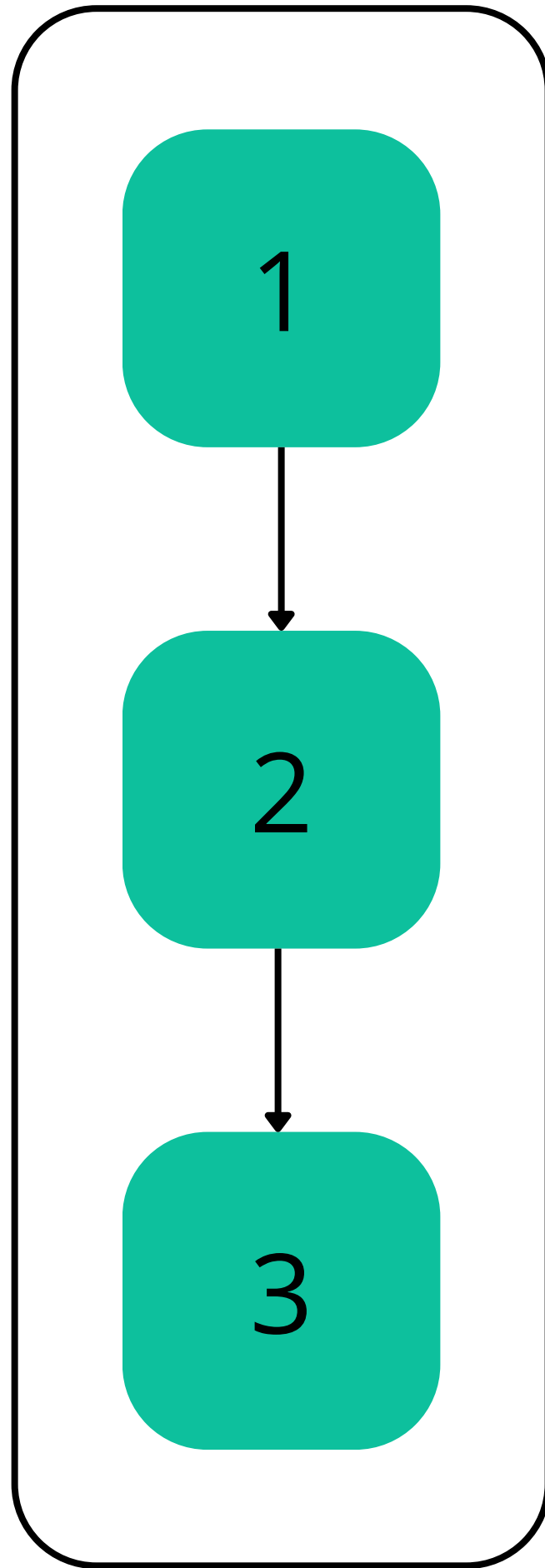
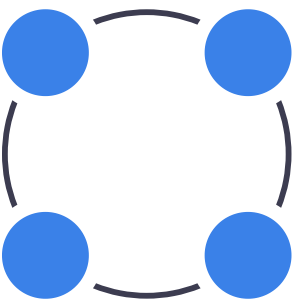


Process

Channel

Workflow





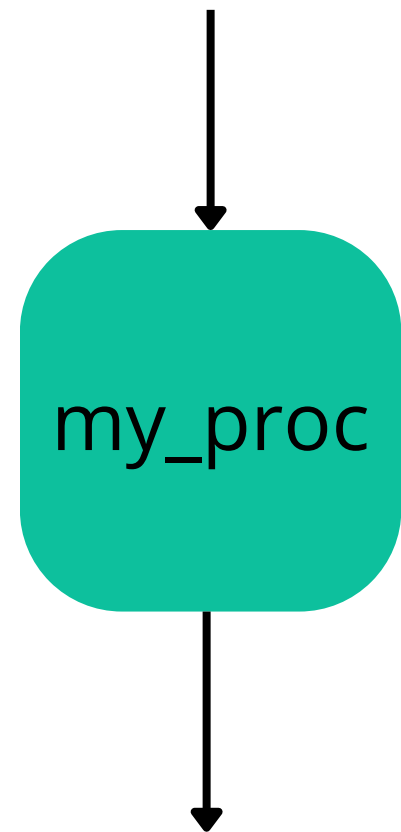
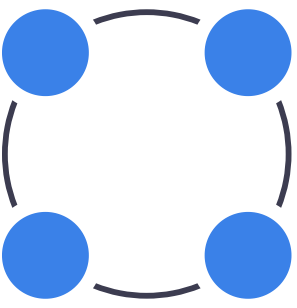
Process

