

---

## Interface générique pour la parallélisation d'applications de recherche en imagerie biomédicale

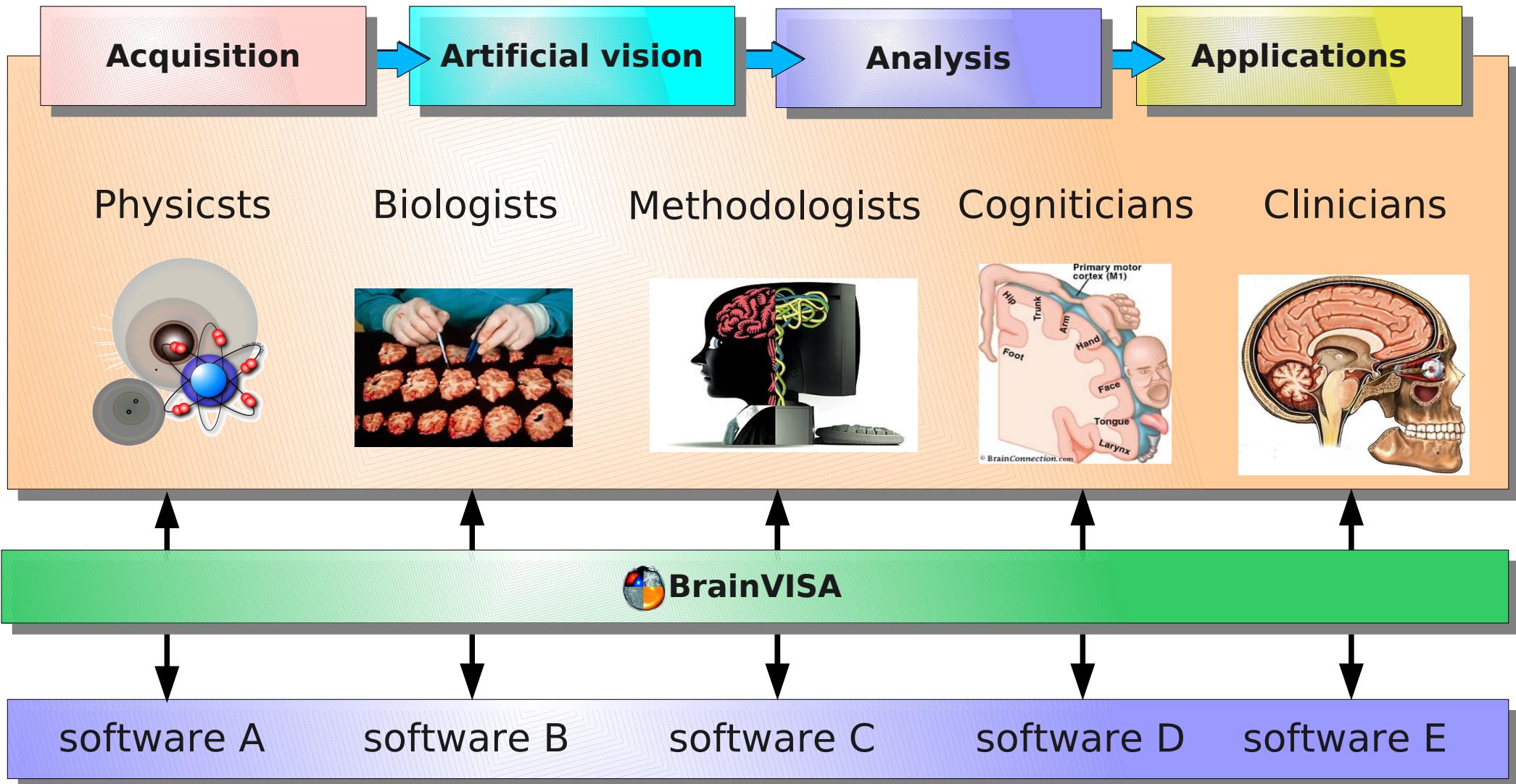
---

Yann Cointepas : [yann@sapetioc.org](mailto:yann@sapetioc.org)  
Soizic Laguitton : [soizic.laguitton@gmail.com](mailto:soizic.laguitton@gmail.com)

