# Making reproducible and publishable large-scale HPC experiments

Philippe SWARTVAGHER
https://ph-sw.fr

*HPC, Big Data & Data Science devroom - FOSDEM'24*

# About me

- Assistant Professor in Computer Science at ENSEIRB-MATMECA engineering school, Bordeaux, France

- Research about HPC, MPI, runtime systems, performance analyzis and visualisation, ...
  > Experimental work: writing/modifying software, evaluating performance
- PhD thesis:
  > *On the Interactions between HPC Task-based Runtime Systems and Communication Libraries*
  > 2019-2022

- Doing HPC experiments for several years now !

# In this presentation…

- Feedbacks from making several reproducible articles
- Advertisement for Guix
- Some advice (hopefully!)

- Apply mostly to scientific publications
  - > But probably also to, e.g., blog posts
  - > And not only HPC!

# What you can find in publications...

Nice results!

selected by the permutation $\sigma$, by applying Algorithms 1 and 2. By manually providing the hierarchy $h$ to Algorithm 3, we are able to consider any level of the hierarchy, extending Slurm's capabilities.

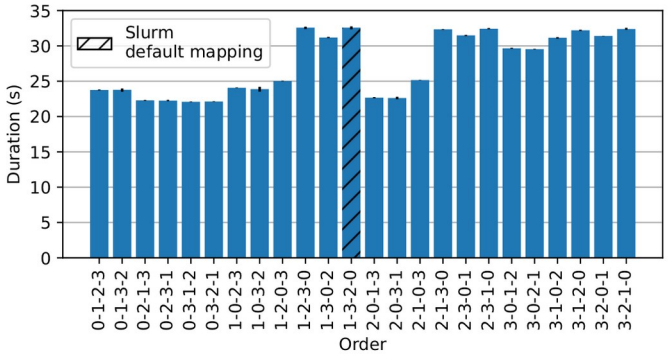
**Algorithm 3** Generate list of cores for `--cpu-bind=map_cpu`

**Inputs**: $h$ : hierarchy of one compute node, $\sigma$ : permutation, $n$ : number of cores to use
**Output**: $l$ : list of core physical IDs

```
1:  l ← []
2:  N ← 1                          ▷ Will be the total number of cores on a node
3:  for i = 0 to |h| − 1 do
4:      N = N × h[i]
5:  end for
6:  for c = 0 to N − 1 do
7:      r ← ComputeNewRank(h, c, σ)        ▷ Apply Alg. 1 and 2
8:      if r < n then
9:          l[r] ← c ▷ Core c will be on r^th position in the array l
10:     end if
11: end for
```

Cool algorithms!

For applications that do not use all cores of compute nodes, the mixed-radix decomposition technique can be used for two distinct




(a) With 1 NIC per compute node

**Figure 8: Impact of process mapping of Splatt executions**

in a different order are evaluated. Bars correspond to the median duration of 5 executions and error bars delimit the best and worst durations. For a given number of MPI processes, orders with bars

Philippe Swartvagher, Sascha Hunold, Jesper Larsson Träff, Ioannis Vardas.
Using Mixed-Radix Decomposition to Enumerate Computational Resources of Deeply Hierarchical Architectures.
SC 2023 - Workshops of The International Conference on High Performance Computing, Network, Storage, and Analysis, Nov 2023, Denver, CO, United States.

# What you can find in publications...

« How did you actually implement this algorithm? »

« How did you run the program? »

Nice results!

« How did you compute statistics? »

« Which options did you use? »

Cool algorithms!

