

An Introduction to GC3Pie Session-based scripts

Riccardo Murri <riccardo.murri@gmail.com>

What is GC3Pie?

GC3Pie is . . .

1. An *opinionated* Python framework for defining and running computational workflows;
2. A *rapid development toolkit* for running user applications on clusters and IaaS cloud resources;
3. The worst name ever given to a middleware piece. . .

As users, you're mostly interested in this part.

What is GC3Pie?

GC3Pie is . . .

1. An *opinionated* Python framework for defining and running computational workflows;
2. *A rapid development toolkit for running user applications on clusters and IaaS cloud resources;*
3. The worst name ever given to a middleware piece. . .

As *users*, *you're mostly interested in this part.*

Who am I?

Systems administrator and programmer.

At UZH since 2010, first at GC3 then at S3IT.

Developer of GC3Pie since 2010.

and what about you?

Name ■ Affiliation ■ Interest in GC3Pie? ■ Programming languages? ■ Experience with command-line or terminal commands?

Outlook

1. Concepts and glossary
2. Usage of a session-based script
3. Command-line tools mainly useful for debugging

This is an *interactive* course!

We'd like the training to be
as interactive and informal as possible.

If you have a question, just ask – don't wait.

After the course is over,
you're very welcome to keep asking questions:

- ▶ in the GC3Pie chat room: <http://gitter.im/gc3pie/chat>
- ▶ on the mailing-list: gc3pie@googlegroups.com
- ▶ or through the forum web interface:
<https://groups.google.com/d/forum/gc3pie>

Where to find the course material

These slides are available for download from:

<http://tinyurl.com/gc3pie-december-2016>

The full GC3Pie documentation can be found at

<http://gc3pie.readthedocs.io/>

This tutorial assumes you're running GC3Pie version 2.5 (i.e., currently the *master* branch on GitHub).

Uses of GC3Pie

Uses of GC3Pie: parameter sweep

You have a simulation code that is dependent on a number of parameters.

Run the code for all possible combinations of parameters.

Then collect all the outputs and post-process to get a statistical overview.

Uses of GC3Pie: model calibration

You have a simulation code that is dependent on a number of parameters.

Run the code for all possible combinations of parameters, and find the ones that “best” approximate a given experimental result.

Uses of GC3Pie: parallel processing

Run the same program over and over again, feeding it different input files each time.

Then collect all the outputs and post-process to get a statistical overview.

Uses of GC3Pie: parallel processing

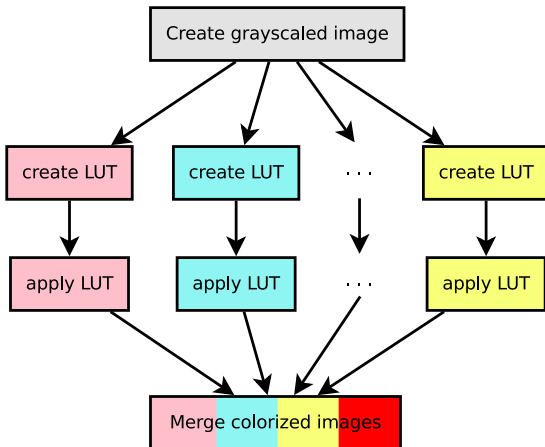
(At times, you chop a large input file into pieces and process each one separately instead.)

“For example, say we have a de novo assembly of 100,000 contigs. If we run 1 BLAST job against NR it could take as long as 50,000 minutes/35 days!! (30sec/query sequence), however if we split this job into subsets of 5,000 sequences and ran 20 jobs in “parallel” on a cluster, our total run-time is reduced to only 41 hours.”

Reference: <http://sfg.stanford.edu/BLAST.html>

Uses of GC3Pie: workflows

Orchestrate execution of several applications: some steps may run in parallel, some might need to be sequenced.



