Experimenting for the Worst-Case Reviewer

Tools and Tips for REP Hygiene

IRIT PhD days – ASR department

Millian Poquet

2025-07-08

Who am I?

Millian Poquet
- MCF Univ. Toulouse / IRIT
- `millian.poquet@irit.fr`
- https://mpoquet.github.io ← slides available

Research interests
- Resource management on distributed systems
- Energy & performance optimization
- Sustainability
- How to experiment on these systems (simulation, real)

Propaganda / bias warning
- Long-term Nix user, NixOS maintainer
- Long-term Grid'5000 user, IRIT group co-manager

Introduction

Stories

Story 1 – Bob is writing its manuscript
When redoing graphs for manuscript, he finds out results have completely changed : (

Story 2 – Bob left, Alice wants to continue his work
Cannot find/run/modify existing work without great effort : (

This talk in a nutshell

Have you read this paper?

- *Want People to Read Your Paper? Consider the Worst-Case Reviewer*[1]

[1] https://www.sigarch.org/want-people-to-read-your-paper-consider-the-worst-case-reviewer/

This talk in a nutshell

Have you read this paper?

- *Want People to Read Your Paper? Consider the Worst-Case Reviewer*[1]

This talk focuses on making your experiments robust to the Worst-Case Reviewer – that's me

Side effects

- Increased likelihood of papers accepted
- Increased likelihood of work being reused
- Increased likelihood of gaining insights?

[1] https://www.sigarch.org/want-people-to-read-your-paper-consider-the-worst-case-reviewer/

Outline

REP & what can go wrong

REP definitions

ACM terminology[1] as of 2020-08-24

- **REPeatability** (same team, same experimental setup)
- **REProducibility** (different team, same experimental setup)
- **REPlicability** (different team, different experimental setup)

Reproducibility definition

*The **measurement** can be obtained with **stated precision** by a **different team** using the **same measurement procedure**, the **same measuring system**, under the **same operating conditions**, in the **same or a different location** on multiple trials. For computational experiments, this means that an **independent group** can obtain the same result using the **author's own artifacts**.*

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

My opinion of these definitions

Positives

- Attempt to standardize terminology
- Same/different team encompasses many problems — lack of code, lack of doc…

Negatives

- Very vague. What kind of measurement? What kind of measurement procedure?
- Statistical REP? Binary REP?
- REP with mutability?
- Longevity of REP?

What can go wrong? vague software (algo ≠ implem)

It is common to see simple subtaks not detailed in algorithms in papers — *e.g.*, `tolower(s)`

- But how they are implemented can change the results of your experimental setup `:/`

What can go wrong? vague software (algo ≠ implem)

It is common to see simple subtaks not detailed in algorithms in papers — *e.g.*, `tolower(s)`

- But how they are implemented can change the results of your experimental setup `:/`

```c
void tolower_simple_libc(const char *input, char *output, size_t size) {
  unsigned int usize = (unsigned int)size;
  for (unsigned int i = 0; i < usize; i++) {
    output[i] = tolower((unsigned char)input[i]);
  }
}
```

What can go wrong? vague software (algo ≠ implem)

It is common to see simple subtaks not detailed in algorithms in papers — *e.g.*, `tolower(s)`

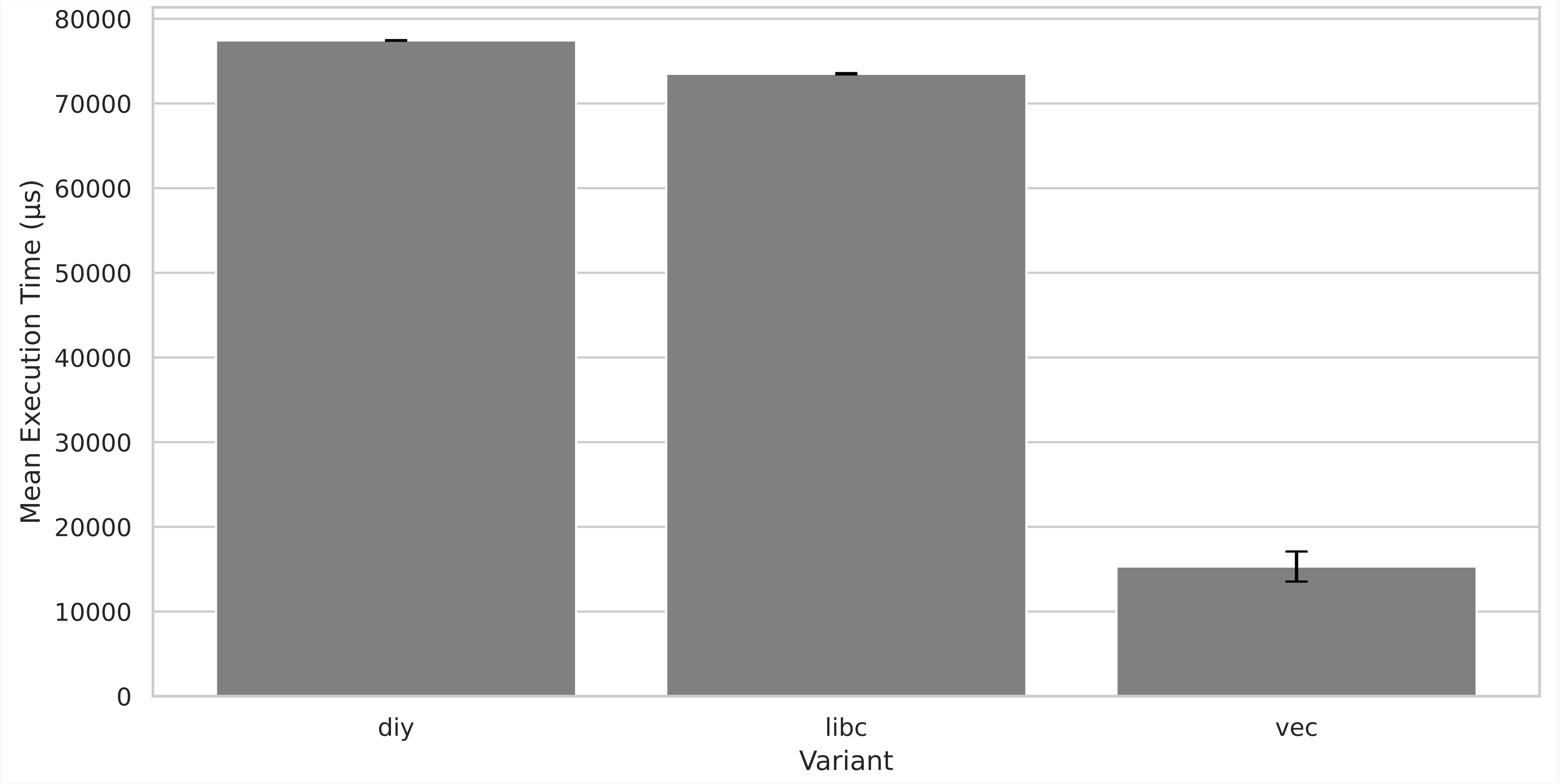
- But how they are implemented can change the results of your experimental setup `:/`

```c
void tolower_simple_libc(const char *input, char *output, size_t size) {
  unsigned int usize = (unsigned int)size;
  for (unsigned int i = 0; i < usize; i++) {
    output[i] = tolower((unsigned char)input[i]);
  }
}

void tolower_simple_diy(const char *input, char *output, size_t size) {
  unsigned int usize = (unsigned int)size;
  for (unsigned int i = 0; i < usize; i++) {
      char c = input[i];
      if (c >= 'A' && c <= 'Z') {
          output[i] = c + 32;
      } else {
          output[i] = c;
      }
  }
}
```

