PSI

October 19 - 25, 2024
CHEP 2024

Efficient and fast container execution using image snapshotters

Max Fatouros, Derek Feichtinger, **Clemens Lange (PSI)**
Jakob Blomer, Amal Thundiyil, Valentin Völkl (CERN)
CHEP2024, 22nd October 2024

Motivation

In its standard configuration,

docker run <image name> <command>

downloads the entire container image from the registry and unpacks it on disk before executing the actual command in the started container

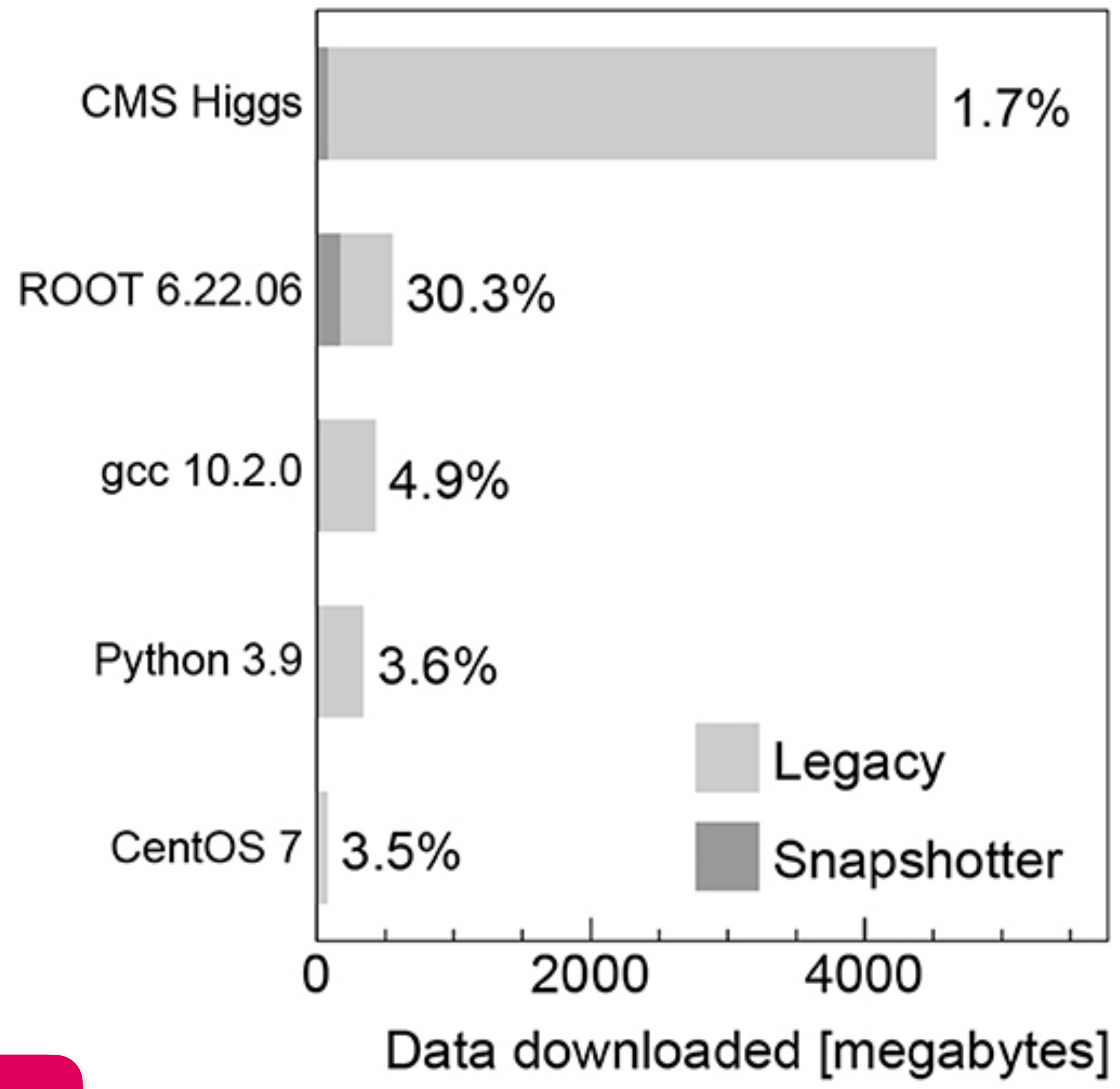
A typical container image used for physics analysis (important e.g. for analysis reusability) has a **size of ~gigabytes**

Executing containerised workloads on batch systems can therefore lead to **hundreds of parallel downloads** of several gigabytes of data

However, only a fraction of the container image is actually needed

→ download only what is actually needed

[Frontiers in Big Data Vol. 4 (2021) 673163]

