Controlling Your Environment

Cristian Ruiz, Michael Mercier
INRIA - France

April 5, 2016 – Reproducible Research Webinar

People involved in preparing this talk

- Michael Mercier (Inria/Atos)
- Cristian Ruiz (Inria)

Grid5000, Kameleon, Expo, ...

Thanks for the feedback of:

- Pierre Neyron (CNRS)
- Arnaud Legrand (CNRS)
- Olivier Richard (UGA)
- Lucas Nussbaum (Loria)

Here is the pad for interactions: http://tinyurl.com/RRW-pad2

Material (demo, slides) available on github

1. A Docker Demo
2. A complete use case

Outline

Motivations

Reproducible research: What does it mean? Watch the first webinar if you need a reminder.

Definition

A way to encapsulate all aspects of our in silico analysis in a manner that would facilitate independent replication by another scientist

Reproducibility is a cornerstone of scientific method

Is code sufficient?

> *An article about computational science in a scientific publication is not the scholarship itself, it is merely advertising of the scholarship. The actual scholarship is the complete software development environment and the complete set of instruction which generated the figures.*
>
> – *David Donoho, 1998*

- Making it available is a great first step
- Making sure others can rerun it is an other move

Problem statement

Experiment replication is not an easy task if you do not have it in mind from the beginning:

> *The path from having a piece of software running on the programmer's own machine to getting it running on someone else's machine is fraught with potential pitfalls*

Philip J. Guo and Dawson Engler, *CDE: Using System Call Interposition to Automatically Create Portable Software Packages*, USENIX LISA Conference,2011

In reproducible research, scientists should care about both the experiments and the analysis:

- All the artifacts (input/outputs files)
- The source code
- Documentation on how to compile, install and run

Still, several problems may prevent someone to rerun an experiment

Dependencies and compilation problems

Unresolved dependencies

```
File  Edit  View  Terminal  Help
ihaveapc@ihaveapc-desktop ~ $ sudo apt-get install gnochm
[sudo] password for ihaveapc:
Reading package lists... Done
Building dependency tree
Reading state information... Done
Some packages could not be installed. This may mean that you have
requested an impossible situation or if you are using the unstable
distribution that some required packages have not yet been created
or been moved out of Incoming.
The following information may help to resolve the situation:

The following packages have unmet dependencies:
  gnochm: Depends: python-gtkhtml2 but it is not installable
E: Broken packages
```

Compilation errors

```
/usr/include/features.h:324:26: fatal error: bits/predefs.h: No such file or directory
compilation terminated.
make[6]: *** [_muldi3.o] Error 1
make[6]: Leaving directory `/home/student1/Downloads/builds/taco9/PopC13/open-stream/gcc/build/x86_64
make[5]: *** [multi-do] Error 1
make[5]: Leaving directory `/home/student1/Downloads/builds/taco9/PopC13/open-stream/gcc/build/x86_64
make[4]: *** [all-multi] Error 2
make[4]: Leaving directory `/home/student1/Downloads/builds/taco9/PopC13/open-stream/gcc/build/x86_64
make[3]: *** [all-target-libgcc] Error 2
make[3]: Leaving directory `/home/student1/Downloads/builds/taco9/PopC13/open-stream/gcc/build'
make[2]: *** [all] Error 2
make[2]: Leaving directory `/home/student1/Downloads/builds/taco9/PopC13/open-stream/gcc/build'
make[1]: *** [/home/student1/Downloads/builds/taco9/PopC13/open-stream/install/bin/gcc] Error 2
make[1]: Leaving directory `/home/student1/Downloads/builds/taco9/PopC13/open-stream'
make: *** [all] Error 2
```

Collberg, Christian *et Al.*, *Measuring Reproducibility in Computer Systems Research*, http://reproducibility.cs.arizona.edu/ 2014,2015

Less than 50% of experimental setups of papers submitted ACM conferences and journals could be built

Other technical issues

Portability issues E.g., BOINC had to rely on homogeneous redundancy to protect against numerical instabilities (OS, hardware, ...).

Imprecise documentation "*I have no clue about how to install it, configure it or run it!*"

Dependency Hell "*I can't install this dependency package without breaking my entire system*"

Code rote "*This dependency package version is buggy! What was the version that was used to run the experiment in the first place?!?*"

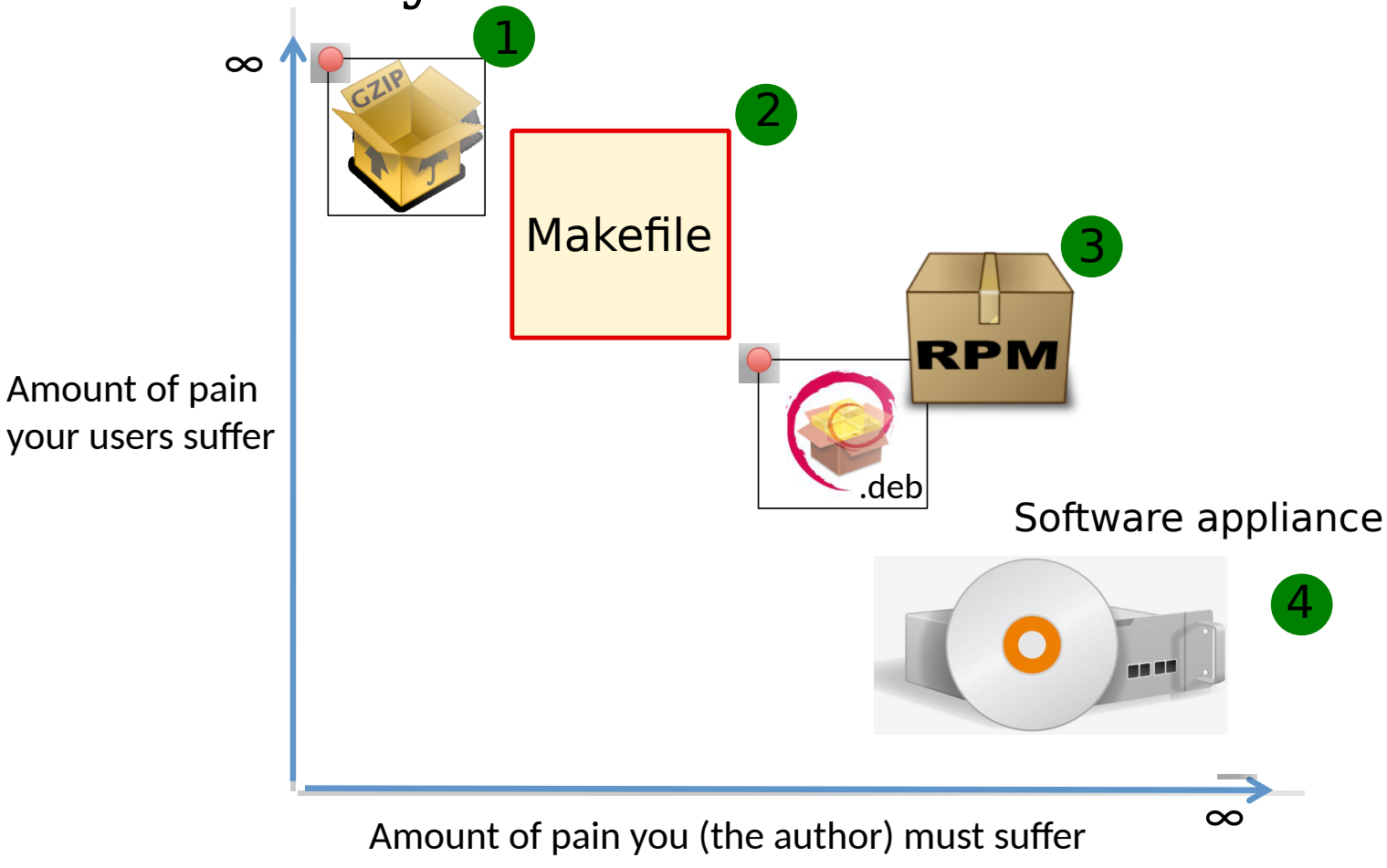
Carl Boettiger, *An introduction to Docker for reproducible research*, ACM SIGOPS Operating Systems Review,2015