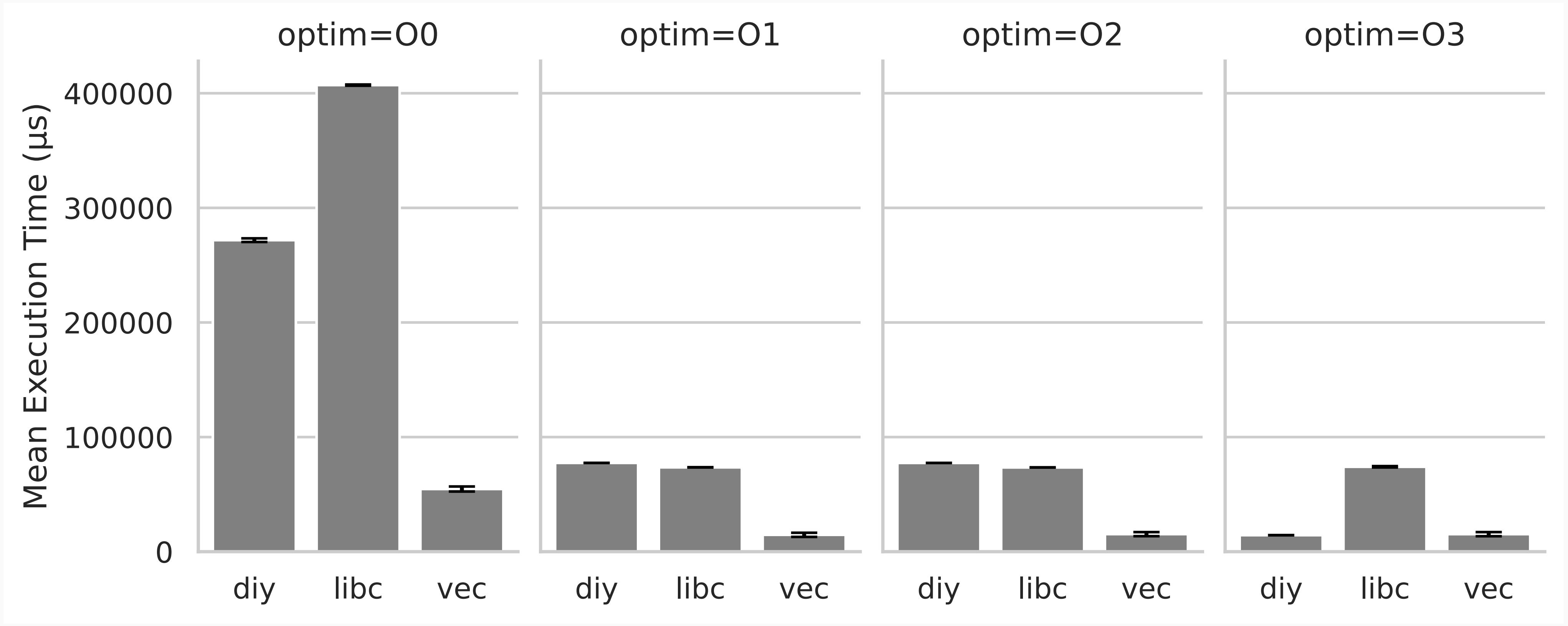
What can go wrong? vague software (algo ≠ implem)

```c
void tolower_vec(const char *input, char *output, size_t size) {
  const __m256i a_minus1 = _mm256_set1_epi8('A' - 1);
  const __m256i z_plus1 = _mm256_set1_epi8('Z' + 1);
  const __m256i delta = _mm256_set1_epi8(32);
  unsigned int usize = (unsigned int)size;
  unsigned int vec_chunks = usize / 32;

  for (unsigned int i = 0; i < vec_chunks; i++) {
      unsigned int offset = i * 32;
      __m256i data = _mm256_loadu_si256((const __m256i*)(input + offset));
      __m256i mask_upper = _mm256_and_si256(
          _mm256_cmpgt_epi8(data, a_minus1),
          _mm256_cmpgt_epi8(z_plus1, data)
      );
      __m256i result = _mm256_or_si256(
          _mm256_and_si256(mask_upper, _mm256_add_epi8(data, delta)),
          _mm256_andnot_si256(mask_upper, data)
      );
      _mm256_storeu_si256((__m256i*)(output + offset), result);
  }
}
```

