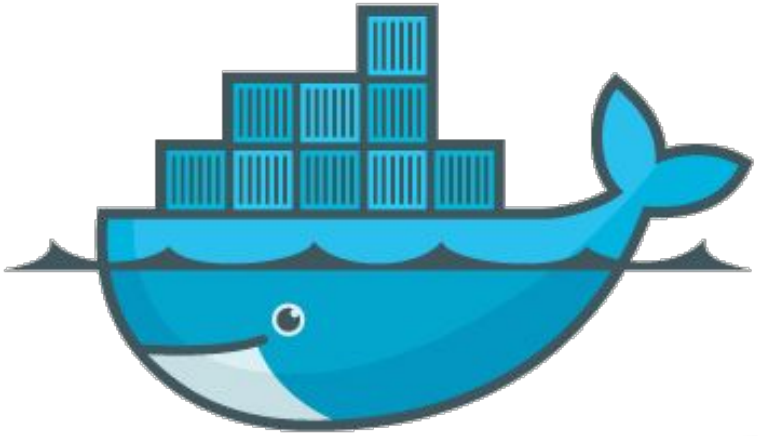
Scientific Filesystem

Vanessa Sochat
vsochat@stanford.edu

Welcome!




SHIFTER

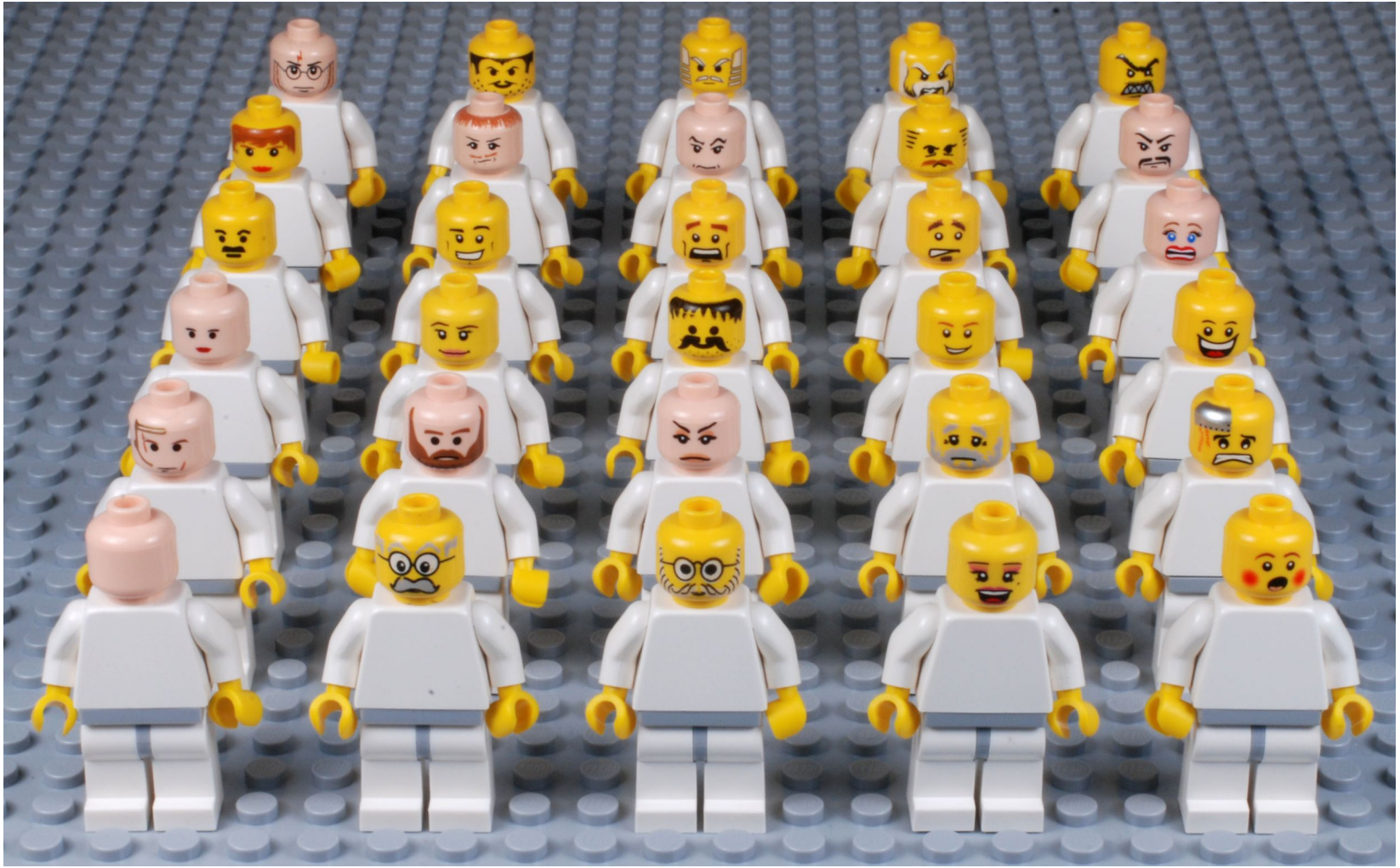

Charliecloud

“

Let’s all be friends!

Once upon a time...

There was a scientist



RESEARCH ARTICLE SUMMARY

PSYCHOLOGY

Estimating the reproducibility of psychological science

Open Science Collaboration*

INTRODUCTION: Reproducibility is a defining feature of science, but the extent to which it characterizes current research is unknown. Scientific claims should not gain credence because of the status or authority of their originator but by the replicability of their supporting evidence. Even research of exemplary quality may have irreproducible empirical findings because of random or systematic error.

RATIONALE: There is concern about the rate and predictors of reproducibility, but limited evidence. Potentially problematic practices increase the likelihood that findings are false. Direct replication is the gold standard for assessing the reproducibility of a previously observed finding and is the means of establishing reproducibility of a finding with new data. We conducted a large-scale, collaborative effort to obtain an initial estimate of the reproducibility of psychological

ON OUR WEB SITE

Read the full article at http://dx.doi.org/10.1126/science.aac4716

substantial decline. Ninety-seven percent of original studies had significant results ($P < .05$). Thirty-six percent of replications had significant results; 47% of original effect sizes were in the 95% confidence interval of the replication effect size; 39% of effects were subjectively rated to have replicated the original result; and if no bias in original results is assumed, combining original and replication results left 68% with statistically significant effects. Correlational tests suggest that replication success was better predicted by the strength of original evidence than by characteristics of the original and replication

NATURE | NEWS

Over half of psychology studies fail reproducibility test

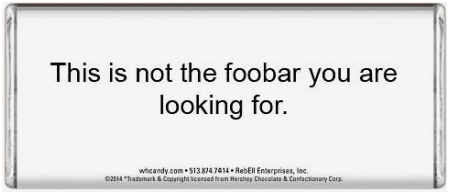
Largest replication study to date casts doubt on many published positive results.