Cultural challenges

- Efforts are not rewarded by the current academic research and funding environment
- Software vendors tend to protect their markets through proprietary formats and interfaces
- Investigators naturally tend to want to own and control their research tools
- Even the most generalized software will not be able to meet the specific needs of every researcher in a field
- The need to derive and publish results as quickly as possible precludes the often slower standards-based development path

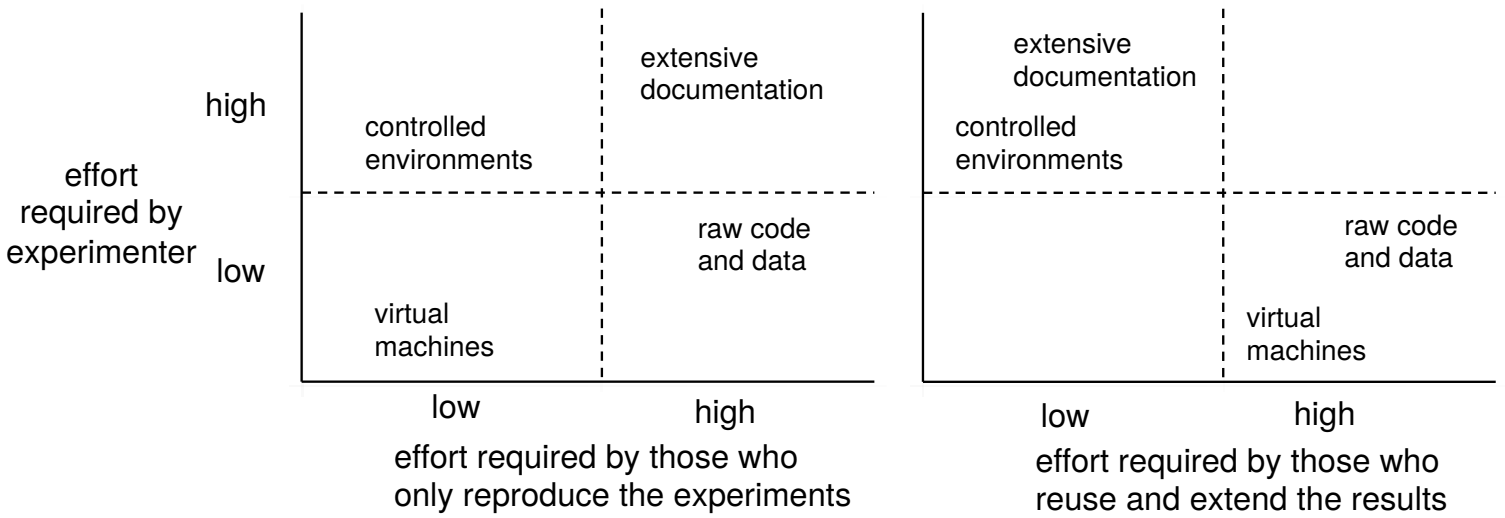
J. T. Dudley and A. J. Butte, *In silico research in the era of cloud computing*, Nature Biotechnology, 2010

Current ways to distribute research code



Courtesy of Philip Guo (AMP Workshop on Reproducible research)