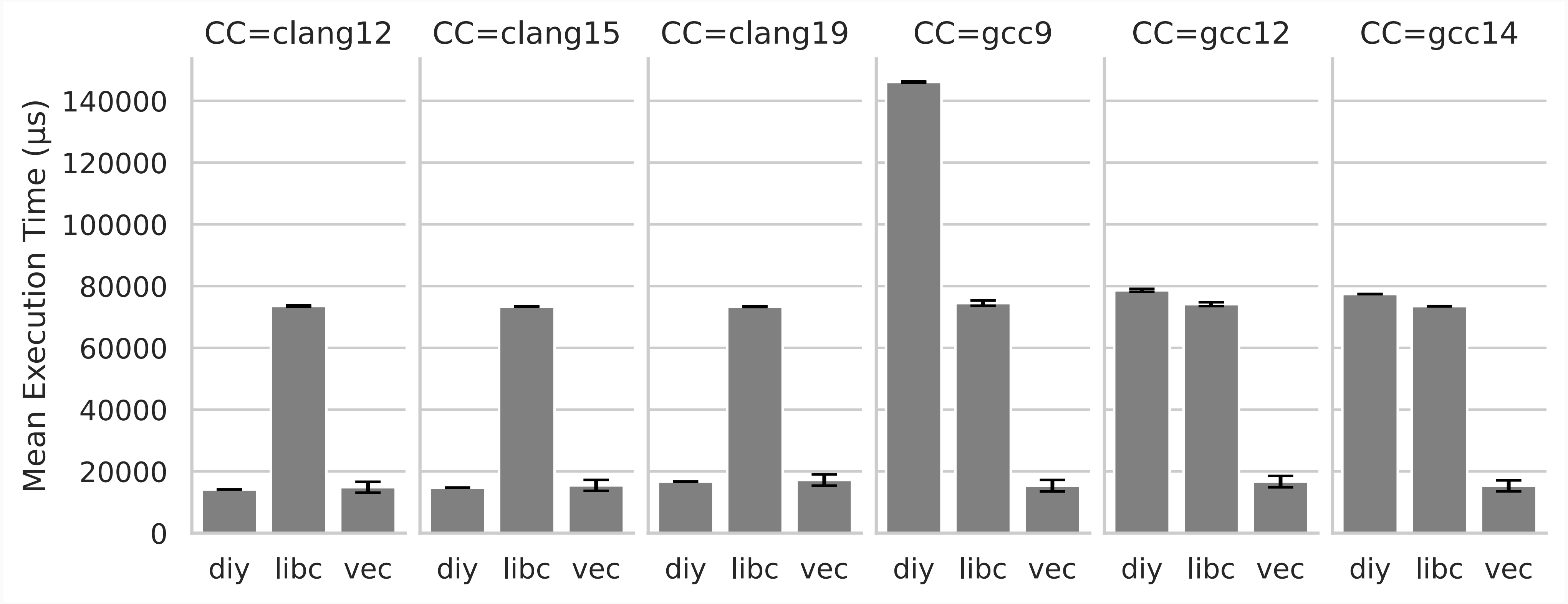
What can go wrong? vague software (algo ≠ implem) — performance



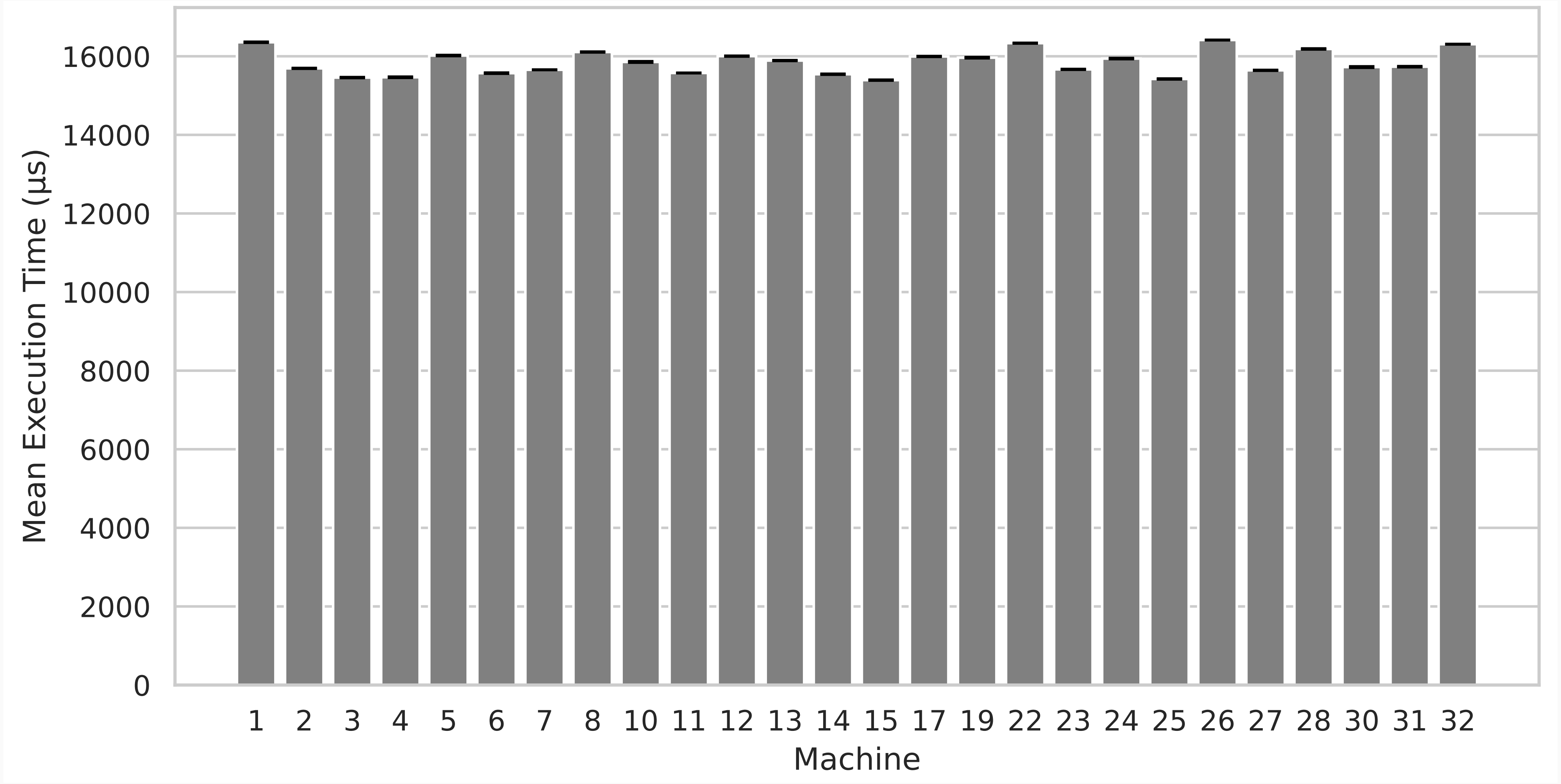
What can go wrong? compilation options



What can go wrong? compilation environment (option=-O2)