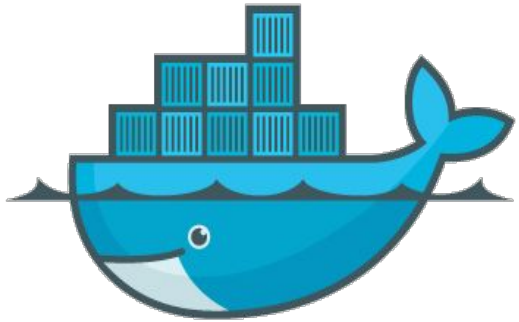
Monya Baker

27 August 2015

Foo → Bar →

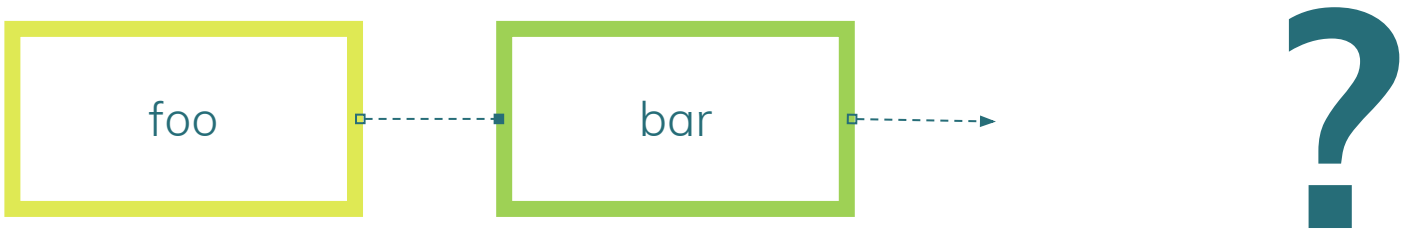
foO → baR →



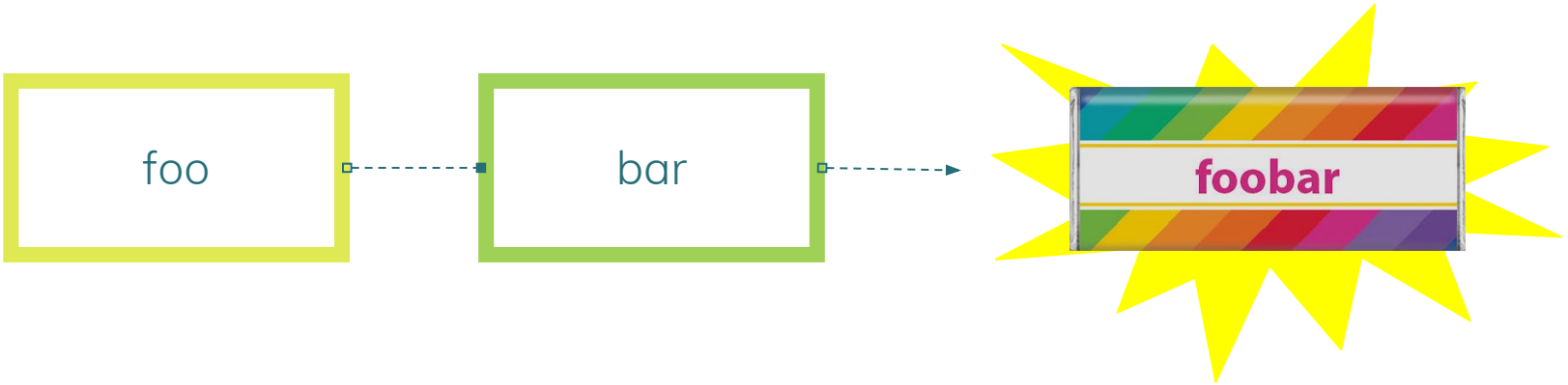


science +

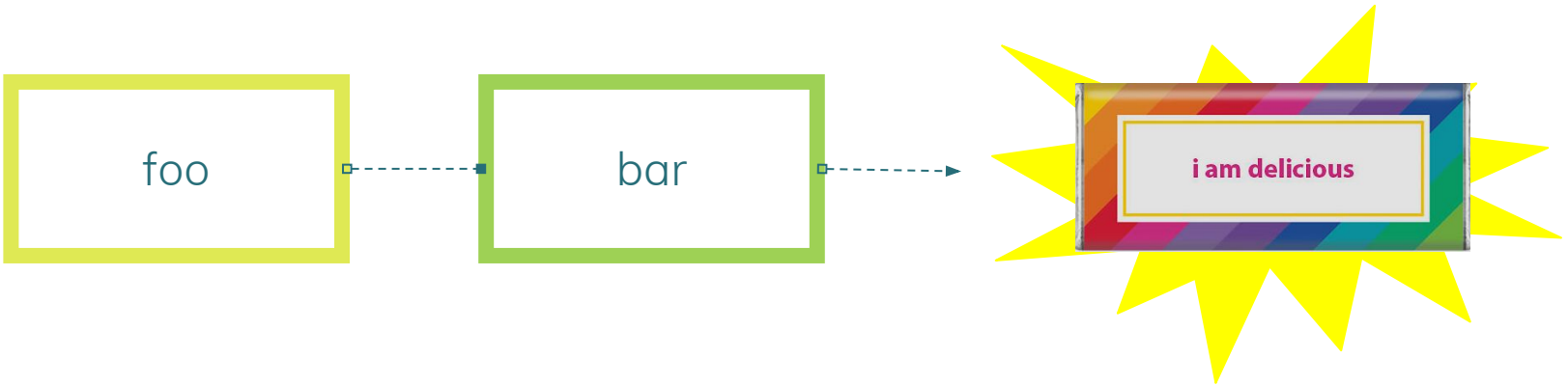
Pipeline foobar



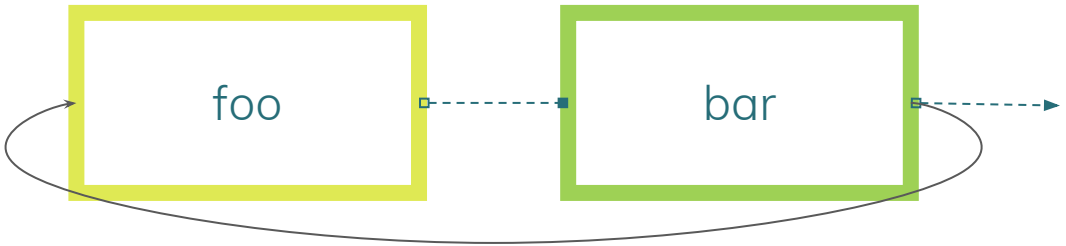
Pipeline foobar



Pipeline foobar

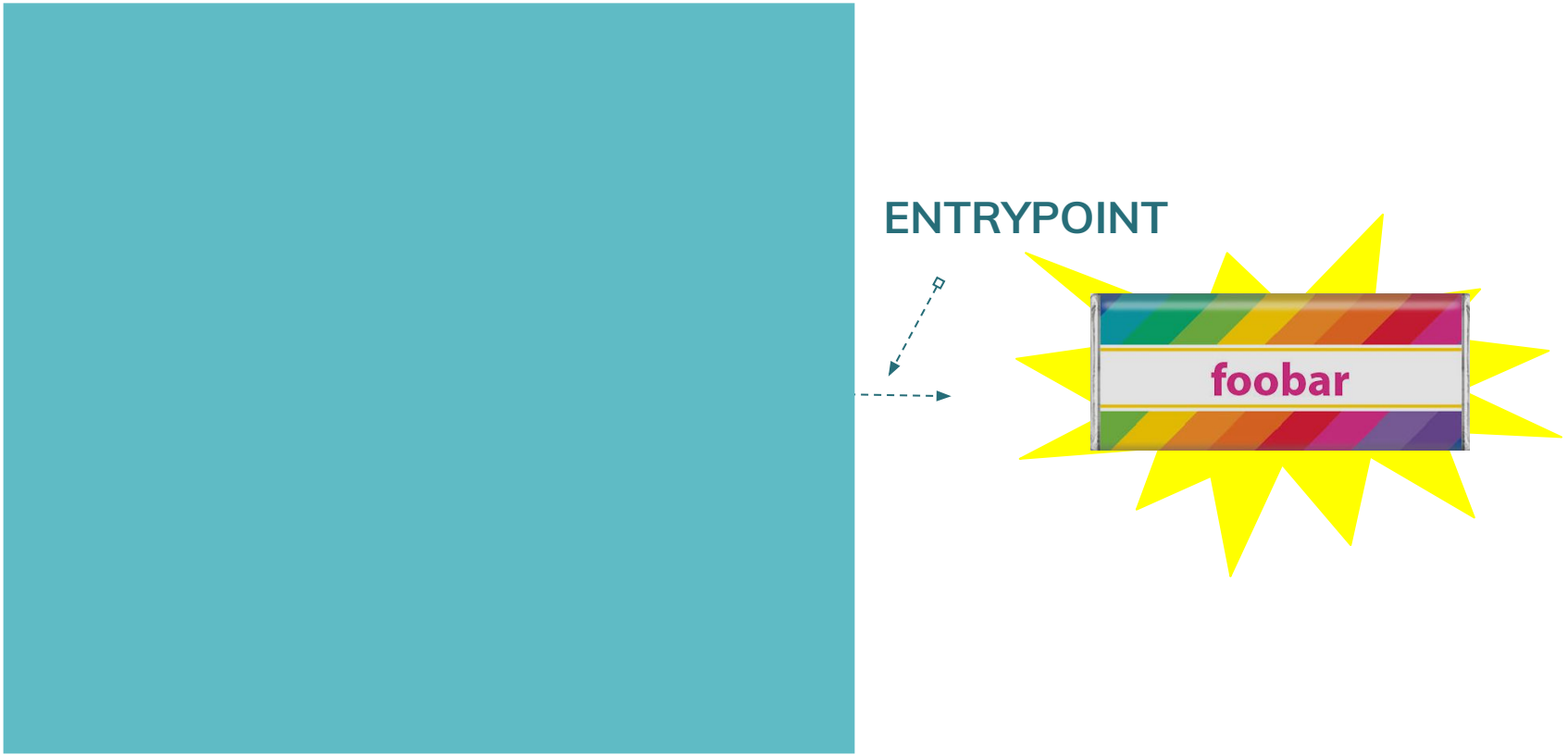


Pipeline foobar

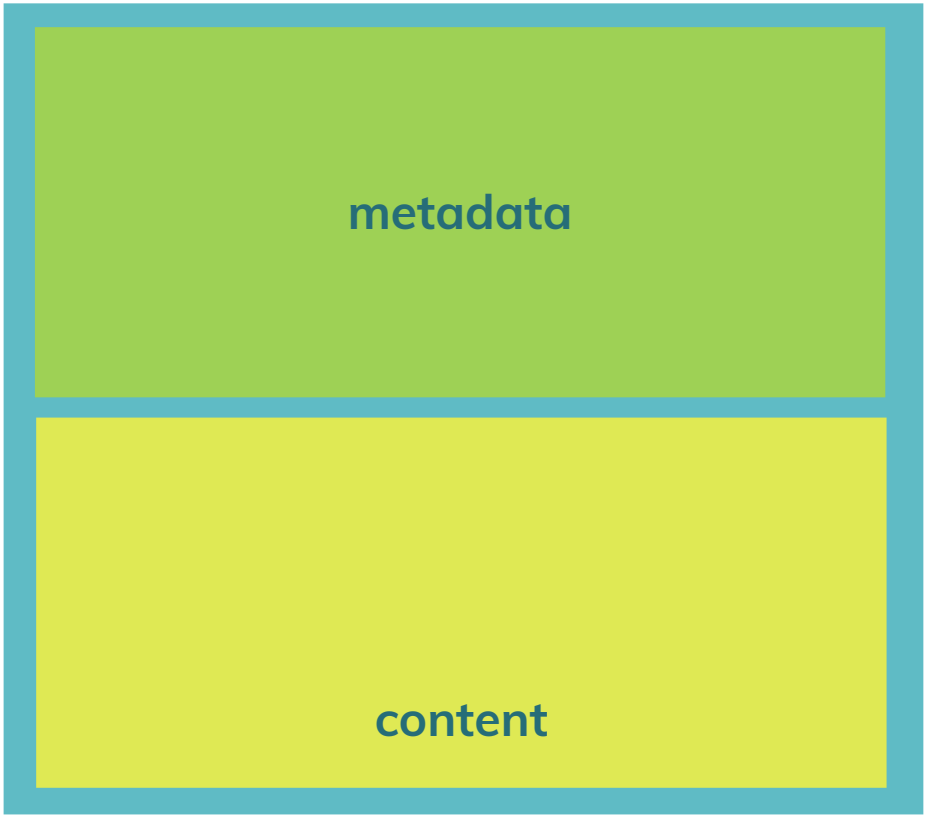


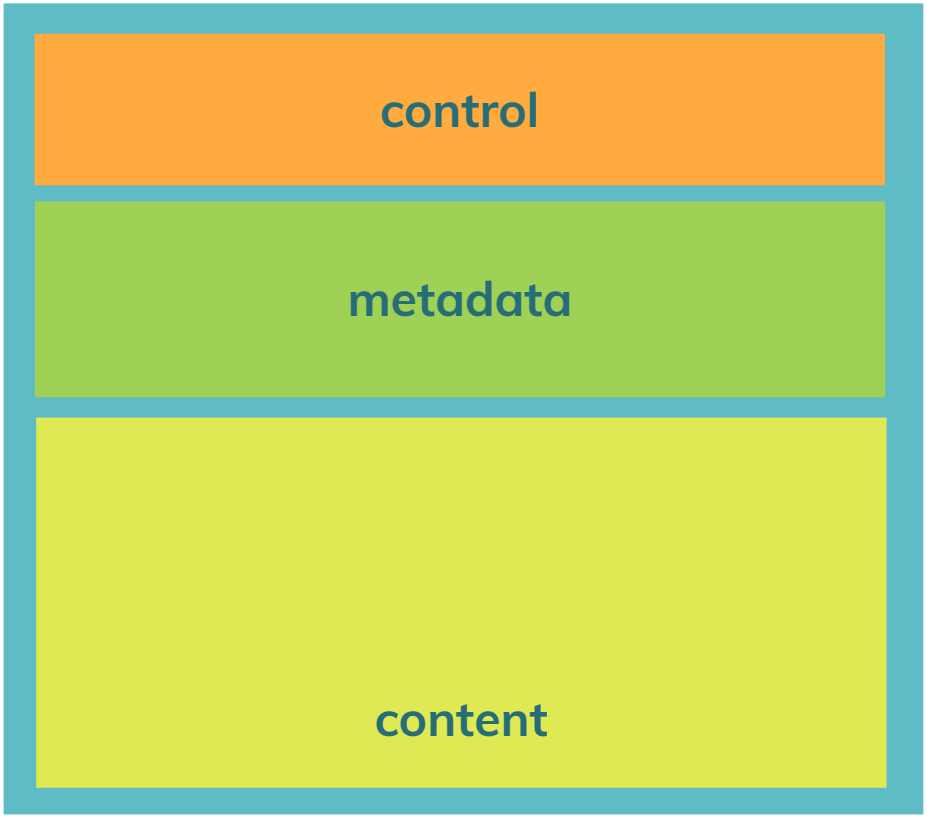
reproducible

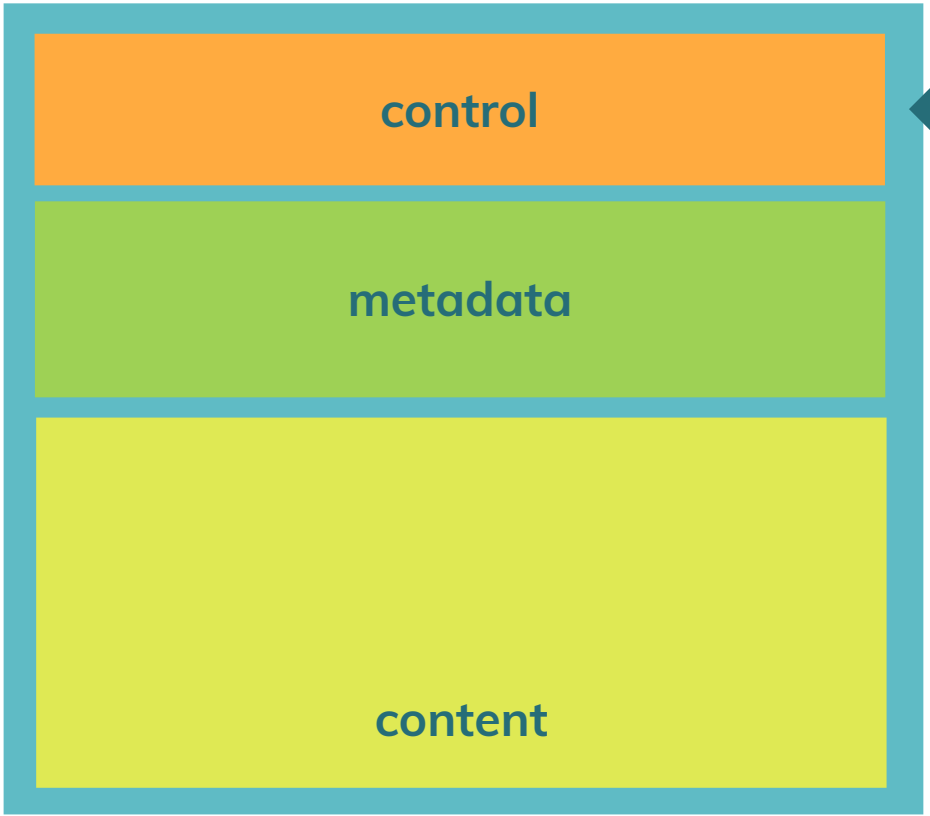
Give it a try?


ENTRYPOINT
foobar


What about the human component of reproducibility?


metadata
content


control
metadata
content

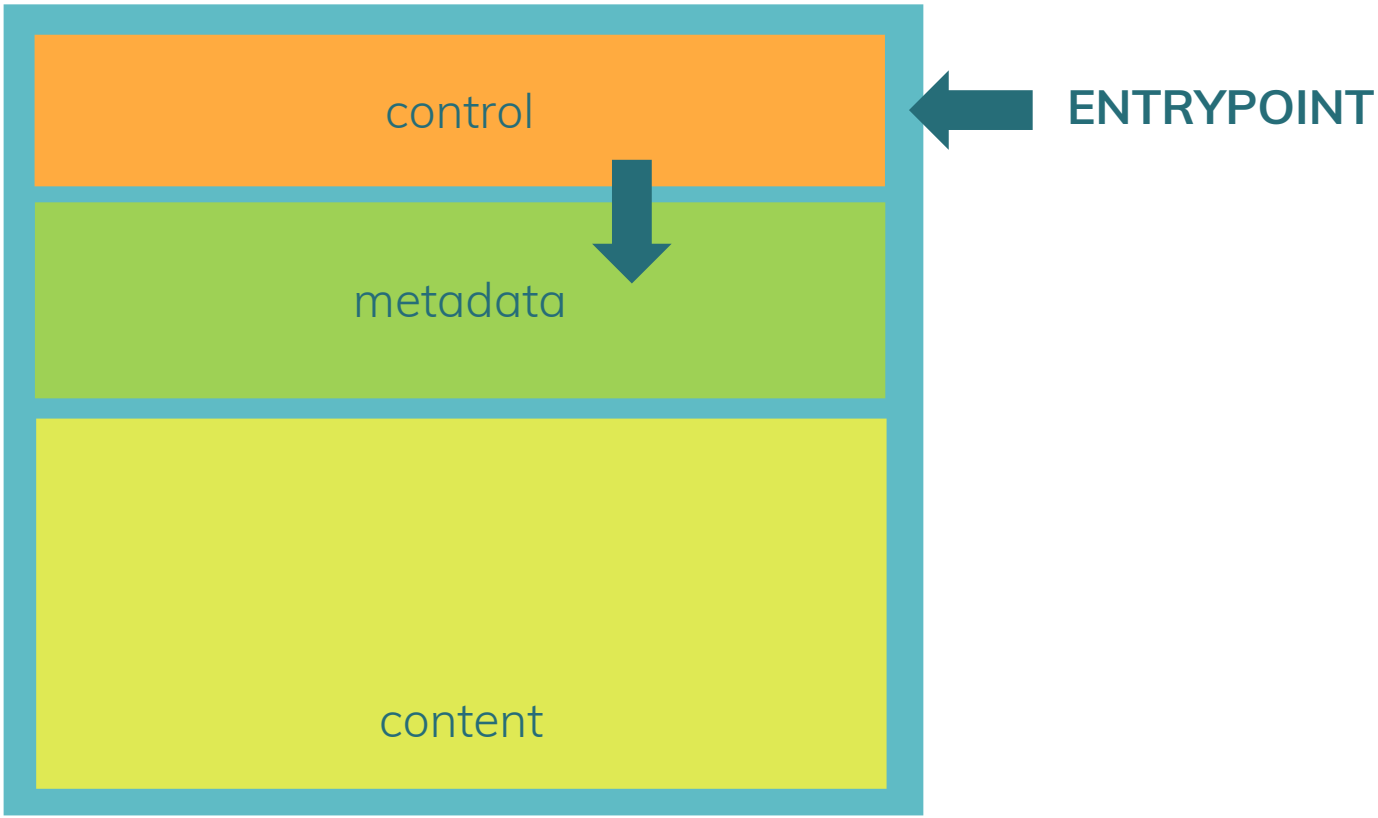


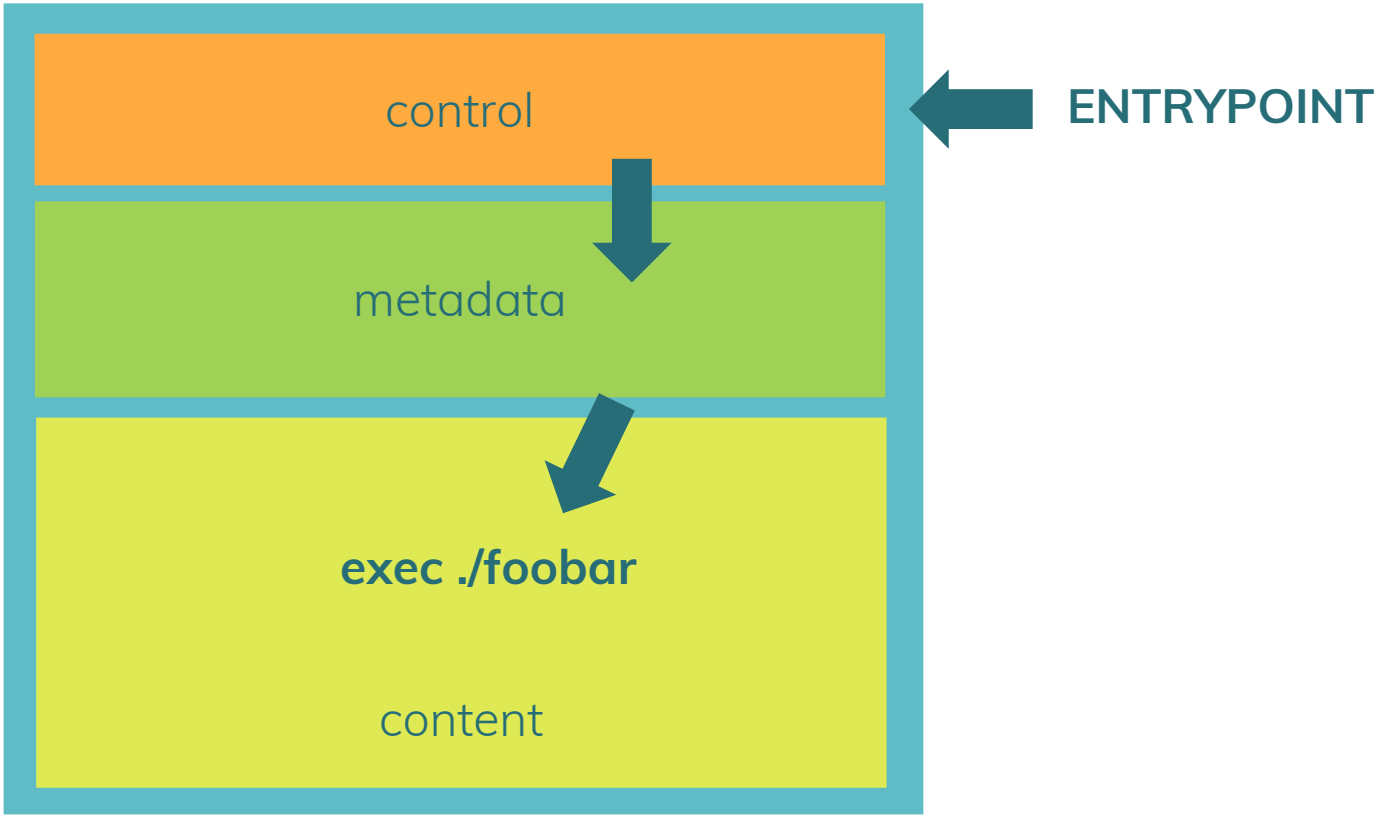
ENTRYPOINT

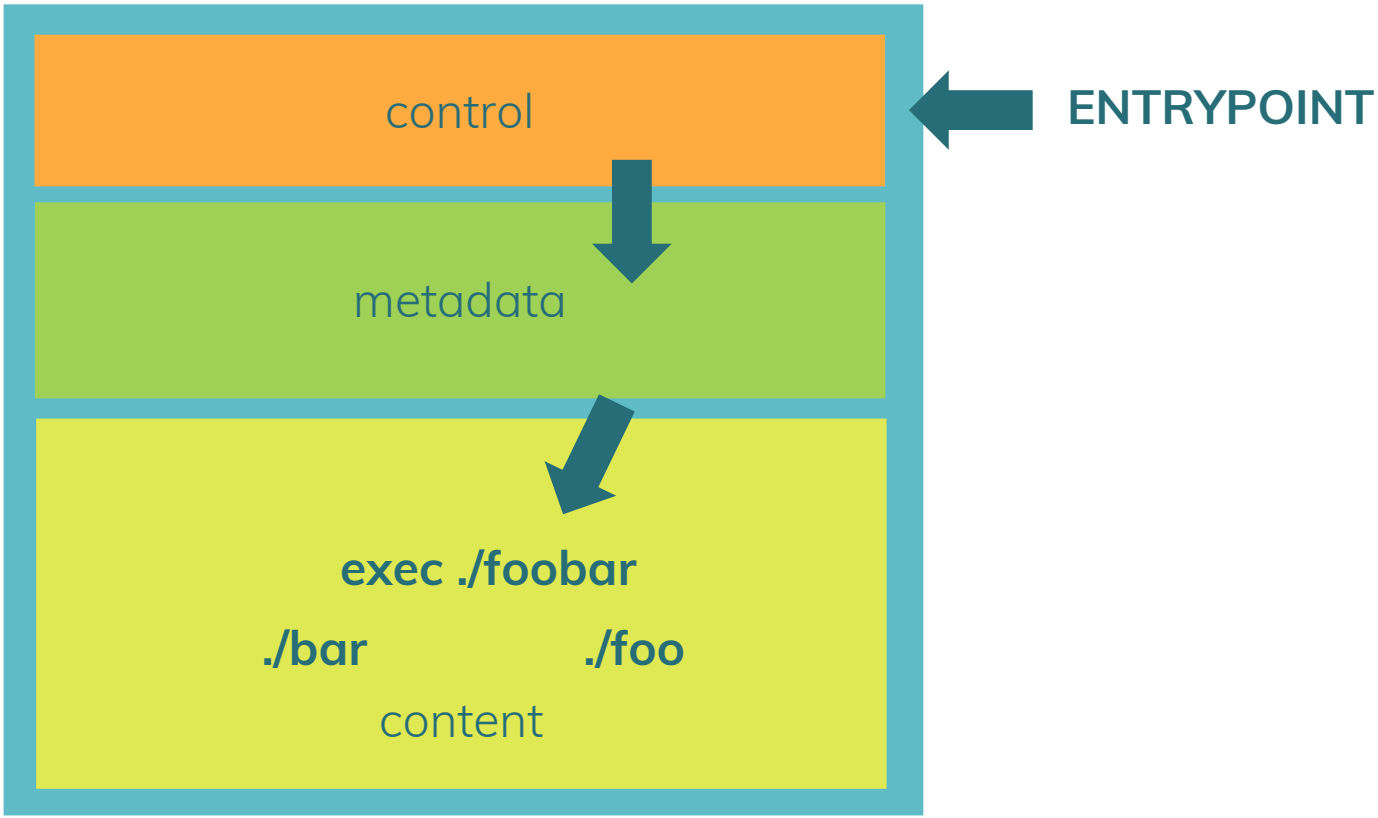
Entrypoint
Environment
Metadata

Software
Libraries
Scripts

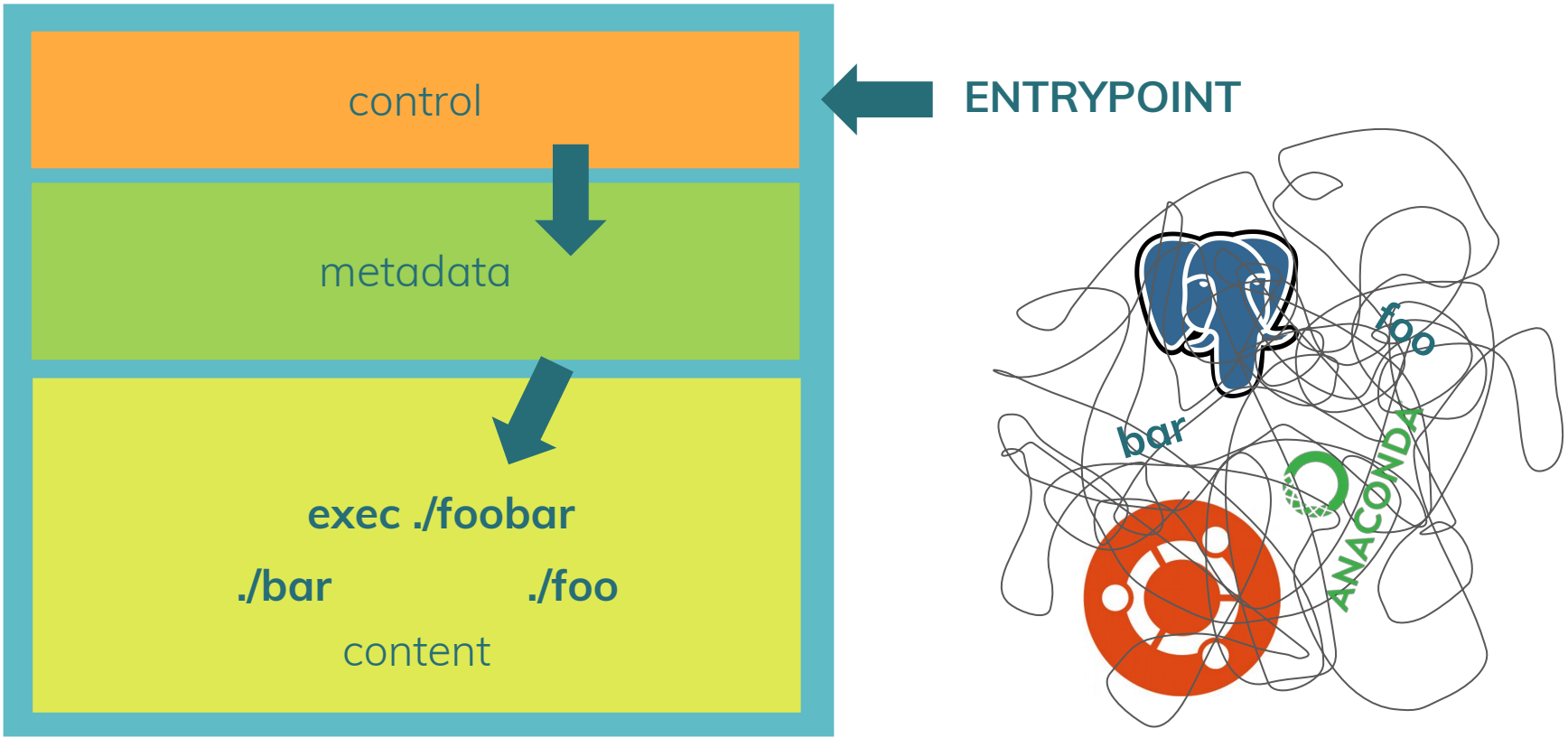

“
Where are foo and bar?