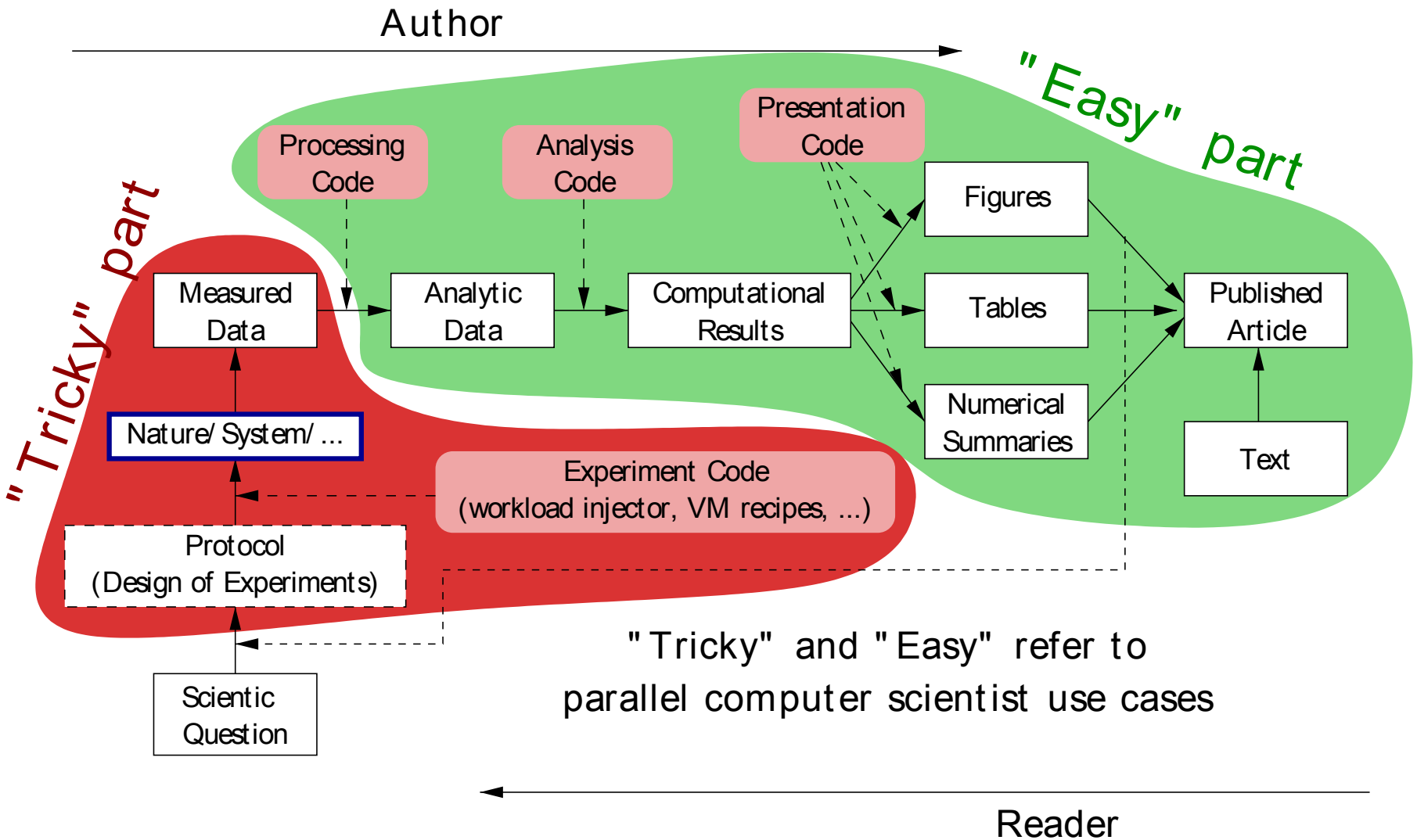
Disseminating science software



Reproducibility, Virtual Appliances, and Cloud Computing, Bill How, University of Washington

- Raw data and code: rely on the published paper for documentation
- Extensive documentation: it may still require certain skills
- Adopt a controlled environment: e.g., rely on a scientific workflow
- Use virtual machines to capture and publish code, data and experimental environment

Everywhere there is code, you need an environment



Why should I take care of my experiment environment?

For myself:

- Be able to reproduce my own experiment later
- Improve my productivity (when preparing articles, PhD, rebuttals, ...)
- Be able to scale my experiment on other machines
- Facilitate experiment extensions and modifications
- Be a better scientist by doing better science 😉

For other people: my students, my colleagues, my peers, ...

- Allow them to reproduce my experiment and corroborate (or not) my results
- Allow them to base their research on my research and extend

For everyone else:

- Improve knowledge sharing
- Increase collaboration possibilities
- Do better science!

Controlling your environment

One way to go is to take care of your experimental environment
There are mainly two approaches:

- Preserving the mess by capturing the already set up environment
- Encourage cleanliness with several options:
  - Using a constrained environment
  - Building your own environment

  See Preserve the Mess or Encourage Cleanliness? (Thain et al., 2015)

Constraint for simplicity, complexity for freedom

Each of them have different levels of constraint and flexibility:

- The more constrained your environment is, the more simple it is
- Freedom comes with responsibility

Outline

Environment definition

Definition (in our case)

An environment is a set of tools and materials that permits a complete reproducibility of a part or of the whole experiment process.

Can be numerous or unique depending on the experiment workflow:

- Experiment environments
  - local, on a testbed, on a dedicated server,...
- Analysis environments
  - Usually a unique local environment

The whole environment contains both hardware and software information

Hardware

Necessary when we carry out performance measures
Tools to capture hardware configuration:

- `dmidecode`
- hwloc (`lstopo`)
- ls* tools (lsblk, lshw, lspci, lsmod,...)
- proprietary tools (bios, nvidia,...)
- Testbeds hardware description API (Grid'5000, Chameleon)

The hardware is not shareable

As it is not shareable the hardware environment needs to be documented as exhaustively as possible.
Of course it depends on how the results of experiments are affected by the underlying hardware.

Software

Different approaches:

Very succinct (usually what is provided, if provided...)

- minimal description in a mail
- README in a git repository
- small documentation

Partial

- bundle of the experiment tool and it dependencies
- linux container image

Full A complete environment backup with the operating system included

- Virtual machine
- A complete system image

Virtual environments: important notions

The role of a virtual environment is to provide some isolation within the host

- A virtual environment can only use a limited part of the resources:
  - filesystem
  - memory/cpu/disk/network
- Has his own software stack $\Rightarrow$ clean dependencies

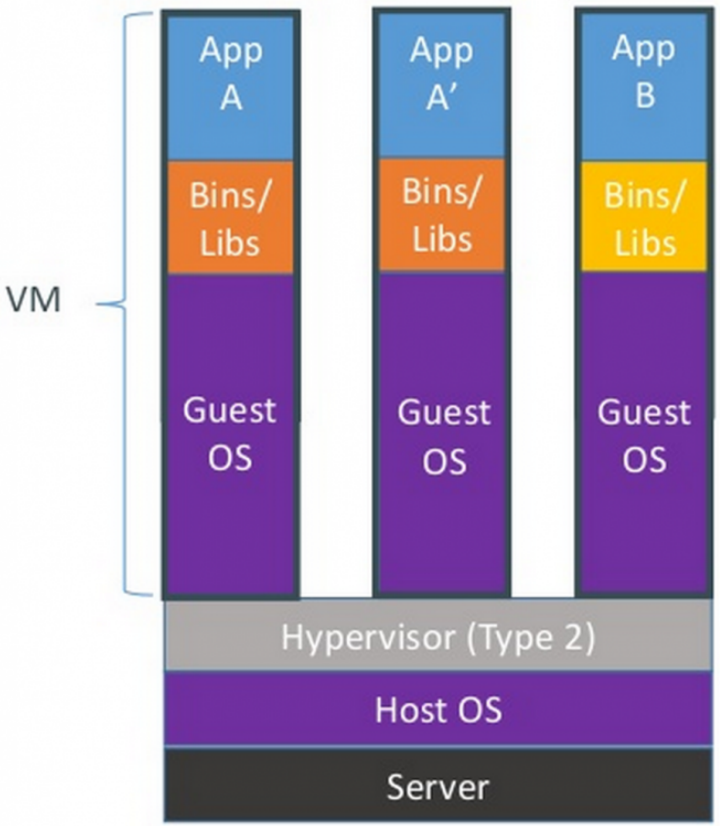
By the way:

What is a container? An isolated part of the system that shares the operating system kernel