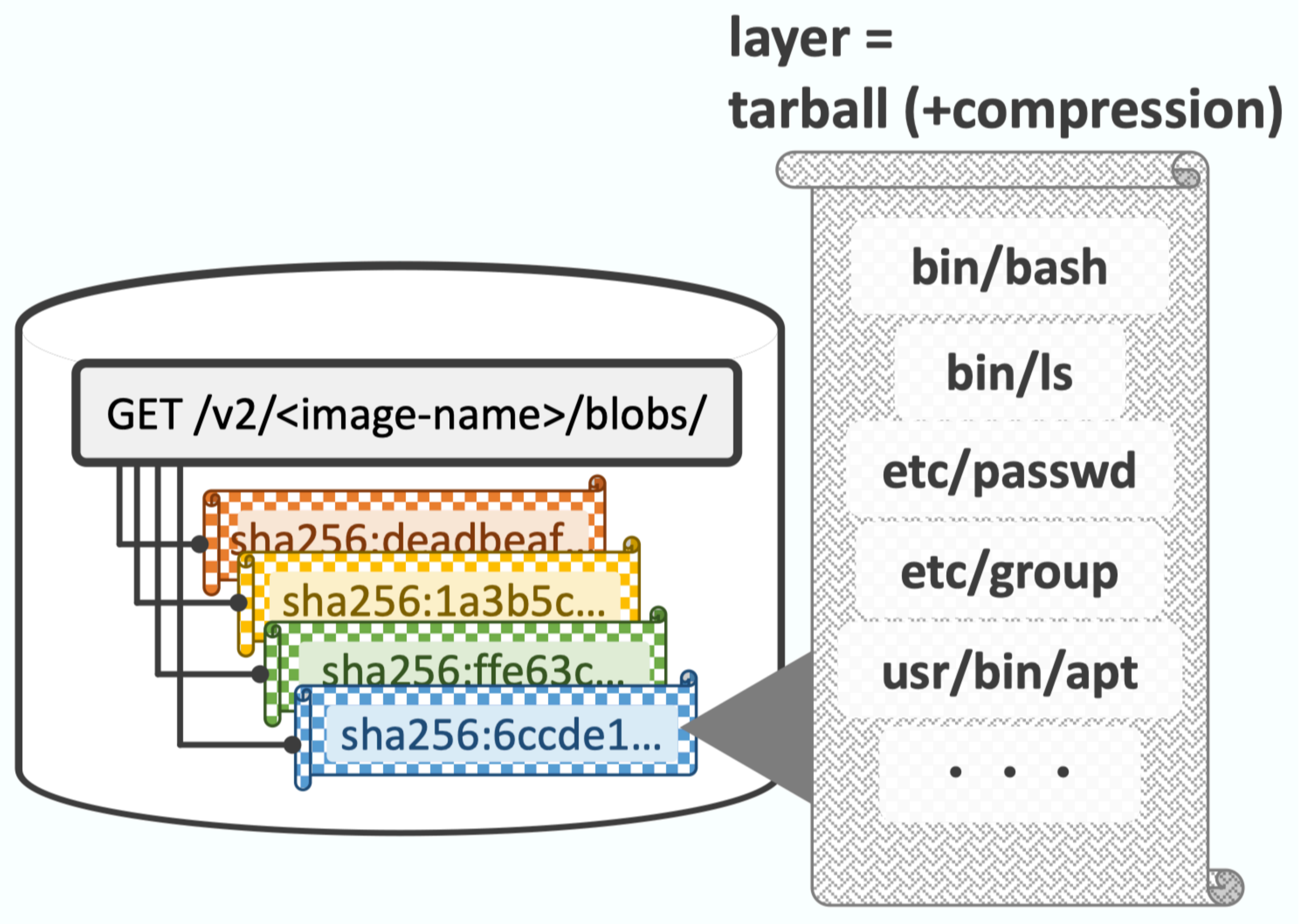


Lazy-pulling of container images

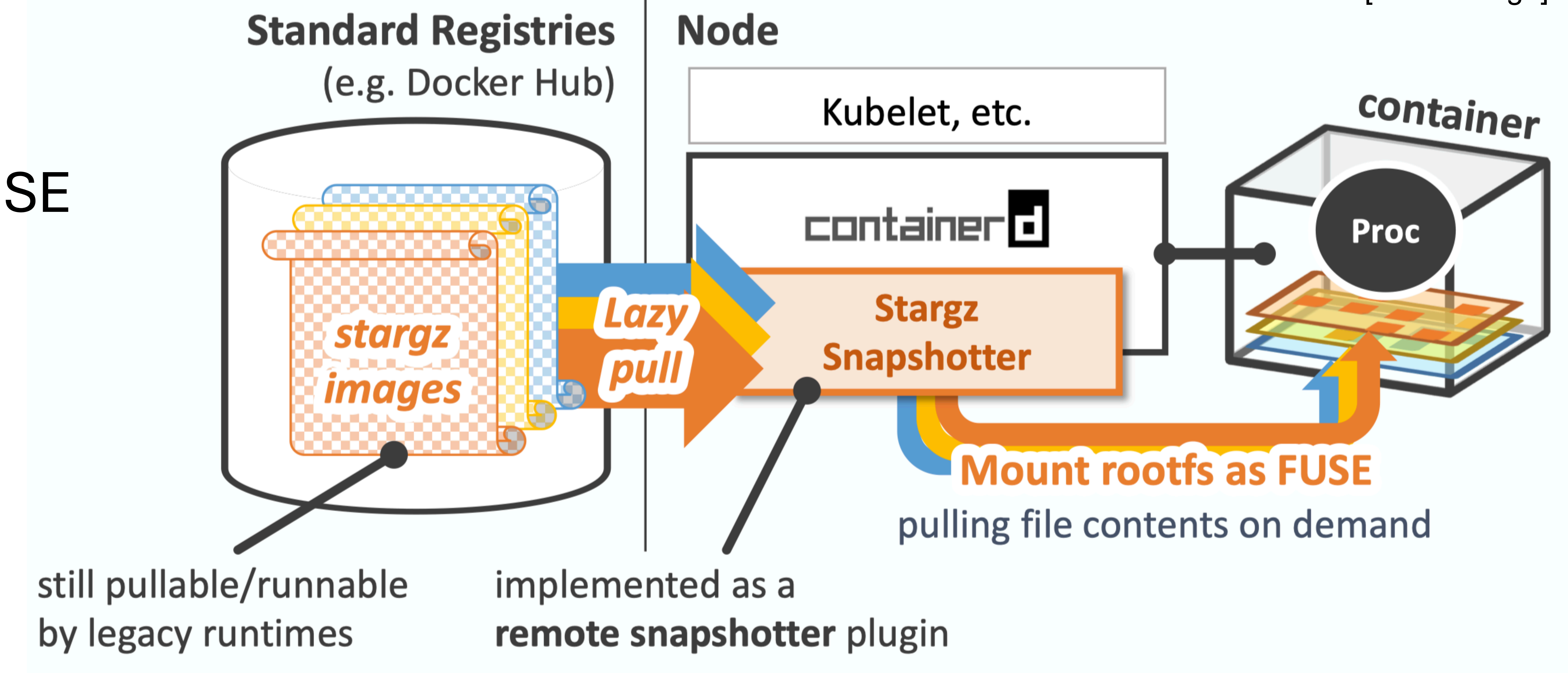
Lazy-pulling = pull/download only what is needed

Container reminder:

- A container is a set of tar-balls plus a manifest (list)
- Downloading and extracting the layers builds the container file system



[K. Tokunaga]



Lazy pulling mounts (rootfs snapshots as FUSE and downloads) accessed file contents on-demand

- Can start container almost immediately
- Can be slower during execution

Implementations of lazy pulling

PSI

Solutions are implemented as so-called image snapshotters for use with containerd

Evaluated tools:

- Stargz snapshotter: use images in searchable tar.gz format
- SOCI snapshotter: add separate index artifact to image (hosted in registry)
- CVMFS snapshotter: use unpacked images on CVMFS
- Overlayfs snapshotter: the default/legacy, non-lazy-loading snapshotter

All snapshotters will fallback to legacy pulling if image (or layer) not available in required format

- Enables use of “protected” layers based on public base images
- Mind: this is something Singularity/Apptainer cannot do

Side note: “lazy pulling” with Apptainer achieved through unpacked images on CVMFS

Benchmarking approach

Use typical particle physics tasks and container images, e.g.:

- ROOT
- Python

Using nerdctl to run workloads with the various snapshotters:

- Parse execution log files to extract timestamps
- Monitor traffic using network monitoring tools
- Repeat process several times, clear cache in between runs