control
metadata
content
ENTRYPOINT


control
metadata
exec ./foobar
content
ENTRYPOINT



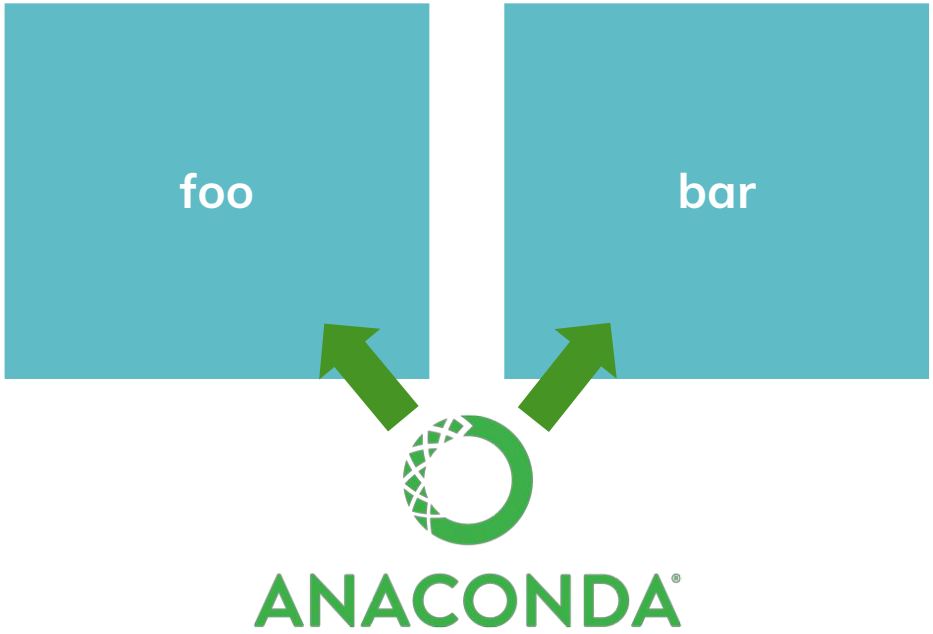
ENTRYPOINT


control
metadata
exec ./foobar
./bar
./foo
content
ENTRYPOINT
foo
bar
ANACONDA


“
I just want to run bar.

Solution 1: More containers

Solution 1: More containers



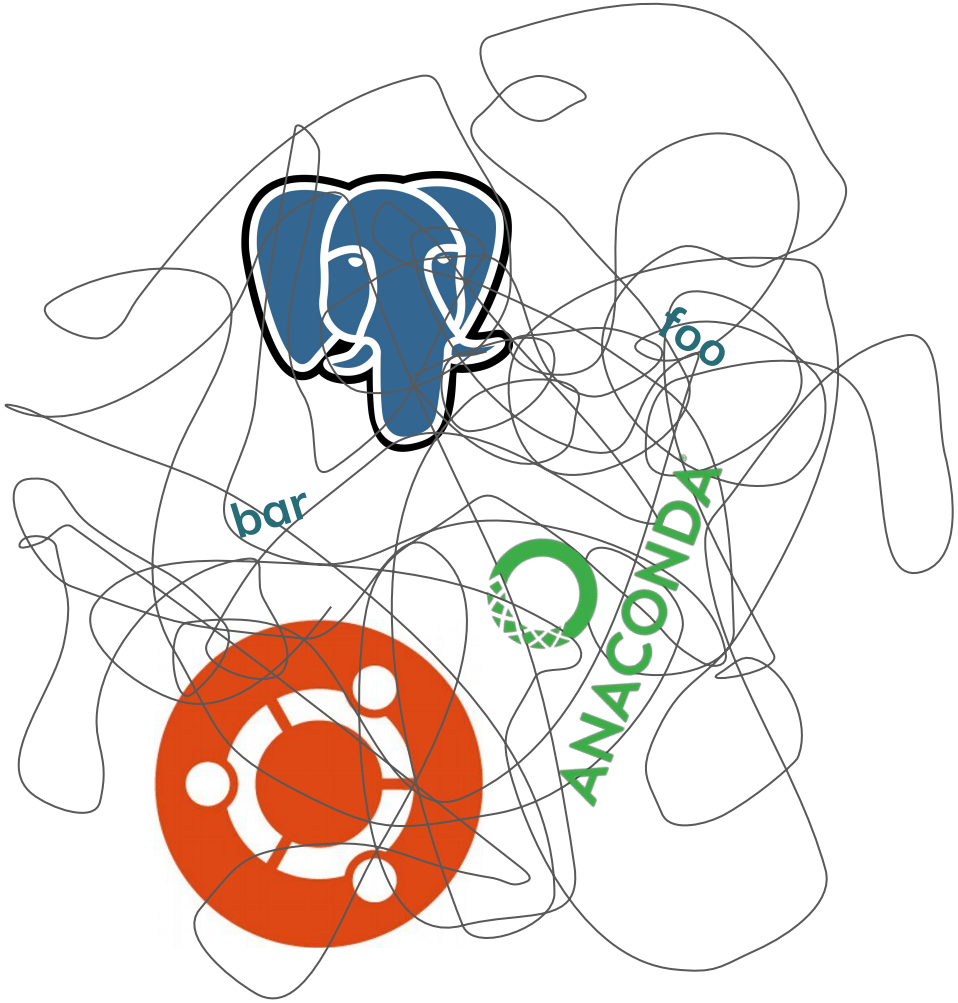
Solution 2: Custom entrypoint

```
If foo:
    exec foo
 else If bar:
    exec bar
else:
    exec foobar
```

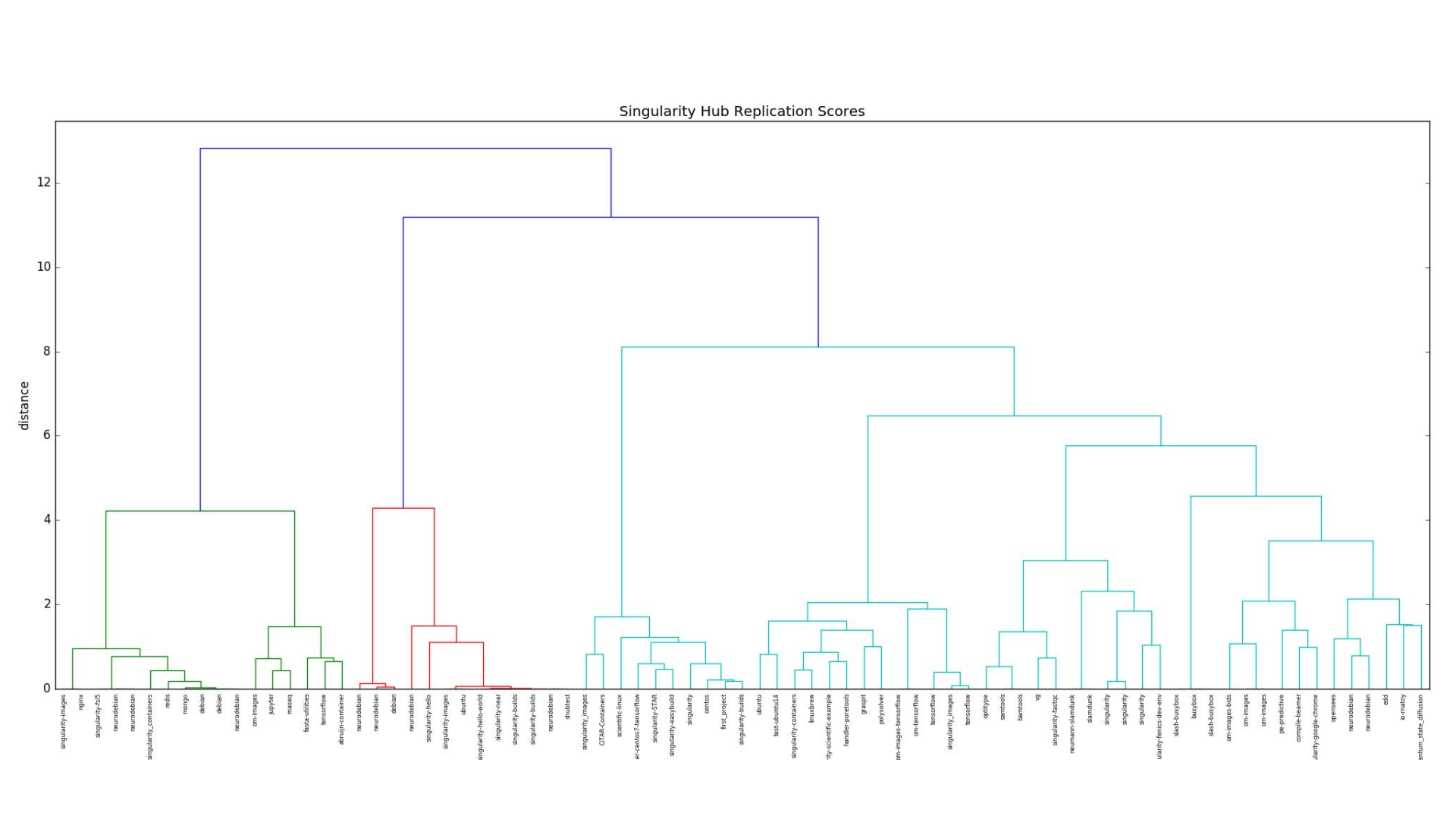
Solution 2: Custom entrypoint

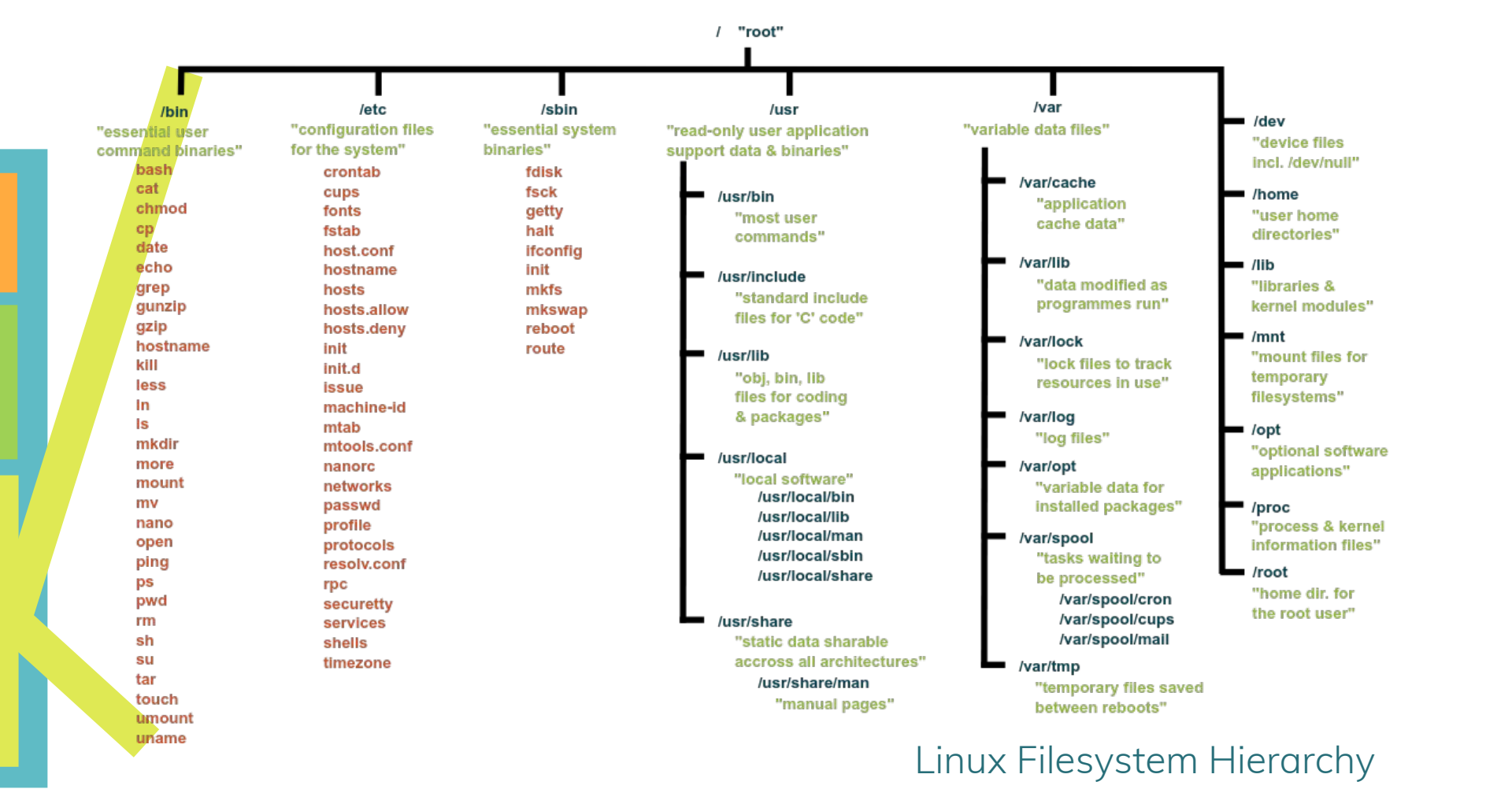
```
If foo:
    exec foo
 else If bar:
    exec bar
else:
    exec foobar
```

```
BESTAPP=FOO
BESTAPP=BAR
EXPORT BESTAPP
```


foo
bar
ANACONDA


“
The Linux Filesystem Hierarchy was
not optimized for scientific applications.


Singularity Hub Replication Scores
distance
0
2
4
6
8
10
12
singularity-images
nginx
singularity-fs5
neurodebian
neurodebian
singularity_containers
redis
mongo
debian
debian
neurodebian
om-images
jupyter
maseq
fasta-utilities
tensorflow
abruijn-container
neurodebian
neurodebian
debian
neurodebian
singularity-hello
singularity-images
ubuntu
singularity-hello-world
singularity-near
singularity-builds
singularity-builds
neurodebian
shubtest
singularity_images
CITAR-Containers
scientific-linux
er-centos7-tensorflow
singularity-STAR
singularity-easybuild
singularity
centos
first_project
singularity-builds
ubuntu
test-ubuntu14
singularity-containers
linuxbrew
rity-scientific-example
handler-ponetools
graspit
polyolver
om-images-tensorflow
om-tensorflow
tensorflow
singularity_images
tensorflow
optitype
samtools
bamtools
vg
singularity-fastqc
neumann-slamdunk
slamdunk
singularity
singularity
singularity
ularity-fenics-dev-env
slash-busybox
busybox
slash-busybox
om-images-bids
om-images
om-images
pe-predictive
compile-beamer
ularity-google-chrome
openseses
neurodebian
neurodebian
edd
io-matoy
antum_state_diffusion



Linux Filesystem Hierarchy

control

foo metadata

bar metadata

foo

bar

global content


“
Let’s make a Scientific Filesystem!

What is SCIF?

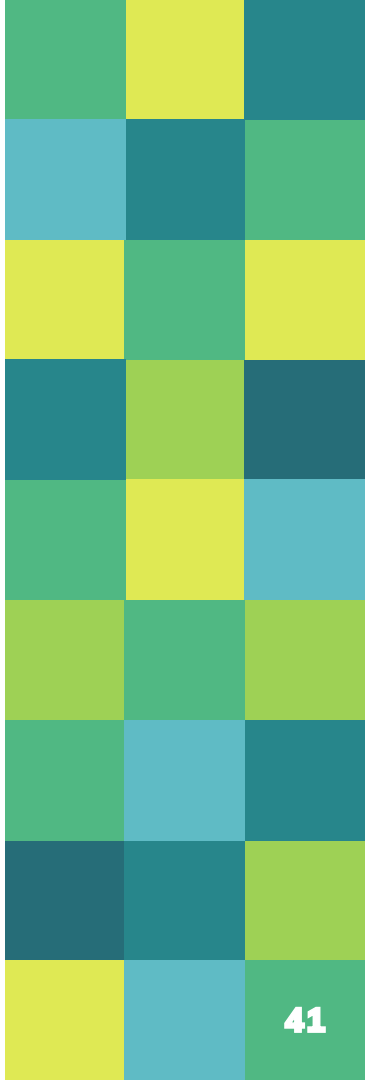
- The specification defines a **filesystem structure** and **environment variable** namespace that interact with **functions** to produce discoverable software applications.