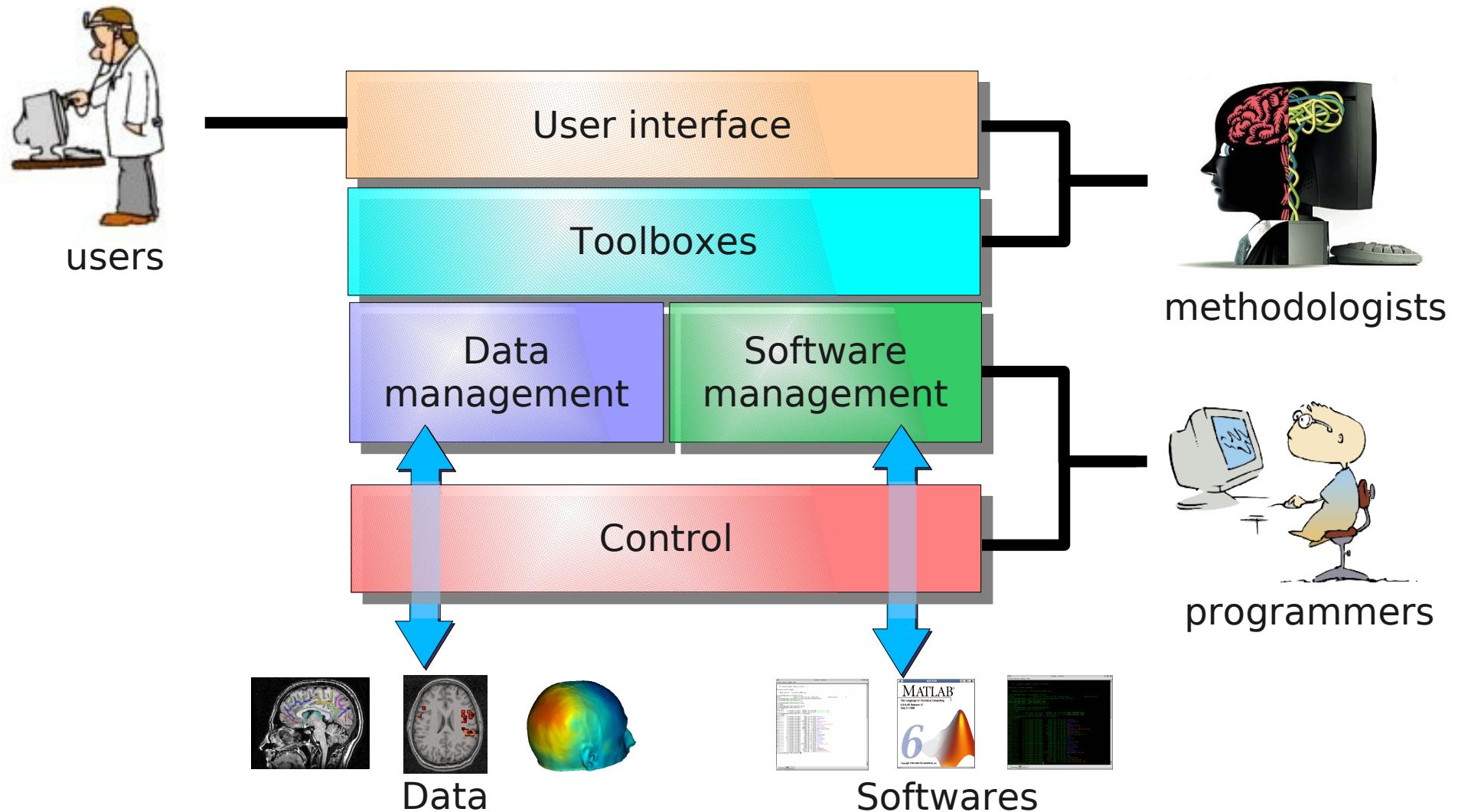


# What is BrainVISA ?