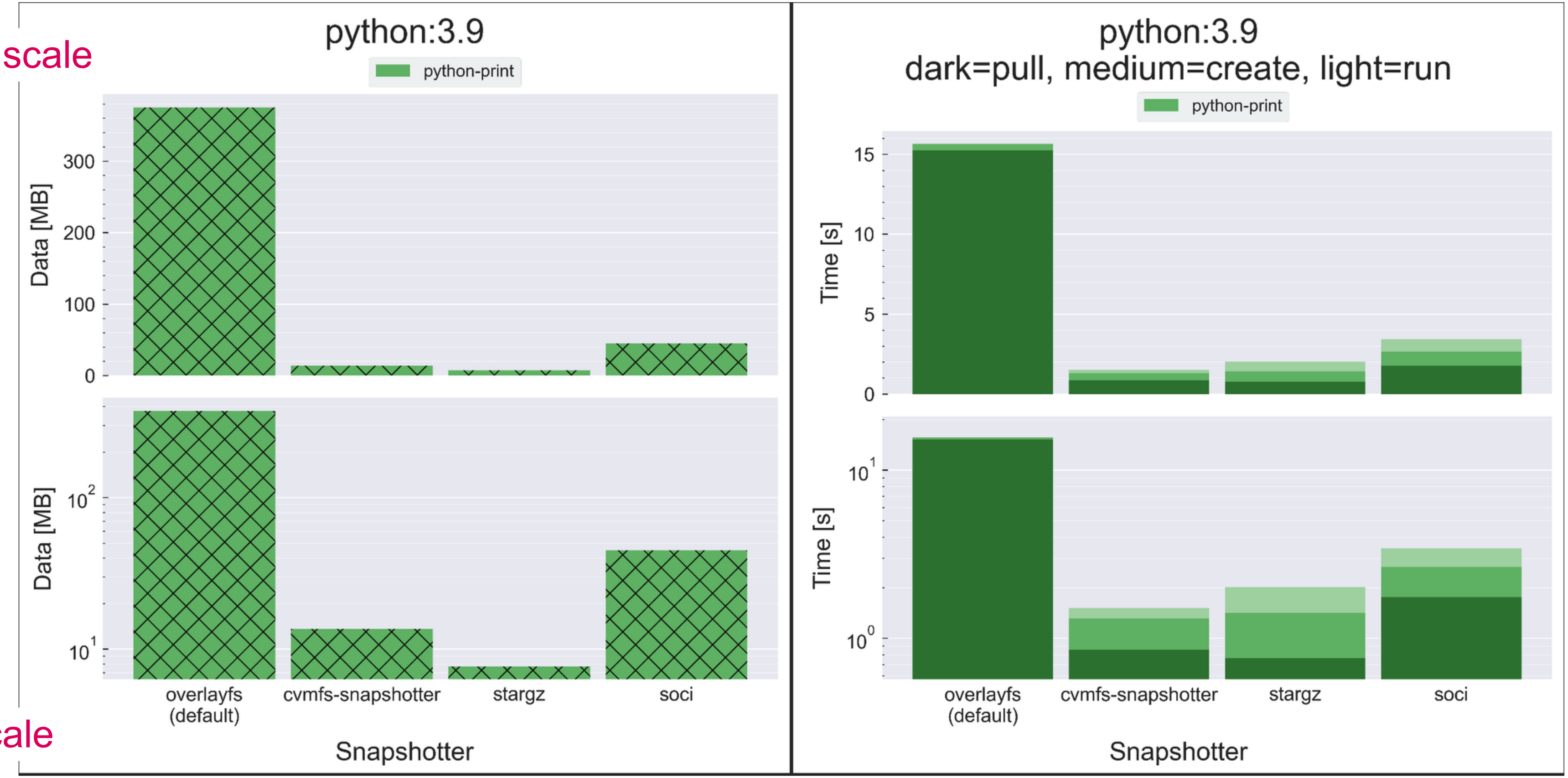
Also compare to “legacy” approach pulling entire image before execution

Using local PC in connection with both a local (same machine) registry, and Harbor registry (in the same network).

Using squid proxy for CVMFS caching (with images pre-cached)