What is a virtual machine? A full system image that shares the system hardware with your guest OS though an hypervisor

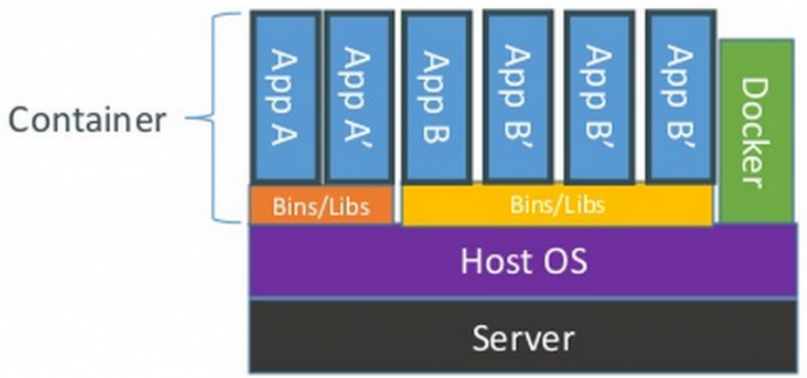
VM vs container

Containers vs. VMs

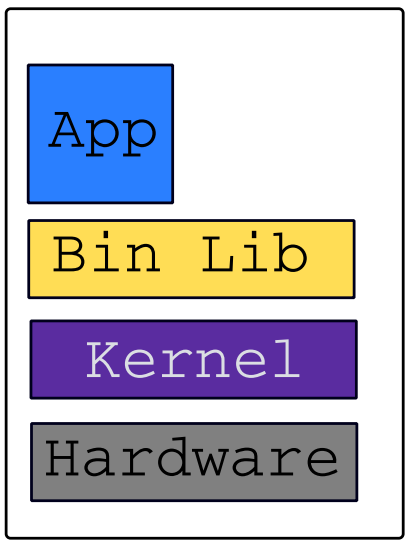


Containers are isolated, but share OS and, where appropriate, bins/libraries

...result is significantly faster deployment, much less overhead, easier migration, faster restart



Types of environments



VM
Image disk + Testbed

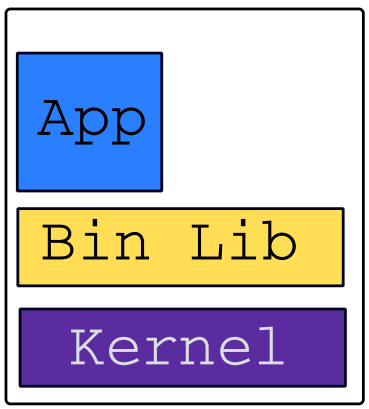
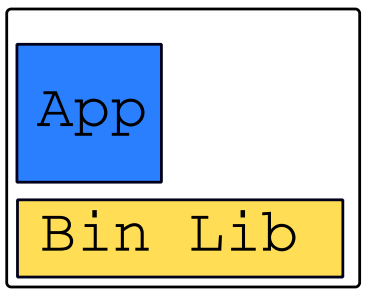
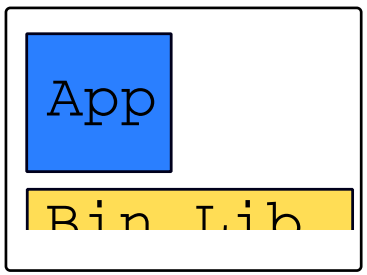


Image disk
QCOW disk
VDI



Tarball for container
LXC and docker



CDE
Reprozip

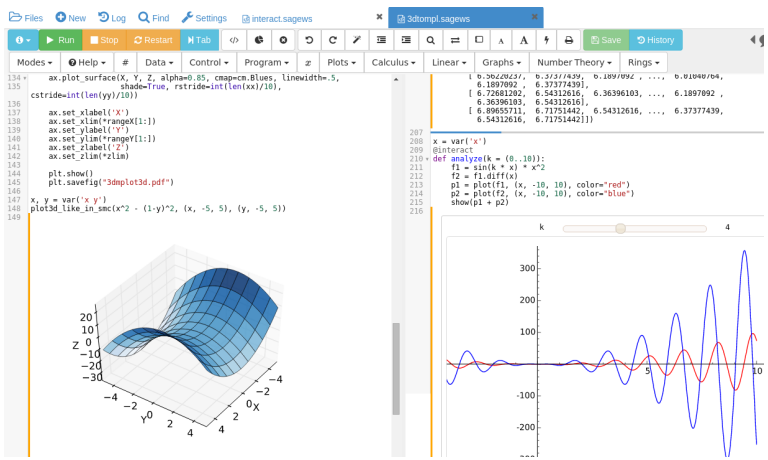
Outline

Use of third party environments

Environment build, specialized, controlled, versioned by somebody else:

- Activepapers (Beta)
  - Python or JVM based language
- SageMathCloud
  - Use Jupyter
    - Julia, Python, R, Haskell, Ruby...
    - 40 languages (partly) supported

Sharing is easy but you have to stick to what the environment provides



Use a controlled environment as a base

Start your experimental setup in a controlled environment from the beginning

- Clean install system in a virtual environment
- Default Testbed (Grid'5000, Cloudlab, Chameleon) environments
- Software appliances market place (e.g., TURNKEY[1], Cloud Market[2])

This encourage cleanliness:
Your environment is controlled (you start from a clean system)

Drawbacks

Nothing is responsible for tracking the modifications applied in this environment
You don't know what is inside the box ☹

[1] http://www.turnkeylinux.org
[2] http://www.thecloudmarket.com

Outline

Capturing an environment

Several approaches for capturing your environment:

- Export everything
  - Kernel + Libraries + Application
  - Heavy but safe
- Capture only what is needed to run on a similar system
  - Libraries (only dependencies) + App
  - Lightweight but can be partial

Copying your experiment environment

A simple capture of an environment is a complete copy of it.