A typical GC3Pie script?

```
> ./warholize.py uzh-logo.png --watch 1
```



**University of
Zurich** ^{UZH}

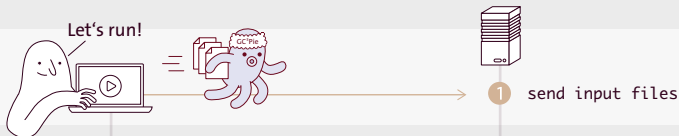


A typical high-throughput script structure

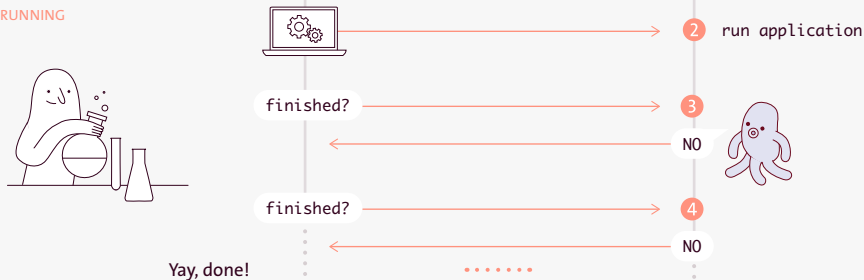
1. Initialize computational resources
2. Prepare programs and inputs for submission
3. Submit tasks
4. Monitor task status (loop)
5. Retrieve results
6. Postprocess and display

TASK STATE

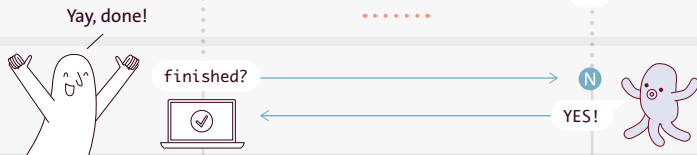
NEW



RUNNING

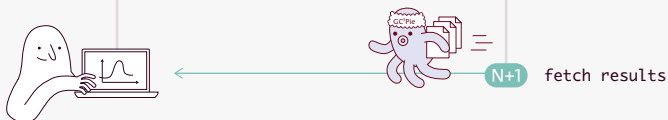


TERMINATING



TERMINATED

Do post-processing
or analyze more



TASK STATE

Now let's run 1'000 tasks!