« I don't understand, show me the code! »

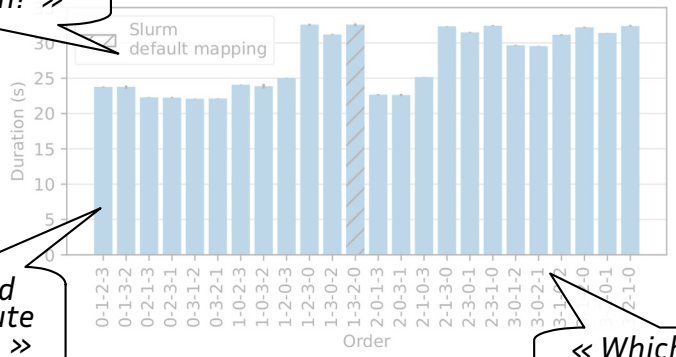
selected by the permutation $\sigma$, by applying Algorithms 1 and 2. By manually providing the hierarchy $h$ to Algorithm 3, we are able to consider any level of the hierarchy, extending Slurm's capabilities.

**Algorithm 3** Generate list of cores for `--cpu-bind=map_cpu`

**Inputs**: $h$ : hierarchy of one compute node, $\sigma$ : permutation, $n$ : number of cores to use
**Output**: $l$ : list of core physical IDs

```
1: l ← []
2: N ← 1                          ▷ Will be the total number of cores on a node
3: for i = 0 to |h| − 1 do
4:     N = N × h[i]
5: end for
6: for c = 0 to N − 1 do
7:     r ← ComputeNewRank(h, c, σ)        ▷ Apply Alg. 1 and 2
8:     if r < n then
9:         l[r] ← c ▷ Core c will be on r^th position in the array l
10:    end if
11: end for
```

For applications that do not use all cores of the compute nodes, the mixed-radix decomposition technique can be used for binding the




(a) With 1 NIC per compute node

Figure 8: Impact of process mapping of Splatt executions

in a different order are evaluated. Bars correspond to the median duration of 5 executions and error bars delimit the best and worst durations. For a given number of MPI processes, orders with bars

Philippe Swartvagher, Sascha Hunold, Jesper Larsson Träff, Ioannis Vardas.
Using Mixed-Radix Decomposition to Enumerate Computational Resources of Deeply Hierarchical Architectures.
SC 2023 - Workshops of The International Conference on High Performance Computing, Network, Storage, and Analysis, Nov 2023, Denver, CO, United States.

# What you can find in publications...

« How did I actually implement this algorithm? »

selected by the permutation $\sigma$, by applying Algorithms 1 and 2. By manually providing the hierarchy $h$ to Algorithm 3, we are able to consider any level of the hierarchy, extending Slurm's capabilities.

**Algorithm 3** Generate list of cores for `--cpu-bind=map_cpu`

**Inputs**: $h$ : hierarchy of one compute node, $\sigma$ : permutation, $n$ : number of cores to use
**Output**: $l$ : list of core physical IDs

```
l ← []
N ← 1                          ▷ Will be the total number of cores on a node
for i = 0 to |h| − 1 do
    N = N × h[i]
end for
for c = 0 to N − 1 do
    r ← ComputeNewRank(h, c, σ)        ▷ Apply Alg. 1 and 2
    if r < n then
        l[r] ← c ▷ Core c will be on r^th position in the array l
    end if
end for
```

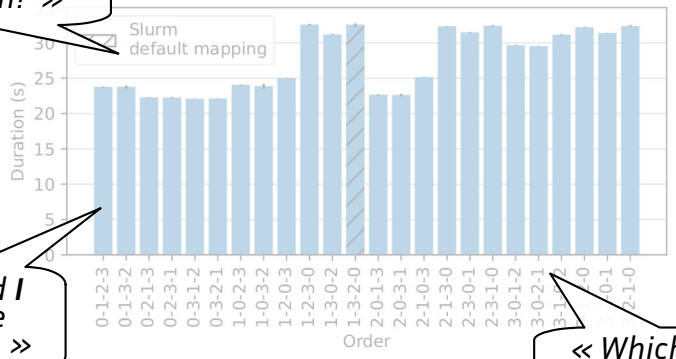
For applications that do not use all cores of the compute nodes, the mixed-radix decomposition technique can be used for [illegible]

Cool algorithms!

« ~~I don't understand, show me the code!~~ »

« How did I run the program? »

Nice results!