Channel

Workflow

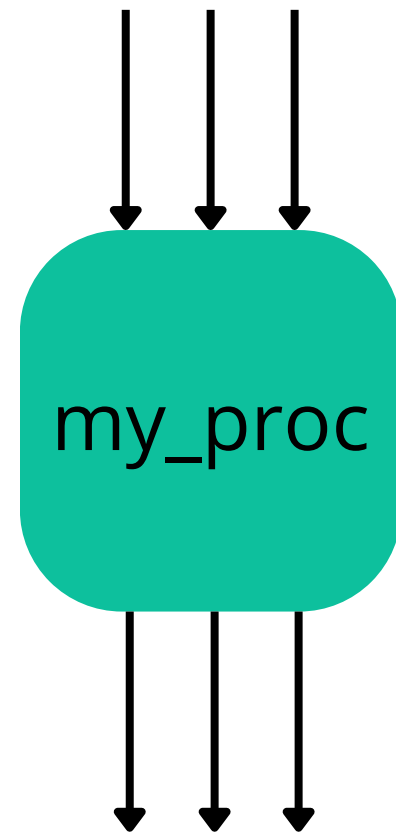
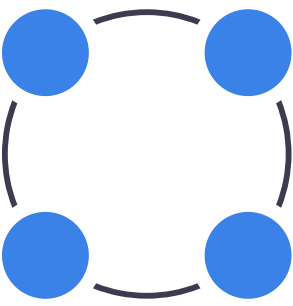
```
workflow {  
    process_1  
    process_2  
    process_3  
}
```

Process

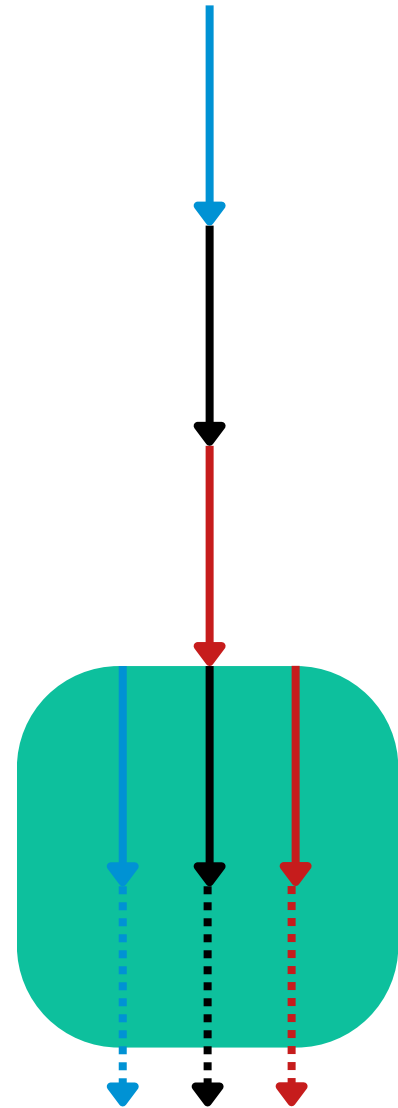
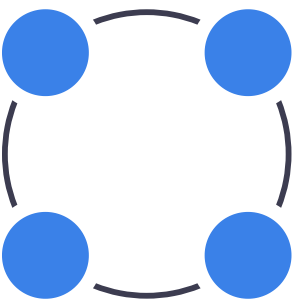


```
process my_proc{  
    input:  
    ...  
    output:  
    ...  
    script:  
    \ \ \  
    . . .  
    \ \ \  
}
```

Channels

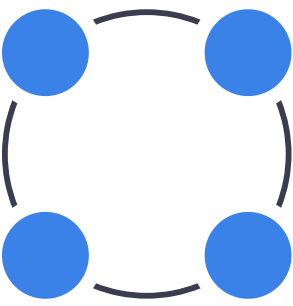


- **Sending** the message is an ***asynchronous*** (i.e. non - closing) operation, which means that the sender does not need to wait for the process of receipt.
- **Obtaining** a message is a ***synchronous*** (i.e., blocking) operation, which means that the acceptor of the process must wait for the message.

