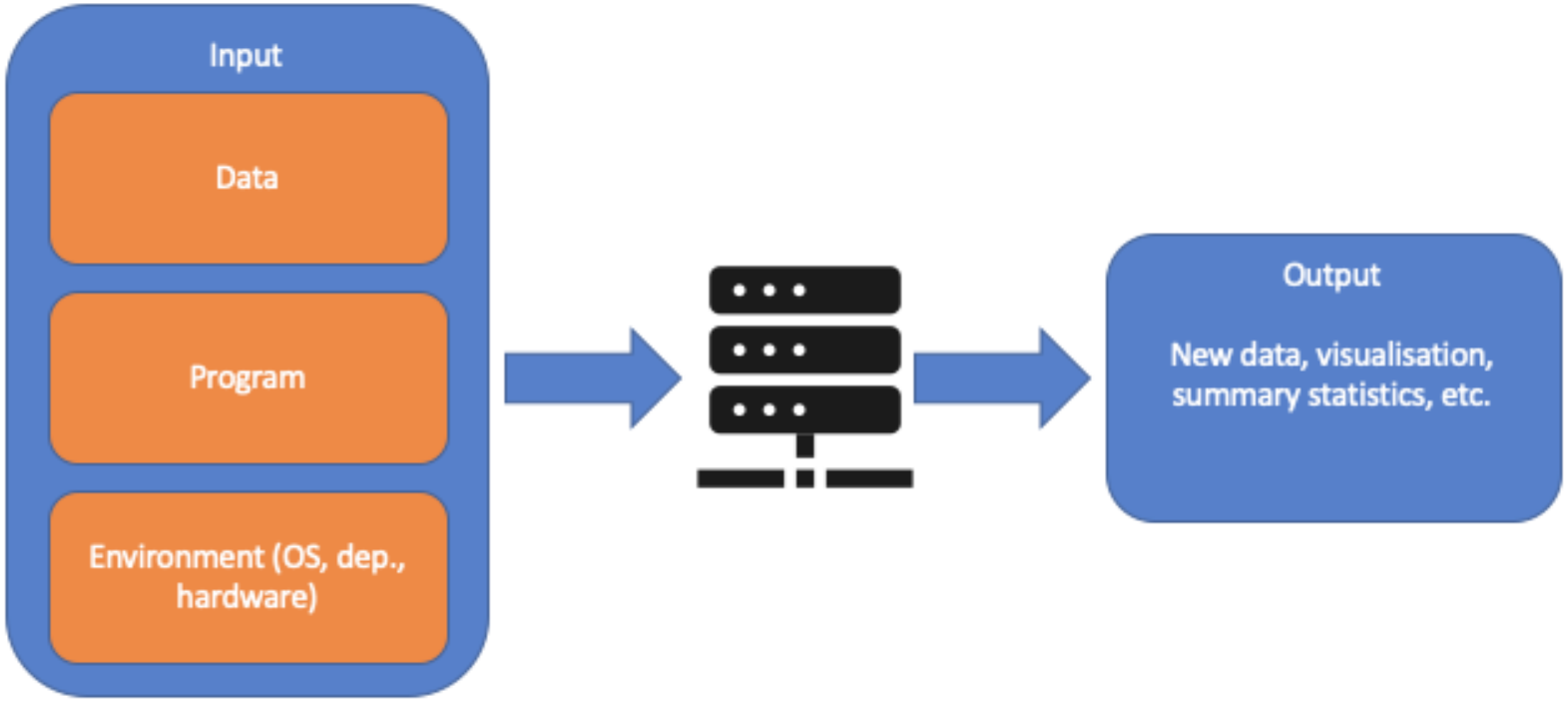
La reproductibilité logicielle en pratique avec GUIX