What can go wrong? “homogeneous” machines – dahu cluster, vec, gcc14, -O2



What can go wrong? “homogeneous” machines

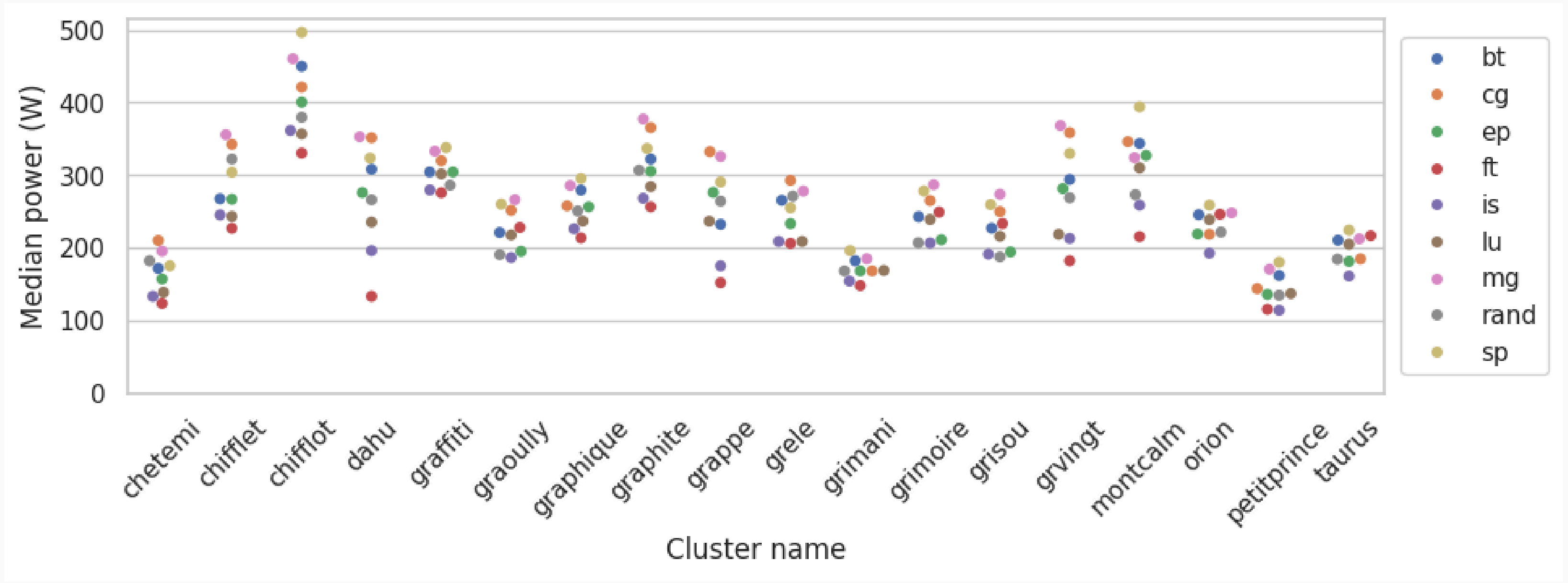
TODO : (

Previous microbenchmark only uses memory and CPU

- CPU have usually similar performance in the same cluster
- That’s not the case of other components – *e.g.*, disks

What can go wrong? different machines

TODO :(



Da Costa, G. Power, performance and system measures of HPC benchmarks on multiple hardware
https://doi.org/10.5281/zenodo.10982239

What can go wrong? filesystem

TODO `:(`

Preliminary run (no repetition) with commands such as this one:

```
fio --randrepeat=1 --ioengine=libaio --direct=0 --gtod_reduce=1 \
    --name=test --numjobs=32 --bs=4k --iodepth=64 --readwrite=randrw \
    --rwmixread=75 --size=256M --filename=/tmp/tmpfs/testfile
```

Gave results such as these ones:

storage	fs	read bw (MiB/s)	write bw (MiB/s)
local ssd	`tmpfs`	4759	1586
local ssd	`ext4`	2741	914
local ssd	`overlayfs`	1676	559
remote ?	`nfs`	490	164

What can go wrong? version of <u>your</u> software

TODO `:(`

Example: 1 Batsim commit divided execution time by 2 — remove `printf` in critical path

What can go wrong? version of <u>some</u> software (dep)

TODO :(

Example: external Batsim users used Batsim with SimGrid compiled without any optimization

Problem: SimGrid has a **HUGE** impact on performance

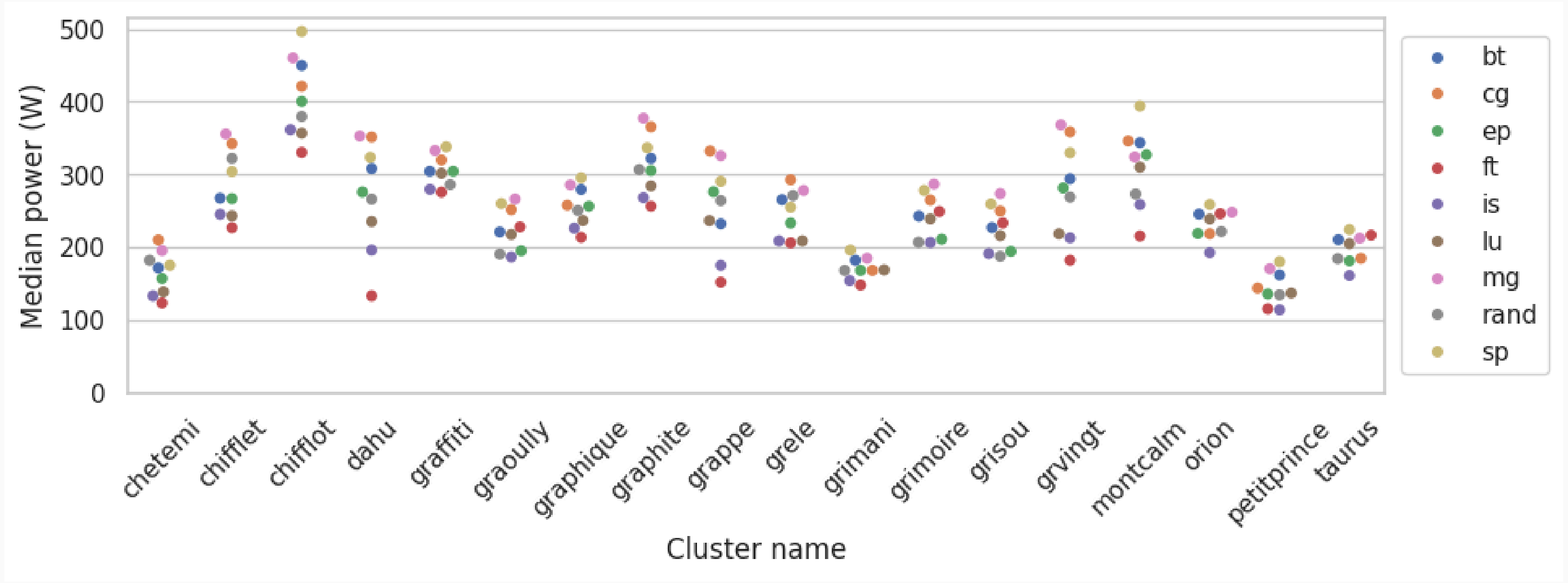
- SimGrid does user-space scheduling
- SimGrid solves hard resource sharing problem for each simulation time step

Execution time of 1 simulation, big picture:

- on my PhD laptop, with optimizations: `< 1 s`
- on external user’s machine, without optimization: timeout after several minutes

What can go wrong? CPU freq changed

TODO :(



Da Costa, G. Power, performance and system measures of HPC benchmarks on multiple hardware https://doi.org/10.5281/zenodo.10982239

What can go wrong? kernel

TODO : (

Idea

- Kernel versions can have impact on (functional) results
- Some kernel options have strong impact on performance (mitigations, scheduling…)

What can go wrong? external resource has changed/disappeared 🙀

Data service

- Input data lost

Source forge

- Google code
- Inria’s gforge
- When will GitHub disappear?