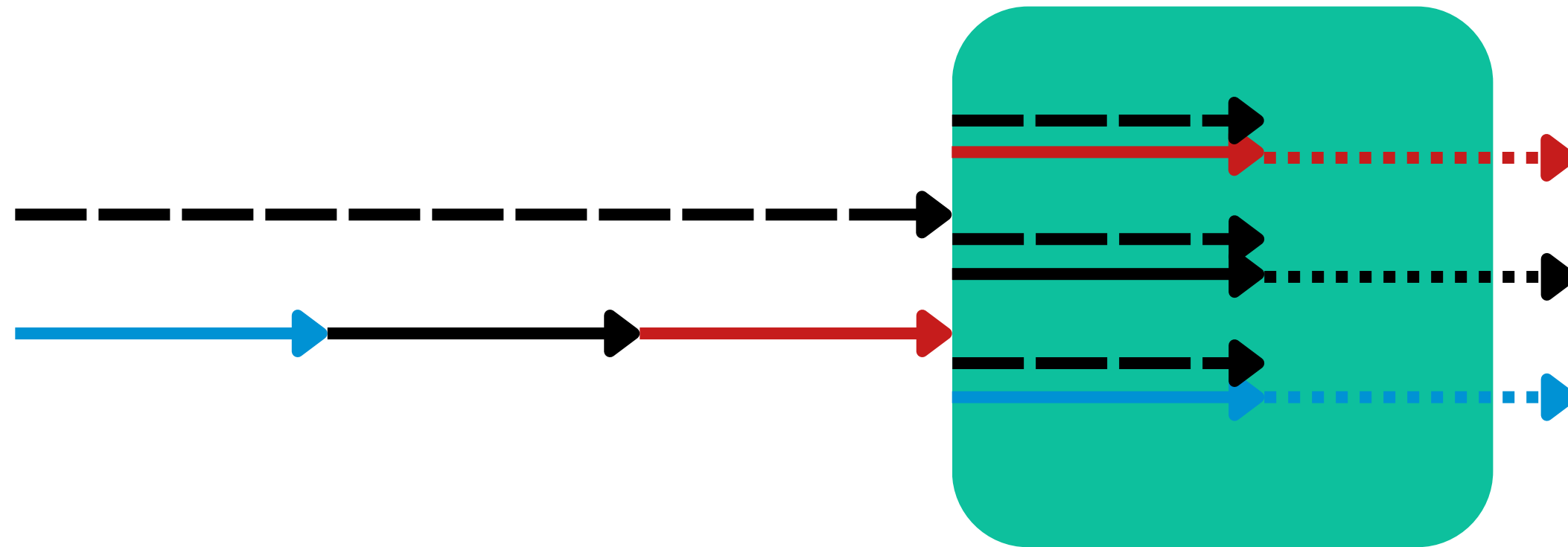


- Non-blocking unidirectional **FIFO** queue
- **FIFO** - First In, First Out.

A value channel can be linked (i.e., assigned) to only one value and can be used by a process or operator any number of times.



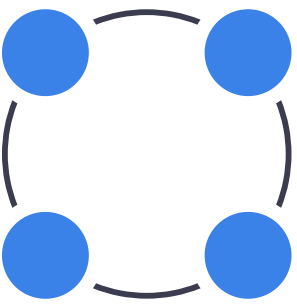
Value channel



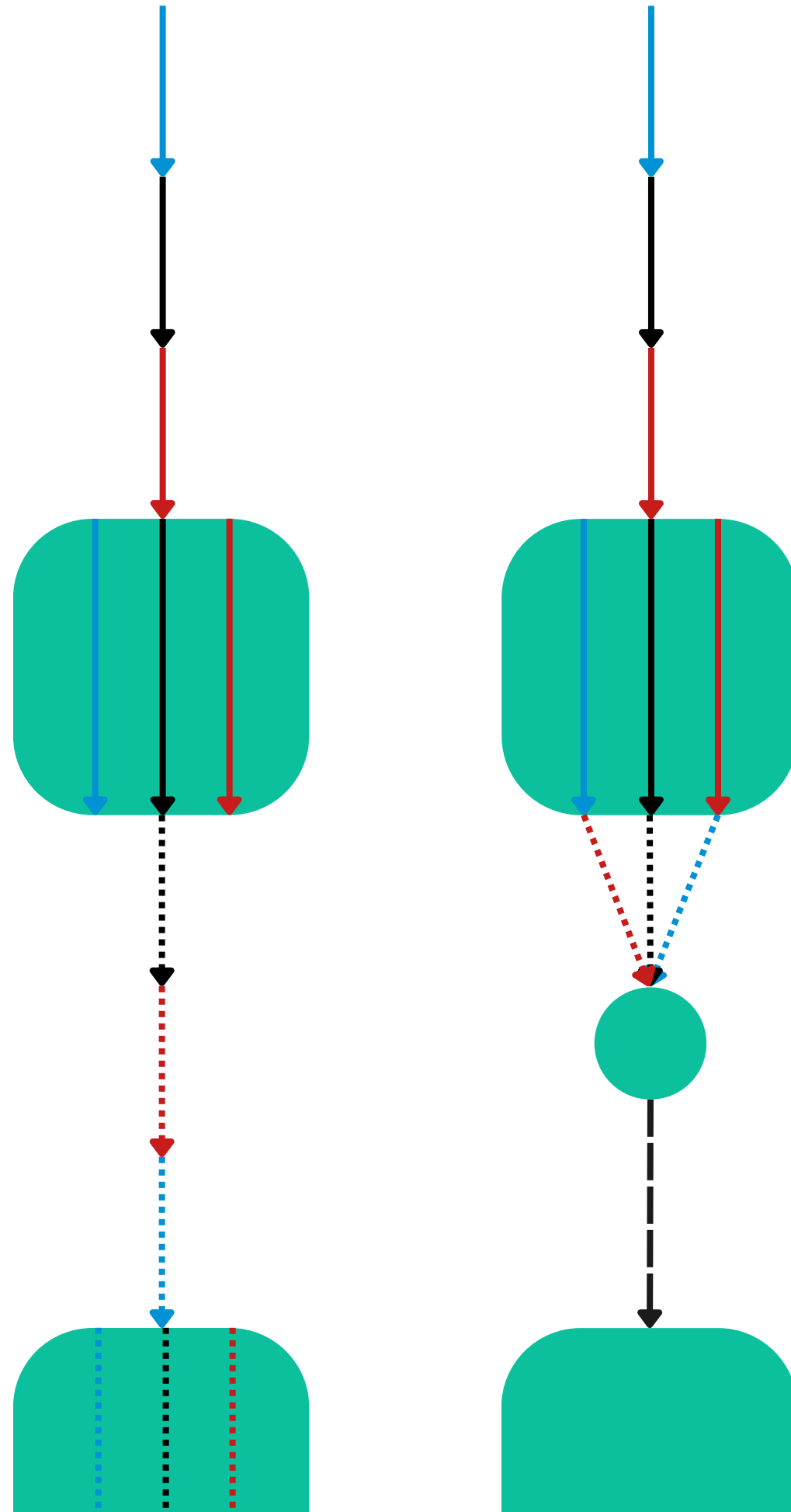
Queue channel

A queue channel is a non-blocking unidirectional FIFO queue that connects a producer process (i.e., one that outputs a value) with a consumer process or operators.