(a) With 1 NIC per compute node

« How did I compute statistics? »

« Which options did I use? »

Figure 8: Impact of process mapping of Splatt executions

in a different order are evaluated. Bars correspond to the median duration of 5 executions and error bars delimit the best and worst durations. For a given number of MPI processes, orders with bars

Philippe Swartvagher, Sascha Hunold, Jesper Larsson Träff, Ioannis Vardas.
Using Mixed-Radix Decomposition to Enumerate Computational Resources of Deeply Hierarchical Architectures.
SC 2023 - Workshops of The International Conference on High Performance Computing, Network, Storage, and Analysis, Nov 2023, Denver, CO, United States.

# Reproducibility

- Many terms, slightly different meanings:
  - > *Reproducibility*
  - > *Replicability*
  - > *Repeatability*
  - > *Availability*

- *Make available everything needed to reproduce something*, e.g., an experiment
  - > In this case: scripts and source code

https://www.acm.org/publications/policies/artifact-review-and-badging-current

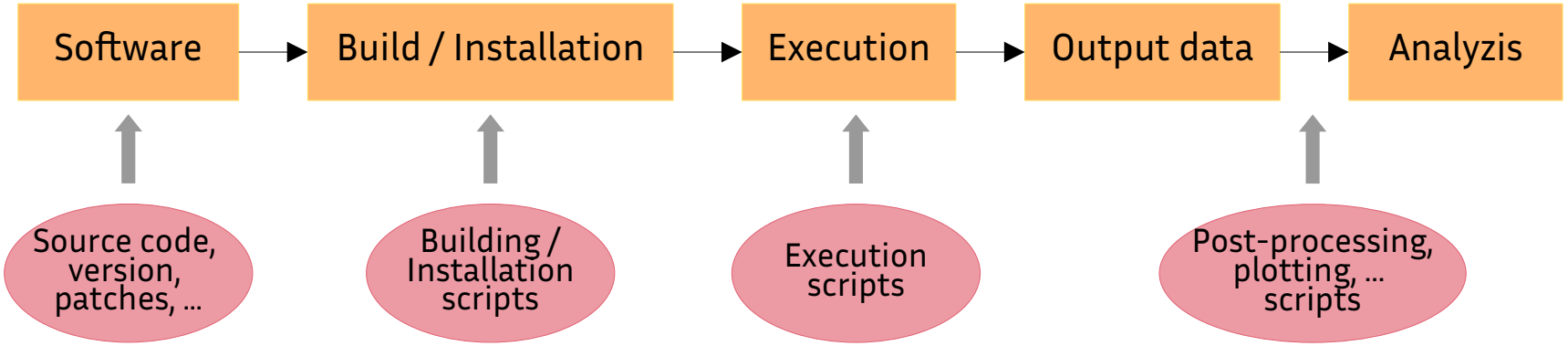
# Why should I care?