Science
Cloud

create VMs

RUNNING

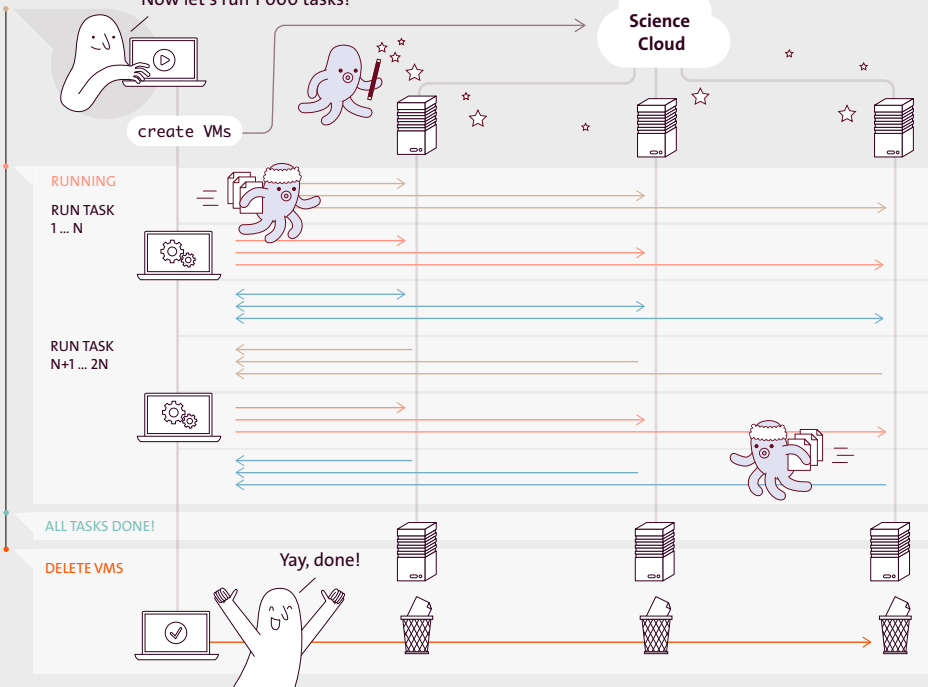
RUN TASK
1 ... N

RUN TASK
N+1 ... 2N

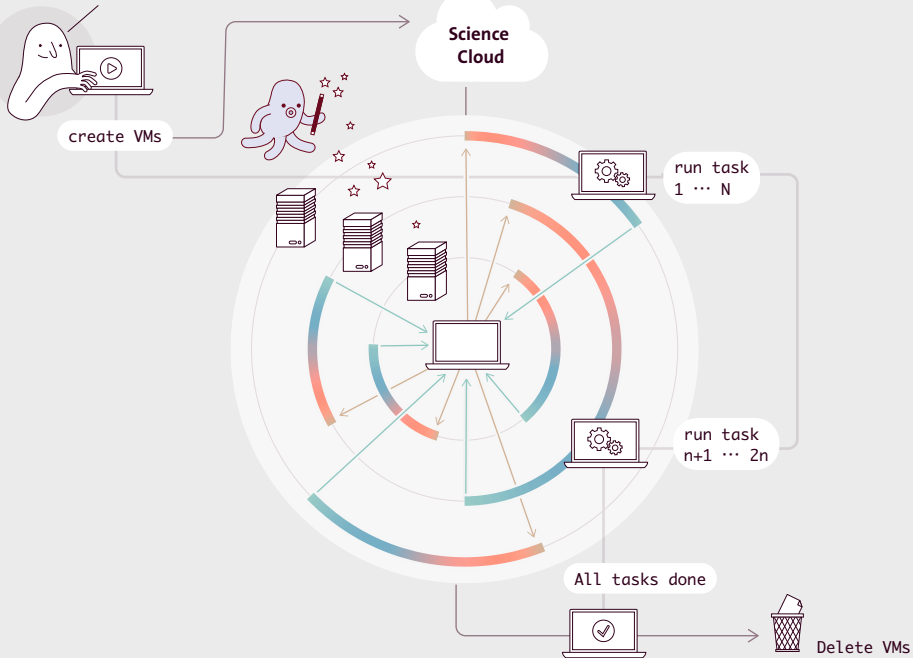
ALL TASKS DONE!

DELETE VMS

Yay, done!



Now let's run 1'000 tasks!



What GC3Pie handles for you

1. Resource allocation (e.g. starting new instances on ScienceCloud)
2. Selection of resources for each application in the session
3. Data transfer (e.g. copying input files in the new instances)
4. Remote execution of the application
5. Retrieval of results (e.g. copying output files from the running instance)
6. De-allocation of resources

Concepts and glossary

What is GC3Pie?

GC3Pie is . . .

1. An *opinionated* Python framework for defining and running computational workflows;
2. *A rapid development toolkit for running user applications on clusters and IaaS cloud resources;*
3. The worst name ever given to a middleware piece. . .

As *users*, *you're mostly interested in this part.*

What is GC3Pie?

GC3Pie is . . .

1. *An opinionated Python framework for defining and running computational workflows;*
2. *A rapid development toolkit for running user applications on clusters and IaaS cloud resources;*
3. The worst name ever given to a middleware piece. . .

However, you need to understand the basic concepts ;-)

GC3Pie glossary: Application

*GC3Pie runs user applications
on clusters and IaaS cloud resources*

An Application is just a command to execute.

GC3Pie glossary: Application

*GC3Pie runs **user applications**
on clusters and IaaS cloud resources*

An Application is just a command to execute.

If you can run it in the terminal,
you can run it in GC3Pie.

GC3Pie glossary: Application

*GC3Pie runs **user applications**
on clusters and IaaS cloud resources*

An Application is just a command to execute.