Operators

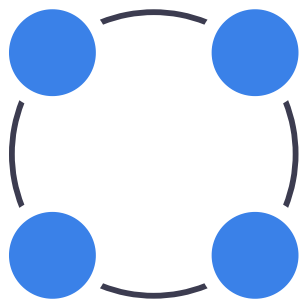


Operators transform channels



```
my_proc.out.collect()
```

Channel factories



Channel factories

Queue channel

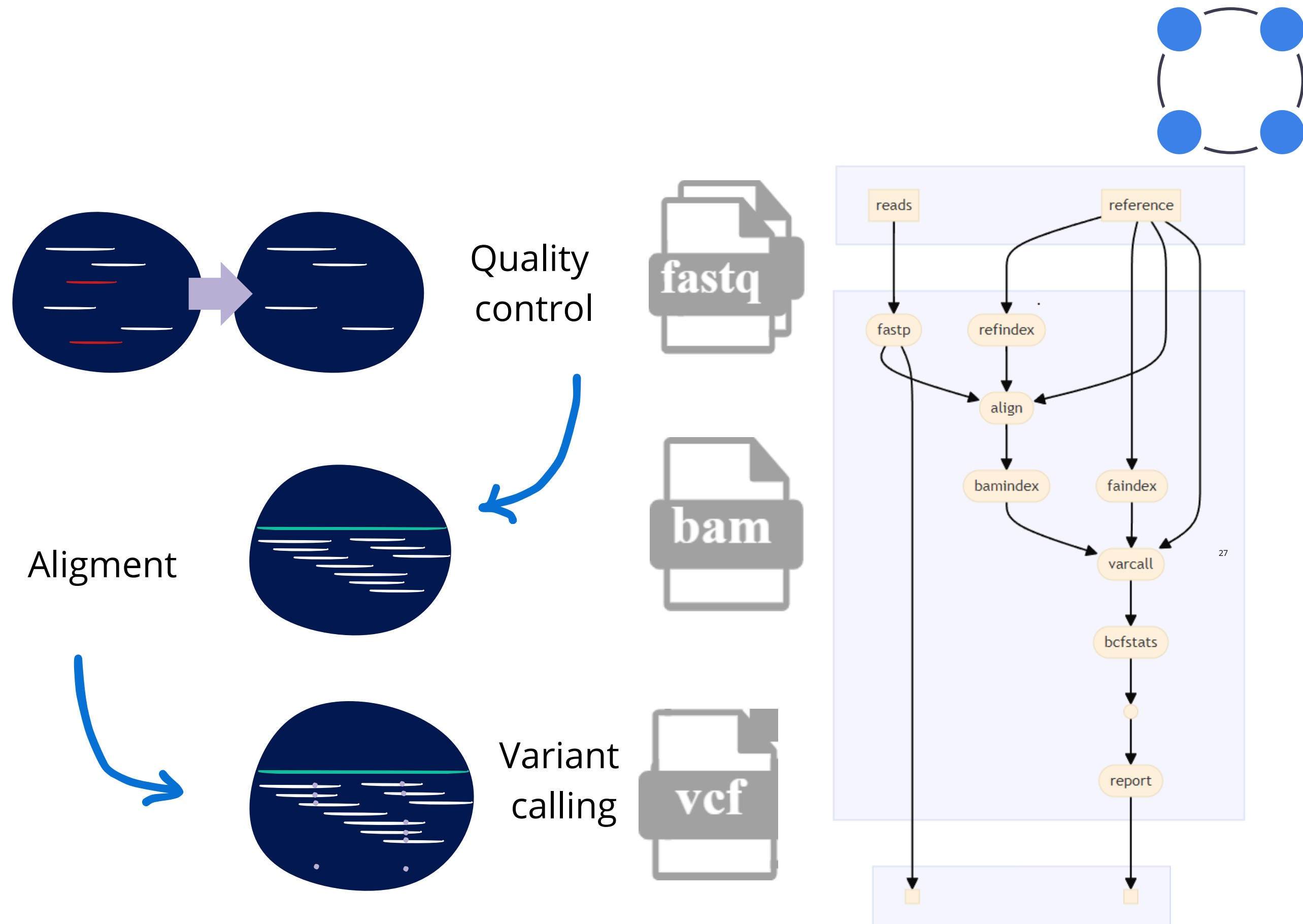
```
Channel.of( 1, 3, 5, 7 )  
Channel.fromPath( '/data/*.txt' )  
Channel.fromFilePairs( '/data/SRR*_{1,2}.fq' )  
Channel.of( 1, 2, 3, 4, 5 )  
    .map { v -> v * v }  
...
```

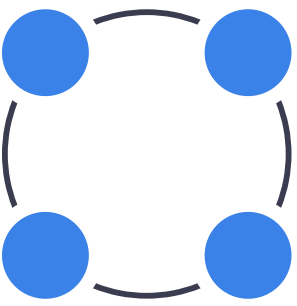
Value channel

```
Channel.of( 1, 2, 3 ).first()  
Channel.value( [1,2,3,4,5] )  
Channel.of( 1, 2, 3, 4 ).collect()  
Channel.of( 1, 2, 3, 4, 5 )  
    .reduce { a, b ->  
        println "a: $a b: $b"  
        a + b  
    }  
...
```