- ~~Conferences / Journals require it and give you nice badges~~

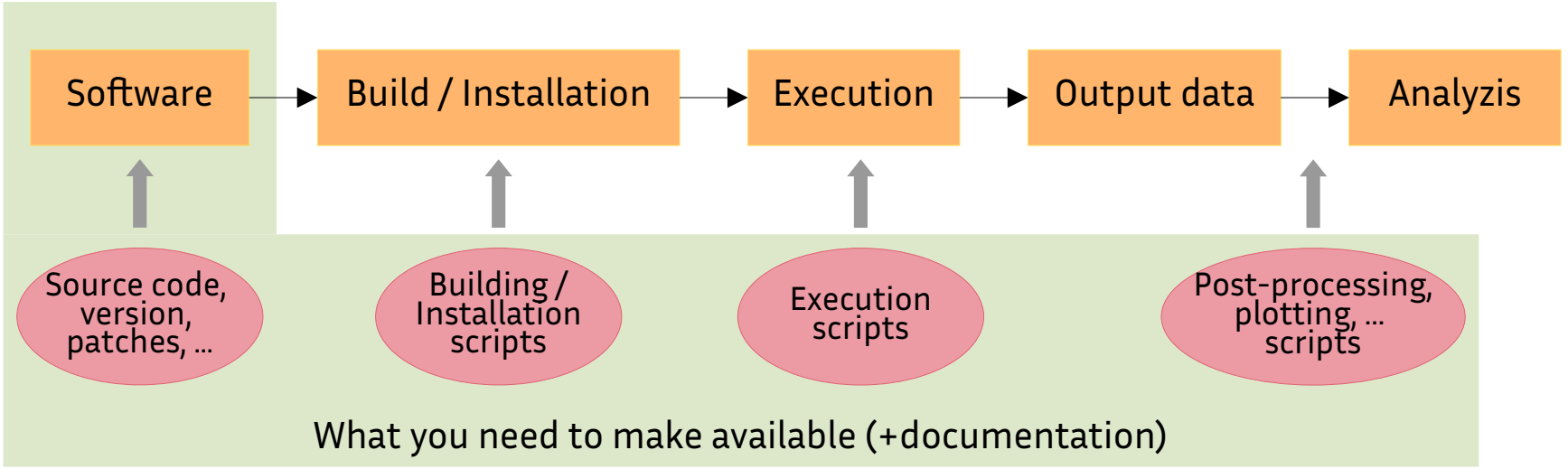

- For you:
  - > To easily come back to an experiments 6 months later
  - > To have trust in your experiments
  - > Some kind of open-source?
- For others:
  - > To see what you actually did in practice
  - > To better understand what you are talking about
  - > To reproduce your experiments (to change or extend it, to compare themselves with it, ...)
  - > To easily share with your colleague / collaborator / ... what you did and how
- Part of the contribution!

# The common workflow

# The common workflow



# The common workflow



# Different levels

Availability  Bit-to-bit reproducibility

- *Availability* is a minimum
- *Bit-to-bit reproducibility* when you manage to rebuild the exact same environment
  - > Not always possible
  - > Not always necessary: on a different system, with different input data or configuration, ...

# Software

Source code, version, patches, ...

- Used software, version, where to download
- But also: which compiler? Which compiler version? Which version of library X and Y? ...
- **The whole software environment is important!**

Building / Installation scripts

- How software were installed? Which building flags? Which dependencies?

# Software

## Minimum

Source code, version, patches, ...

Building / Installation scripts

| Package | Version | Website | Code location | Commit |
|---|---|---|---|---|
| NewMadeleine | 2021-05-21 | https://pm2.gitlabpages.inria.fr/NewMadeleine | https://gitlab.inria.fr/pm2/pm2.git | 2765619bd2c77af733778643f369ba086a29715a |
| StarPU | 1.3.8 | http://starpu.gforge.inria.fr | https://gitlab.inria.fr/starpu/starpu.git | starpu-1.3.8 |
| Chameleon | 1.1.0 | https://gitlab.inria.fr/solverstack/chameleon | https://gitlab.inria.fr/solverstack/chameleon | 4db899ca30d29927018d83964b9b6d517269abe1 |
| StarVZ | 0.5.1 | https://github.com/schnorr/starvz | https://github.com/schnorr/starvz | b789296a90e22ae8cd73a6a58df3ff2bd1ff02e3 |
| GCC | 11.2.0 | https://gcc.gnu.org/ | | |
| Slurm | 19.05.8 | https://slurm.schedmd.com/ | https://download.schedmd.com/slurm/slurm-19.05.8.tar.bz2 | |
| Intel MKL | 2019.1.144 | https://software.intel.com/en-us/mkl | http://registrationcenter-download.intel.com/akdlm/irc_nas/tec/14895/l_mkl_2019.1.144.tgz | |
| FxT | 0.3.14 | https://savannah.nongnu.org/projects/fkt | http://download.savannah.nongnu.org/releases/fkt/fxt-0.3.14.tar.gz | |
| Python | 3.8.2 | https://www.python.org | https://www.python.org/ftp/python/3.8.2/Python-3.8.2.tar.xz | |
| Matplotlib | 3.1.2 | https://matplotlib.org/ | https://files.pythonhosted.org/packages/source/m/matplotlib/matplotlib-3.1.2.tar.gz | |
| Pandas | 1.3.0 | https://pandas.pydata.org | https://files.pythonhosted.org/packages/source/p/pandas/pandas-1.3.0.tar.gz | |