A single execution of an Application
is indeed called a Run.

(Other systems might call this a “Job”.)

GC3Pie glossary: Task

*GC3Pie **runs** user applications
on clusters and IaaS cloud resources*

More generally, GC3Pie runs Tasks.

Tasks are a superset of applications,
in that they include workflows.

► More on this later!

GC3Pie glossary: Resources

*GC3Pie runs user applications
on clusters and IaaS cloud **resources***

**Resources are the computing infrastructures
where GC3Pie executes applications.**

Resources include: your laptop, the “Hydra” cluster,
the Science Cloud, Amazon AWS.

Hands-on time!

Start a VM on Science Cloud, using the “GC3Pie Tools Training” snapshot.

Exercise A:

1. Run the `./warholize.py` script to get a new “warholized” version of the UZH logo.
2. What command-line option lets you run the whole workflow in one go?
3. How can you warholize multiple images at once?

Warholize!

```
> ./warholize.py uzh-logo.png -C 1
```



**University of
Zurich** ^{UZH}



Session-based scripts

`warholize.py` is a typical *session-based script*.

A *session* is just a named collection of jobs.

A *session-based script* creates a session and runs all the jobs in it until completion.

The output directory

If you don't specify an output directory for your job, a session-based script *usually* collects output in the current working directory.

If an output directory already exists, it will be *renamed* and never overwritten.

If you pass the option `-o DIRECTORY` to the script, all the output dirs will be saved inside that directory.

Create a session

A session-based script **creates a session** and runs all the jobs in it until completion.

Create session logo:

```
> ./warholize.py uzh-logo.png -s logo
```

Run a session until done

A session-based script creates a session and **runs all the jobs in it until completion.**

Run jobs in session `logo`, polling for updates every 5 seconds:

```
> ./warholize.py uzh-logo.png -s logo --watch 5
```

You can stop a GC3Pie script by pressing *Ctrl+C*.
Run it again to resume activity from where it stopped.

Run a session until done

A session-based script creates a session and **runs all the jobs in it until completion.**

Run jobs in session `logo`, polling for updates every 5 seconds:

```
> ./warholize.py uzh-logo.png -s logo --watch 5
```

You can stop a GC3Pie script by pressing *Ctrl+C*.
Run it again to resume activity from where it stopped.

Alternate display of session contents, I

Display top-level tasks in session `logo`:

```
> gsession list logo
```

Exercise B: Now try this yourself.

WTF??

```
> gsession list logo
gc3.gc3libs: WARNING: Failed loading file '/home/ubuntu/logo/jobs/WarholizeWorkflow.108': In
...
LoadError: Failed retrieving object from file '/home/ubuntu/logo/jobs/WarholizeWorkflow.108'
gc3.gc3libs: WARNING: Ignoring error from loading 'ParallelTaskCollection.107': Failed retr
+-----+-----+-----+-----+
| JobID | Job name | State | Info |
+-----+-----+-----+-----+
+-----+-----+-----+-----+
```

In order to work, all GC3Pie utilities need to access the Python script that generated the tasks and session.

To fix: set the PYTHONPATH variable to the directory containing your script:

```
> export PYTHONPATH=$PWD
```

WTF??

```
> gsession list logo
gc3.gc3libs: WARNING: Failed loading file '/home/ubuntu/logo/jobs/WarholizeWorkflow.108': In
...
LoadError: Failed retrieving object from file '/home/ubuntu/logo/jobs/WarholizeWorkflow.108
gc3.gc3libs: WARNING: Ignoring error from loading 'ParallelTaskCollection.107': Failed retr
+-----+-----+-----+-----+
| JobID | Job name | State | Info |
+-----+-----+-----+-----+
+-----+-----+-----+-----+
```

In order to work, all GC3Pie utilities need to access the Python script that generated the tasks and session.