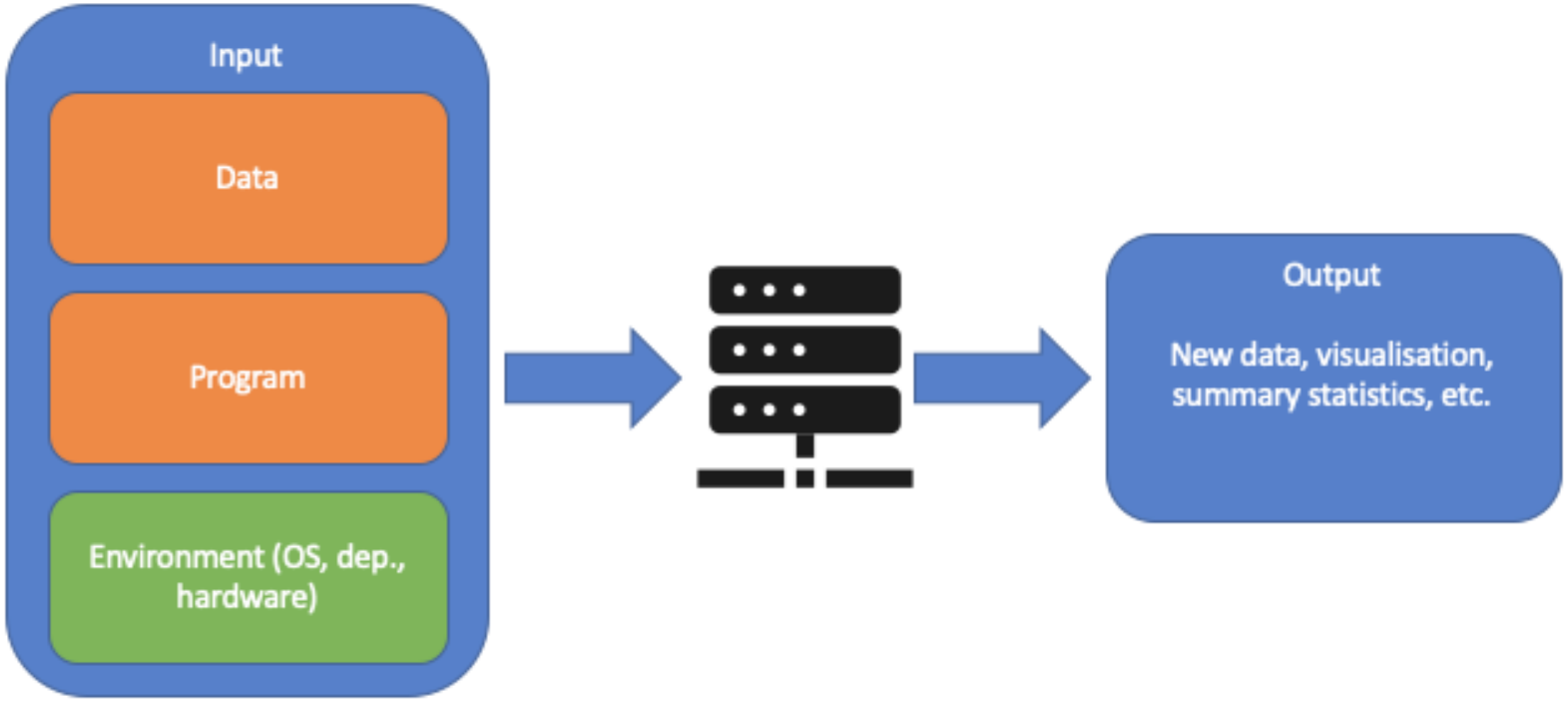
Séminaire recherche reproductible - 25/11/2021