Results: python image: print() — remote registry/cache

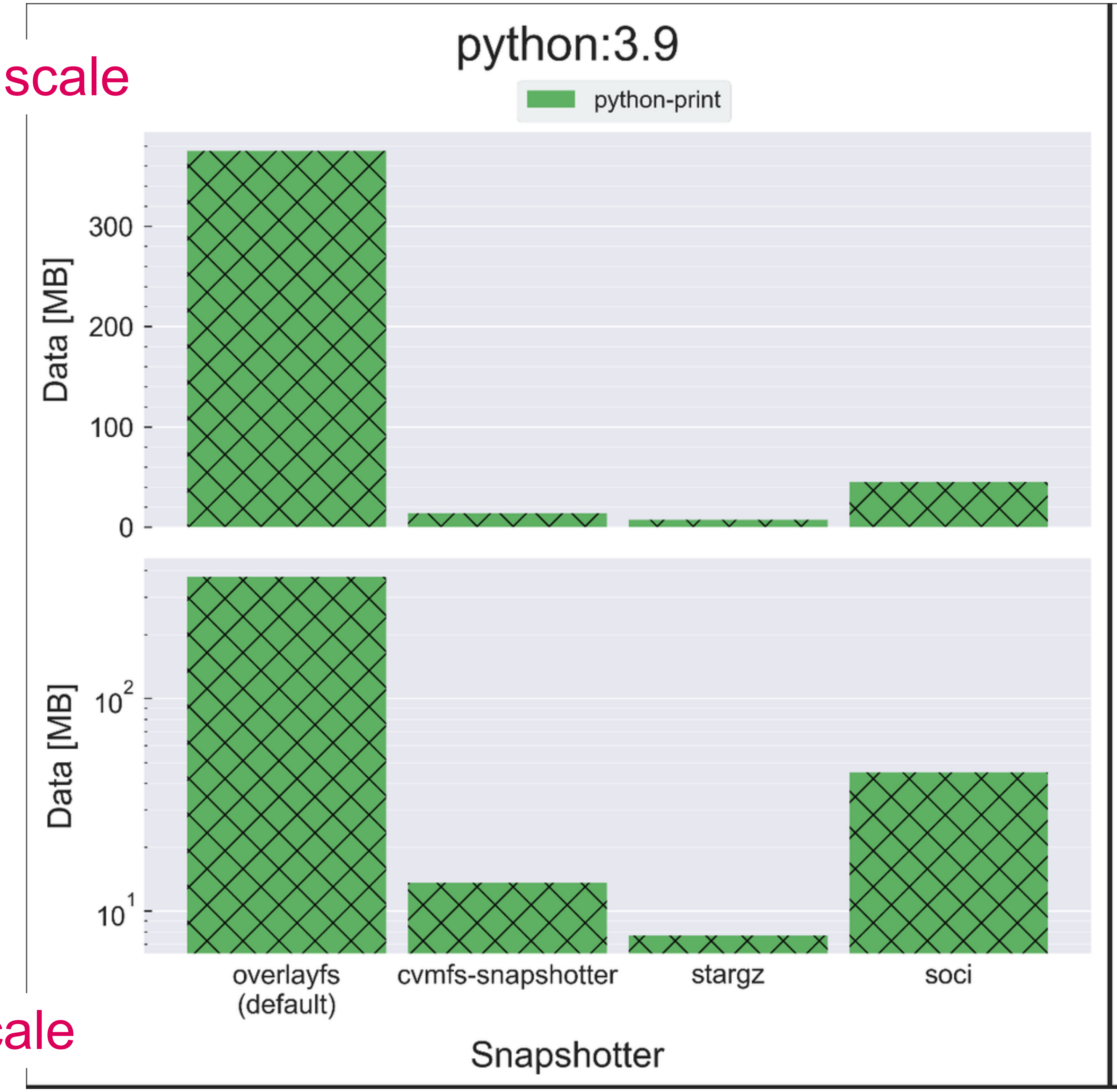
linear scale



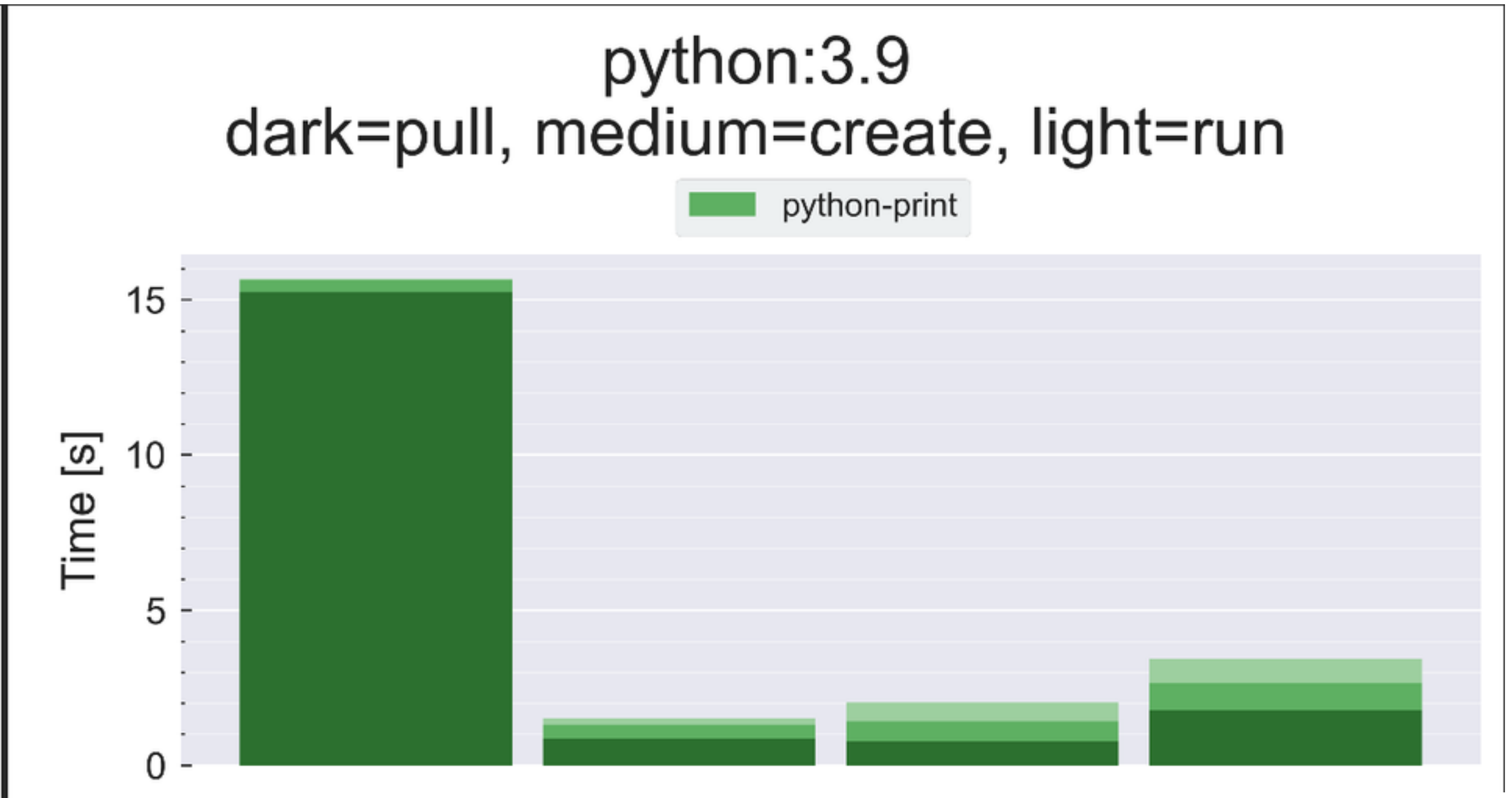
log scale

Results: python image: print() — remote registry/cache

linear scale



log scale



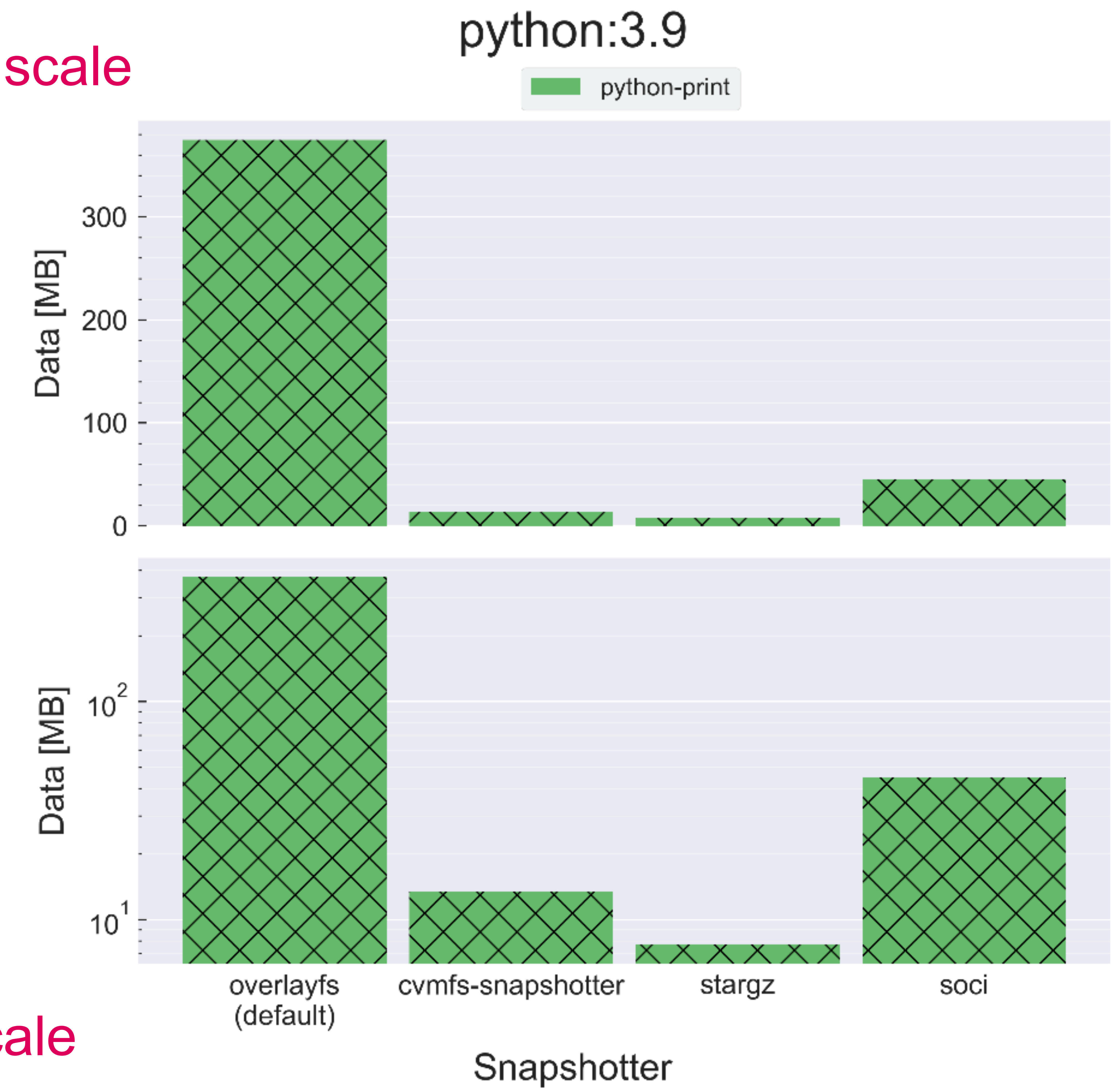
Observations:

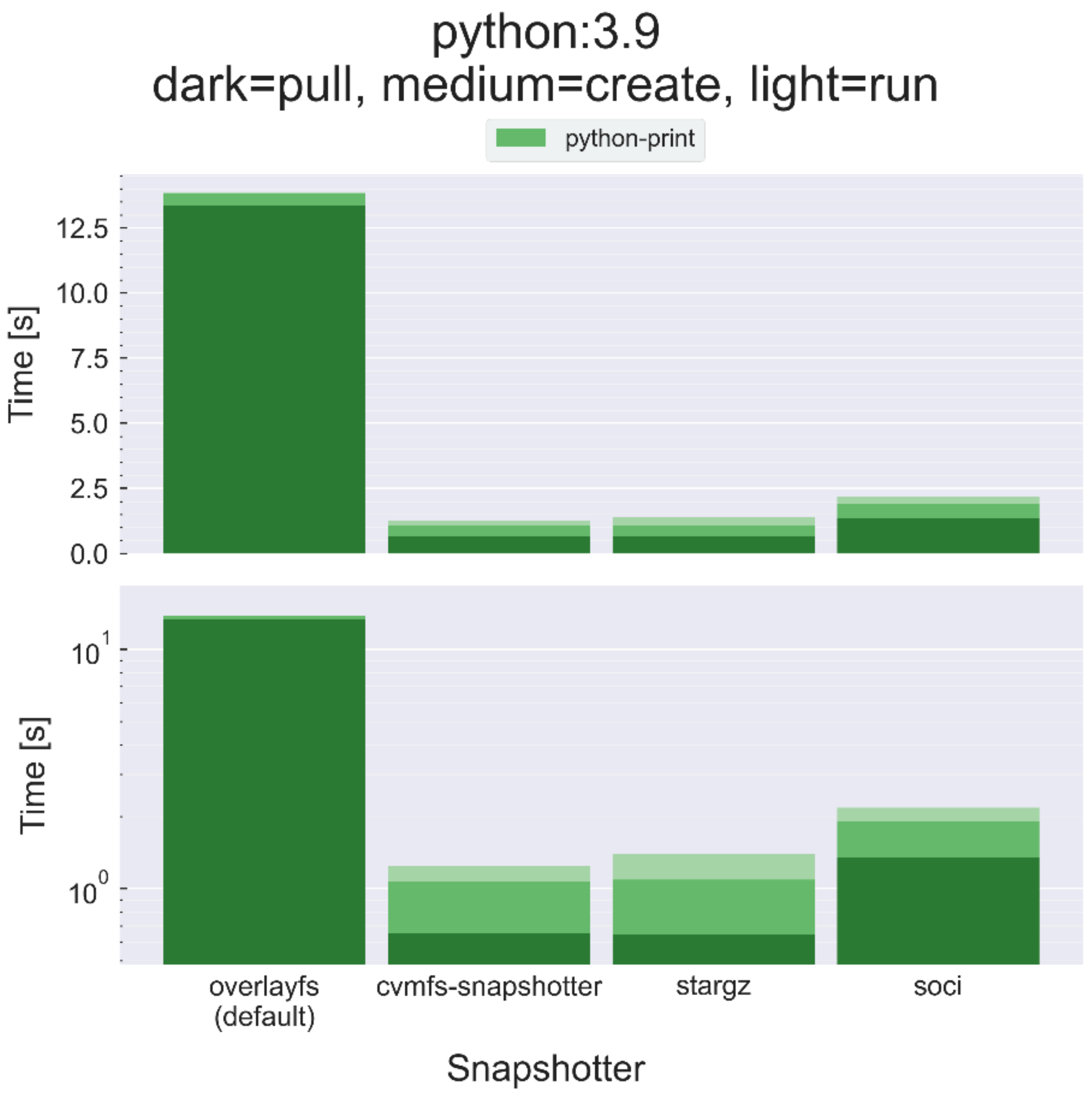
- Time to start image drastically reduced for all lazy snapshotters
- Only a few megabytes downloaded
- SOCI snapshotter loads more data because of layer minimum size requirement (configurable)