Client

A controller for a SCIF, either for a developer or a user

Integration

A third party software or tool that understands the SCIF structure and interacts with all or some portion of it



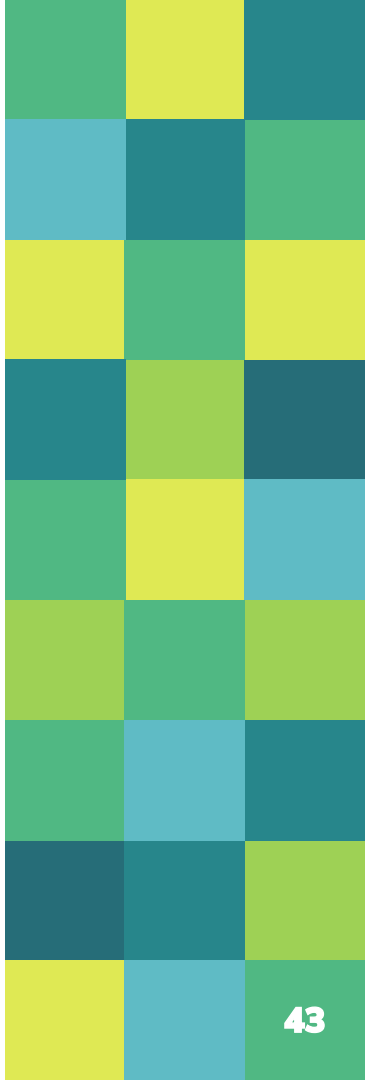
Goals

- Enable users to easily generate predictable, discoverable applications
- A recipe format that maps to a SCIF, and vice versa
- Software to generate and manage SCIF

Non-Goals

The Scientific Filesystem (SCIF) explicitly does not implement, define, or provide in v0.2:

- Package Management
- Workflow Management
- A mechanism for authentication and authorization or management of file permissions.
- Support or integration with non-linux operating systems.



The SCIF Recipe

The SCIF Recipe

Metadata

BESTAPP=foo
VERSION=1.0

Entry Points

./run-foo.sh
./run-bar.sh

Content

Files
Install Routine

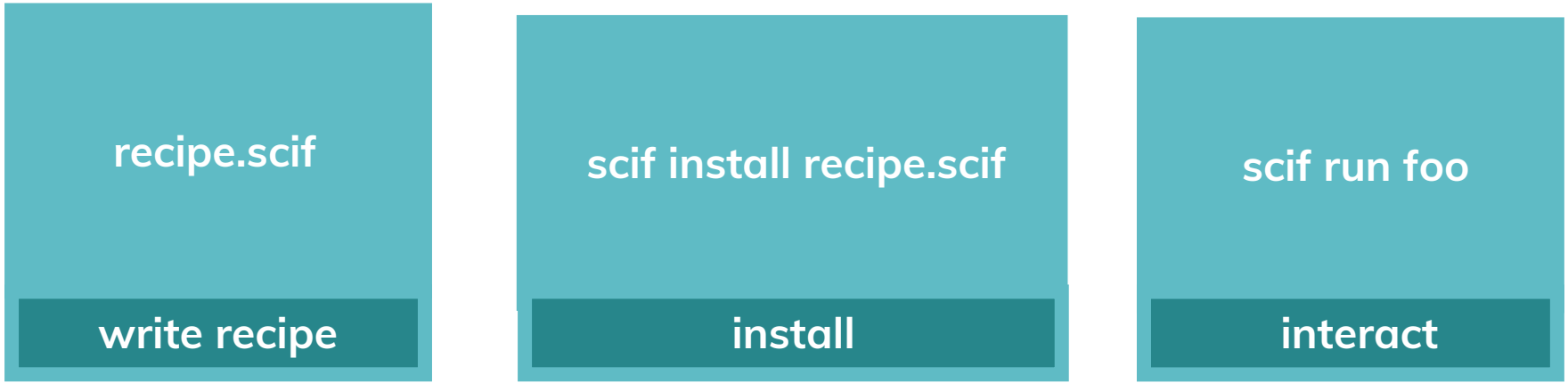
%app[action] [name]

```
%apprun hello-world
   /bin/bash hello-world.sh
%appinstall hello-world
   echo "echo 'Hello World!'" >> $SCIF_APPBIN/hello-world.sh
   chmod u+x $SCIF_APPBIN/hello-world.sh
%appenv hello-world
   THEBESTAPP=$SCIF_APPNAME
   export THEBESTAPP
%applabels hello-world
   MAINTAINER Vanessasaur
%apphelp hello-world
   This is an example "Hello World" application.
   You can install it to a Scientific Filesystem
   (scif) with the command:
       scif install hello-world.scif
```

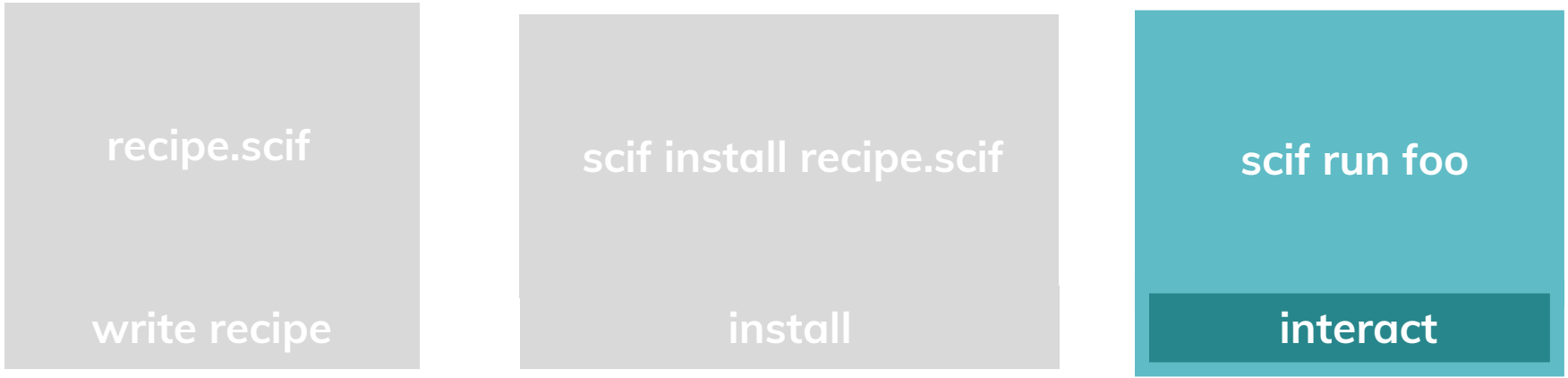
https://sci-f.github.io/tutorial-quick-start

hello-world.scif

Developer Interaction



User Interaction



User Interaction


```
./container  ...

apps
run
exec
help
inspect
shell
pyshell
```

./scif **apps**

docker run **vanessa/scif:hw** apps

./**scif-cli** apps

./scif **apps**

docker run **vanessa/scif:hw** apps

./**scif-cli** apps

./scif **run** hello-world

```
%apprun hello-world-echo
   exec echo “Hello World!”
```

docker run **vanessa/scif:hw** run hello-world-echo

./**scif-cli** run hello-world-echo

./scif **run** hello-world

```
%apprun hello-world-echo
    exec echo “Hello World!”
```

docker run **vanessa/scif:hw** run hello-world-echo

./**scif-cli** run hello-world-echo

./scif **exec** hello-world

docker run **vanessa/scif:hw** exec hello-world echo **[e]OMG**

./**scif-cli** exec hello-world echo **[e]OMG**

```
%appenv hello-world
    OMG=TACOS
    export OMG
```

./scif **exec** hello-world

docker run **vanessa/scif:hw** exec hello-world echo **[e]OMG**

./**scif-cli** exec hello-world echo **[e]OMG**

```
%appenv hello-world
    OMG=TACOS
    export OMG
```

sticker	description	example
`[e]`	an environment variable prefix	`[e]OMG` converts to `$OMG`
`[pipe]`	pipe (usually `|`)	env `[pipe]` grep _SCIF
`[out]`	output direction (usually `>`)	cat input.txt `[out]` output.txt
`[in]`	input direction (usually `<`)	
`[append]`	append to a file (usually `>>`)	echo “pancakes” » recipe.txt

./scif **inspect** hello-world

./scif **help** hello-world

docker run **vanessa/scif:hw** help hello-world

./**scif-cli** help hello-world

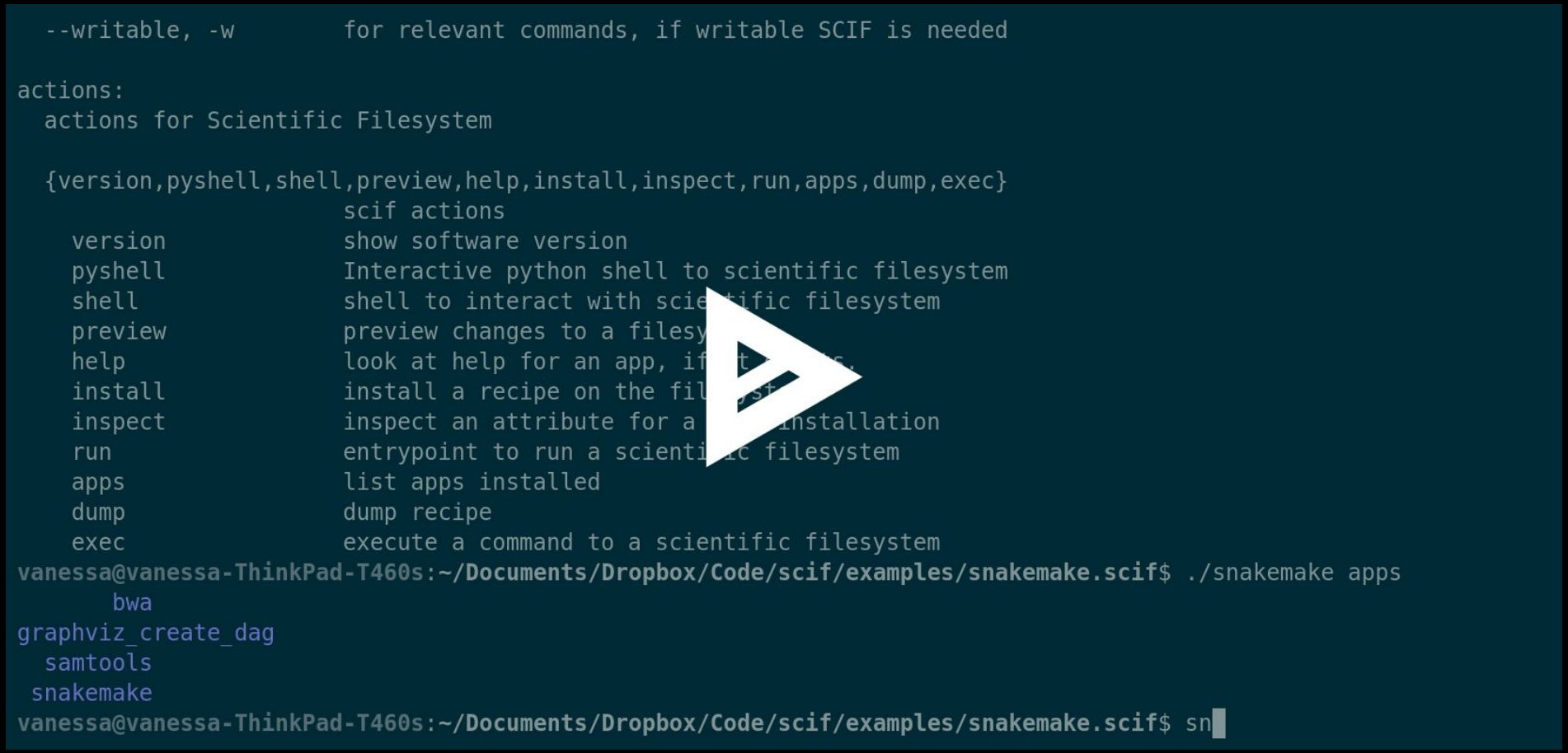
```
%apphelp hello-world
    This is an example "Hello World" application.
    You can install it to a Scientific Filesystem
    (scif) with the command:
        scif install hello-world.scif
```

./scif **shell** hello-world

```
%appenv hello-world
    OMG=TACOS
    export OMG
```

./scif **shell** hello-world

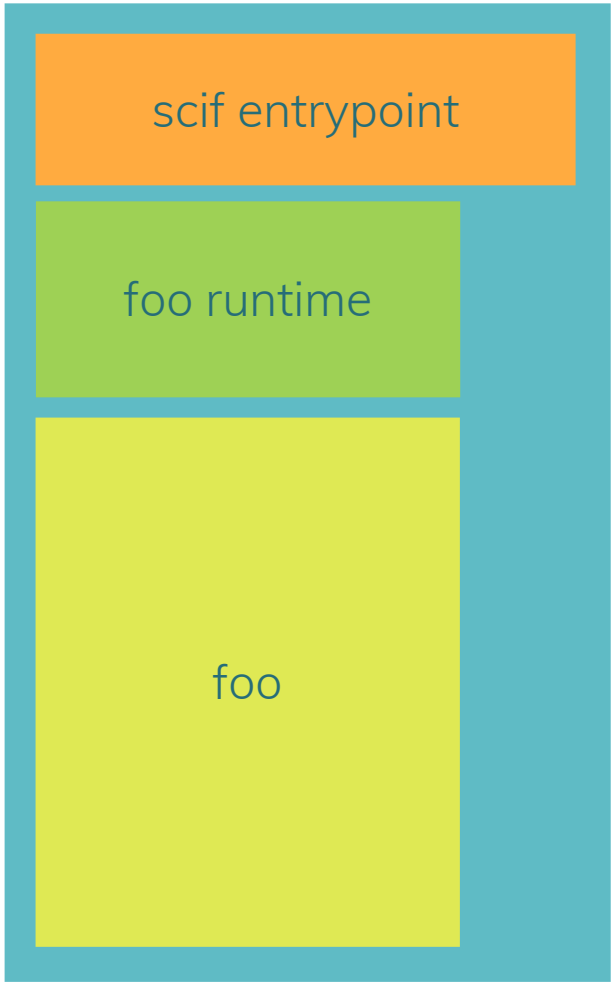
./scif **pyshell**


--writable, -w for relevant commands, if writable SCIF is needed
actions:
actions for Scientific Filesystem
{version,pyshell,shell,preview,help,install,inspect,run,apps,dump,exec}
scif actions
version show software version
pyshell Interactive python shell to scientific filesystem
shell shell to interact with scientific filesystem
preview preview changes to a filesy
help look at help for an app, if
install install a recipe on the fil
inspect inspect an attribute for a installation
run entrypoint to run a scientific filesystem
apps list apps installed
dump dump recipe
exec execute a command to a scientific filesystem
vanessa@vanessa-ThinkPad-T460s:~/Documents/Dropbox/Code/scif/examples/snakemake.scif$ ./snakemake apps
bwa
graphviz_create_dag
samtools
snakemake
vanessa@vanessa-ThinkPad-T460s:~/Documents/Dropbox/Code/scif/examples/snakemake.scif$ sn

Architecture

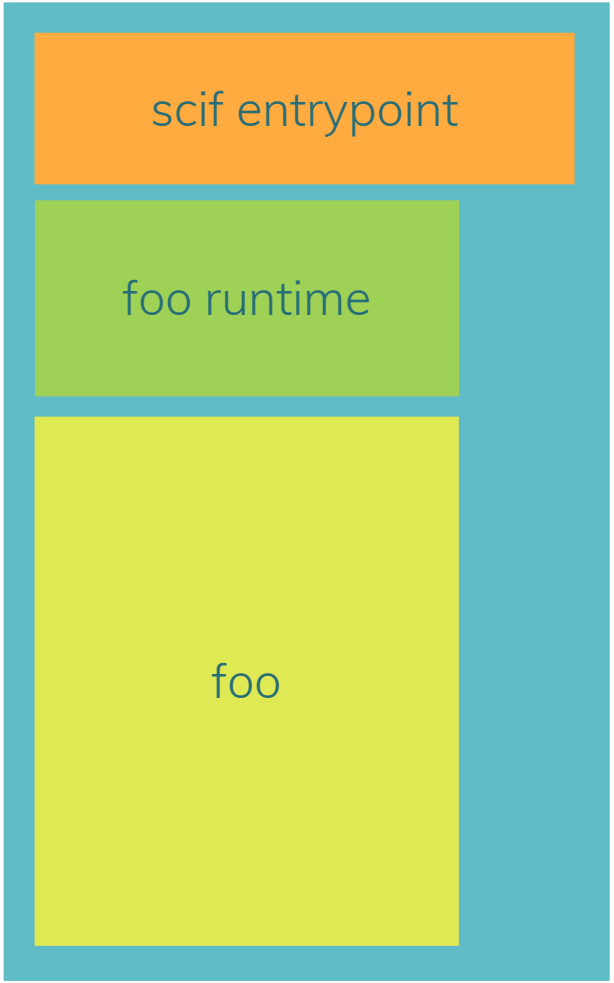
The Scientific Filesystem has a default root on the host to intentionally be separate from standard linux folders, and to not interfere with likely existing folders (e.g., just `/data` or `/apps`).

```
/scif
    /apps
    /data
```



Organization

```
/scif
   /apps
      /foo
         /bin
         /lib
         /scif
            runscript
            runscript.help
            labels.json
            environment.sh
```



Organization

```
/scif
   /apps
      /foo
         /bin      ← $PATH
         /lib      ← $LD_LIBRARY_PATH
         /scif
            runscript
            runscript.help
            labels.json
            environment.sh
```

Interaction

```
./container apps ...
    foo
    bar
```

```
/scif
   /apps
      /bar
      /foo
```

Interaction

```
./container exec bar [command]
```

```
/scif
   /apps
      /foo
         /bin
         /lib
         /scif
            runscript
            runscript.help
            labels.json
            environment.sh
```

Interaction

```
./container run bar [args]
```

```
/scif
   /apps
      /foo
         /bin
         /lib
         /scif
            runscript
            runscript.help
            labels.json
            environment.sh
```

Interaction

```
./container shell bar $SCIF_SHELL
./container pyshell bar
```

```
/scif
   /apps
      /foo
         /bin
         /lib
         /scif
            runscript
            runscript.help
            labels.json
            environment.sh
```