To fix: set the `PYTHONPATH` variable to the directory containing your script:

```
> export PYTHONPATH=$PWD
```

Alternate display of session contents, II

Display *all* tasks in session log:

```
> gsession list --recursive log
```

► Workflows and task hierarchy

Alternate display of session contents, III

Display summary of tasks in session `logo`:

```
> gstat -b -s logo
```


Display session history

Show log of activity on tasks in session log:

```
> gsession log log
```

Resource definition

The gservers command

The `gservers` command is used to see configured and available resources.

```
> gservers
```

	localhost	
frontend	(Frontend host name)	localhost
type	(Access mode)	shellcmd
updated	(Accessible?)	True
queued	(Total queued jobs)	0
user_queued	(Own queued jobs)	0
user_run	(Own running jobs)	0
max_cores_per_job	(Max cores per job)	2
max_memory_per_core	(Max memory per core)	2GiB
max_walltime	(Max walltime per job)	12hour

Resources are defined in file `$HOME/.gc3/gc3pie.conf`

The gservers command

The `gservers` command is used to see **configured** and available resources.

```
> gservers
```

	localhost	
frontend	(Frontend host name)	localhost
type	(Access mode)	shellcmd
updated	(Accessible?)	True
queued	(Total queued jobs)	0
user_queued	(Own queued jobs)	0
user_run	(Own running jobs)	0
max_cores_per_job	(Max cores per job)	2
max_memory_per_core	(Max memory per core)	2GiB
max_walltime	(Max walltime per job)	12hour

Resources are defined in file `$HOME/.gc3/gc3pie.conf`

Example execution resources: local host

Allow GC3Pie to run tasks
on the local computer.

This is the default installed
by GC3Pie into

`$HOME/.gc3/gc3pie.conf`

```
[resource/localhost]
enabled = yes
type = shellcmd
frontend = localhost
transport = local
max_cores_per_job = 2
max_memory_per_core = 2GiB
max_walltime = 12 hours
max_cores = 2
architecture = x86_64
auth = none
override = no
```

Example execution resources: SLURM

Allow submission of jobs to the “Hydra” cluster.

```
[resource/hydra]
enabled = yes
type = slurm
frontend = login.s3it.uzh.ch
transport = ssh
auth = ssh_user_rmurri
max_walltime = 1 day
max_cores = 96
max_cores_per_job = 64
max_memory_per_core = 1 TiB
architecture = x86_64
prologue_content =
    module load cluster/largemem

[auth/ssh_user_rmurri]
type=ssh
username=rmurri
```

Example execution resources: OpenStack

```
[resource/sciencecloud]
enabled=no
type=openstack+shellcmd
auth=openstack
```

```
vm_pool_max_size = 32
security_group_name=default
security_group_rules=
  tcp:22:22:0.0.0.0/0,
  icmp:-1:-1:0.0.0.0/0
network_ids=
  c86b320c-9542-4032-a951-c8a068894cc2
```

```
# definition of a single execution VM
instance_type=1cpu-4ram-hpc
image_id=2b227d15-8f6a-42b0-b744-ede52ebe59f7
```

```
max_cores_per_job = 8
max_memory_per_core = 4 GiB
max_walltime = 90 days
max_cores = 32
architecture = x86_64
```

```
# how to connect
vm_auth=ssh_user_ubuntu
keypair_name=rmurri
public_key=~/.ssh/id_dsa.pub
```

```
[auth/ssh_user_ubuntu]
# default user on Ubuntu VM images
type=ssh
username=ubuntu
```

```
[auth/openstack]
# only need to set the 'type' here;
# any other value will be taken from
# the 'OS_*' environment variables
type = openstack
```

Allow running tasks on the
“ScienceCloud” VM
infrastructure.

Cannot access the cloud?

You need to tell GC3Pie what credentials to use for authenticating to OpenStack / ScienceCloud.