NewMadeleine is built with the following commands:

```
git clone https://gitlab.inria.fr/pm2/pm2.git
cd pm2
git checkout 5eaab4cc1e8d70061db813a598af227efba52dc9
cd scripts
./pm2-build-packages ./pukabi+madmpi-mini.conf --prefix=<installation prefix>
# Set environment variables as indicated at the end of installation
```

memory-contention is built with the following commands:

```
git clone https://gitlab.inria.fr/pswartva/memory-contention.git
cd memory-contention
git checkout e2d788f5718386c818f0aa07e826fd9e8c6b4870
./autogen.sh
mkdir build
cd build
../configure
make
```

# Software

Source code, version, patches, ...

Building / Installation scripts

- What about...
  - > Module files?
  - > Spack / Conda / Easybuild / ... environments?
  - > Docker / Singularity / ... images?

# Software

Source code, version, patches, ...

Building / Installation scripts

- What about...
  - > Module files → specific to a system, don't say how it is build, may disappear after some time
  - > Spack / Conda / Easybuild / ... environments → don't fully isolate the environment, depends on what is already installed on the system
  - > Docker / Singularity / ... images → what it inside the image? Building the image 6 months later may contain different things

**No garantee of being always reproducible!**

Even worse: give the feeling of being reproducible

# Enters... Guix!


- From `https://hpc.guix.info`:
  - Transactional package manager
  - Create as many software environments as you like (like `virtualenv` but not only for Python; like `module`, but for every software defined in Guix)
  - The software environments created with Guix are fully reproducible: a package built from a specific Guix commit on your laptop will be exactly the same as the one built on the HPC cluster you deploy it too, usually bit-for-bit.

# Software


Source code, version, patches, ...

Building / Installation scripts

```
module load openmpi/4.1.2
# Build chameleon
cmake .. -DFOO=BAR
make && make install
mpirun ...
```

```
guix shell --pure chameleon -- mpirun ...
```



Inria

# Software


Source code, version, patches, ...

Building / Installation scripts

- All Guix package definitions are located in Guix channels
  - > Actually Git repositories
  - > → Commit hashes of used Guix channels pin versions of all packages
    (solves problem of pinning versions of dependencies of dependencies)
  - > Everything except the kernel



Inria

# Software

Guix

Source code, version, patches, ...

Building / Installation scripts

- Export currently used channels **(and their versions)**:

```
guix describe -f channels > channels.scm
```

- Explicitly use pinned channels:

```
guix time-machine
--channels=./channels.scm -- shell --pure
chameleon -- mpirun …
```

- Backup `channels.scm`: to be sure to execute the same code, even 6 months later

```
(list (channel
        (name 'guix)
        (url "https://git.savannah.gnu.org/git/guix.git")
        (branch "master")
        (commit
          "ec66f84824198f380d20126d3e4b2ea795fd205a")
        (introduction
          (make-channel-introduction
            "9edb3f66fd807b096b48283debdcddccfea34bad"
            (openpgp-fingerprint
              "BBB0 2DDF 2CEA F6A8 0D1D  E643 A2A0 6DF2 A33A 54FA"))))
      (channel
        (name 'guix-hpc-non-free)
        (url "https://gitlab.inria.fr/guix-hpc/guix-hpc-non-free.git")
        (branch "master")
        (commit
          "58aaac8c18773d900511d441e935145d73cdfc5e"))
      (channel
        (name 'guix-hpc)
        (url "https://gitlab.inria.fr/guix-hpc/guix-hpc.git")
        (branch "master")
        (commit
          "74840c47b744ad7342e7a86852831009a2831630")))
```



Inria

# The Guix killer feature: package transformations


Source code, version, patches, ...