Hardware

- : (

What can go wrong? Summary

Impact?

- Can no longer run anything
- Can run but main outputs differ (binary or statistical diff)
- Can run but non-functional metrics differ (performance, power...)

Why?

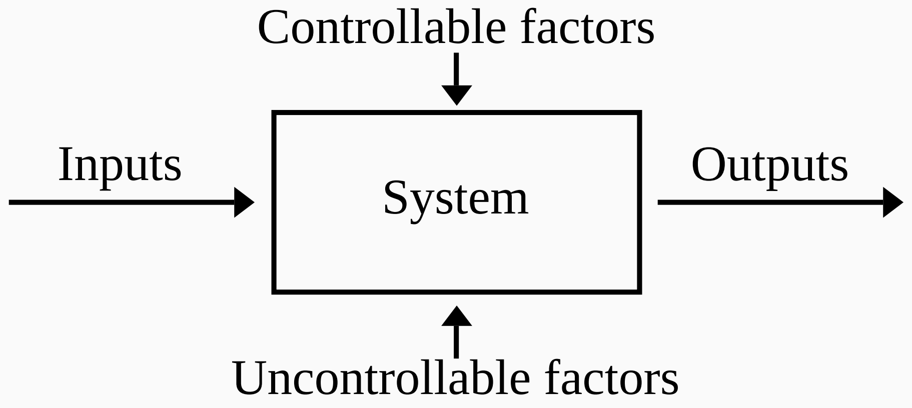
- Unreliable external sources (random service, closed-source stuff...)
- Uncontrolled software
  - Your software
  - Your dependencies
  - OS and various *hidden* services
- Uncontrolled hardware
  - State: voltage/frequency, temperature...
  - (Real) network connections...

Expe & control

Slide from SMPE[1]'s lecture on Design of Experiments

Blackbox model for the system under study

- All inputs may not be known
  - Some can be controlled
  - Some cannot
- All outputs may not be known



Main idea

- Define the set of relevant response
- Determine the set of relevant factors/variables

[1] https://github.com/alegrand/SMPE

What should be controlled? How?

IMO, **as much as is reasonably possible** and is useful for your experiment

- Your software : your code + all your user-space dependencies
- Your software execution environment: how your software is run
- OS config & version
- Hardware setup

Even if you don't control it, try to **carefully record** factors you think relevant

In particular, **timestamps** allow to check many things a posteriori

Enabling control with **mutability** is critical in computer science IMO

- Performance strongly depends on hardware/software environments, that evolve

What should be controlled? How?

IMO, **as much as is reasonably possible** and is useful for your experiment

- Your software : your code + all your user-space dependencies
- Your software execution environment: how your software is run
- OS config & version
- Hardware setup

Even if you don't control it, try to **carefully record** factors you think relevant
In particular, **timestamps** allow to check many things a posteriori

Enabling control with **mutability** is critical in computer science IMO

- Performance strongly depends on hardware/software environments, that evolve

The next sections of this talk give examples on how to control

- Your user-space software via **Nix**
- Some OS config/version & hardware setup via **Grid'5000**

Nix

Nix's main idea

Definition

An operation is said to have a **side effect** if it has any observable effect other than its primary effect of reading the value of its arguments and returning a value to the invoker of the operation.

Nix’s main idea

Definition

An operation is said to have a **side effect** if it has any observable effect other than its primary effect of reading the value of its arguments and returning a value to the invoker of the operation.

How to manage software **without** side effects ?

- **Immutable** file system to store packages
- **Pure** functions to build packages
- **Cryptographic hashes** for external data (source code...)

What is Nix?

Nix: package manager

- Download, store and *install* packages
- Get into tmp well-defined environments (shells)
  - `virtualenv`, but for any language
  - `docker run`, but without isolation

Nix: programming language

- λ DSL
- Define how to build packages
- Define environments (set of packages)

NixOS: Linux distribution

- Declarative system config via Nix language
- Sources: https://github.com/NixOS/nixpkgs

How to store packages?

Filesystem Hierarchy Standard

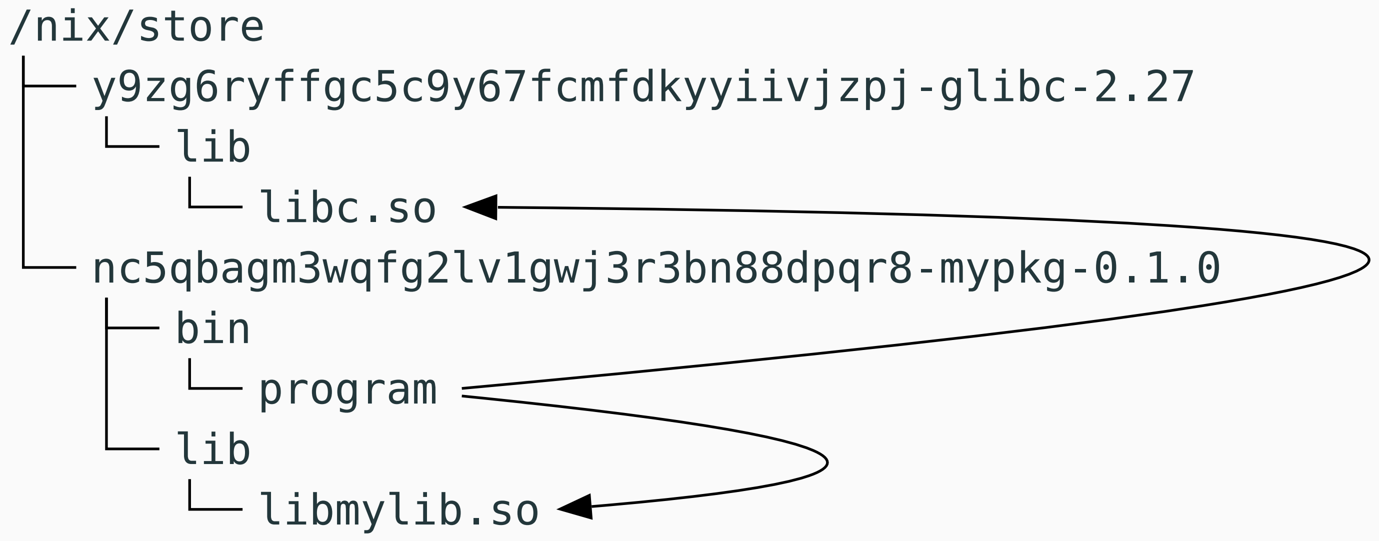
- All packages merged together
- Multiversion is tedious
- Always in the *default* environment
  - ELFs with vague deps – `require libmylib.so`
  - Libs from default paths (`/lib`, `/usr/lib`, or `ldconfig`)
  - PATH to default paths or hacked

```
/usr
├─ bin
│  └─ program
└─ lib
   ├─ libc.so
   └─ libmylib.so
```

How to store packages?