You can write them in the configuration file, or use the `sc-authenticate.sh` script to load them in the environment:

```
> source sc-authenticate.sh
```

```
Please enter your ScienceCloud project name [training]:
```

```
Please enter your ScienceCloud username [ubuntu]: shortname
```

```
Please enter your OpenStack Password:
```

Important: at the “username” prompt, enter your “UZH shortname” – *do not* accept the default!

Hands-on time!

Exercise C: Change the configuration file `~/.gc3/gc3pie.conf` to enable the `sciencecloud` resource. Verify with the `gservers` command that it works.

Select execution resource

Select where applications will be run with option `-r`:

```
> ./warholize.py -s logo -r localhost
```

The resource name must exist in the configuration file (i.e., check `gservers`' output).

Stopping a script and re-starting it with a different resource will likely result in an error: old tasks can no longer be found.

Passing requirements to the application

Some options are used to specify some requirements of *all* applications in a session:

- c NUM Set the number of CPU cores required for each job.
- m GB Set the amount of memory required per execution core
- w DURATION Set the time limit for each job; default is script-dependent.

These options have proven not to be much useful except for debugging/experimentation, so **they might be removed in a future release!**

Cloud backend management

Hands-on time!

Exercise D: Run the “Warholize” workflow on the ScienceCloud.
Does it succeed? Why?

See what VMs are in use

Use the `gcloud` command to show cloud usage:

```
> gcloud list
```

Limit concurrently-running jobs

Limit the maximum number of concurrently-running tasks with option `-J`:

```
> ./warholize.py -s logo -J 1
```

In large computational campaigns, it is important not to flood the execution resources with too many tasks.

Hands-on time!

Exercise E:

1. Run the “Warholize” workflow on the ScienceCloud. Stop the script while it’s running. Now start it again. What happens?
2. Run the “Warholize” workflow on the ScienceCloud. Stop the script while it’s running; list the jobs in the session and note down the IDs. Now run the script again, **adding the “-N” option**. When the script terminates, inspect the session again and note the IDs. What has happened? Why?

Use -N with caution!

From the session-based script's --help output:

-N, --new-session

Discard any information saved in the session dir (see '--session' option) and start a new session afresh. **Any information about previous jobs is lost.**

Cleanup unused VMs

Use the `gcloud` command again:

- To stop *all* unused VMs:

```
> gcloud cleanup
```

Exercise F: Do it. Now.

- To stop a specific VM:

```
> gcloud terminate f031d6ad-bd6c-439e-9a98-4d64
```

Session management

Aborting a single task

To stop and abort a single task, use the `gkill` command:

```
> gkill -s logo MyApplication.123
```

Exercise : What happens if you try to abort a task collection?

Aborting a single task

To stop and abort a single task, use the `gkill` command:

```
> gkill -s logo MyApplication.123
```

Exercise : What happens if you try to abort a task collection?

Aborting a whole session

Kill all the running tasks in a session again using the `gkill` command:

```
> gkill -s logo -A
```

Alternatively you can use `gsession abort`:

```
> gsession abort logo
```

Selecting tasks from a session, I

The `gselect` command is the go-to tool for selective listing of tasks in a session. For example, to list finished tasks:

```
> gselect -s logo --state TERMINATED
```

The output of `gselect` is a list of task IDs, to be fed into another GC3Pie command. For example, to kill all queued tasks:

```
> gselect -s logo --state SUBMITTED | xargs gkill -s logo
```

Selecting tasks from a session, II

The `gselect` command has many different options to select tasks:

```
> gselect --help
usage: gselect [-h] [-V] [-v] [--config-files CONFIG_FILES] -s SESSION
               [--error-message REGEXP] [--input-file FILENAME]
               [--jobname REGEXP] [--jobid REGEXP] [-l STATES]
               [--output-file FILENAME] [--output-message REGEXP]
               [--successful] [--submitted-after DATE]
               [--submitted-before DATE] [--unsuccessful]
```

Exercise : Use `gselect` to print the IDs of the “TricolorizeImage” tasks in the last “Warholize” session.

Thank you!

Any questions?