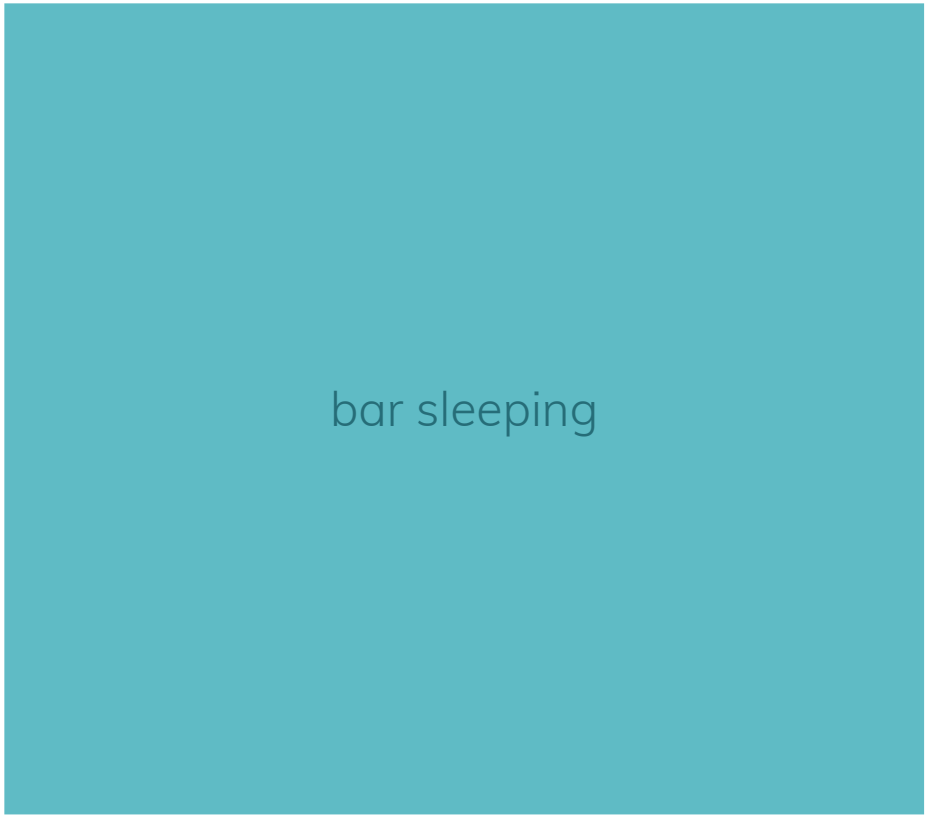
Environment Namespace

Environment Namespace

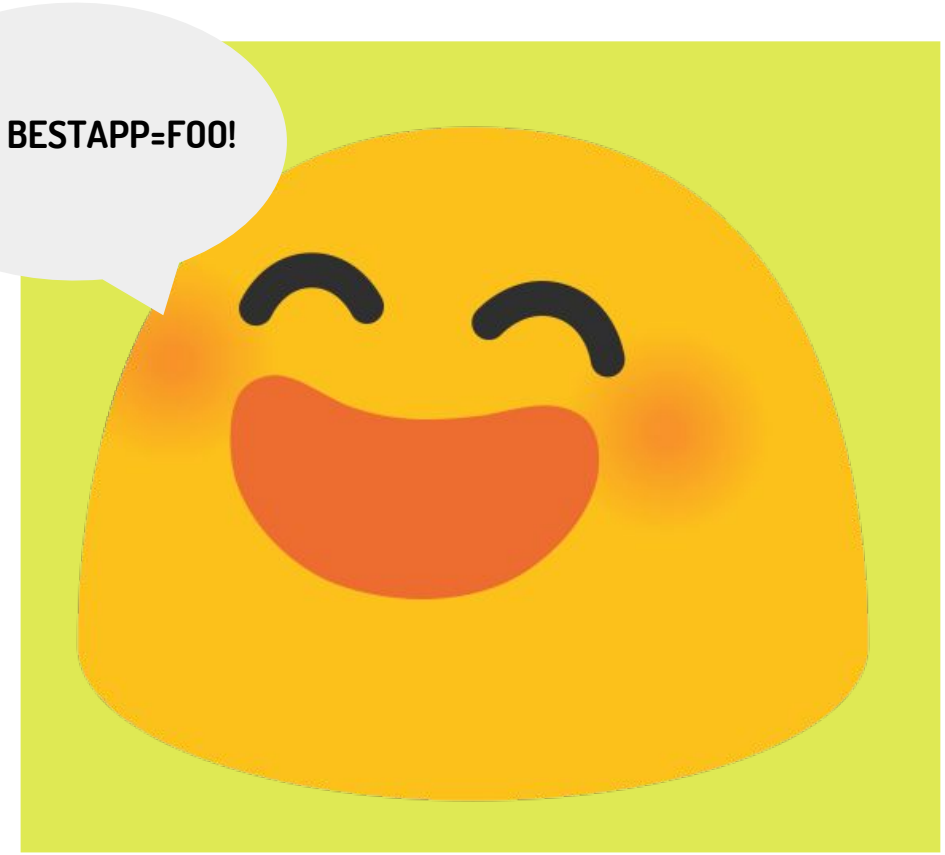
Variable	Default	Definition
SCIF_BASE	/scif	the root location for SCIF
SCIF_DATA	/scif/data	the root location for apps data
SCIF_APPS	/scif/apps	the root location for installed apps
SCIF_SHELL	/bin/bash	shell to use for “shell” command
SCIF_PYSHELL	ipython	interactive python shell for pyshell command
SCIF_ENTRYPOINT	/bin/bash	the command to run given no runscript or app defined
SCIF_ENTRYFOLDER	SCIF_BASE	the entry folder to run the entrypoint command
SCIF_MESSAGELEVEL	INFO	a client level of verbosity. Must be one of `CRITICAL`, `ABORT`, `ERROR`, `WARNING`, `LOG`, `INFO`, `QUIET`, `VERBOSE`, `DEBUG`

Table 1 During interaction and runtime of SCIF, the following environment variables must be defined.

```
./container run foo
```

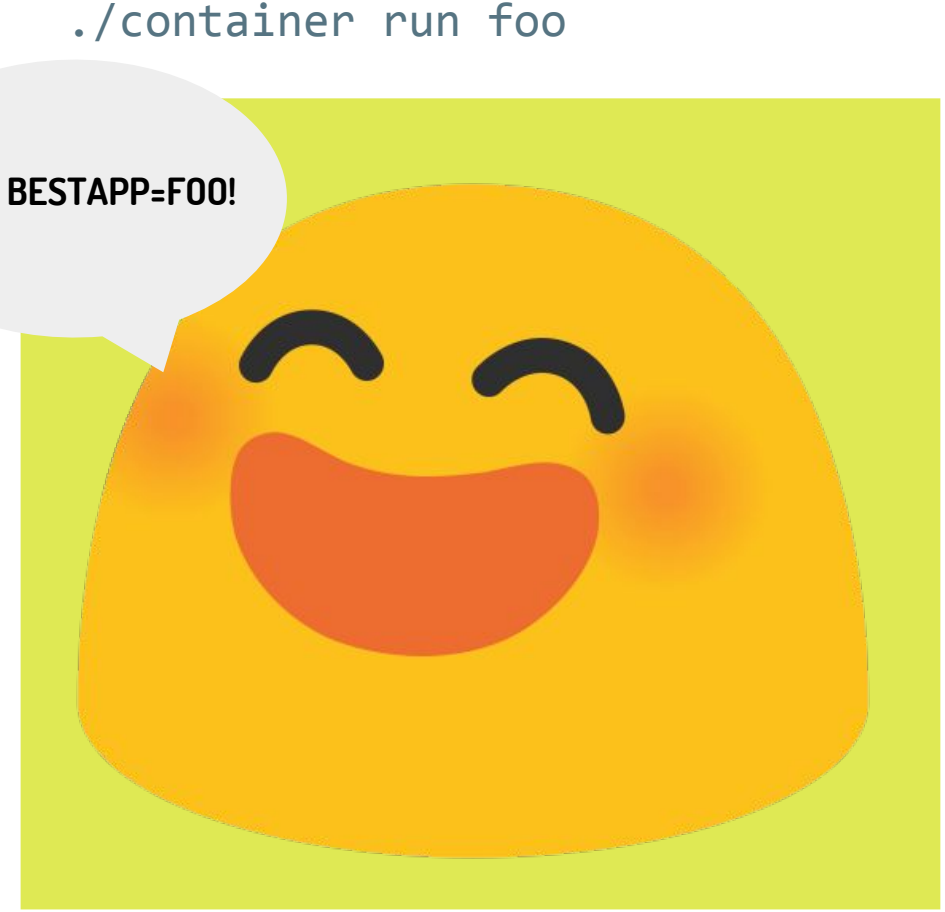


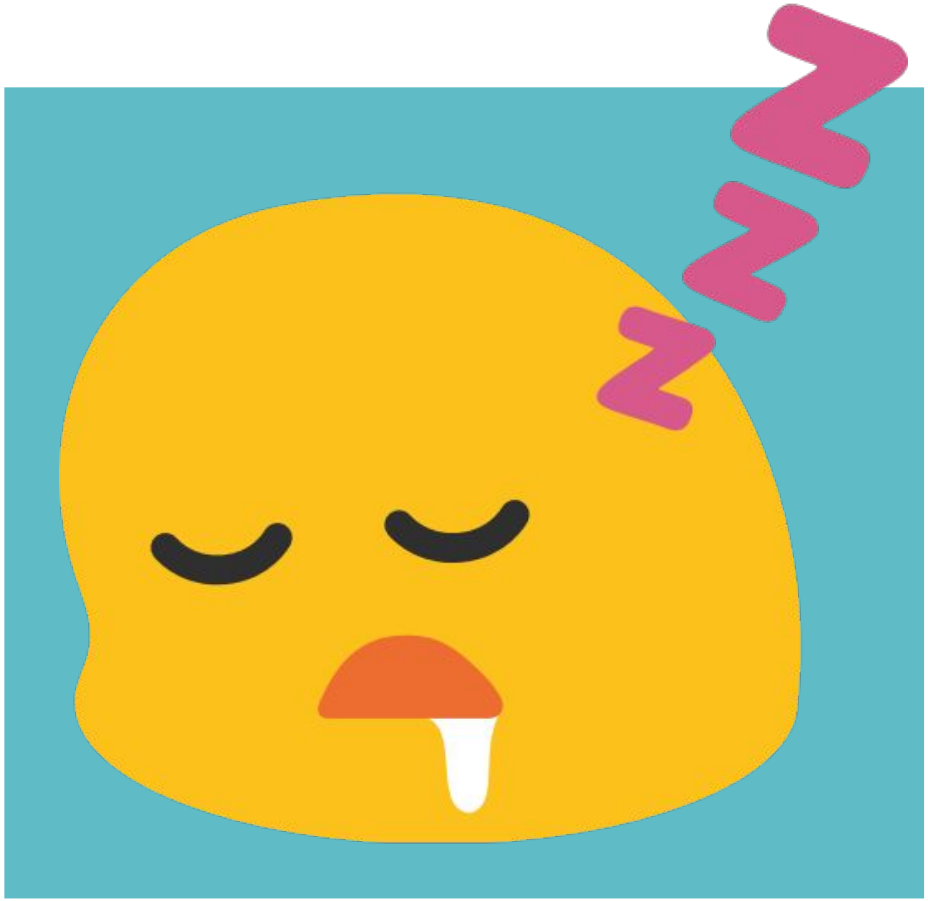
```
./container run foo
```



bar sleeping

./container run foo





Active App Environment Namespace

Variable	Default	Definition
SCIF_APPNAME	example	the active software app
SCIF_APPDATA	/scif/data/example	the data root for the active software app
SCIF_APPROOT	/scif/apps/example	the install root for the active software app
SCIF_APPBIN	/scif/apps/example/bin	the app bin, which is automatically added to the path when active
SCIF_APPLIB	/scif/apps/example/lib	the app bin, which is automatically added to the path when active
SCIF_APPMETA	/scif/apps/example/scif	the metadata folder
SCIF_APPHELP	/scif/apps/example/scif/runscript.help	a text file with help to print for the user to the terminal
SCIF_APPRUN	/scif/apps/example/scif/runscript	the commands to run as the app entrypoint
SCIF_APPTEST	/scif/apps/example/scif/test	the commands to run to test the app
SCIF_APPLABELS	/scif/apps/example/scif/labels.json	a key:value json lookup dictionary of labels
SCIF_APPENV	/scif/apps/example/scif/environment.sh	a shell script to source for the software app environment

Sleeping App Environment Namespace

Variable	Default	Definition
SCIF_APPNAME_sleeper	sleeper	the inactive software app
SCIF_APPDATA_sleeper	/scif/data/sleeper	the data root for the inactive software app
SCIF_APPROOT_sleeper	/scif/apps/sleeper	the install root for the active software app
SCIF_APPBIN_sleeper	/scif/apps/sleeper/bin	the app bin, which is automatically added to the path when active
SCIF_APPLIB_sleeper	/scif/apps/sleeper/lib	the app bin, which is automatically added to the path when active
SCIF_APPMETA_sleeper	/scif/apps/sleeper/scif	the metadata folder
SCIF_APPHELP_sleeper	/scif/apps/sleeper/scif/runscript.help	a text file with help to print for the user to the terminal
SCIF_APPRUN_sleeper	/scif/apps/sleeper/scif/runscript	the commands to run as the app entrypoint
SCIF_APPTEST_sleeper	/scif/apps/sleeper/scif/test	the commands to run to test the app
SCIF_APPLABELS_sleeper	/scif/apps/sleeper/scif/labels.json	a key:value json lookup dictionary of labels
SCIF_APPENV_sleeper	/scif/apps/sleeper/scif/environment.sh	a shell script to source for the software app environment

What can I do with a SCIF?

- expose *multiple* container entrypoints
- package metrics/evaluation alongside the analysis
- SCIF apps as different working environments, runtime
- SCIF apps for different runtime contexts

PAPER

The Scientific Filesystem (SCIF)

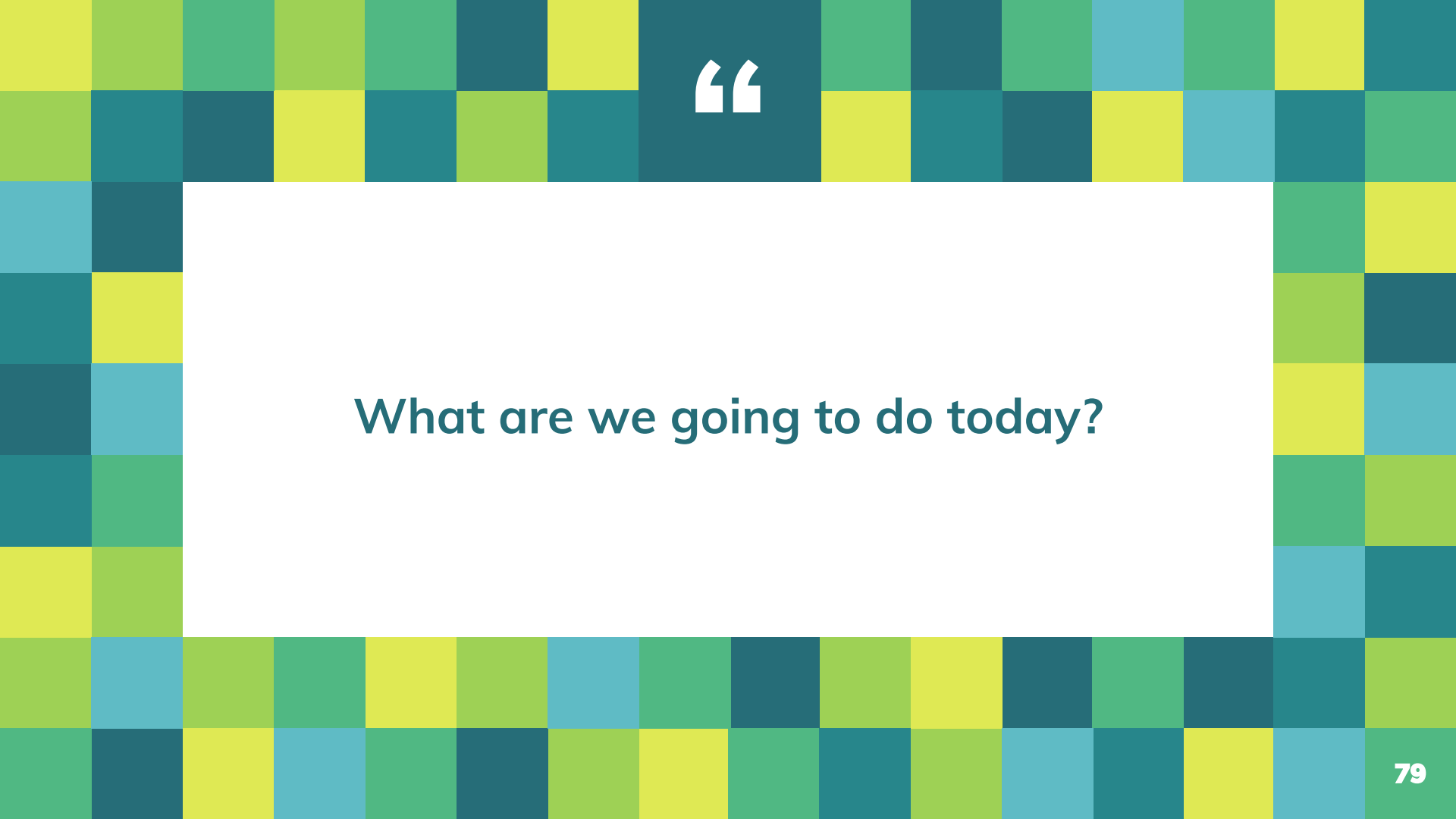
Vanessa Sochat[1,*,†]

[1]Stanford Research Computing Center and [2]Stanford University School of Medicine

[*]vsochat@stanford.edu

Abstract

Background, Here we present the Scientific Filesystem (SCIF), an organizational format that supports exposure of executables and metadata for discoverability of scientific applications. The format includes a known filesystem structure, a definition for a set of environment variables describing it, and functions for generation of the variables and interaction with the libraries, metadata, and executables located within. SCIF makes it easy to expose metadata, multiple environments, installation steps, files, and entrypoints to render scientific applications consistent, modular, and discoverable. A SCIF can be installed on a traditional host or in a container technology such as Docker or Singularity. We will start by reviewing the background and rationale for the Scientific Filesystem, followed by an overview of the specification, and the different levels of internal modules ("apps") that the organizational format affords. Finally, we


What are we going to do today?

What are we going to do today?

Build Containers

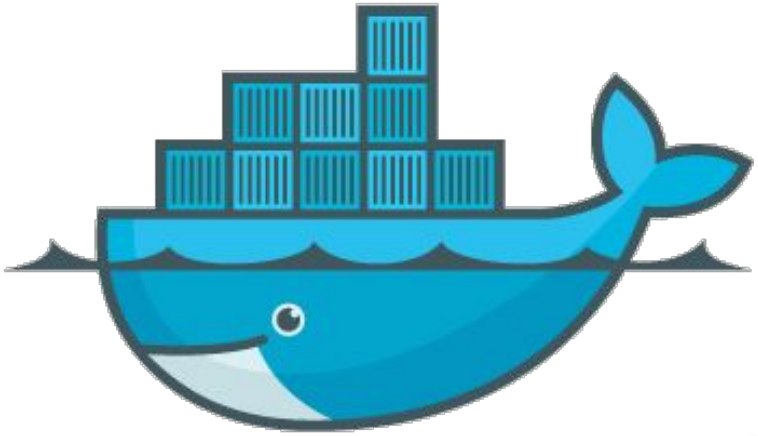
Across different container technologies, all with the same scientific filesystem (SCIF)

Run a Workflow

The same snakemake workflow made modular by SCIF applications

Evaluate

How do the different container technologies compare?