Pierre-Antoine Bouttier, UAR GRICAD, CNRS

Expérimentations numériques

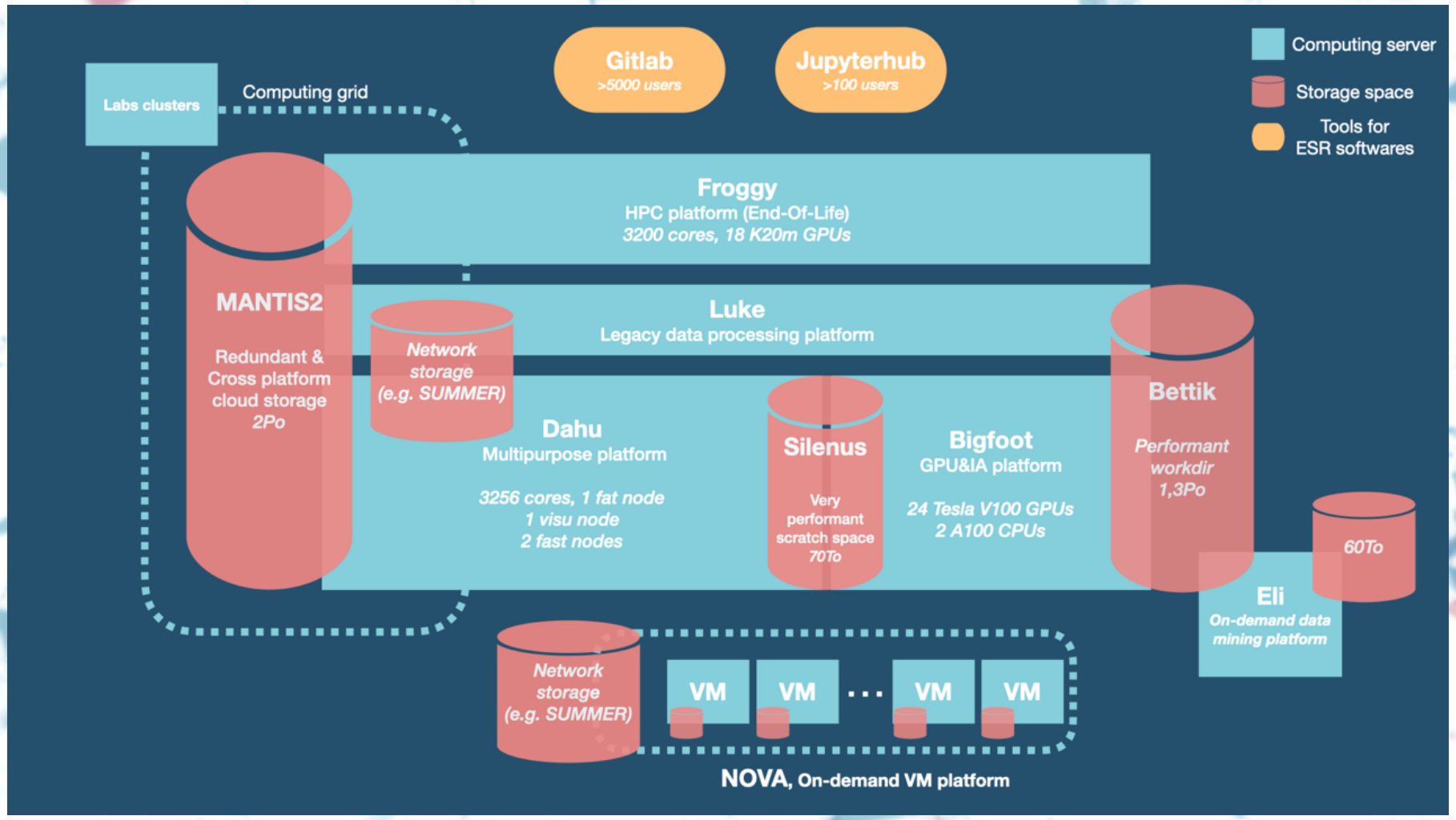


Expérimentations numériques



Comment, avec Guix, mettre en place un environnement logiciel reproductible ?

Vue d'ensemble des infras GRICAD



Notre contexte

- 3 clusters de calcul en propre (environ 10000 coeurs CPU)...
- ...reliés entre eux et à des clusters de labos par une grille de calcul local (*CiGri*)
- Utilisations :
  - HPC
  - Traitement de données
  - HTC
  - Visu, formation & développement
- **Grande hétérogénéité** des usages, des communautés, des niveaux de compétences utilisateur
- **Le défi** : trouver un outil de gestion d'environnement logiciel qui réponde à des contraintes très hétérogènes

Pourquoi GUIX ?

- Mêmes raisons que pour nix :
  - Reproductibilité (et portabilité d'un cluster à l'autre)
  - Versatilité
  - Souplesse pour l'utilisateur en général (comparativement à `module`)

Mais pourquoi GUIX alors ?

- Mésocentre fait aussi pour expérimenter différentes solutions en conditions réelles
- Interface utilisateur et langage différent
- Liens plus directs avec le développeur historique (L. Courtès, INRIA Bordeaux)

Revers de la médaille

- Dispersion des forces (rien d'ingérable) côté support/ASR
- Utilisateurs parfois un peu perdus
- => Effort d'accompagnement (que l'on fait avec plaisir)

Quelques définitions

- Définition de paquet : code source définissant l'installation un logiciel sur un système cible, notamment :
  - emplacement des sources
  - métadonnées
  - dépendances
- Channel : dépôt git contenant un ensemble de définition de paquets (codes sources)
  - Channel principal guix : ensemble des paquets guix disponibles par défaut
  - Channel GRICAD : paquets développés pour les besoins des utilisateurs GRICAD

Principaux mécanismes de GUIX

- Chaque paquet est unique et identifié. Une fois construits à partir du fichier de définitions, les paquets sont tous stockés sous le même répertoire `/gnu/store`
- Quand l'utilisateur installe un paquet, `guix` le construit puis créée des liens symboliques dans un répertoire (= profil) appartenant à l'utilisateur (par défaut, `$HOME/.guix-profile/`)
- Ensuite, pilotage par la variables d'env (*e.g.* `PATH` , `PYTHONPATH` ...)
- A chaque modification de la définition du paquet, si on souhaite le reconstruire, un nouveau paquet sera créée (unicité des paquets)
- Dans le cadre de logiciels open-source, l'ensemble de l'arbre de dépendance est construit à partir des sources.