Variant calling

DAG





sample1_R1.fastq
sample1_R2.fastq

sample4_R1.fastq
sample4_R2.fastq

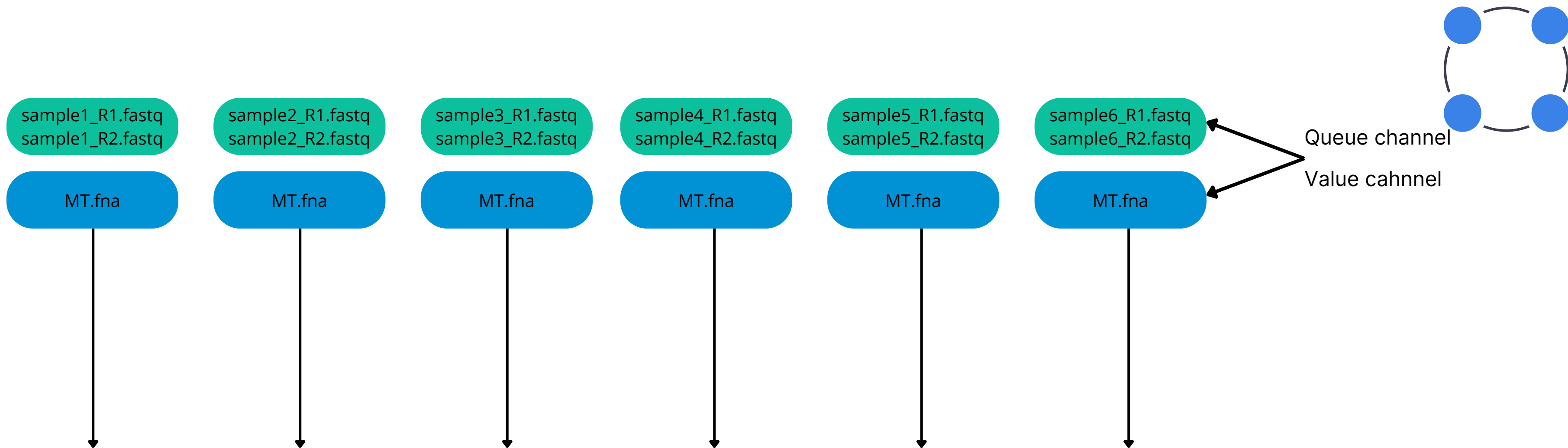
sample6_R1.fastq
sample6_R2.fastq

sample2_R1.fastq
sample2_R2.fastq

MT.fna

sample5_R1.fastq
sample5_R2.fastq

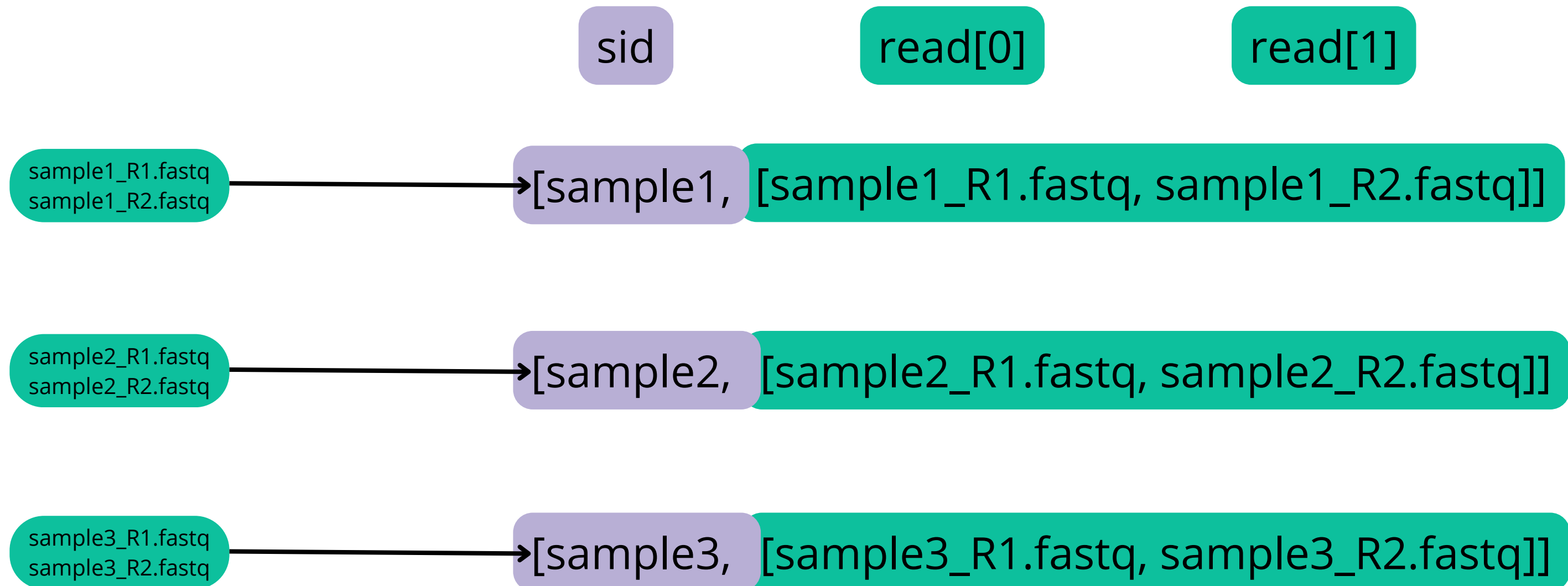
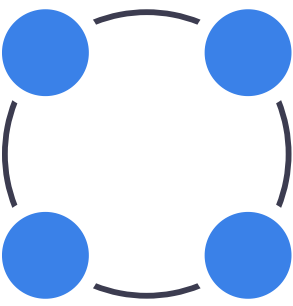
sample3_R1.fastq
sample3_R2.fastq