It depends on what your environment is:

1. On a classical local machine:
   - Problem: A simple backup bundle is not easily usable by others
   - Partial solution: Clone your hard drive to a VM (excluding personal data)
2. On a VM or any Copy-on-write environment use the instant snapshot capability
   - Faster and simpler backup
   - VM need to be used from the beginning (mentioned previously)
3. On a testbed machine use the provided snapshot mechanism

In either case sharing is complicated

- Huge environment images of several Gigabytes are common
- Need a dedicated place to store them (a repository or some market place)

You still don't know what is inside the box 

J. T. Dudley and A. J. Butte, *In silico research in the era of cloud computing*, Nature Biotechnology 2010

Capture only what is needed

Use a tracking tool to capture only what is necessary

- Instrumenting a run of your experiment to catch every used material
  - Binaries/Scripts (experiment.py, Python 2.7)
  - Configuration files (conf.yaml)
  - Libraries (libc, numpy, matplotlib)

  Then create a compressed bundle
- Rerun the experiment on another machine:
  1. Import the provided bundle
  2. Initialize the environment (depends on the tools...)
  3. Rerun the exact same experiment

Capture is not foolproof:

- Running with only one set of parameters is not enough
- More risk to miss something ☹

Less messy than virtual environment copy ☺ but it is not easy to modify it to extend an experiment ☹

Capture tools

Existing tools:

- CDE (Guo et al., 2011)
  - First to bring the idea
  - Seems not maintained since 2013
- ReproZip (Freire et al., 2013)
  - One tool to trace and pack
  - Several tools to unpack and run (install package, chroot, docker, vagrant)
  - More during the demo
- CARE (Janin et al., 2014)
  - Only for experts
  - Seems unmaintained since 2014
- Parrot
  - Limited to the Parrot filesystem...

Outline

Environment generation (some facts)

- If you're moving a computation to a new system, it should be simple and straightforward to set up the environment almost identical to that of the original machine
- A major challenge in reproducing computations is installing the prerequisite software environment ☹
- Modern open computational science relies on complex software stacks
- So, it is necessary to know:
  - How was it built?
  - What does it contains?
  - How can I modify it to extend the experiment?

How is software installed and configured?

Source code compilation

```
$ tar -xzf pdt-3.19.tar.gz && cd pdtoolkit-3.19/
$./configure -prefix=/usr/local/pdt-install
$ make clean install
```

- Need to install all dependencies by hand
- Some skills are required

Package manager

A PM is a collection of software tools that automates the process of installing, upgrading, configuring, and removing computer programs for a computer's operating system in a consistent manner

- Examples in the Linux world: APT, yum, pacman, Nix ...
- There also exists package managers for programming languages: Bundler, CPAN, CRAN, EasyInstall, Go Get, Maven, pip, RubyGems, ...

Outline

The DevOps Approach

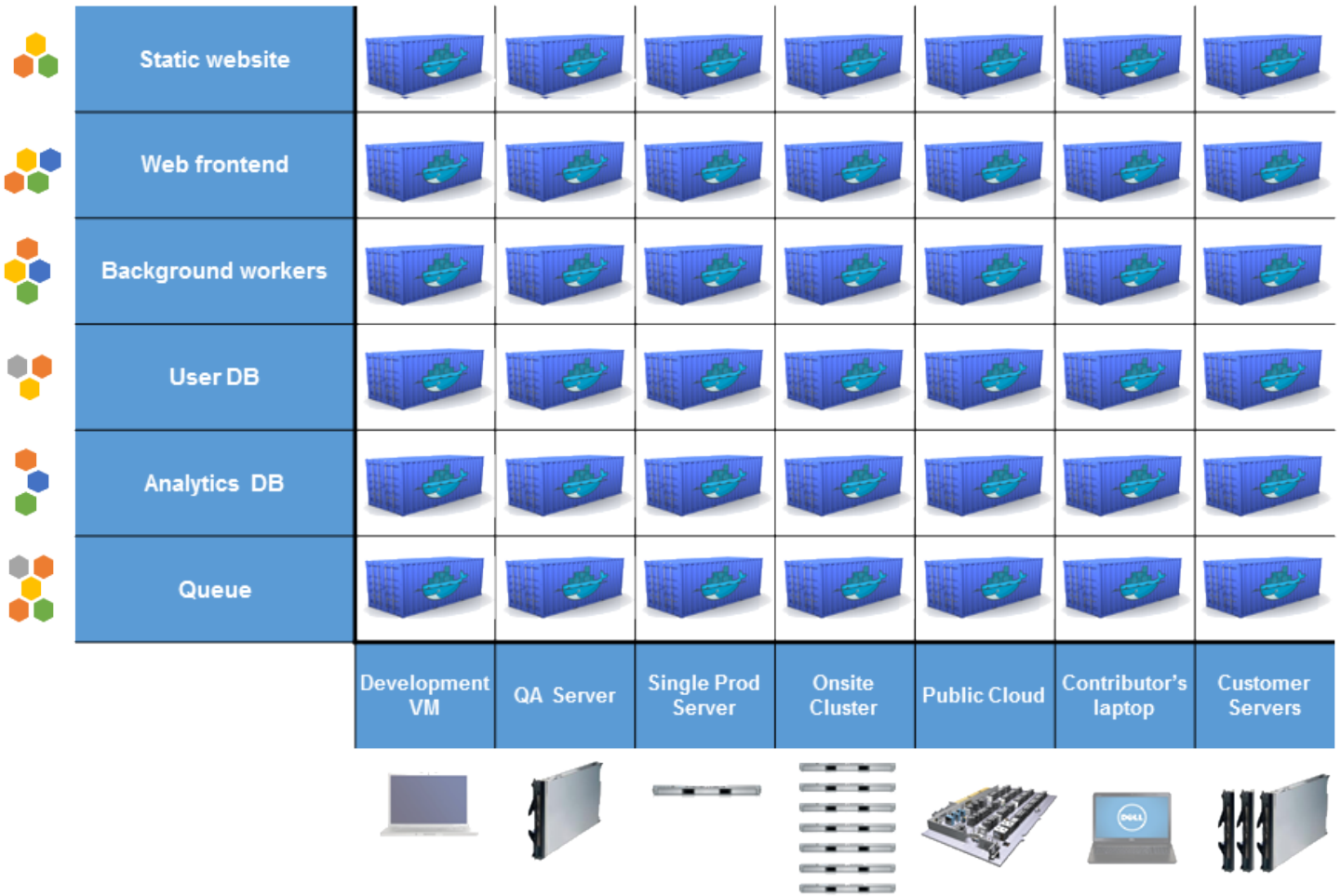
- Dev = Development, Ops= (System) operation
- You have a pile of crusty code that's hard to install
- And documenting how to install it is almost as hard! 
- Why not develop scripts that reliably install your toolset?
  - Because that sounds hard ? 
  - But it's more fun than writing documentation!