Building / Installation scripts

- Change package definition on-the-fly
- Simple package substitution:

```
guix shell --pure chameleon --with-input=openblas=mkl -- mpirun …
```

- Use a specific upstream branch, commit, version...:

```
guix shell --pure chameleon --with-commit=starpu=acae6e -- mpirun …
```

- Apply a patch to package source code:

```
guix shell --pure chameleon --with-patch=chameleon=./foo.patch -- mpirun …
```

- And others
- Makes the installation of software much easier!
  > No need for installation scripts and instructions anymore!

https://guix.gnu.org/en/manual/devel/en/html_node/Package-Transformation-Options.html



Inria

# Guix: what about performance?


- Should be the same
- What is not in Guix: tuning of libraries made by cluster providers
  > @Cluster providers: please share these modifications!
- What is in Guix: package transformation `--tune` to rebuild package for a specific processor architecture
- One of the goal of the Guix-HPC effort

https://hpc.guix.info
https://hpc.guix.info/blog/2022/01/tuning-packages-for-a-cpu-micro-architecture/
https://hpc.guix.info/blog/2019/12/optimized-and-portable-open-mpi-packaging/

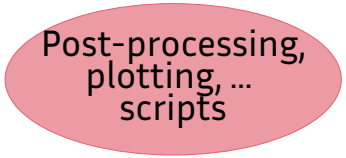


Inria

# Execution scripts


- Comment!
  - > What you are doing and why
- Try to separate what is specific to *your* experiment (platform, input data, ...) and experiment logic:
  - > Job scheduler system
  - > Used resources (number of nodes, cores, ...)
  - > Paths, usernames, problem size, input data, ...

# Post-processing, plotting, ... scripts




- Also executed inside a Guix environment!
- Seperate post-processing (analyzing data, computing what will be plotted) from plotting
  > Ease (and accelerate!) the writting of plotting scripts
- Factorization (moving things to functions, modules, ...) is not always a good idea
  > You may need to add annotation to this specific plot
  > You may need to compute this specific value only for this kind of data
  > Scripts have to remain flexible enough

# Post-processing, plotting, ... scripts (2)

Post-processing, plotting, ... scripts

- Directly generate from scripts codes of table to be included in your TeX file
  - > Think if you need to change how you compute all the values...

```
\begin{tabular}{|c|c|c|c|c|c|c|c|}
  % ...
      \hline
      \henri & \percent{2.62} & \percent{3.53} & \percent{3.08} \\
      \hline
      \henrisubnuma & \percent{2.90} & \percent{3.80} & \percent{3.69} \\
      \hline
      \bora & \percent{4.39} & \percent{5.14} & \percent{4.77} \\
      \hline
      \dahu & \percent{2.76} & \percent{2.38} & \percent{2.57} \\
      \hline
      \diablo & \percent{2.32} & \percent{1.54} & \percent{1.93} \\
      \hline
      \grvingt & \percent{3.41} & \percent{8.06} & \percent{5.74} \\
      \hline
      \pyxis & \percent{1.15} & \percent{13.32} & \percent{7.24} \\
      \hline
      \occigen & \percent{0.01} & \percent{0.01} & \percent{0.01} \\
      \hline
      \textbf{Average} & \percent{3.09} & \percent{5.40} & \percent{4.28} \\
      \hline
\end{tabular}
```

| orm | Communications | | |
|---|---|---|---|
| | *on Samples* | *on non-Samples* | *all* |
| ri | 2.62 % | 3.53 % | 3.08 % |
| ubnuma | 2.90 % | 3.80 % | 3.69 % |
| ly | 8.22 % | 10.84 % | 9.53 % |
| ra | 4.39 % | 5.14 % | 4.77 % |
| hu | 2.76 % | 2.38 % | 2.57 % |
| blo | 2.32 % | 1.54 % | 1.93 % |
| ngt | 3.41 % | 8.06 % | 5.74 % |
| is | 1.15 % | 13.32 % | 7.24 % |
| gen | 0.01 % | 0.01 % | 0.01 % |
| rage | 3.09 % | 5.40 % | 4.28 % |