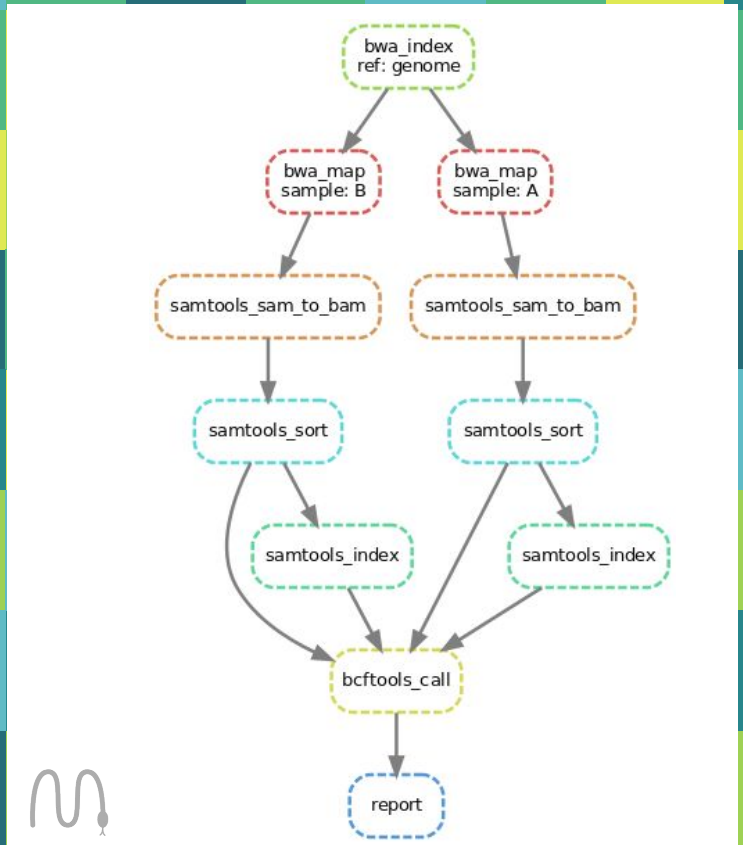


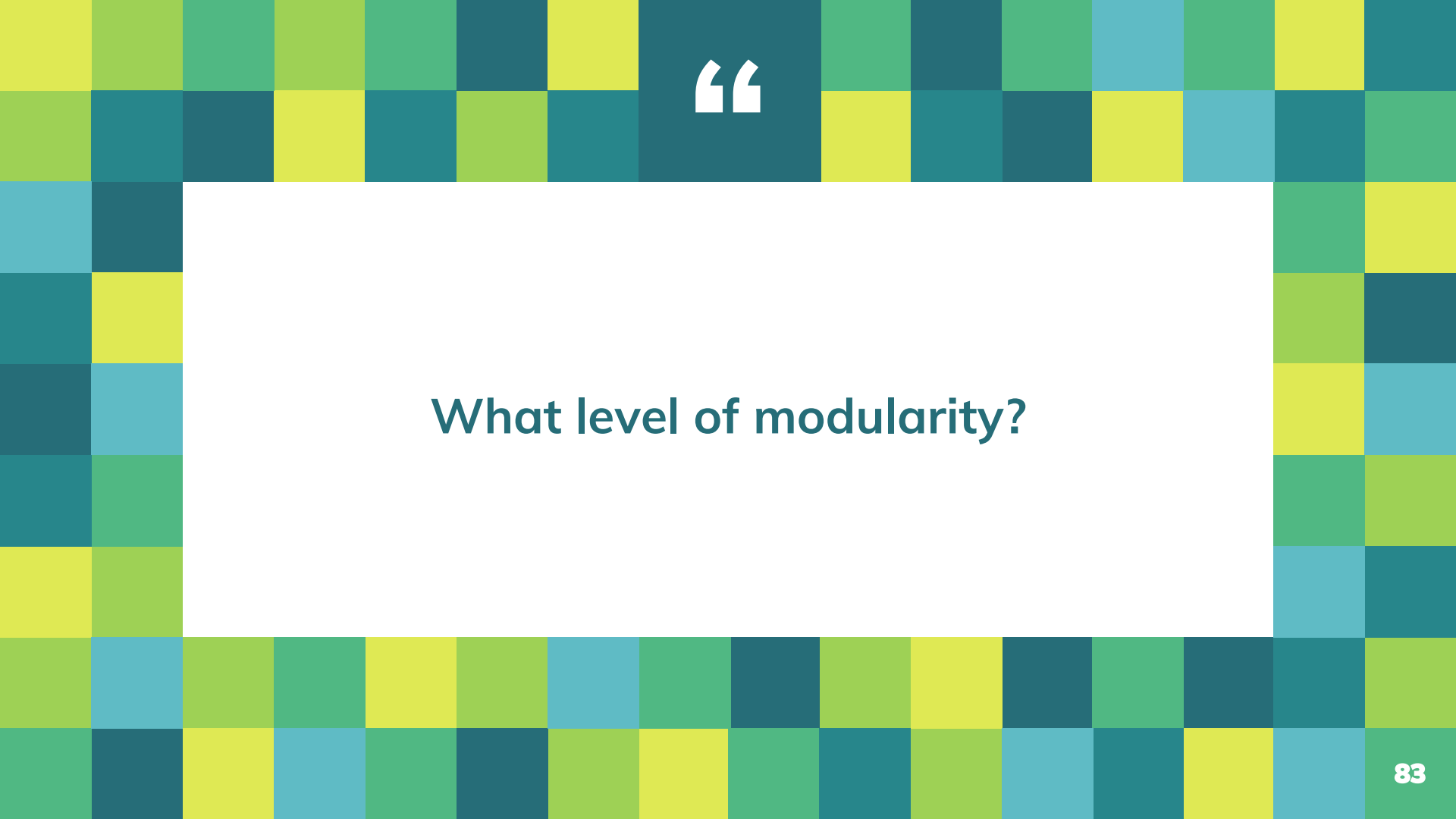

SHIFTER


Charliecloud

Scientific Pipeline

- *Felix Bartusch*
- variant calling
- Snakemake




What level of modularity?

- Level 1: Container Modularity

  - One entrypoint
  - Run the entire thing, or don’t

```
./container
```

- Level 2: Pipeline Modularity

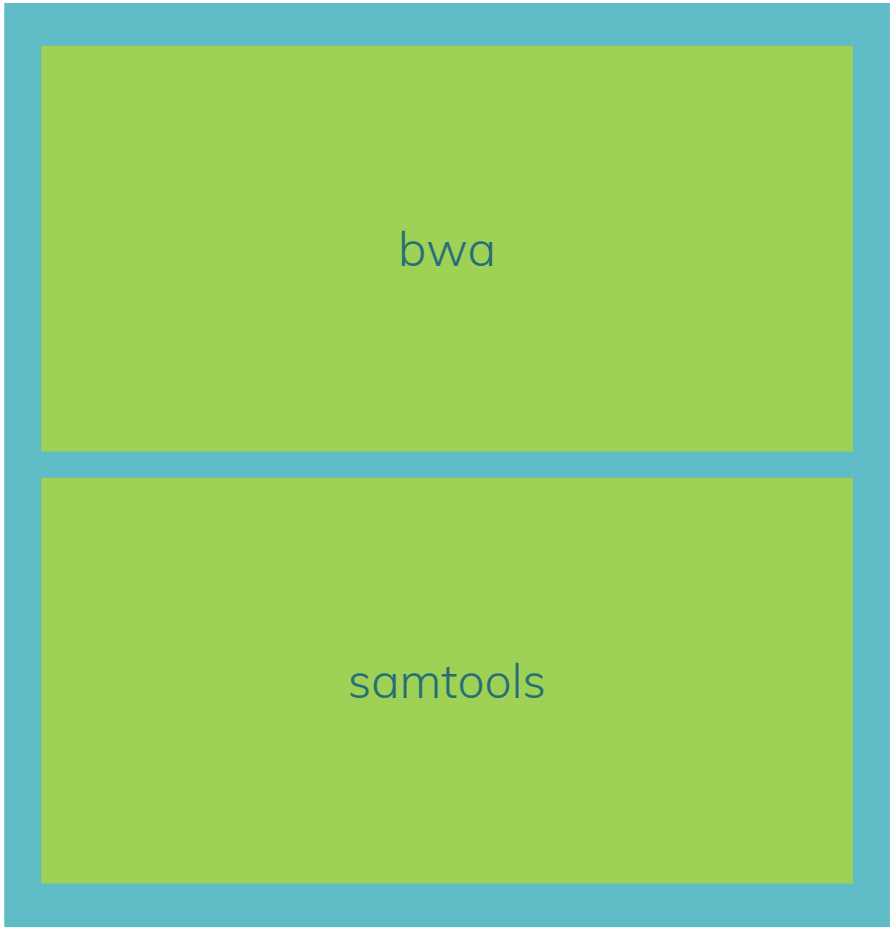
  - Multiple entrypoints
  - Correspond to pipeline steps
  - Step → collection of commands

```
./container apps
  mapping
  sorting

./container run mapping
```

- Level 3: Developer Modularity

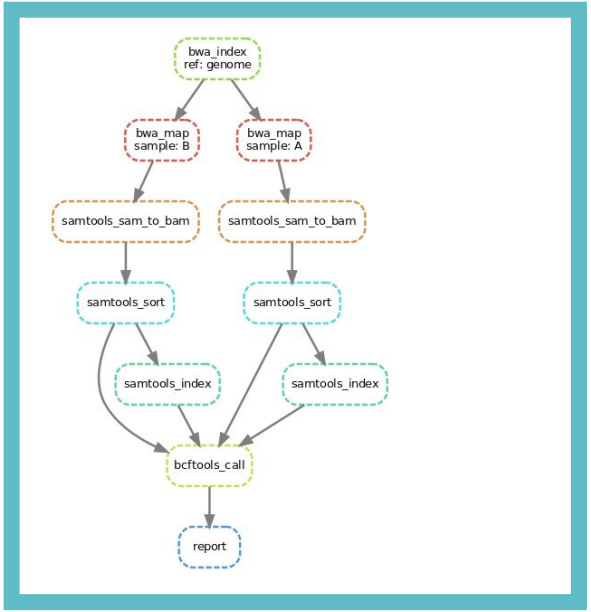
  - Expose lowest level executables
  - Ideal for building pipelines
  - Step → core tools

```
./container apps
  bwa
  samtools

./container run samtools
```



A Scientific Filesystem Design for Variant Calling

What level of modularity should be exposed to interact with the container?



bwa

samtools

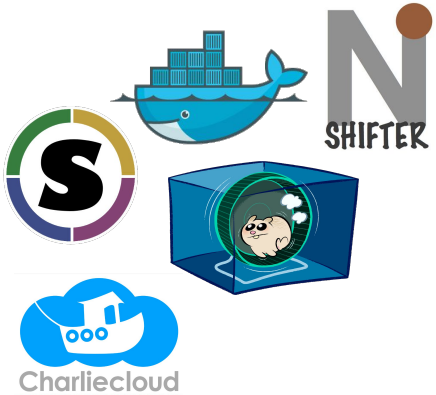
graphviz_create_dag

snakemake

```
./container apps
    bwa
    samtools
    graphviz_create_dag
    snakemake
```

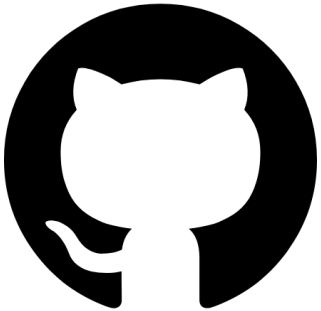
What did I do with the container?

Build Containers



Run a Workflow

Evaluate



https://github.com/sci-f/snakemake.scif

Container Comparisons

What steps did we take across container technologies?

0.	install the software
1.	build the container
2.	run components of the workflow
3.	run the entire workflow

What did we learn?

What did I learn about user interaction?

Good software without documentation and tutorial for several use cases is akin to a treasure chest on a remote planet.

Container Comparisons

What steps did we take across container technologies?

0.	install the software

Installation

- Docker
- Singularity
- Charliecloud
- runc
- Shifter

Container Comparisons

What steps did we take across container technologies?

0. install the software

1. *Documentation is the first interaction*
    a. *Finding it*
    b. *Is it up to date?*
    c. *Is there too much detail?*
    d. *Is there a clear avenue to get more help?*

Container Comparisons

What steps did we take across container technologies?

0.	install the software
1.	build the container

Container Comparisons

What steps did we take across container technologies?

0.	install the software
1.	build the container

1. *Examples are needed for learning*
    a. *Does it show me all the things I might need?*
        i. *binds*
        ii. *options*
        iii. *permissions*

Container Comparisons

What steps did we take across container technologies?

0.	install the software
1.	build the container
2.	run components of the workflow

Container Comparisons

What steps did we take across container technologies?

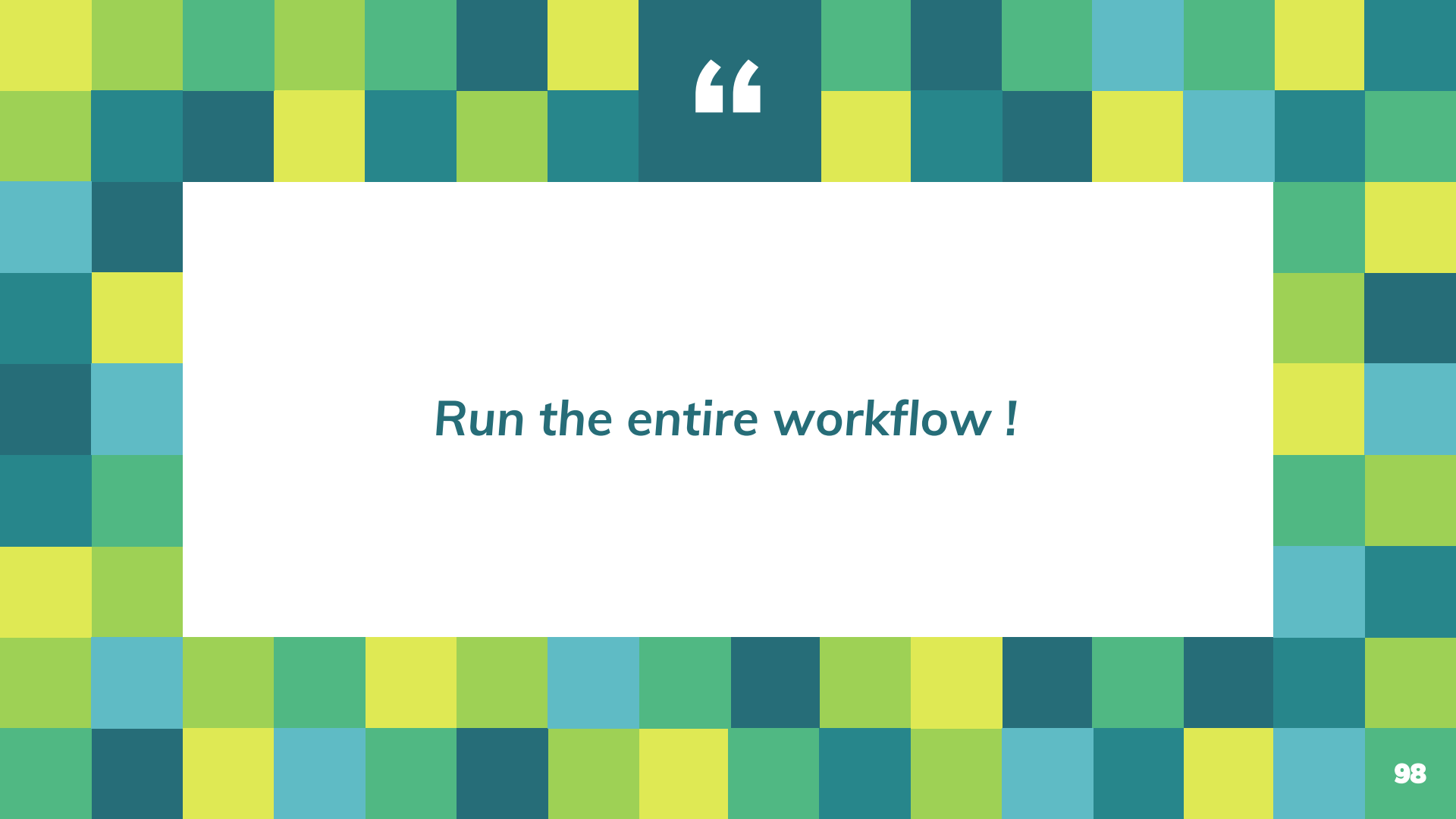
0.	install the software
1.	build the container
2.	run components of the workflow

1. *Establish boundaries*
    a. *show inside vs. outside*

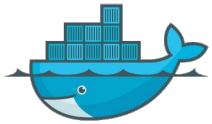
Container Comparisons

What steps did we take across container technologies?