Nix Store

- Each package in its own directory + links
- Naming: hash of inputs + package name + package “version”
- Precise dependencies
  - ELFs have set `DT_RUNPATH`
  - Wrapper scripts to set `PATH`, `PYTHONPATH`…



```
/nix/store
├── y9zg6ryffgc5c9y67fcmfdkyyiivjzpj-glibc-2.27
│   └── lib
│       └── libc.so
└── nc5qbagm3wqfg2lv1gwj3r3bn88dpqr8-mypkg-0.1.0
    ├── bin
    │   └── program
    └── lib
        └── libmylib.so
```

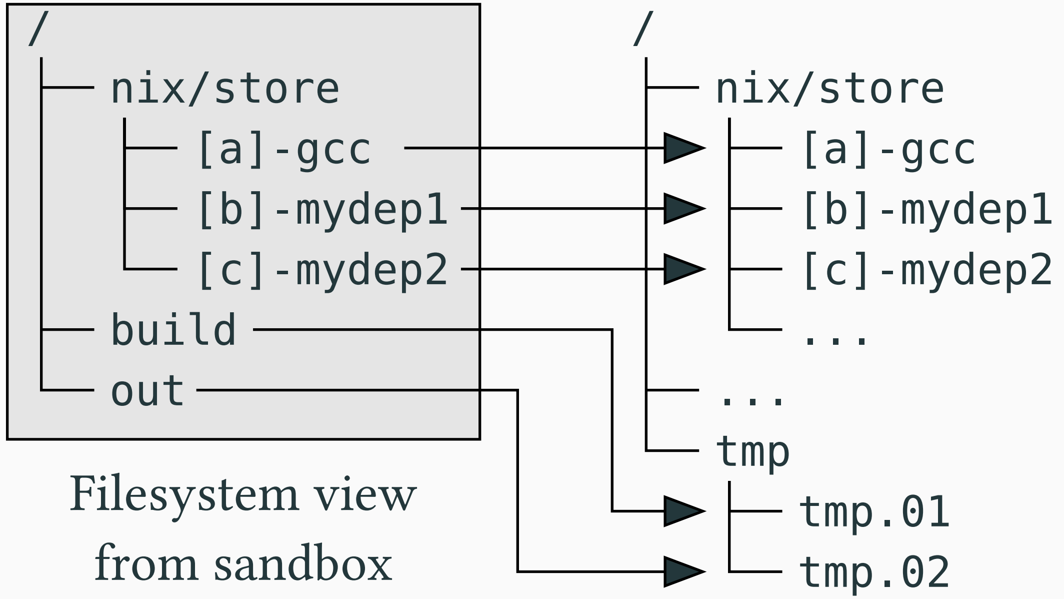
How does Nix achieve purity?

Main ideas

- Content hash used for external data (source code / bootstrap)
- Packages are built in a sandbox (Linux namespaces)
  - Controlled builder args and env vars
  - No filesystem access outside of build script / sources / deps
  - No network/ipc/... access

Workflow to build a package

1. Build all deps
2. Run build in sandbox
   - Input: build script (read-only)
   - Inputs: all deps paths (read-only)
   - Outputs: temp dirs
3. temp dirs $\rightarrow$ Store (DB transaction)



Filesystem view from sandbox

How does Nix achieve purity? derivation tree

TODO : (

Idea: Tree of λ calls

- Leaves: content-hashed external data (sources, bootstrap compilers)
- Intermediate nodes: compilers, your indirect dependencies, your direct dependencies
- Root: your package

Nix λ example – explicit hello world package

```
{ stdenv }:

stdenv.mkDerivation {
  name = "hello";
  src = ./.;

  phases = [ "unpackPhase" "buildPhase" "installPhase" ];
  # default unpackPhase is used
  buildPhase = ''
    gcc -o ./hello ./hello.c
  '';
  installPhase = ''
    mkdir -p $out/bin
    mv ./hello $out/bin/
  '';
}
```

Entry points

1. `prePhases`
2. `unpackPhase`
3. `patchPhase`
4. `(pre)configurePhase`
5. `(pre)buildPhase`
6. `checkPhase`
7. `(pre)installPhase`
8. …
9. `postPhases`

Customized by

- setting `phases`
- using another builder

Nix λ example – real package, using a build system

```
{ stdenv, fetchgit, meson, ninja, pkg-config, boost, gtest }:

stdenv.mkDerivation rec {
  pname = "intervalset";
  version = "1.2.0";
  src = fetchgit {
    url = "https://framagit.org/batsim/intervalset.git";
    rev = "v${version}";
    hash = "sha256-+mG5cPgB+wAxao/8epXWrWcyvYmzmc8Un6At+6U00qs=";
  };
  buildInputs = [ meson ninja pkg-config boost gtest ];
  # configurePhase = "meson build";
  # buildPhase = "meson compile -C build";
  # checkPhase = "meson test -C build";
  # installPhase = "meson install -C build";
}
```

Usual build layers

- Package manager: `nix`, `apt`…
- Build system: `meson`, `cmake`…
- DAG builder: `ninja`, `make`…
- Compiler: `clang`, `gcc`…

Package customization – tune λ args via override

```
packages = rec {
  intervalset = pkgs.callPackage ./intervalset.nix { };
  intervalset-as-debian10 = intervalset.override {
    boost = boost-167;
    meson = meson-049;
  };

  boost-176 = ...;
  boost-167 = ...;
  boost = boost-176;

  meson-058 = ...;
  meson-049 = ...;
  meson = meson-058;
};
```