# Document things

- In a `README.md`

- What is this about? Link to the paper
- What do I need for described experiments?
  - > Which hardware, how many nodes, ...
  - > How much storage, RAM, ...
  - > How much time
- Installation and execution instructions with **and** without Guix
- For each table, plot, ... in the paper:
  - > Section about how it is done, folder containing scripts and other relevant resources

# Where to make it available?

- A Git repository
- Dedicated repository: `paper-title-reproducibility` → `paper-title-r13y`

# Where to make it available?

- A Git repository
- Dedicated repository: `paper-title-reproducibility` → `paper-title-r13y`

- **Archive it forever on SoftwareHeritage!**
  - > Will give you a SWHID to identify your repository / snapshot / directory
  - > Like a DOI but computed based on archived content (like a Git commit ID)

`https://archive.softwareheritage.org/`

# Reference it in publications

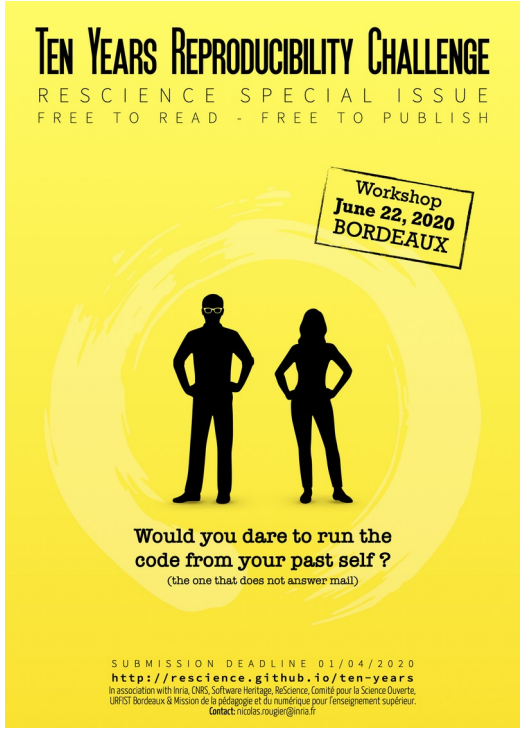
A public companion contains the instructions to reproduce our study: https://gitlab.inria.fr/pswartva/paper-model-memory-contention-r13y, archived on https://www.softwareheritage.org/ with the ID `swh:1:snp:306f7c10cf69a5860587e5aad62b76070b798ecd`.

# (Almost) Last remarks

- Have reproducibility in mind from the beginning of your experiments
- If *bit-to-bit reproducibility* seems difficult, at least publish your code and scripts
  - > To have at least *availability*
- Guix is not mandatory!
  - > But very handy tool to get *bit-to-bit reproducibility* for (almost) free
- What about data (input, output, pre- and post-processed)?
  - > Don't have a strong opinion yet
  - > Host everything on Zenodo or equivalent?
  - > Mandatory for reproducibility

# Some initiatives

- ReScience C Journal
  - > Publication of articles explaining how another article was replicated (or not)
  - > Open and public reviewing process
- Ten Years Reproducibility Challenge
  - > Reproduce one of your 10 year old articles
- Guix Past
  - > Guix channel containing old software, old versions
- Follow the activity of Guix-HPC for blog posts and events!




https://rescience.github.io/
https://rescience.github.io/ten-years/
https://gitlab.inria.fr/guix-hpc/guix-past
https://hpc.guix.info/

# Conclusion

- Publish source code of your software and experiment scripts
- Document it

- Reproducibility adds more value to experiments, results, research
- You will be more confident about your experiments
  > Especially in case you need to run them again
- It contributes to makes a better science!
  > Spread the word!

https://hpc.guix.info/blog/2023/06/a-guide-to-reproducible-research-papers/
https://gitlab.tuwien.ac.at/philippe.swartvagher/paper-mpi-rank-reordering-r13y
https://gitlab.inria.fr/pswartva/paper-starpu-traces-r13y/
https://gitlab.inria.fr/pswartva/paper-model-memory-contention-r13y