GC3Pie manual: <http://gc3pie.readthedocs.io/>

Mailing-list: gc3pie@googlegroups.com
or read online at
<https://groups.google.com/d/forum/gc3pie>

GC3Pie chat room: <http://gitter.im/gc3pie/chat>

Workflows

The “Warholize” workflow

How do we “warholize” an arbitrary image?

1. Convert the original image to grayscale.
2. Colorize the grayscale image using three different colors for each tile.
3. Arrange all the colorized images into an $N \times N$ frame.

Reference: <http://gc3pie.readthedocs.org/en/master/programmers/tutorials/warholize/warholize.html>

GC3Pie glossary: Task Collections

The basic unit of work in a GC3Pie workflow is called a Task.

The `Application` class that you already know is a kind of Task (in programming speak, it's a derived class).

A set of Tasks is itself a Task, and is called a `TaskCollection`.

Running tasks in sequence

To run tasks in an ordered sequence, one after the other, GC3Pie provides a `SequentialTaskCollection` class.

It is created with a list of tasks, and runs all of them in the order given. The sequence is dynamic, in that you can add new tasks on the fly, re-run existing ones, or remove future tasks.

A `SequentialTaskCollection` is itself a task.

Running tasks in parallel

To run tasks in parallel (i.e., they have no inter-dependency), GC3Pie provides a `ParallelTaskCollection` class.

It is created with a list of tasks, and runs all of them in parallel (compatibly with the computational resource limits).

A `ParallelTaskCollection` is itself a task.

Putting it all together

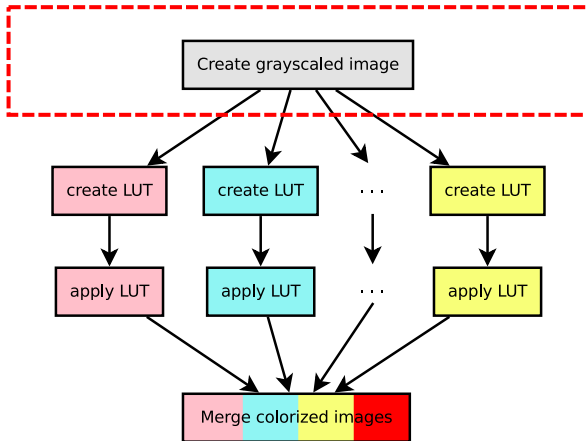
So tasks can be:

- Application instances,
- SequentialTaskCollections,
- ParallelTaskCollections.

So you can nest them, and create parallelly-running sequences, or sequences of “job explosions” (many jobs in parallel), or any combination of this.