Results: python image: print() — local registry/cache

linear scale

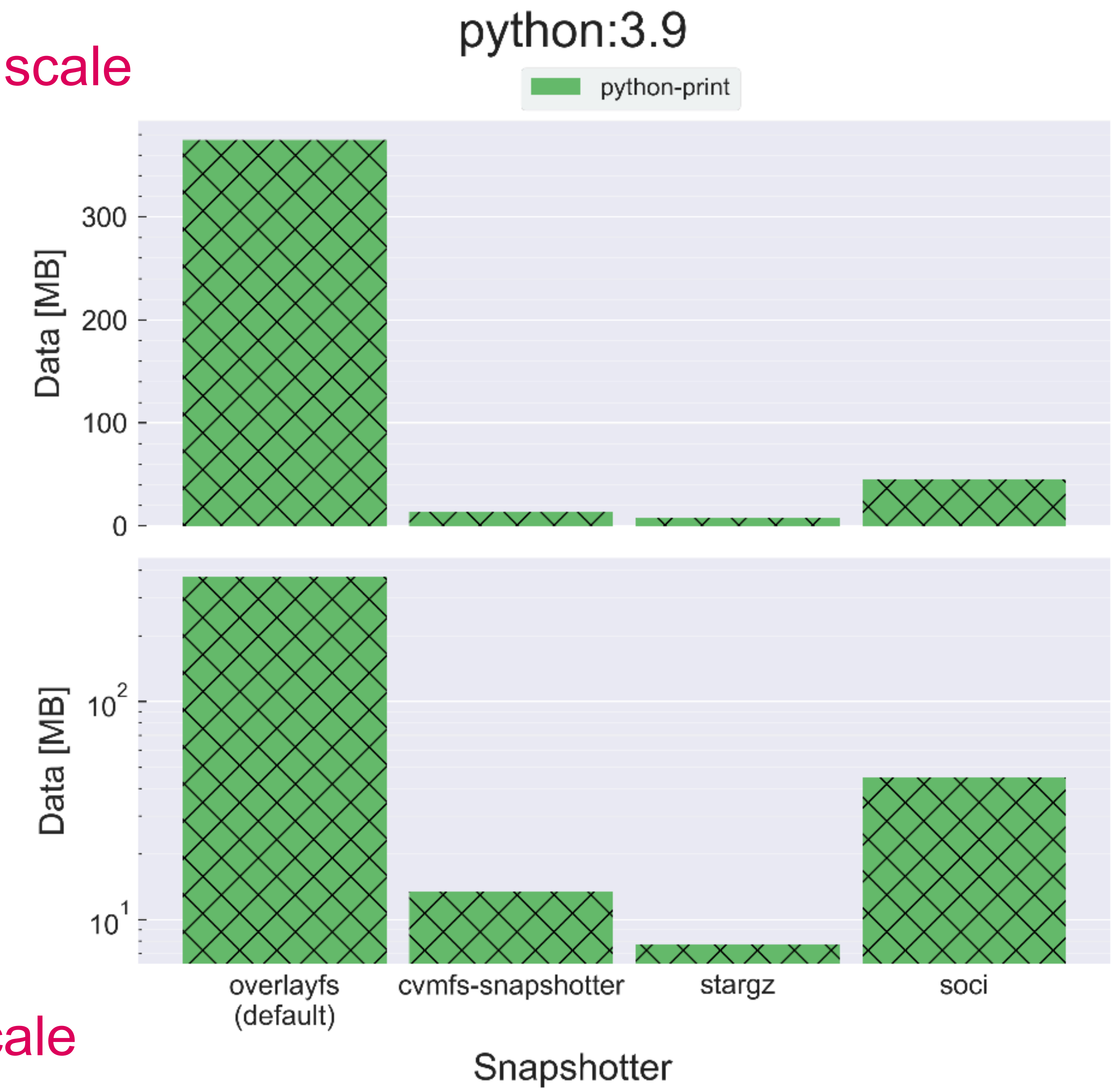
log scale



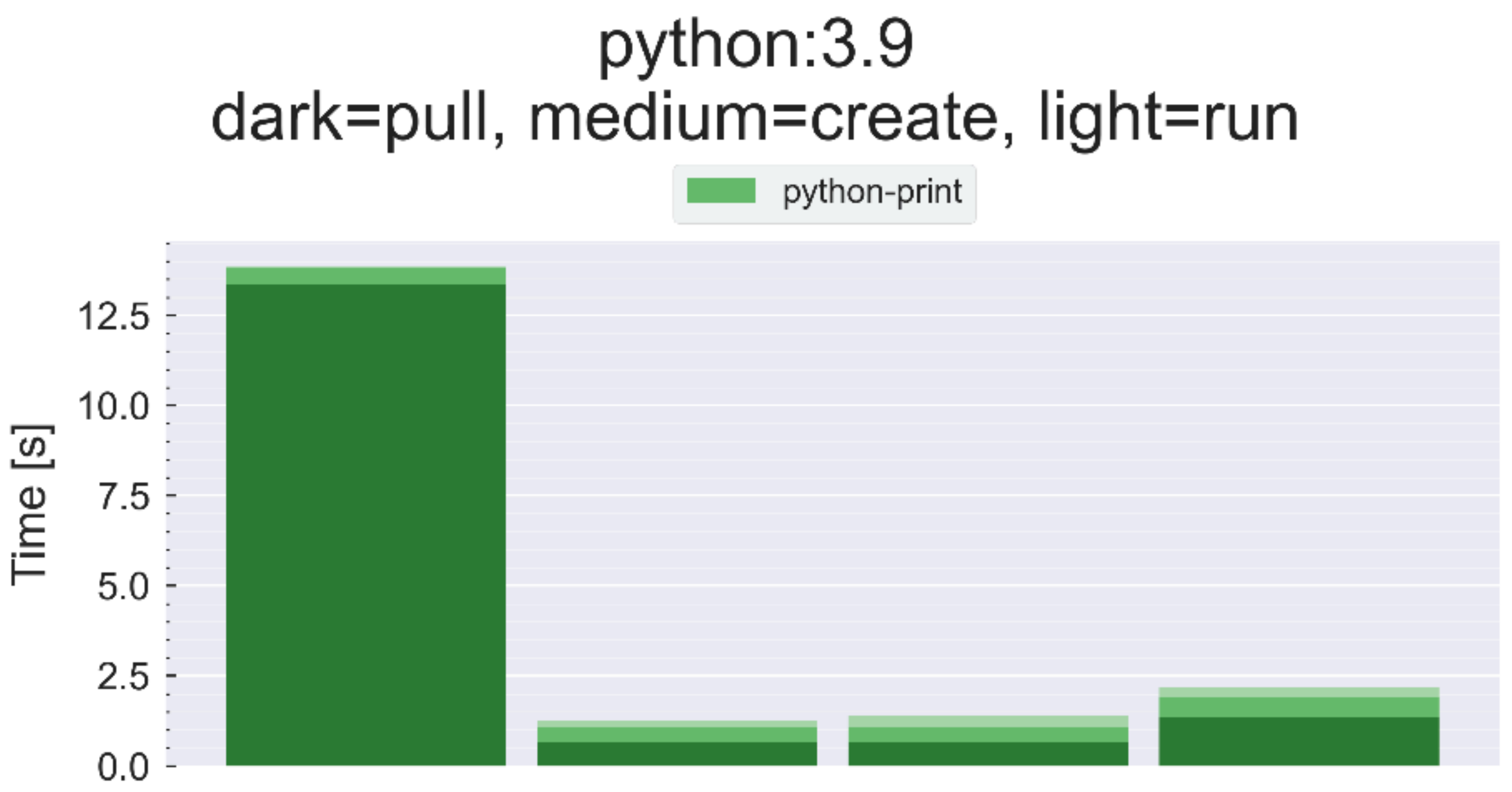


Results: python image: print() — local registry/cache

linear scale



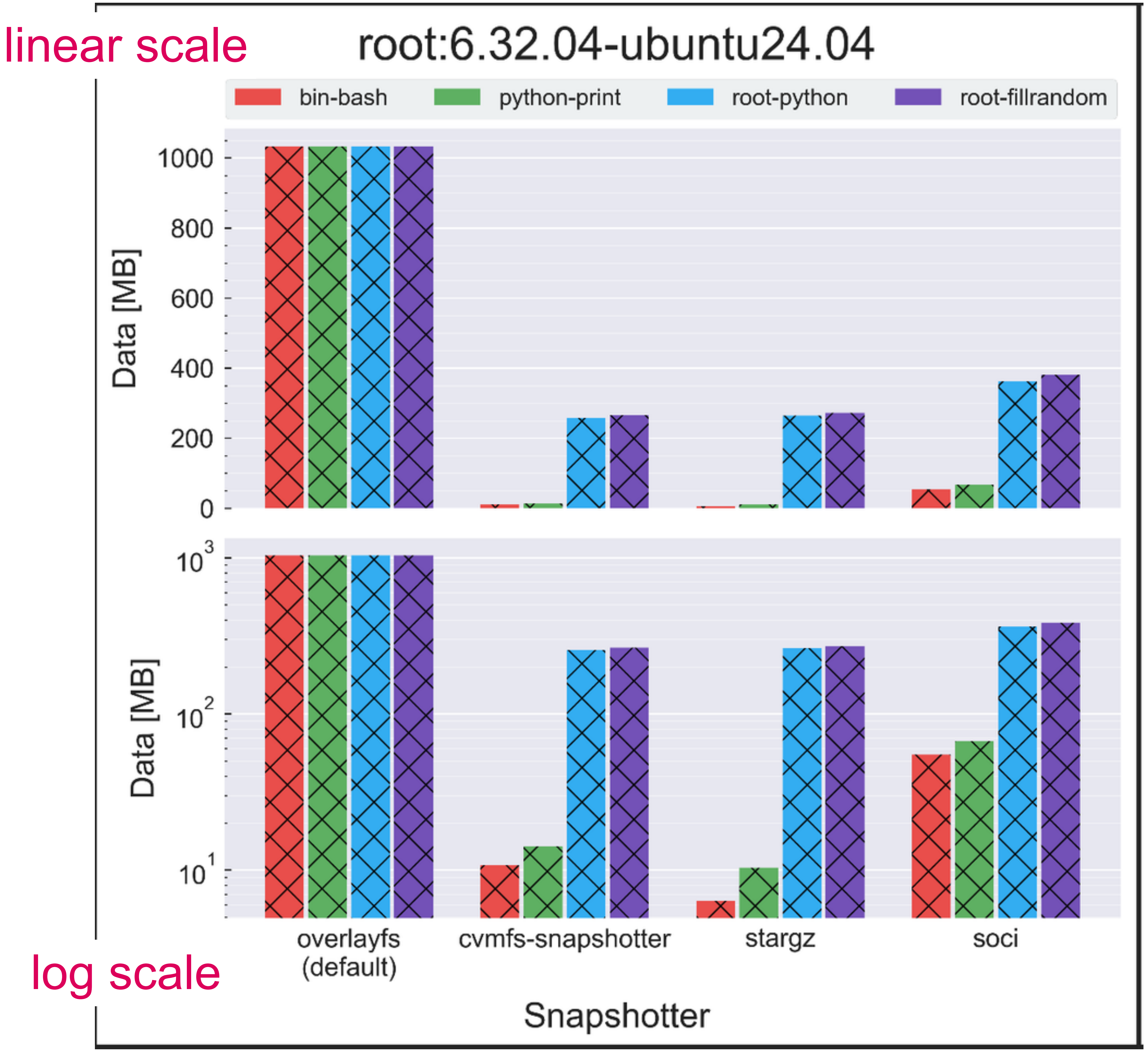
log scale



Observations:

- Very similar behaviour w.r.t. remote registry/ cache
- Slightly faster due to reduced network overhead

Results: ROOT image (1)



Investigate performance with workloads of increasing complexity:

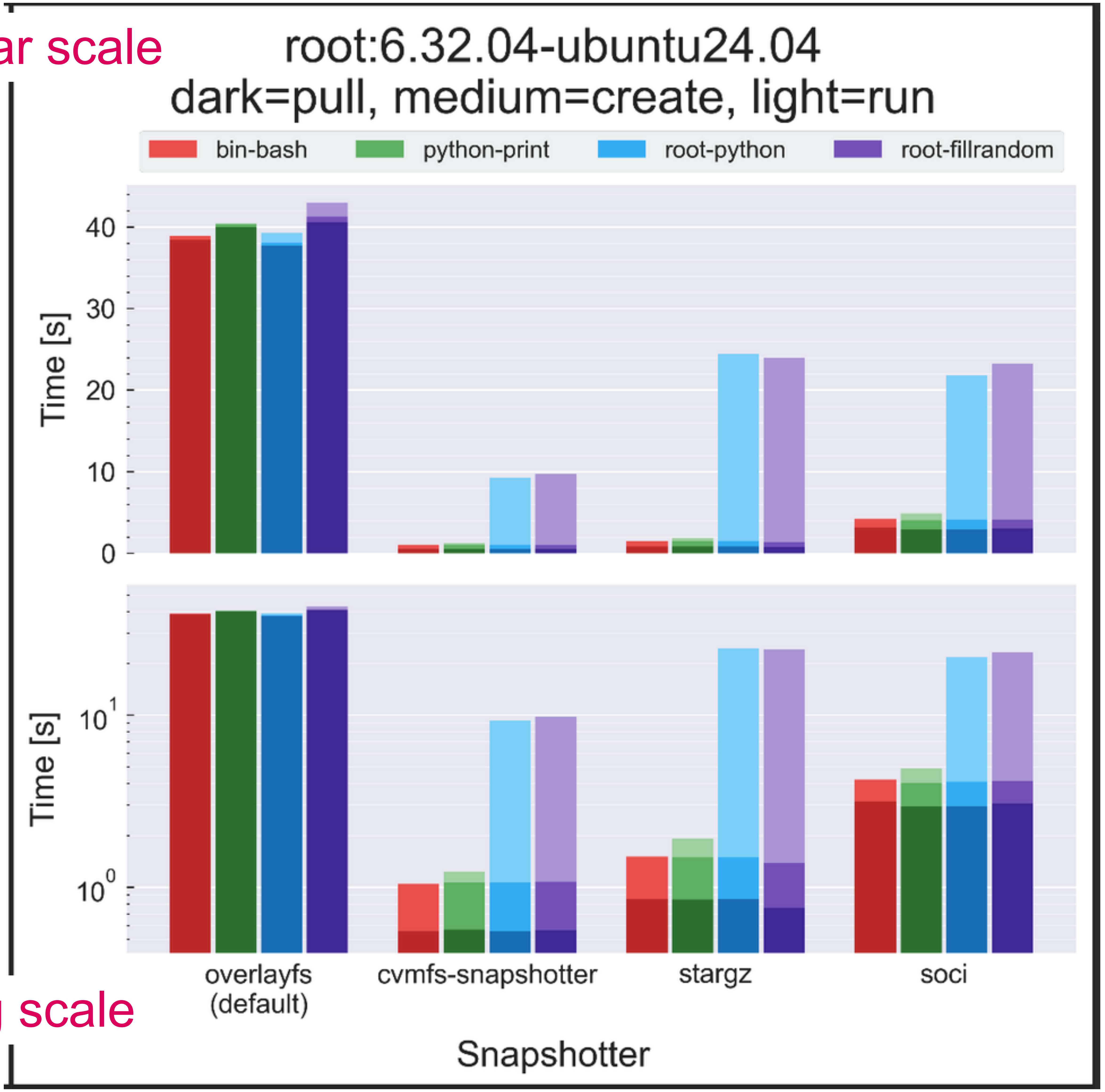
- */bin/bash*
- *print()* in python
- *import ROOT* in python
- *fillrandom.py* using pyROOT

Observations:

- Comparable performance
- *import ROOT* loads a lot of data

Results: ROOT image (2)

linear scale



log scale

Investigate performance with workloads of increasing complexity:

- */bin/bash*
- *print()* in python
- *import ROOT* in python
- *fillrandom.py* using pyROOT

Observations:

- CVMFS snapshotter faster than other two lazy snapshotters
- For complex workloads, pull time small compared to execution time (but mind significant data savings!)