Use all the good things that software engineering has created along decades for ensuring isolation and reproducibility

Creating recipes: text based description

- README
- Shell scripts
- Configuration management tools: automate software configuration and installation
  - Software stacks can be easily transportable
  - Some CM tools: Puppet, Salt, Ansible
  - A lot of work has to be done to write recipes 

DevOps response: Docker for deployment

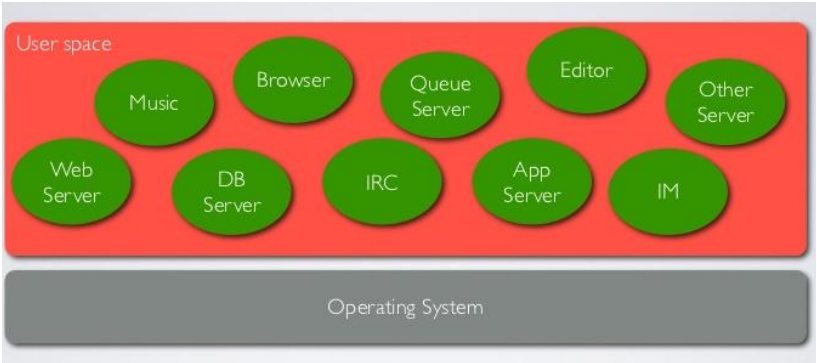


Any application can be easily moved through different environments

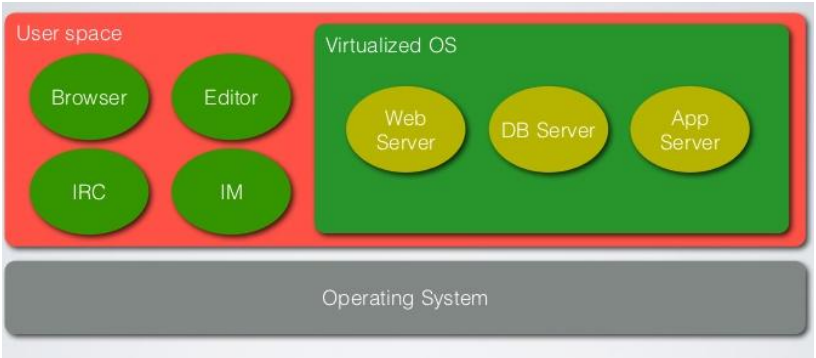
DevOps response: Docker for deployment

- Docker is an open-source engine that automates the deployment of any application as a lightweight, portable, self-sufficient container that will run virtually anywhere
- Docker tries to achieve deterministic builds by isolating your service, building it from a snapshotted OS and running imperative steps on top of it
- Dependency hell: Docker works with images that consume minimal disk space, are versioned, archivable, and shareable (DockerHub)
- Dockerfiles: resolving imprecise documentation

DevOps response: Vagrant for building



No isolation
No repeatability
No verification



Isolation
repeatability
Faster onboarding

- It automates the build of development environment using a base environment called box and a series of text-based instructions

DevOps response: Vagrant for building

- Researchers write text-based configuration files that provide instruction to build virtual machines
- Somehow solves way the problem of sharing a VM. Since these files are small, researchers can easily share them and track different versions via source-control repositories
- VMs are not seen as black boxes anymore
- Researchers can automate the process of building and configuring virtual machines
- It is possible to use different providers: EC2, Virtualbox, VMware, Docker, etc ...

Outline

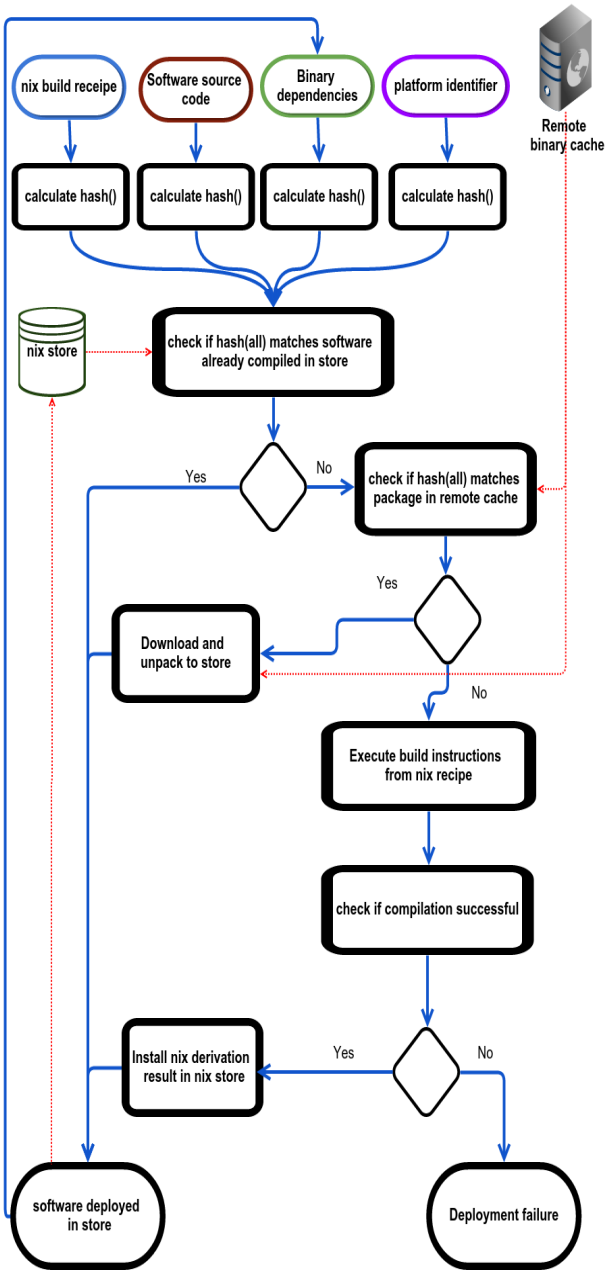
Reproducible builds: a functional package management (Nix)

- Apply functional model to packaging

  A package is the output of a function that is deterministic (it depends only on a function inputs, without any side effects)