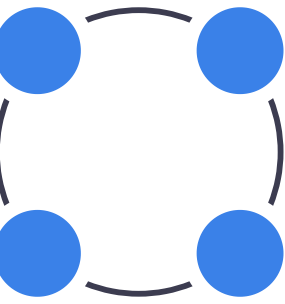


```
reads = Channel.fromFilePairs(params.reads)
```

```
reference = Channel.fromPath(params.reference).collect()
```

```
params.reads = '../data/*_R{1,2}.fastq'  
reads = Channel.fromFilePairs(params.reads)
```





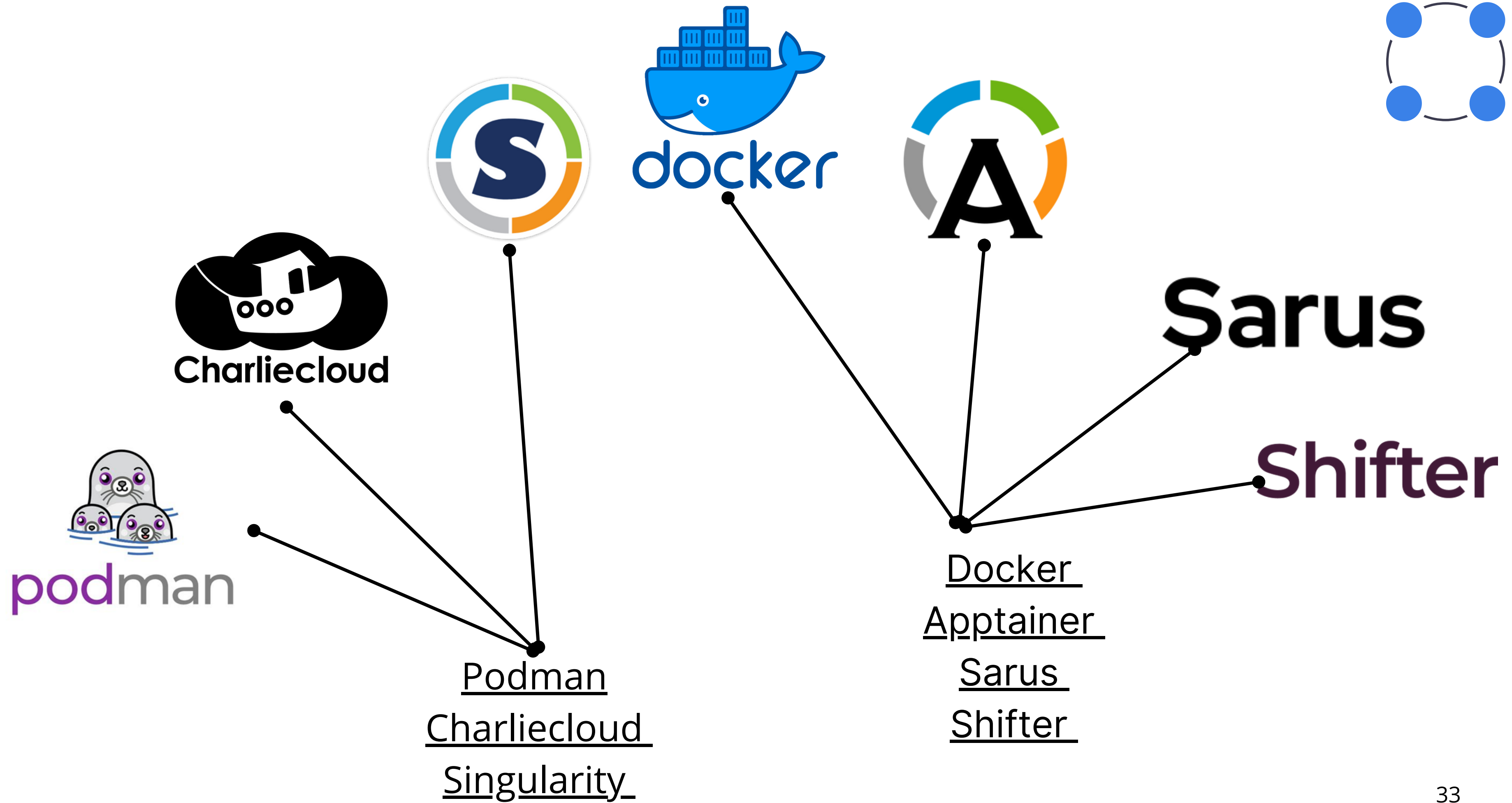
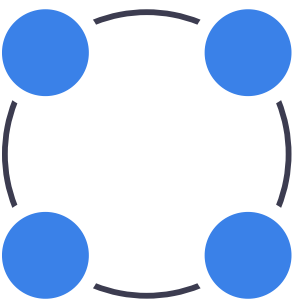
Lifecode