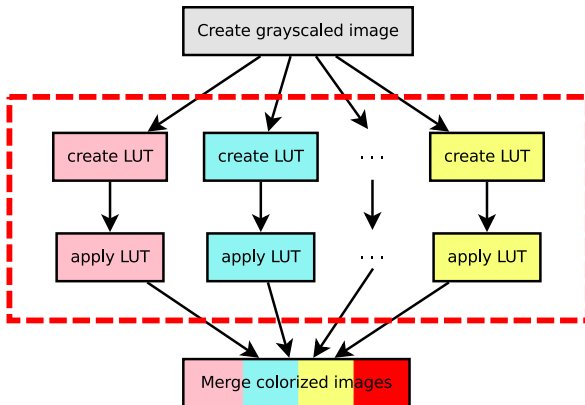
The Warholize workflow, I

1. Convert the original image to grayscale.



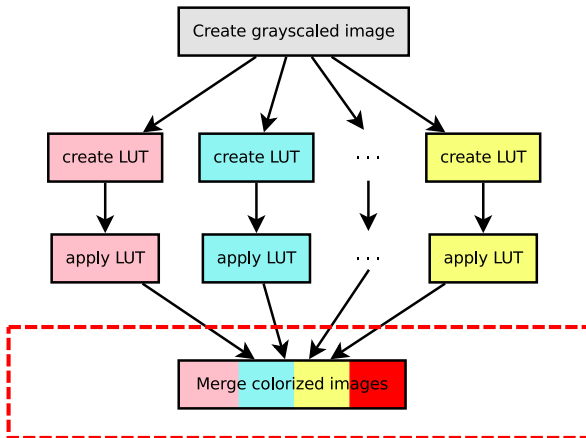
The Warholize workflow, II

2. Colorize the grayscale image using three different colors for each tile.



The Warholize workflow, III

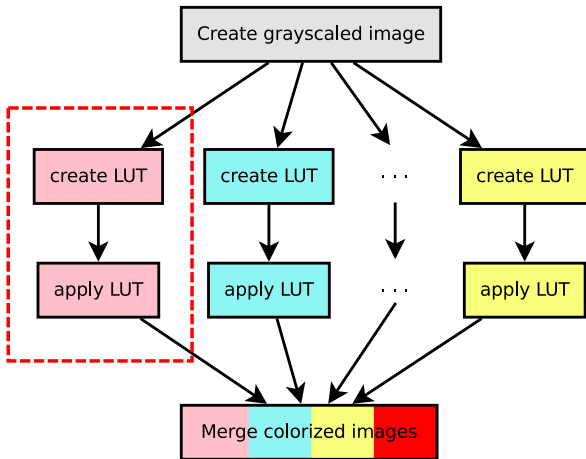
3. Arrange all the colored images into an $N \times N$ frame.



The Warholize workflow, IV

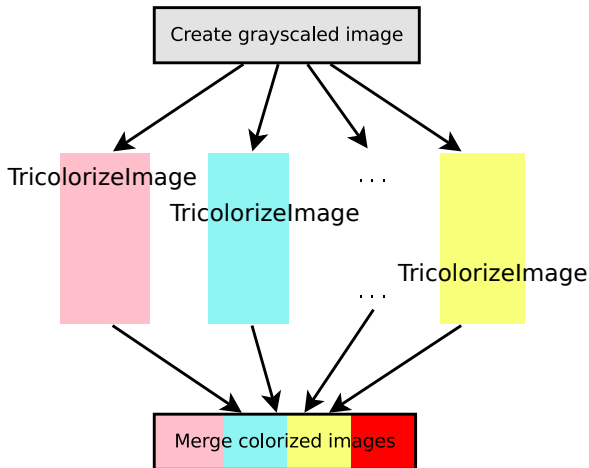
Step 2 actually entails two sub-steps:

- a) mapping greyscale levels to random colors,
- b) applying this mapping to produce new images



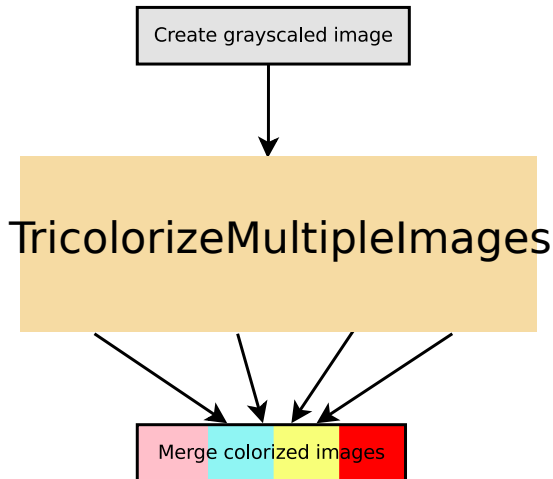
The Warholize workflow, V

So, Step 2 is a `SequentialTaskCollection`-type task. Let's call this two-pass sequence `TricolorizeImage`.



The Warholize workflow, VI

All the `TricolorizeImage` instances run in parallel. Collect them into a `ParallelTaskCollection`-type task, called `TricolorizeMultipleImages`.



The Warholize workflow, VII

Now we are left with a three-step sequence: greyscale, TricolorizeMultipleImages, montage. This can be defined again as a SequentialTaskCollection-type task, the WarholizeWorkflow.