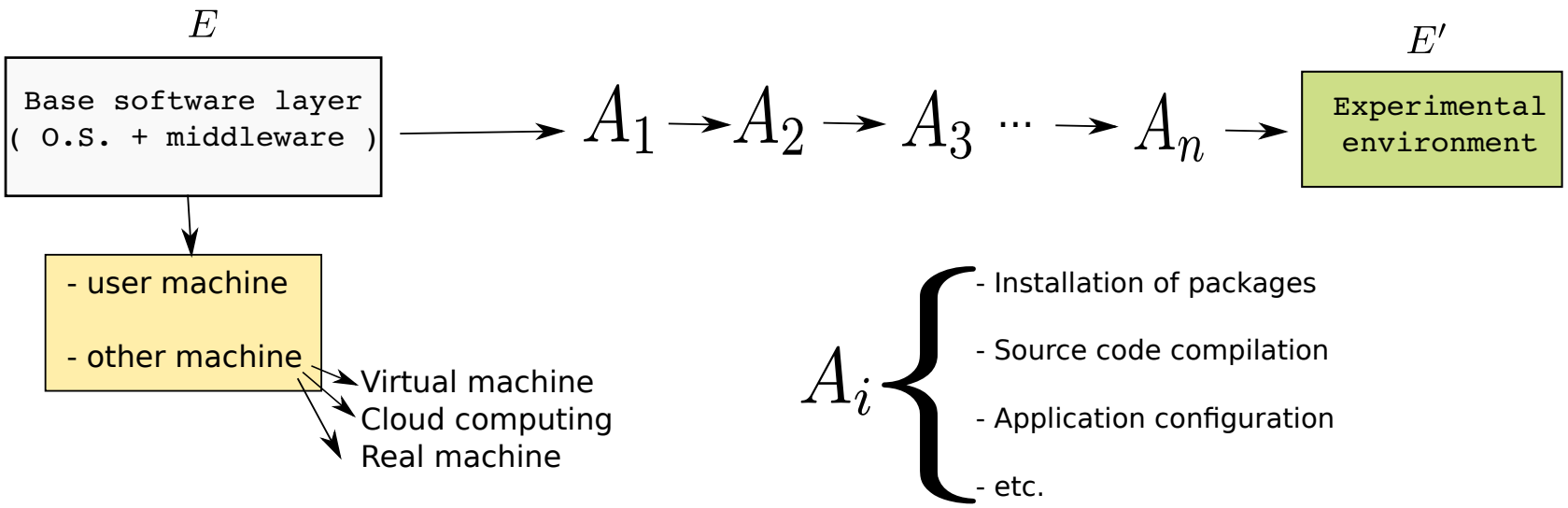
- The principle: two independent runs of a given build process for a given set of inputs should return the same value
- Functional hash-based immutable package management
- Isolated build
- Deterministic
- No dependency hell

Reproducible builds: Nix workflow



Devresse, Adrien et Al., Nix based fully automated workflo and ecosystem to guarantee scientific result reproducibilit across software environments and systems, 2015

Environment generation



Reconstrucability

An experimental setup E' is reconstructable if the following three facts hold:

1. Experimenters have access to the original base experimental setup E.
2. Experimenters know exactly the sequence of actions $\langle A_1, A_2, A_3, ..., A_n \rangle$ that produced E'.
3. **Experimenters are able to change some action A_i and successfully re-construct an experimental setup E''**

Ruiz, Cristian et Al., *Reconstructable Software Appliances with Kameleon* ACM SIGOPS Operating Systems Review,2015

Reconstrucability

It can be expressed as $E' = f(E, \langle A_i \rangle)$ where f applies $\langle A_i \rangle$ to E to derive the experimental setup E'.
Few cases where this hypothesis does not hold:

- An action A_i is composed of sub-tasks that are executed concurrently making the process not deterministic. For example: `Makefile -j`
- packages are validated based on timestamps (Debian 8)
- The compiler may purposedly non-deterministic (e.g., *stack-smashing protector* canaries)
- Leaked information from the host: `hostname`, `/proc/cpuinfo`

Additional problems:

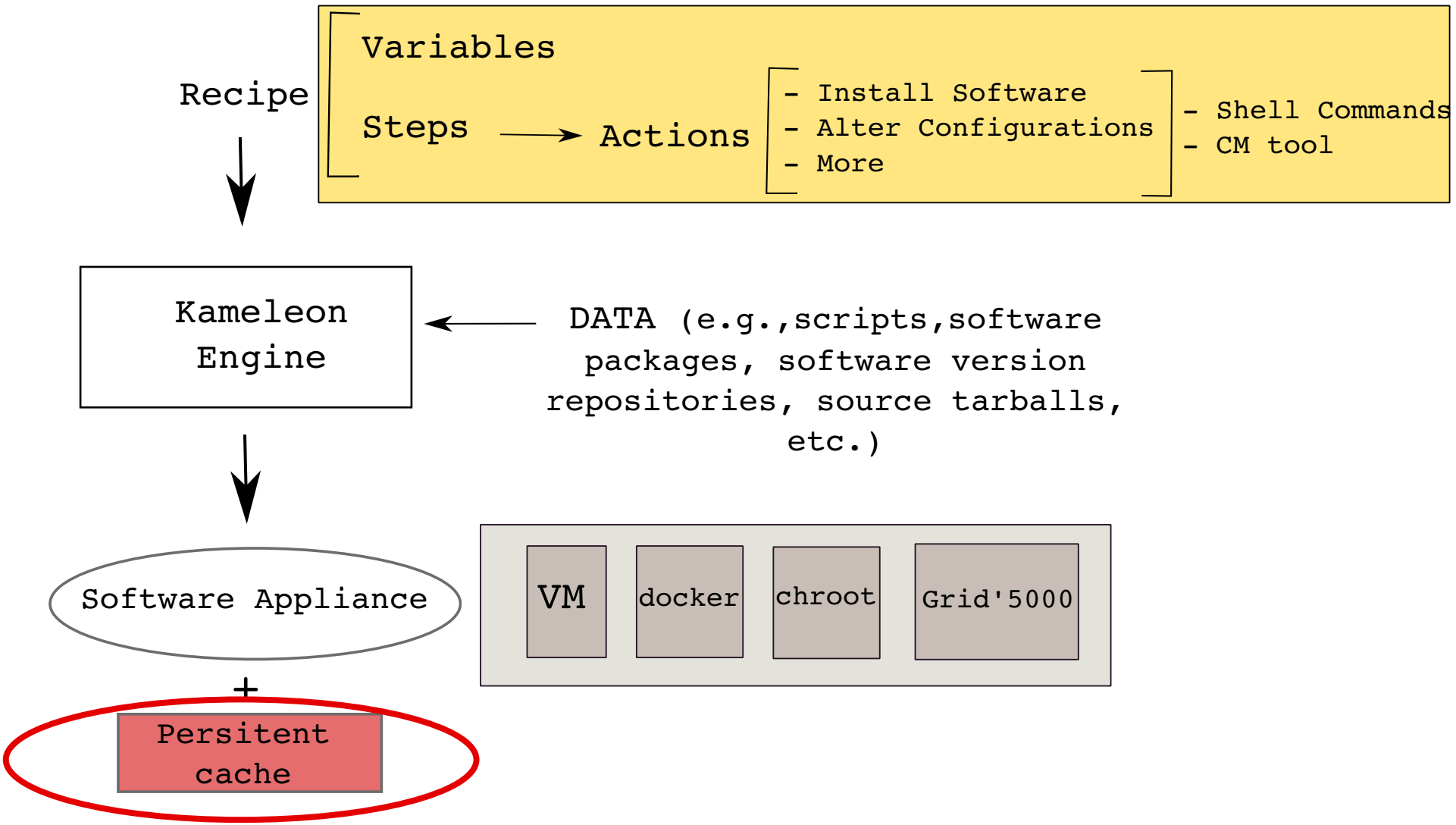
- Accessing the same base setup E
- **Software used is not available anymore**

Dealing with software availability (Debian Snapshot)

The Debian community is quite active on the reproducibility front.