0.	install the software
1.	build the container
2.	run components of the workflow
3.	run the entire workflow


“
Run the entire workflow !

runc --root /tmp/runc run --bundle /tmp/runc snakemake.runc



docker run -v $PWD/data:/scif/data:z -v $PWD:/scif/data/snakemake:z -it vanessa/snakemake.scif run snakemake all

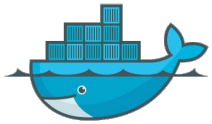
singularity run --bind data/:/scif/data snakemake.simg run snakemake all

ch-run --cd $PWD -b data:/scif/data /var/tmp/snakemake.ch -- /opt/conda/bin/scif run snakemake all

shifter --image=vanessa/snakemake.scif --volume `pwd`/data:/scif/data --workdir /scif/data /opt/conda/bin/scif run snakemake all

command

runc --root /tmp/runc run --bundle /tmp/runc snakmake.runc



docker run -v $PWD/data:/scif/data:z -v $PWD:/scif/data/snakemake:z -it vanessa/snakemake.scif **run snakemake all**

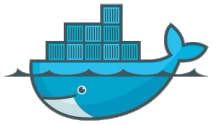
singularity run --bind data/:/scif/data snakemake.simg **run snakemake all**

ch-run --cd $PWD -b data:/scif/data /var/tmp/snakemake.ch -- /opt/conda/bin/scif **run snakemake all**

shifter --image=vanessa/snakemake.scif --volume `pwd`/data:/scif/data --workdir /scif/data /opt/conda/bin/scif **run snakemake all**

executable

runc --root /tmp/runc run --bundle /tmp/runc snakemake.runc



docker run -v $PWD/data:/scif/data -v $PWD:/scif/data/snakemake -it vanessa/snakemake.scif run snakemake all

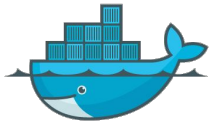
singularity run --bind data/:/scif/data snakemake.simg run snakemake all

ch-run --cd $PWD -b data:/scif/data /var/tmp/snakemake.ch -- /opt/conda/bin/scif run snakemake all

shifter --image=vanessa/snakemake.scif --volume `pwd`/data:/scif/data --workdir /scif/data /opt/conda/bin/scif run snakemake all

executable + action

runc --root /tmp/runc run --bundle /tmp/runc snakmake.runc



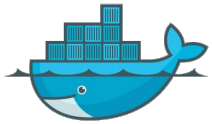
docker run -v $PWD/data:/scif/data -v $PWD:/scif/data/snakemake -it vanessa/snakemake.scif run snakemake all

singularity run --bind data/:/scif/data snakemake.simg run snakemake all

ch-run --cd $PWD -b data:/scif/data /var/tmp/snakemake.ch -- /opt/conda/bin/scif run snakemake all

shifter --image=vanessa/snakemake.scif --volume `pwd`/data:/scif/data --workdir /scif/data /opt/conda/bin/scif run snakemake all

runc --root /tmp/runc run **--bundle /tmp/runc** snakemake.runc



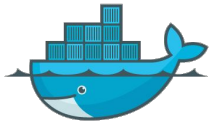
docker run -v $PWD/data:/scif/data -v $PWD:/scif/data/snakemake -it **vanessa/snakemake.scif** run snakemake all

singularity run --bind data/:/scif/data **snakemake.simg** run snakemake all

ch-run --cd $PWD -b data:/scif/data **/var/tmp/snakemake.ch** -- /opt/conda/bin/scif run snakemake all

shifter **--image=vanessa/snakemake.scif** --volume `pwd`/data:/scif/data --workdir /scif/data /opt/conda/bin/scif run snakemake all

runc --root /tmp/runc run --bundle /tmp/runc snakmake.runc

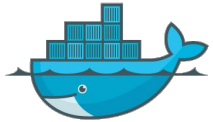


docker run **-v $PWD/data:/scif/data** **-v $PWD:/scif/data/snakemake** -it vanessa/snakemake.scif run snakemake all

singularity run **--bind data/:/scif/data** snakemake.simg run snakemake all

ch-run --cd $PWD **-b data:/scif/data** /var/tmp/snakemake.ch -- /opt/conda/bin/scif run snakemake all

shifter --image=vanessa/snakemake.scif **--volume `pwd`/data:/scif/data** --workdir /scif/data /opt/conda/bin/scif run snakemake all



executable action options container options command

https://github.com/sci-f/snakemake.scif

Are the outputs comparable?

Results are the same

$ diff results/docker/calls.vcf results/singularity/calls.vcf
28c28
< ##bcftools_callCommand=call -mv -; Date=Sat Mar 10 01:07:35 2018

> ##bcftools_callCommand=call -mv -; Date=Fri Mar 9 20:07:47 2018

$ diff results/docker/calls.vcf results/runc/calls.vcf
28c28
< ##bcftools_callCommand=call -mv -; Date=Sat Mar 10 01:07:35 2018

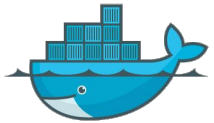
> ##bcftools_callCommand=call -mv -; Date=Sat Mar 10 01:08:05 2018

$ diff results/docker/calls.vcf results/shifter/calls.vcf
28c28
< ##bcftools_callCommand=call -mv -; Date=Sat Mar 10 01:07:35 2018

> ##bcftools_callCommand=call -mv -; Date=Mon Mar 12 20:45:21 2018

$ diff results/docker/calls.vcf results/charliecloud/calls.vcf
28c28
< ##bcftools_callCommand=call -mv -; Date=Sat Mar 10 01:07:35 2018

> ##bcftools_callCommand=call -mv -; Date=Sat Mar 10 01:07:57 2018



vs

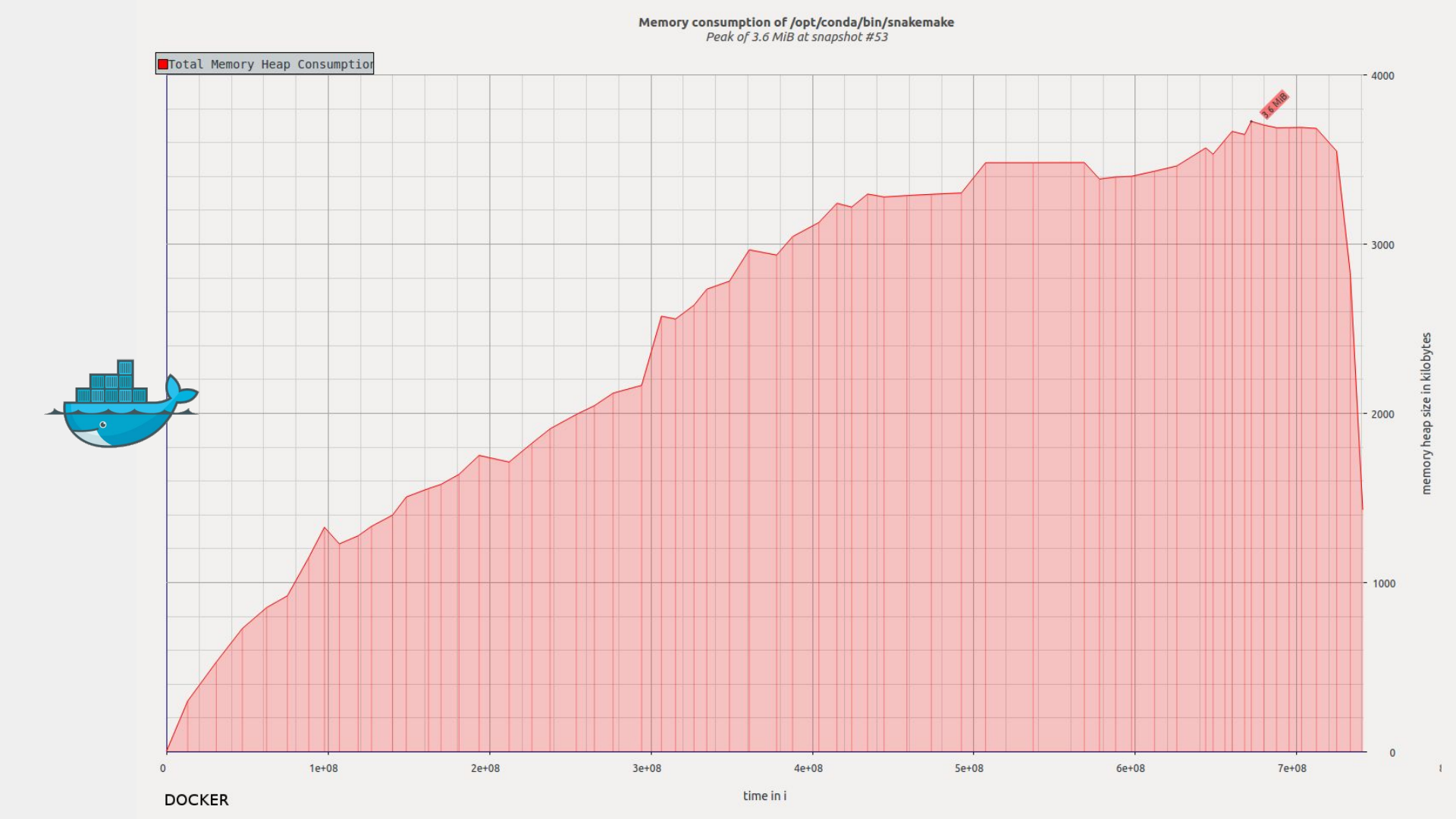
What did we learn?

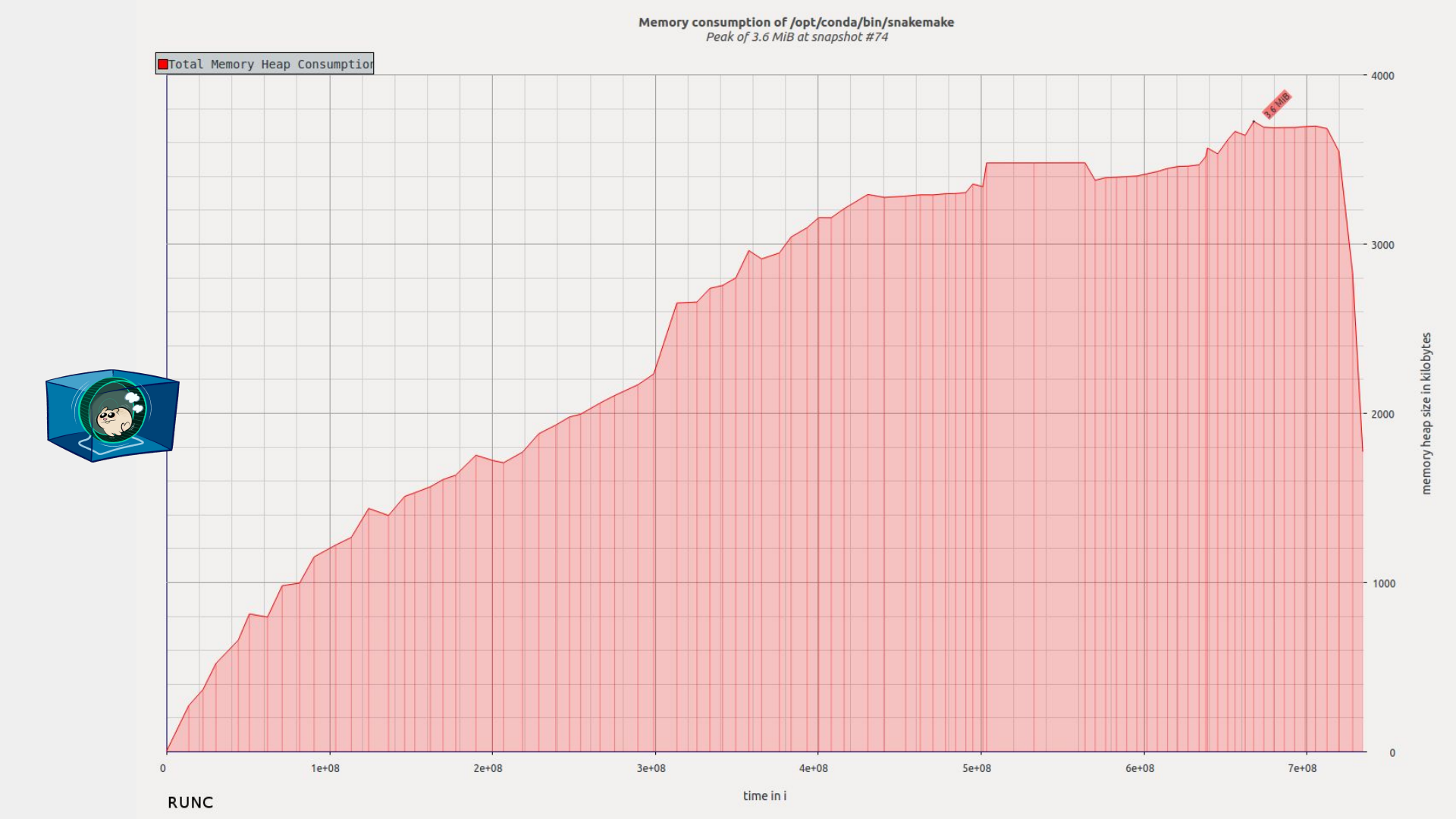
Do the technologies produce the same result?

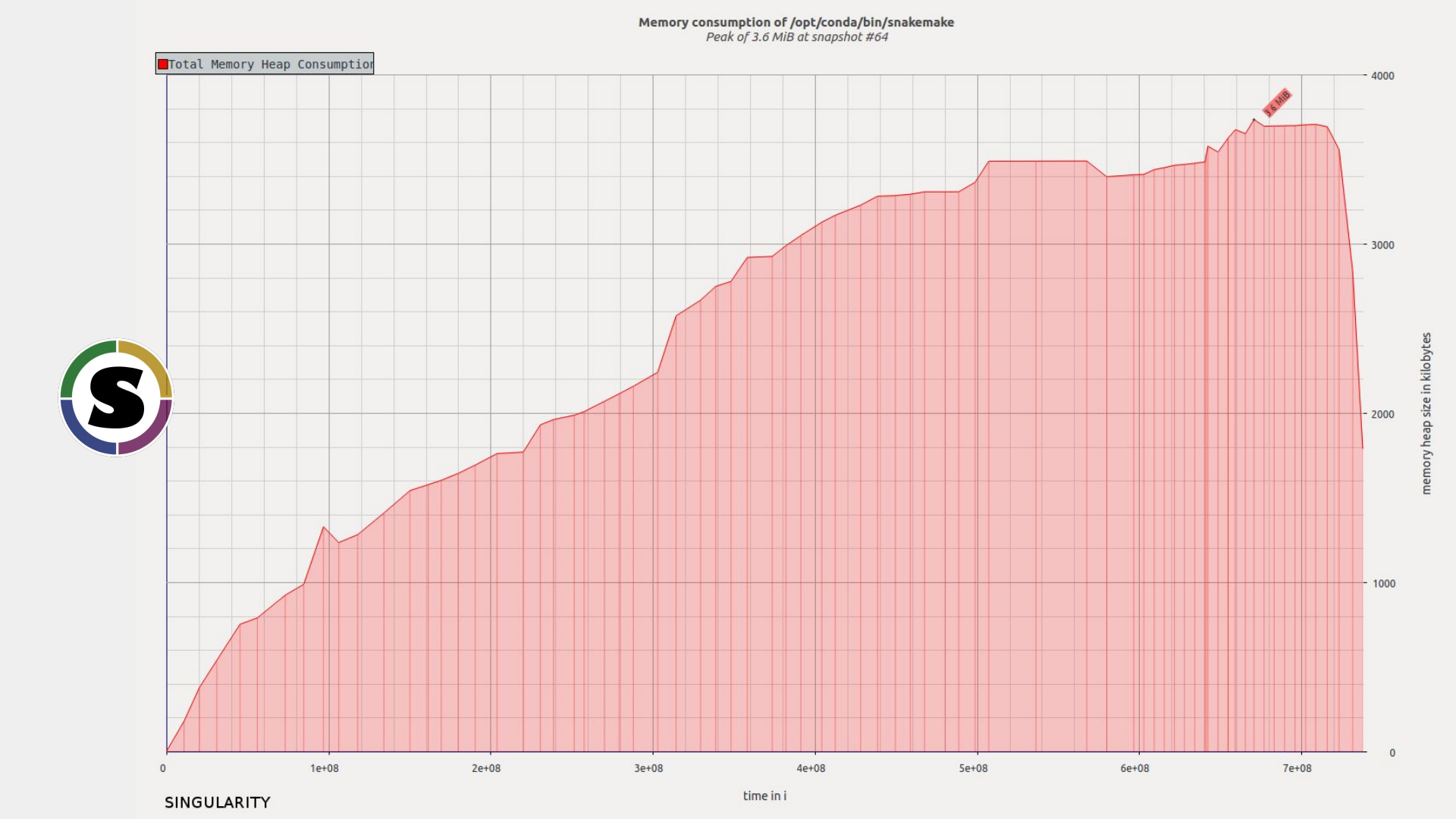
From the perspective of the scientist, we have comparable resource usage and produce the same output.

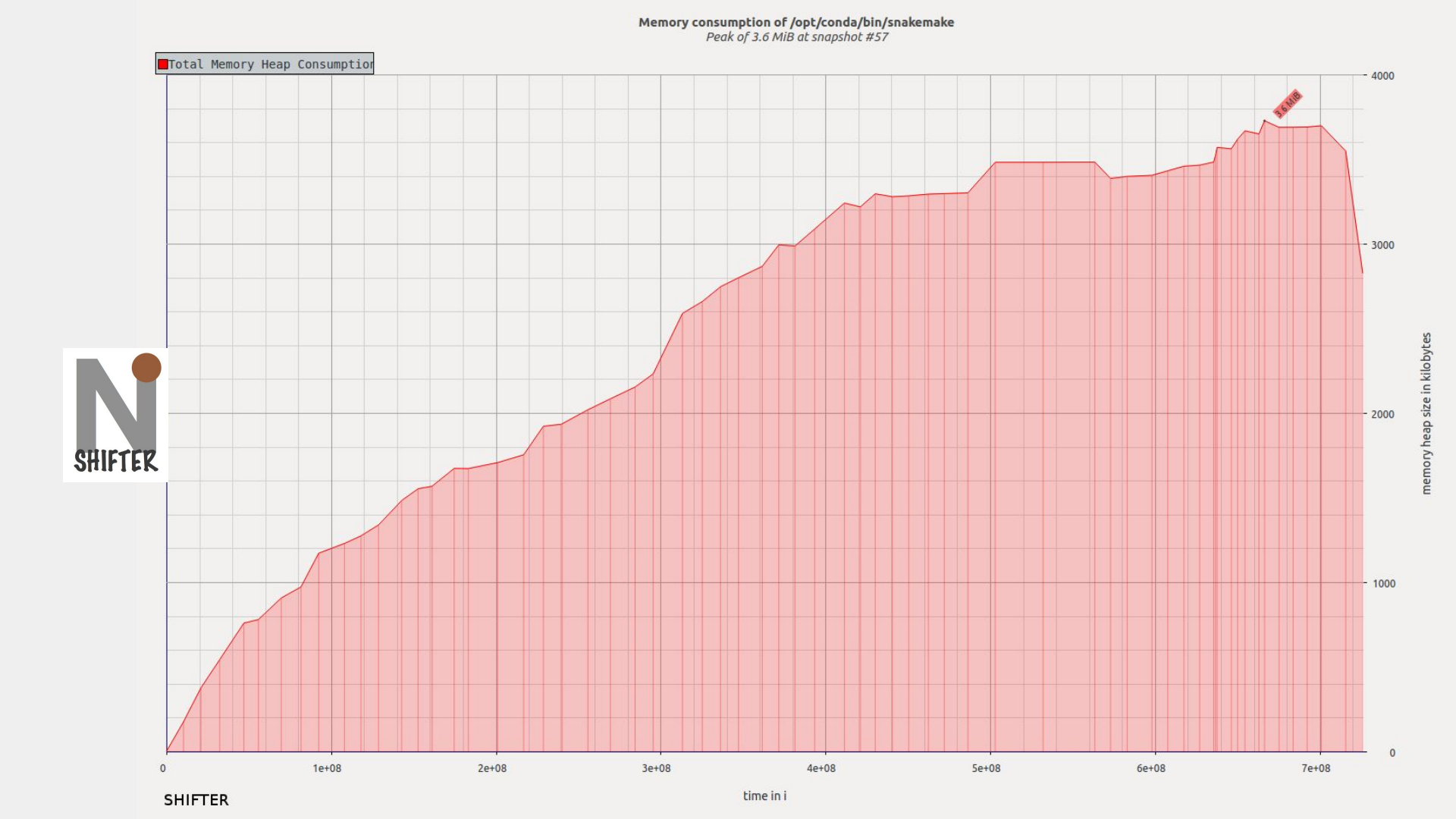

“
What about resource usage?