Package customization – tune λ definition via `overrideAttrs`

```
packages = rec {
  intervalset = pkgs.callPackage ./intervalset.nix { };
  intervalset-110 = intervalset.overrideAttrs (old: rec {
    version = "1.1.0";
    src = pkgs.fetchgit {
      url = "https://framagit.org/batsim/intervalset.git";
      rev = "v${version}";
      hash = "sha256-auMx9J8h3nqNVB4qLcnxVRua4E3jy5bNc2e/REPOek4=";
    };
  });
  intervalset-local = intervalset.overrideAttrs (old: rec {
    version = "local";
    src = "/home/user/projects/intervalset";
    mesonBuildType = "debug";
  });
};
```

Nix code example – shell

```
{ pkgs }:

pkgs.mkShell {
  buildInputs = [
    pkgs.sysbench

    pkgs.curl

    pkgs.python3
    pkgs.python3Packages.numpy
  ];
}
```

```
$ sysbench --version
sh: sysbench: command not found
$ python --version
sh: python: command not found

$ nix-shell ...
(nix-shell) $ sysbench --version
sysbench 1.0.20
(nix-shell) $ python --version
Python 3.12.7
(nix-shell) $ python
Python 3.12.7 (main, Oct  1 2024, 02:05:46) [GCC 13.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import numpy
>>> numpy.__version__
'1.26.4'
>>> exit()
```

Nix code example – container

Big image

```nix
packages = rec {
  hello = pkgs.callPackage ./hello.nix {};
  hello-cont = pkgs.dockerTools.buildImage {
    name = "hello";
    tag = "latest";
    created = "now";
    copyToRoot = pkgs.buildEnv {
      name = "image-root";
      paths = [ hello ];
      pathsToLink = [ "/bin" ];
    };
    config.Cmd = [ "/bin/hello" ];
  };
};
```

2 layers

```nix
{ dockerTools, batsim, bash }:
let self = rec {
  layer-dependencies = dockerTools.buildImage {
    name = "oarteam/batsim-deps";
    tag = batsim.version;
    copyToRoot = batsim.runtimeDeps ++ [ bash ];
  };
  layer-batsim = dockerTools.buildImage {
    fromImage = layer-dependencies;
    fromImageName = layer-dependencies.name;
    fromImageTag = layer-dependencies.tag;
    tag = layer-dependencies.tag;
    name = "oarteam/batsim";
    config = {
      # ...
    };
  };
};
in
  self.layer-batsim
```

Where to write your Nix expressions?

Evaluating remote (git, some https server...) Nix expressions is as easy as local ones (files)
→ You are free to decide where to write your Nix files

Common locations

- Git repo of your software
- Git repo of your experiment
- Git repo of a set of tools you (or your team/lab/...) manage
- Nixpkgs

Nix $\longrightarrow$ control software env. Limits?

Reproducible builds only if

- Compiler is deterministic
- Build chain (build system…) is deterministic

Quote from the Meson's doc

Meson aims to support reproducible builds out of the box with zero additional work (assuming the rest of the build environment is set up for reproducibility). If you ever find a case where this is not happening, it is a bug. Please file an issue with as much information as possible and we'll get it fixed.

Contaminant approach – cannot reuse non-Nix packages

- Nixpkgs has much more packages than any other Linux distro[1]
- Writing a Nix package is straightforward most of the time

[1] https://repology.org/repositories/statistics
As of 2025-06-23, nixpkgs=106475, AUR=78079, debian12=34451

Grid’5000

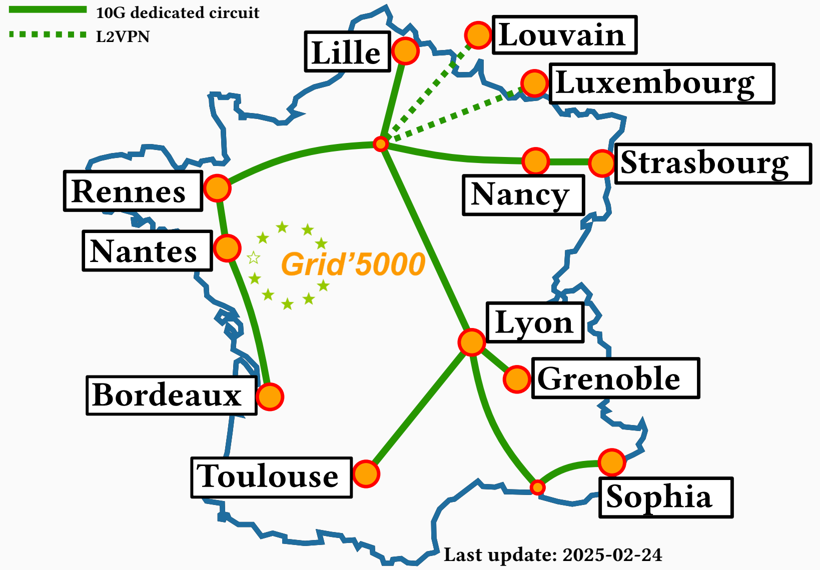
Grid'5000 overview

Testbed for computer science research

- large scale
- flexible
- experiment-driven

Key features

- 800 nodes, 15000 cores
- highly reconfigurable & controllable
- advanced monitoring / measurements
- strongly relies on OSS



Hardware heterogeneity

Some clusters

- `dahu` : 32 nodes – 2x Intel Xeon Gold 6130, 192 GiB, 240 GB SSD, Omni-Path
- `drac` : 12 nodes – 2x IBM POWER8NVL 1.0, ..., InfiniBand, 4x Nvidia Tesla P100
- `estats` : 12 nodes – 1x Nvidia Carmel (`aarch64`), 1x Nvidia AGX Xavier
- `sirius` : 1 node – 2x AMD EPYC 7742, 1.92 TB SSD, 8x Nvidia A100
- `servan` : 2 nodes – 2x AMD EPYC 7352, ..., 2x 100 Gbps FPGA
- `engelbourg` : 8 nodes – 1x Intel Pentium D1517, ..., P4 switch

Environment control

Operating system

- Debian (updated regularly) on most clusters
- Specific OS on some clusters – *e.g.*, Nvidia-customized Ubuntu on `estats`
- **What you want** with `kadeploy`

Build your own system image

- With `kameleon` – imperative approach
- With `NixOS (Compose)` – functional approach
- How you want as long as you provide `kadeploy`-compatible files

Network

- VLAN with `kavlan`
- Real topology is hard to change, but emulation[1] via `netem`/`distem`/`enoslib`

[1] https://www.grid5000.fr/w/Network_emulation

How useful is Grid'5000 to your research?