- It's an archive that allows to access old packages based on dates and version numbers
- It provides a valuable resource for tracking down when regressions were introduced, or for providing a specific environment that a particular application may require to run
- Only concerns software that is packaged ☹

Kameleon: Reconstructable Appliance Generator



Kameleon Features

- Easy to use $\rightsquigarrow$ structured language based on few constructs and which relies on shell commands
- Allows shareability thanks to the hierarchical structure of recipes and the extend mechanism
- Kameleon supports the build process by providing debugging mechanisms such as interactive shell sessions, break-points and checkpointing
- Allows the easy integration of providers using the same language for the recipes
- Persistent cache makes reconstructability a reality

Outline

Docker

It's time for a Docker Demo (follow the links from https://github.com/alegrand/RR_webinars/)
Docker advantages for reproducible research:

- Integrating into local development environments
- Modular reuse
- Portable environments
- Public repositories for sharing
- Versioning

Carl Boettiger, *An introduction to Docker for reproducible research*, ACM SIGOPS Operating Systems Review,2015

Docker advantages

- Portable computation & sharing

```
$ docker export container-name > container.tar
$ docker push username/r-recommended
```

- Re-usable modules

```
$ docker run -d --name db training/postgres
$ docker run -d -P --link db:bd training/webapp \
   python app.py
```

- Versioning

```
$ docker history r-base
$ docker tag  d7e5801bb7ac ttimbers/mmp-dyf-skat:latest
```

A complete use case: Batsim

Let's demo a complete use case (follow the links from https://github.com/alegrand/RR_webinars/).

Use case: A *not that simple* simulation

- SimGrid (C library) + BatSim (C++) + OAR scheduler (Python) + A new scheduler (perl)
- Python script for the glue

Steps:

- Build an environment with Kameleon
- Capture an experiment with ReproZip
- Export the corresponding bundle
- Rerun the experiment on another machine (ReproZip + Docker)
- Compare the results (csv + python $\rightsquigarrow$ graphics) still using the environment

Outline

Conclusion

Reproducibility is easier when you have it in mind from the beginning

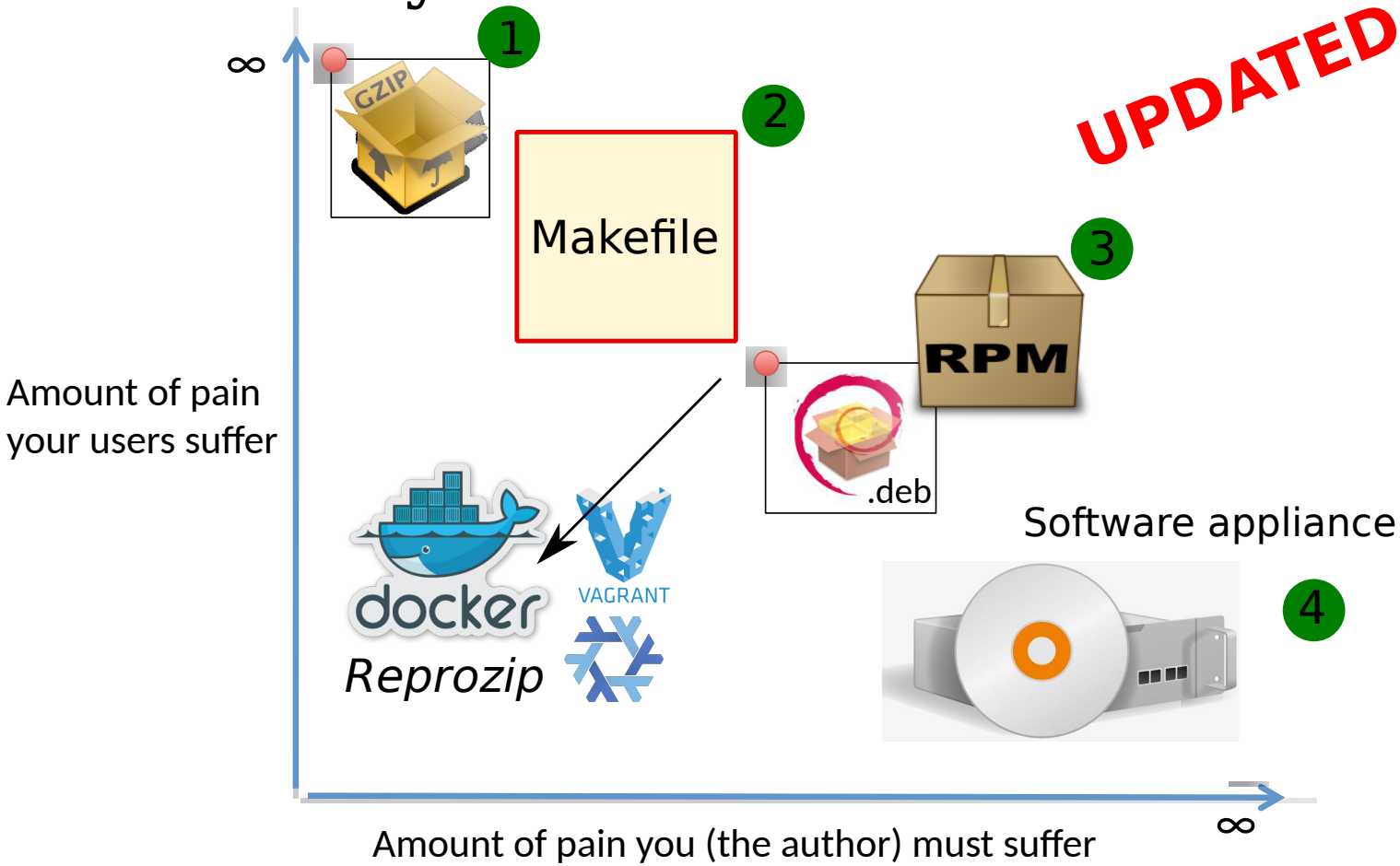
Choose your tools Reproducibility brings some complexity but more and more tools to manage this complexity for you

Provide environments Whatever the environment quality you provide, it is better than no environment at all 😉

Better if you provide the recipe Providing experiment environment is good. Providing the recipe to build this environment is better!

Conclusion

Current ways to distribute research code



Courtesy of Philip Guo (AMP Workshop on Reproducible research)