Let's write our pipeline

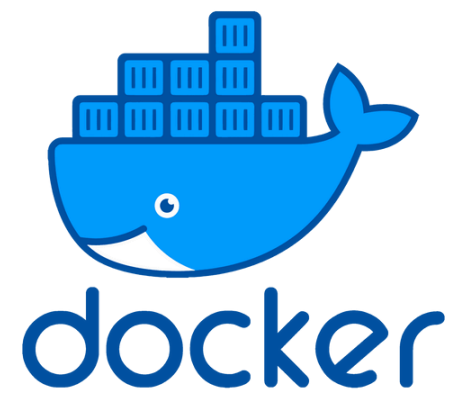
- variant_calling

part1

Containerization

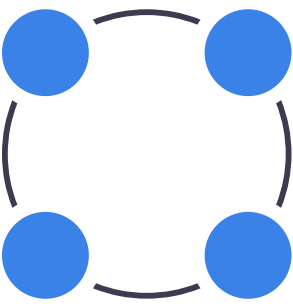


👍 More popular
👎 Problems with root

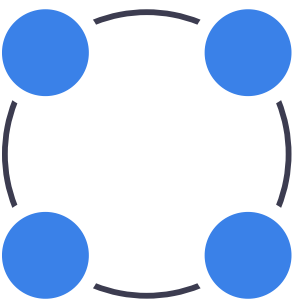


VS

👎 Less popular
👍 More secure



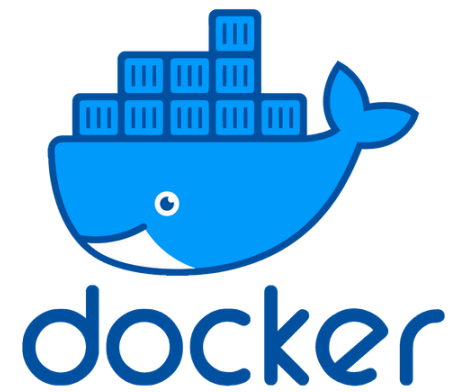
more details on
link ->



 Docker compatible

 Compatibility
with HPC

 docker compose



VS



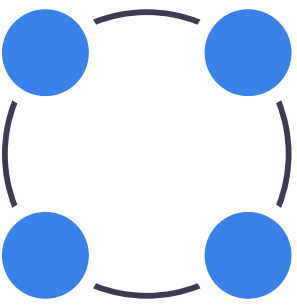
Does
not require
constant daemon work

Lifecode

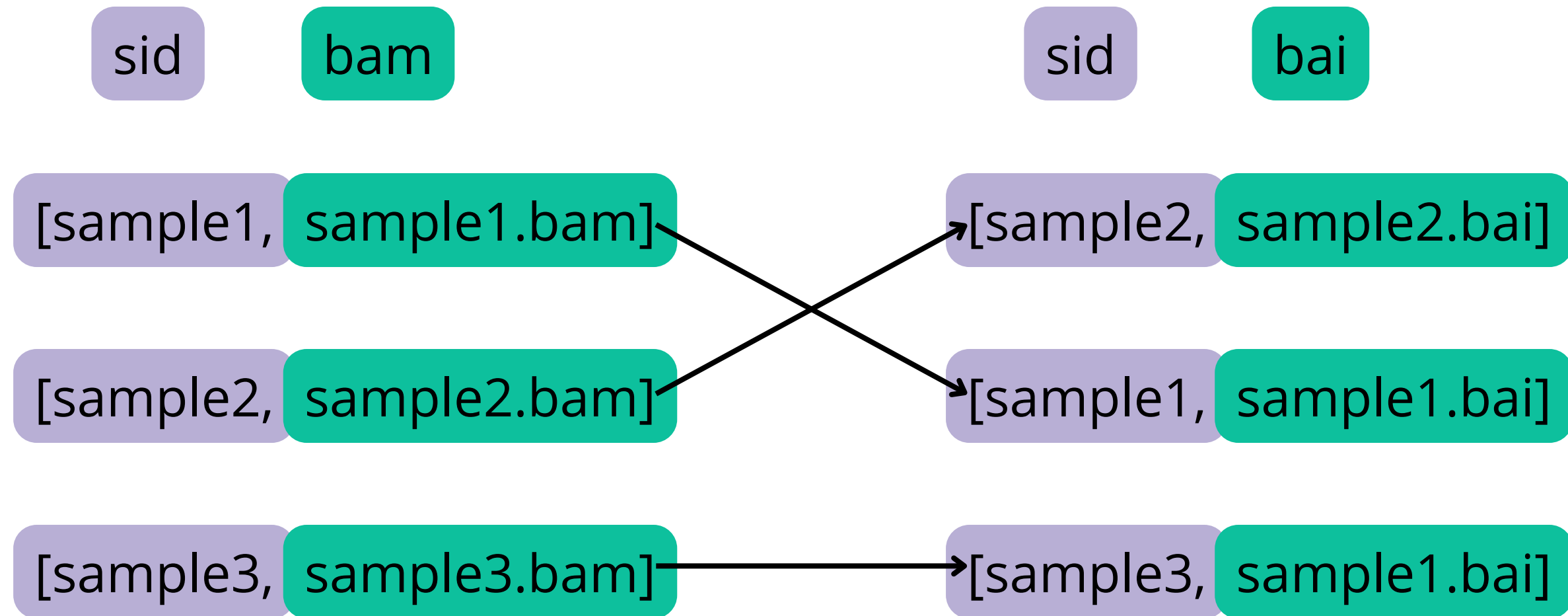
Let's write pipeline

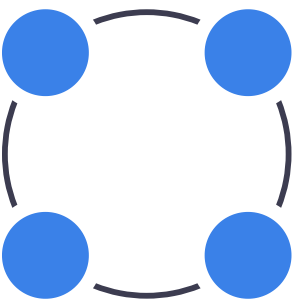
- variant_calling

part 2



align.out, bamindex.out





```
align.out.join(bamindex.out)
```

sid

bam

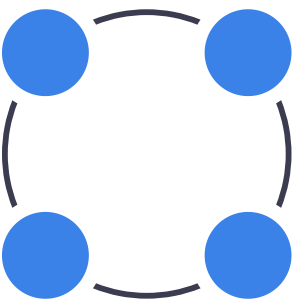
bai

[sample1, sample1.bam, sample1.bai]

[sample2, sample2.bam, sample2.bai]

[sample3, sample3.bam, sample3.bai]

Deploy



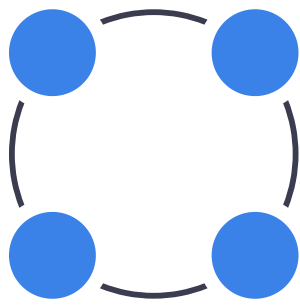
```
nextflow run nextflow-io/hello
nextflow run -latest nextflow-io/hello
nextflow run -latest nextflow-io/hello -r dev
```

- **Pull the pipeline**
- **Display a list of pipelines**
- **Derive information about the pipeline**
- **Delete the local version**

```
nextflow pull nextflow-io/hello
nextflow list
nextflow info nextflow-io/hello
nextflow drop nextflow-io/hello
```

- **Launch the script immediately from the githab**
- **Launch the latest version**
- **Run a certain branch**

Error strategy



- **terminate (default)**

When a task fails, terminate the pipeline immediately and report an error. Pending and running jobs are killed.

- **finish**