I cannot tell, I don't know what you work on `:/`

Most likely useful if you

- Work on distributed systems (lots of different configurable nodes)
- Work on operating systems (clean automatic bare-metal deployments)
- Need computing resources to run programs – *e.g.*, simulation campaigns
- Work on performance/power evaluation, if your hw/network needs are not too specific

Accessing Grid'5000

Getting Access

1. Create your account[1]
2. Accept the usage policy[2]
3. Follow the Getting Started tutorial[3]
4. Follow other tutorials[4]
5. Communicate with users or admins[5]

Technically

1. `ssh` on a frontend
2. reserve resources with OAR
3. use resources interactively or not

[1] https://www.grid5000.fr/w/Grid5000:Get_an_account

[2] https://www.grid5000.fr/w/Grid5000:UsagePolicy

[3] https://www.grid5000.fr/w/Getting_Started

[4] https://www.grid5000.fr/w/Users_Home

[5] https://www.grid5000.fr/w/Support

Artifact

Artifact?

From USENIX OSDI'25 Call for Artifacts[1]

A scientific paper consists of a constellation of artifacts that extend beyond the document itself: software, hardware, evaluation data and documentation, raw survey results, mechanized proofs, models, test suites, benchmarks, and so on. ***In some cases, the quality of these artifacts is as important as that of the document itself.***

Forgotten before, encouraged now, **forced soon?**

- SC since 2016: description **mandatory**, evaluation optional
- ACM REP since 2023: AD+AE **mandatory** for full papers with experimental results
- USENIX OSDI'24: 69% of accepted papers participated in the artifact evaluation process

[1] https://www.usenix.org/conference/osdi25/call-for-artifacts

How to do an artifact?

Use cases

- Reviewer/Reader wants to check result analysis, using existing data
- Reviewer/Reader wants to check a result, running a subpart of the experiment
- Researcher wants to reuse parts of the experiment setup

How to do an artifact?

Use cases

- Reviewer/Reader wants to check result analysis, using existing data
- Reviewer/Reader wants to check a result, running a subpart of the experiment
- Researcher wants to reuse parts of the experiment setup

How to document your experiment?

- Big README with shell blocks?
- Bunch of scripts?
- Automate everything with some workflow system?

How to do an artifact?

Use cases

- Reviewer/Reader wants to check result analysis, using existing data
- Reviewer/Reader wants to check a result, running a subpart of the experiment
- Researcher wants to reuse parts of the experiment setup

How to document your experiment?

- Big README with shell blocks?
- Bunch of scripts?
- Automate everything with some workflow system?

The best way to it depends on your experiment, **but**

- Enable reusing parts of the experiments (sofware engineering)
  - ▸ Can be tricky – *e.g.*, custom OS deployment without being Grid'5000 specific
- Provide a high-level documentation
- Enable exploration: Keep things as stupid as possible
- Enable variation: Show how to tune important parameters, software environment...

Artifact example : toy “experiments” for this talk

https://gitlab.irit.fr/sepia-pub/open-science/non-rep-src-irit-phd-days

Artifact example : one we did for Euro-Par'24

Code of experimental setup in Git repo, duplicated for longevity

- Public repo `at irit`
- Public repo `at framagit`
- Long-term archive `on Software Heritage`

Various needed software in their own repos

- All their source codes & Nix descriptions archived on Software Heritage

Measurements & analysis notebooks (more details than what fit in paper)

- `On Zenodo`

Methods

REP hygiene VS patching holes and cracks

Current **REP** crisis is mostly a **methodological** crisis

REP is too often an afterthought of your experiment

- How to trace your scientific decisions?
- How to enable people joining the work along the way easily?

IMO sane REP practice should be used early on in your experiment

- Very similar to software engineering good practices

A fool with a tool…

is still a fool!

Tools such as Nix and Grid'5000 help, but **method** is the key to achieve REP

IMO Computer science is immature and lacks standardized protocols **but**

- You need to learn about most common *bad research* practices[1]
  - Data dredging (p-hacking): Perform many statistical tests on the data and only report those that come back with significant results
  - HARKing: Hypothesizing after the results are known
  - Publication bias towards *positive* results
  - …
- You need to plan your experiments
  - Exploration ≠ Testing your hypotheses
  - Consider €/environmental costs

[1] https://www.polytechnique-insights.com/en/braincamps/society/what-does-it-mean-to-trust-science/science-can-suffer-from-lack-of-reproducibility-of-results/

Ten Simple Rules for Reproducible Computational Research — comp. biology

From Sandve GK, Nekrutenko A, Taylor J, Hovig E (2013)[1]

1. For Every Result, Keep Track of How It Was Produced
2. Avoid Manual Data Manipulation Steps
3. Archive the Exact Versions of All External Programs Used
4. Version Control All Custom Scripts
5. Record All Intermediate Results, When Possible in Standardized Formats
6. For Analyses That Include Randomness, Note Underlying Random Seeds
7. Always Store Raw Data behind Plots
8. Generate Hierarchical Analysis Output,
   Allowing Layers of Increasing Detail to Be Inspected
9. Connect Textual Statements to Underlying Results
10. Provide Public Access to Scripts, Runs, and Results

[1] https://doi.org/10.1371/journal.pcbi.1003285

Best Practices for Scientific Computing — bioinformatics

From Wilson G, Aruliah DA, Brown CT, Chue Hong NP, Davis M, Guy RT, et al. (2014)[1]

1. Write programs for people, not computers.
2. Let the computer do the work.
3. Make incremental changes.
4. Don't repeat yourself (or others).
5. Plan for mistakes.
6. Optimize software only after it works correctly.
7. Document design and purpose, not mechanics.
8. Collaborate.

[1] https://doi.org/10.1371/journal.pbio.1001745

The Practice of Reproducible Research: Case Studies and Lessons from the Data-Intensive Sciences

Quotes from chapter *Lessons Learned*, wrote by *Kathryn Huff*[1]

- *The case studies made clear [...] that humans must be incentivized to spend time on tasks intended solely for reproducibility*
- *a lack of access to restricted data or hardware can hobble the reproducibility efforts of even the most determined scientists*
- *portability of one's workflow is still a challenge for those intent on openness, since packaging — especially installation of dependencies — remains a critical stumbling block to sharing and extending work*

[1] https://www.ucpress.edu/books/the-practice-of-reproducible-research/paper

Conclusion

Conclusion — REP and tools for REP in computer science

Take home message

- Plan your experiments
- Consider REP in your experiment design
- Consider writing artifacts — and evaluating them? :)
- Some tools (Nix, Grid'5000...) can help

To go further

- *Scientific Methodology and Performance Evaluation for Computer Scientists*
  https://github.com/alegrand/SMPE/
- REP MOOCs on https://www.fun-mooc.fr/fr/cours/?query=repro
  - *Recherche reproductible : principes méthodologiques pour une science transparente*
  - *Reproducible Research II: Practices and tools for managing computations and data*
- https://nix-tutorial.gitlabpages.inria.fr/nix-tutorial
- https://www.grid5000.fr