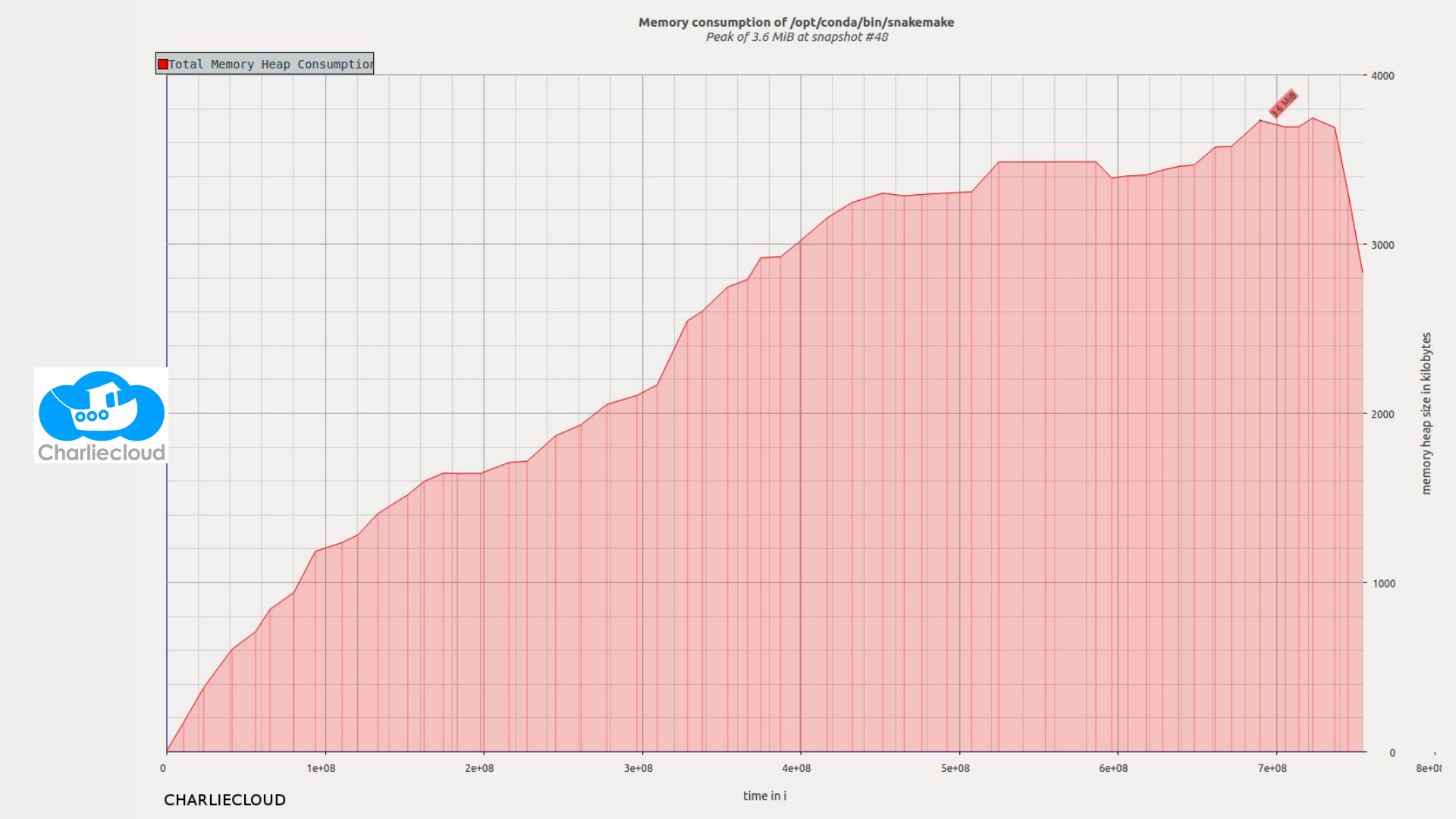
```
singularity run --bind data/:/scif/data snakemake.simg run valgrind
singularity run --bind data/:/scif/data snakemake.simg run timeit
```


Memory consumption of /opt/conda/bin/snakemake
Peak of 3.6 MiB at snapshot #53
Total Memory Heap Consumption
3.6 MiB
4000
3000
2000
1000
0
memory heap size in kilobytes
0
1e+08
2e+08
3e+08
4e+08
5e+08
6e+08
7e+08
time in i
DOCKER

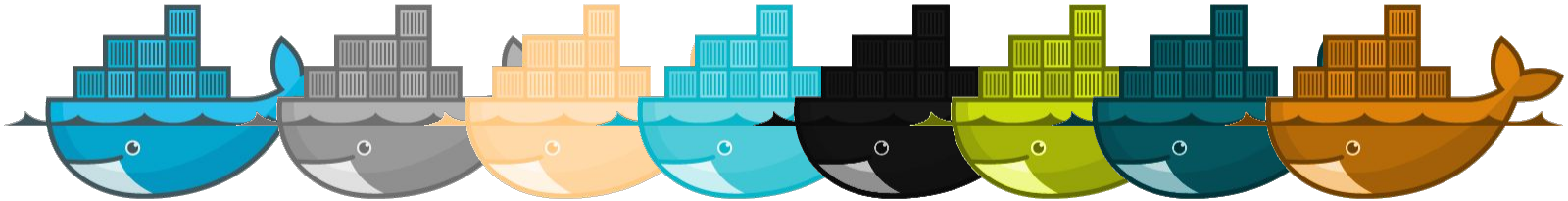

Memory consumption of /opt/conda/bin/snakemake
Peak of 3.6 MiB at snapshot #74
Total Memory Heap Consumption
3.6 MiB
4000
3000
2000
1000
0
memory heap size in kilobytes
0
1e+08
2e+08
3e+08
4e+08
5e+08
6e+08
7e+08
time in i
RUNC


Memory consumption of /opt/conda/bin/snakemake
Peak of 3.6 MiB at snapshot #64
Total Memory Heap Consumption
3.6 MiB
4000
3000
2000
1000
0
memory heap size in kilobytes
0
1e+08
2e+08
3e+08
4e+08
5e+08
6e+08
7e+08
time in i
SINGULARITY


Memory consumption of /opt/conda/bin/snakemake
Peak of 3.6 MiB at snapshot #57
Total Memory Heap Consumption
3.6 MiB
4000
3000
2000
1000
0
memory heap size in kilobytes
0
1e+08
2e+08
3e+08
4e+08
5e+08
6e+08
7e+08
time in i
SHIFTER


Memory consumption of /opt/conda/bin/snakemake
Peak of 3.6 MiB at snapshot #48
Total Memory Heap Consumption
3.6 MiB
4000
3000
2000
1000
0
memory heap size in kilobytes
0
1e+08
2e+08
3e+08
4e+08
5e+08
6e+08
7e+08
8e+08
time in i
Charliecloud
CHARLIECLOUD

...but how do my choices of the content, and host, influence my result?



Does Heap Consumption Vary Across Machine Type?

```
vanessa@vanessa-ThinkPad-T460s:~/Documents/Dropbox/Code/scif/examples/snakemake.scif/results/cloud$ sregistry build --config config-n1-highcpu-2.json https://ww
w.github.com/sci-f/snakemake.scif
[client|google-compute] [database|sqlite:////home/vanessa/.singularity/sregistry.db]
[bucket][sregistry-vanessa]
INSTANCE purple-lentil-2448
sci-f-snakemake.scif-builder https://vsoch.github.io/builders/cloud/google/compute/ubuntu/metrics/securebuild-2.4.3
Robot Logger: http://35.203.190.60
Allow a few minutes for web server install, beepboop!
vanessa@vanessa-ThinkPad-T460s:~/Documents/Dropbox/Code/scif/examples/snakemake.scif/results/cloud$ sregistry build --config config-n1-highcpu-4.json https://ww
w.github.com/sci-f/snakemake.scif
[client|google-compute] [database|sqlite:////home/vanessa/.singularity/sregistry.db]
[bucket][sregistry-vanessa]
INSTANCE placid-signal-8619
sci-f-snakemake.scif-builder https://vsoch.github.io/builders/cloud/google/compute/ubuntu/metrics/securebuild-2.4.3
Robot Logger: http://104.198.10.164
Allow a few minutes for web server install, beepboop!
vanessa@vanessa-ThinkPad-T460s:~/Documents/Dropbox/Code/scif/examples/snakemake.scif/results/cloud$ sregistry build --config config-n1-highcpu-8.json https://ww
w.github.com/sci-f/snakemake.scif
[client|google-compute] [database|sqlite:////home/vanessa/.singularity/sregistry.db]
[bucket][sregistry-vanessa]
INSTANCE gloopy-nalgas-6369
sci-f-snakemake.scif-builder https://vsoch.github.io/builders/cloud/google/compute/ubuntu/metrics/securebuild-2.4.3
Robot Logger: http://35.230.72.13
Allow a few minutes for web server install, beepboop!
vanessa@vanessa-ThinkPad-T460s:~/Documents/Dropbox/Code/scif/examples/snakemake.scif/results/cloud$ sregistry build --config config-n1-highcpu-16.json https://w
ww.github.com/sci-f/snakemake.scif
[client|google-compute] [database|sqlite:////home/vanessa/.singularity/sregistry.db]
[bucket][sregistry-vanessa]
INSTANCE bricky-despacito-9882
sci-f-snakemake.scif-builder https://vsoch.github.io/builders/cloud/google/compute/ubuntu/metrics/securebuild-2.4.3
Robot Logger: http://35.230.90.118
Allow a few minutes for web server install, beepboop!
vanessa@vanessa-ThinkPad-T460s:~/Documents/Dropbox/Code/scif/examples/snakemake.scif/results/cloud$ sregistry build --config config-n1-highcpu-32.json https://w
ww.github.com/sci-f/snakemake.scif
[client|google-compute] [database|sqlite:////home/vanessa/.singularity/sregistry.db]
[bucket][sregistry-vanessa]
INSTANCE expressive-pedo-7623
sci-f-snakemake.scif-builder https://vsoch.github.io/builders/cloud/google/compute/ubuntu/metrics/securebuild-2.4.3
Robot Logger: http://35.230.21.109
Allow a few minutes for web server install, beepboop!
vanessa@vanessa-ThinkPad-T460s:~/Documents/Dropbox/Code/scif/examples/snakemake.scif/results/cloud$ sregistry build --config config-n1-highcpu-64.json https://www.
github.com/sci-f/snakemake.scif
[client|google-compute] [database|sqlite:////home/vanessa/.singularity/sregistry.db]
[bucket][sregistry-vanessa]
INSTANCE butterscotch-house-3317
sci-f-snakemake.scif-builder https://vsoch.github.io/builders/cloud/google/compute/ubuntu/metrics/securebuild-2.4.3
Robot Logger: http://35.197.117.86
Allow a few minutes for web server install, beepboop!
vanessa@vanessa-ThinkPad-T460s:~/Documents/Dropbox/Code/scif/examples/snakemake.scif/results/cloud$
```

The Builders

Welcome to the Builders documentation! [builder-manifest][static-demo].

What are the Builders?

Hi friends! Welcome to the builders documentation! The Builder robots are served faithfully by the Singularity Global Client so you can run your own builds on different clouds of your choosing. This documentation is intended for **developers** of builders, and not using the builders. For using the builders, see the Singularity Global Client. The builder that you use (e.g., Google Compute) will logically map to the client there!

```
> SINGULARITY BUILDER ROBOT LOGGER, REPORTING FOR DUTY!

> INSTALLING SINGULARITY DEPENDENCIES
PREPARING LOGGING...
LOGS AVAILABLE AT HTTP://10.138.0.6/

# SINGULARITY

SINGULARITY_REPO: HTTPS://GITHUB.COM/CCLERGET/SINGULARITY.GIT
    THE SINGULARITY REPOSITORY BEING CLONED BY THE BUILDER.
SINGULARITY_BRANCH: FEATURE-SQUASHBUILD-SECBUILD-2.4.3
    THE BRANCH OF THE REPOSITORY BEING USED.
SINGULARITY_COMMIT:
    IF DEFINED, A PARTICULAR COMMIT TO CHECKOUT.

# SETTINGS

SREGISTRY_USER_REPO: HTTPS://GITHUB.COM/EILON-S/SHERLOCK_VEP
    YOUR REPOSITORY WE ARE BUILDING FROM!
```

Local machine

sregistry build https://www.github.com/vsoch/hello-world

build template

cloud builder

cloud storage

sregistry pull sregistry pull vsoch/hello-world:latest...

n1-standard-1

n1-standard-2

n1-standard-4

n1-standard-8

n1-standard-1

n1-standard-2

n1-standard-4

n1-standard-8

n1-standard-1

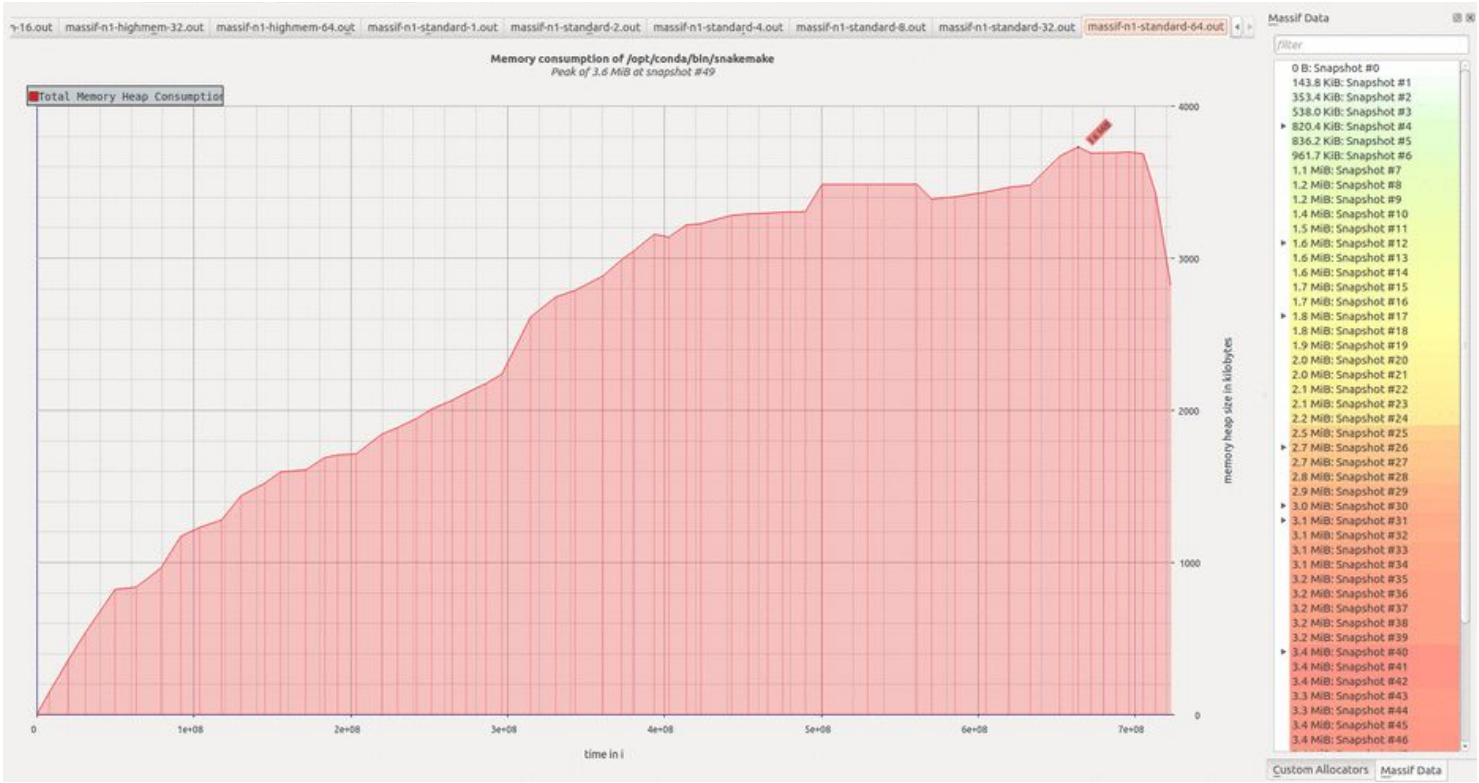
n1-standard-2

n1-standard-4

n1-standard-8

Does Heap Consumption Vary Across Machine Type?

Variant calling memory usage is comparable from 1 to 64 cores



What is the optimal design?

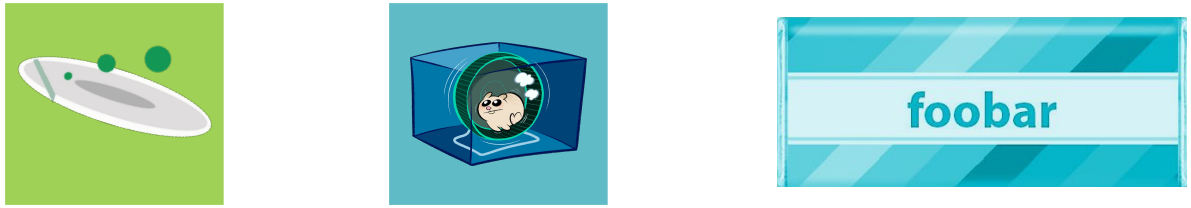
tensorflow 17.0

tensorflow 17.1

theano 1.0.1

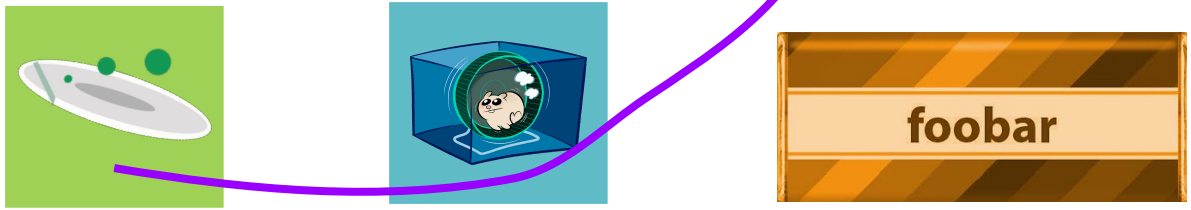
theano 1.0.0

tensorflow 17.0



tensorflow 17.1

theano 1.0.1



theano 1.0.0

“

TLDR


It does the same thing!

Containerization seems to produce a reproducible workflow.

We need to make tools that are usable across technologies.

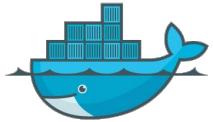
Transparency, ease of use, and ability to contribute are important.

It’s our job to encourage introspection and evaluation of variation.

Once upon a time...

There was a scientist

There was a
reproducibility crisis



...but friends were there to help :)


WHY CAN'T WE BE
FRIENDS?
https://sci-f.github.io

PRESENTER: VANESSASAURUS

THIS IS THE END

vsochat@stanford.edu