Passons à la pratique

Arrivée sur les clusters

```
# On active GUIX.
$ source /applis/site/guix-start.sh
```

- A faire dès que l'on veut utiliser la commande guix ou les environnements précédemment installés avec guix, y compris au sein des jobs que vous lancez.
- Cette commande peut être mise dans le `.bashrc` si et seulement si vous utilisez uniquement `guix` pour gérer votre environnement logiciel (bravo !)
- À la première utilisation, le script va initialiser votre profil par défaut.

Les commandes de base

```
# Liste des paquets installés
$ guix package -I
# Rechercher un paquet
$ guix search python
# Installer un paquet
$ guix install python
# Commande gricad pour rafraîchir guix lui-même et le profil courant
$ refresh_guix
# Mise à jour des définitions de paquet (non des paquets eux-mêmes) et de guix lui-même
$ guix pull
# Mettre à jour un paquet
$ guix upgrade python
# Pour supprimer un paquet
$ guix remove python
```

Les profils (sur les clusters GRICAD)

Profil : répertoire contenant les liens symboliques d'un ensemble de paquets que vous avez installés.

```
# Liste des profils de l'utilisateur
$ guix package --list-profiles
# Activation d'un profil
$ refresh_guix nom_profil
# Création d'un profil
$ guix install -p $GUIX_USER_PROFILE_DIR/nom_profil <some_package>
# Ajouter un nouveau profil au profil courant
$ refresh_guix nom_profil
```

Sur les clusters, on est obligé (pour le moment) d'empiler les profils.

Un exemple (de la vraie vie véritable)

Installation du logiciel NCO (suite d'opérateur pour des fichiers netcdf)

A l'ancienne (compilation à la main)

- Avantages :
  - L'utilisateur sait déjà faire
  - avec Guix, l'environnement de compilation et de runtime est reproductible ! (c'est déjà pas mal)
- Inconvénients :
  - Plusieurs étapes.
  - Chaque utilisateur, dans le temps et l'espace doit tout refaire.
  - => optimal : il vaut mieux faire ou nous demander de faire le paquet guix !

Le paquet GUIX

Voir IDE

Les environnements

Intérêt des environnements : env. log. isolé, temporaire et reproductible rapidement (*e.g.* conda environment, virtualenv)

```
# Lance un shell avec python, numpy et scipy disponibles
$ guix shell python python-numpy python-scipy
# Idem mais lance la commande python3 ensuite
$ guix shell python python-numpy python-scipy -- python3
```

Reproduire un environnement

Comment faire pour reproduire, au bit près, l'environnement logiciel ?

- On écrit un `README` exhaustif en espérant que les confrères le suivent à la lettre ?
- On ne fait plus jamais de guix pull et on croise les doigts pour que les confrères aussi ?
- On envoie le disque dur ?

Reproduire un environnement simplement

En 3 commandes, voici comment reproduire un environnement à l'identique dans le temps et l'espace :

```
bob@laptop$ guix describe --format=channels > bob-channels.scm
```

Bob envoie le fichier texte `bob-channels.scm` à Alice qui exécute ailleurs et plus tard :

```
alice@supercomputer$ guix pull --channels=bob-channels.scm
```

Merci de votre attention

Et à propos des conteneurs ?

2 parties dans les systèmes de conteneurs (*e.g.* docker, singularity):

- La construction de l'image (Et docker ou singularity sont plutôt mauvais du point de vue de la reproductibilité)
- L'exécution de l'image

And what about containers?

Heureusement, on peut **construire des images avec guix qui sont parfaitement reproductible**!

```
# Create a tar.gz file...
$ guix pack -f docker -S /bin=bin guile guile-readline
# ...that can be directly passed to docker load...
$ docker load < file
# Then docker run
$ docker run -ti guile-guile-readline /bin/guile
```

Cheat codes pour créer un paquet

```
guix import pypi --recursive nomDuPaquetPypi
```