When a task fails, wait for submitted and running tasks to finish and then terminate the pipeline, reporting an error.

- **ignore**

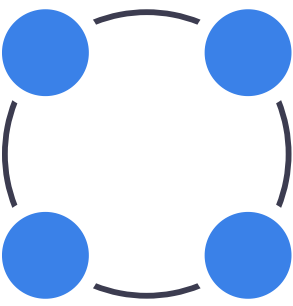
When a task fails, ignore it and continue the pipeline execution. If the `workflow.failOnIgnore` config option is set to true, the pipeline will report an error (i.e. return a non-zero exit code) upon completion. Otherwise, the pipeline will complete successfully.

- **retry**

When a task fails, retry it.

```
errorStrategy 'terminate'
errorStrategy 'finish'
errorStrategy 'ignore'
    debug true
errorStrategy 'retry'
    maxErrors 5
    maxRetries 3
```

Logs

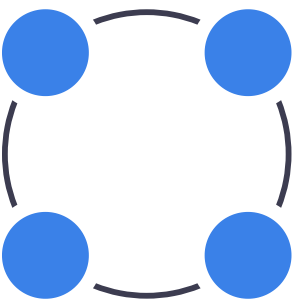


```
nextflow log  
nextflow log tiny_leavitt  
nextflow log -before tiny_leavitt  
nextflow log tiny_leavitt -f 'process,exit,hash,duration'
```

log and clear teams help control launches and control the cache and logs of launches without resorting to `rm -rf`

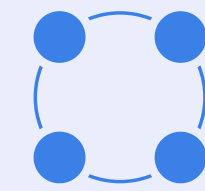
```
nextflow clean tiny_leavitt -n  
nextflow clean tiny_leavitt -f  
nextflow clean -but tiny_leavitt -f  
nextflow clean -keep-logs tiny_leavitt -n
```

More

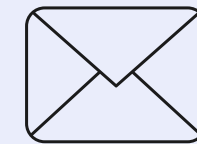


Additional sources

- [Nextflow documentation](#)
- [Nf-core community](#)
- [About best practice](#)
- [Nextflow youtube channel](#)
- [nf-core youtube channel](#)
- [Nextflow training](#)

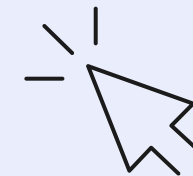


Thanks for your attention



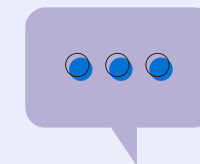
Email

glebus-sasha@mail.ru



Github

<https://github.com/glebus-sasha>



Telegram

@Glebus_Sasha

Ways to reach out