Usability today

PSI

rootless docker available since RHEL 8 and kernel 4.18/5.11 — still requires (cluster) admin action

Use of Stargz snapshotter requires images to be converted (programatically) to a specific format → adoption might be slow/difficult

SOCI snapshotter only requires small addition to existing image—however, only certain registries support additional artifacts (Harbor ✅, GitLab ❌)

CVMFS snapshotter requires images to be “unpacked” → delay between building them and having them available (and they need to be added to the unpacker “sync” list)

Example configurations for docker and kubernetes

containerd configs for CVMFS snapshotter:

```
# /etc/containerd/config.toml
version = 2

# Ask containerd to use this particular snapshotter
[plugins."io.containerd.grpc.v1.cri".containerd]
    snapshotter = "cvmfs-snapshotter"
    # important: the cvmfs snapshotter needs
    # annotations to work.
    disable_snapshot_annotations = false

# Set the communication endpoint between containerd
# and the snapshotter
[proxy_plugins]
    [proxy_plugins.cvmfs-snapshotter]
        type = "snapshot"
        address =
            "/run/containerd-cvmfs-grpc/containerd-cvmfs-grpc.sock"
```

```
# /etc/containerd-cvmfs-grpc/config.toml
# Source of image layers
repository = "unpacked.cern.ch"
absolute-mountpoint = "/cvmfs/unpacked.cern.ch"
```

kubernetes (k3s) CVMFS snapshotter config:

```
# /var/lib/rancher/k3s/agent/etc/containerd/config.toml.tmpl
version = 2
[plugins."io.containerd.grpc.v1.cri".containerd]
    snapshotter = "cvmfs-snapshotter"
    disable_snapshot_annotations = false
[proxy_plugins]
    [proxy_plugins.cvmfs-snapshotter]
    type = "snapshot"
    address = "/run/containerd-cvmfs-grpc/containerd-cvmfs-grpc.sock"
[plugins."io.containerd.grpc.v1.cri".cni]
    bin_dir = "/var/lib/rancher/k3s/data/current/bin"
    conf_dir = "/var/lib/rancher/k3s/agent/etc/cni/net.d"
```

Conclusions

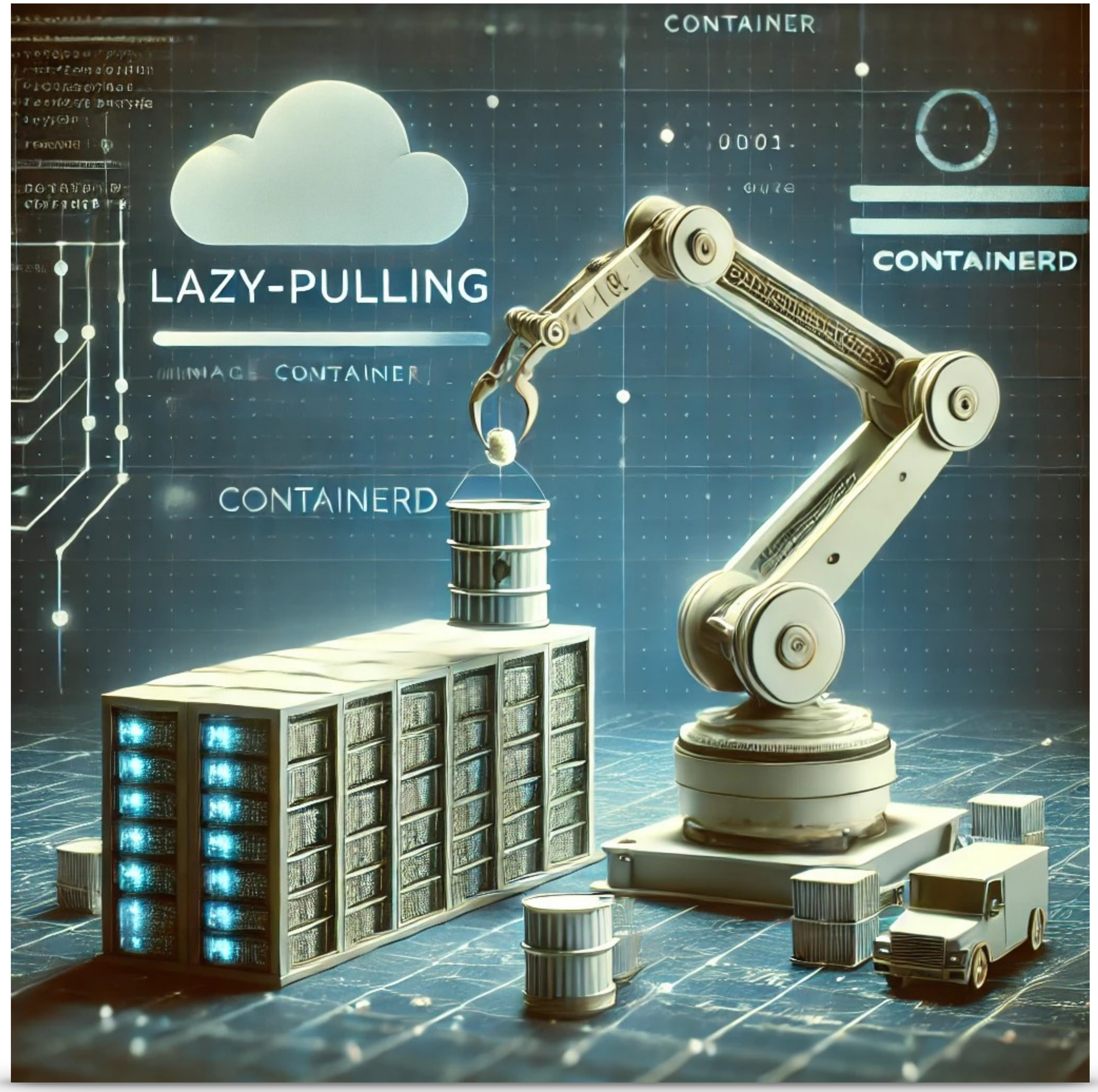
Container image snapshotters open up new possibilities for image distribution and access

- Significant bandwidth/data savings observed
- Time saving depends on workload/image details

Overall, evaluated snapshotters all have advantages and disadvantages in usability/requirements

Performance similar

- CVMFS snapshotter seems to be a bit faster than the other two snapshotters evaluated



PSI