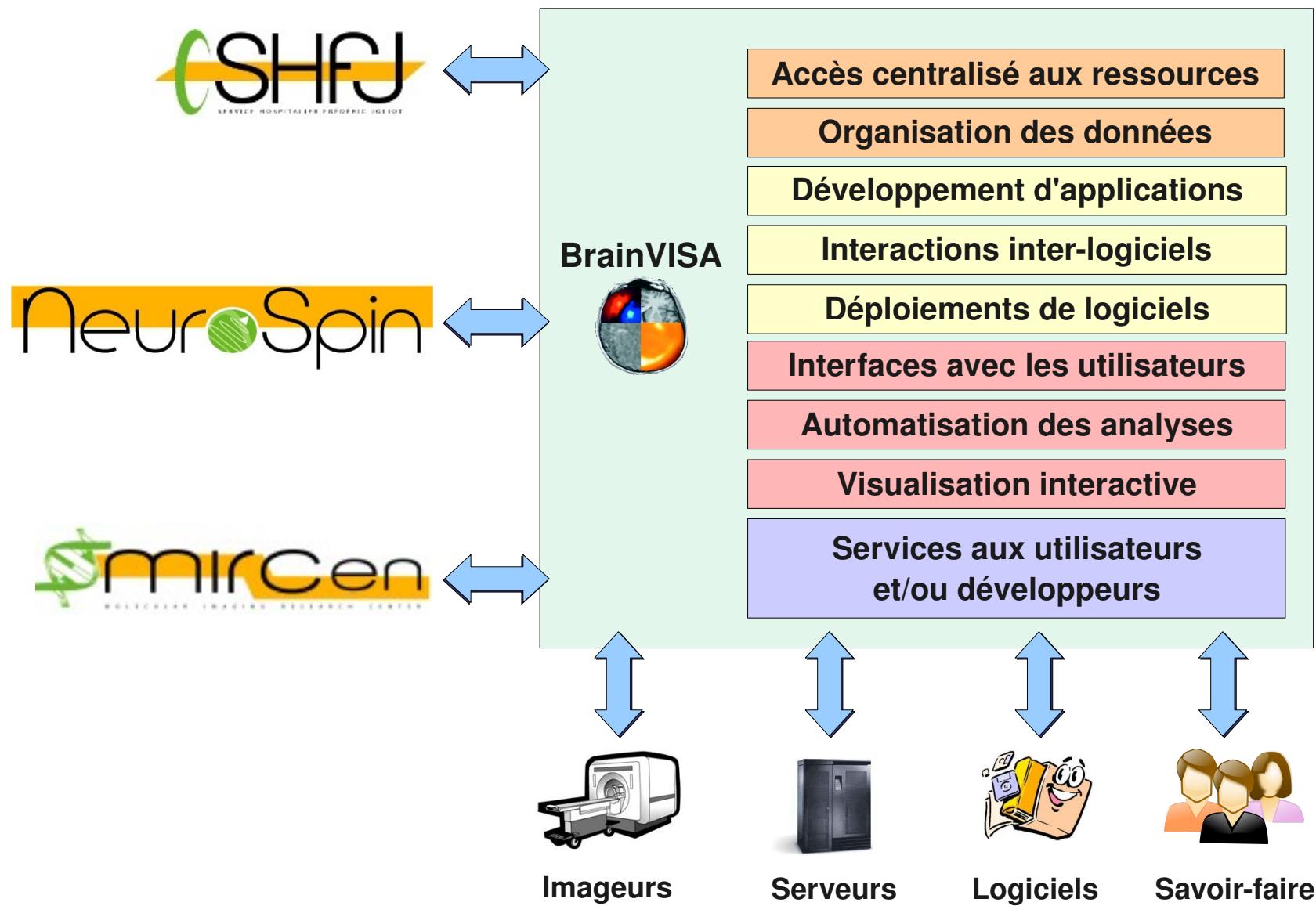
Image processing pipeline



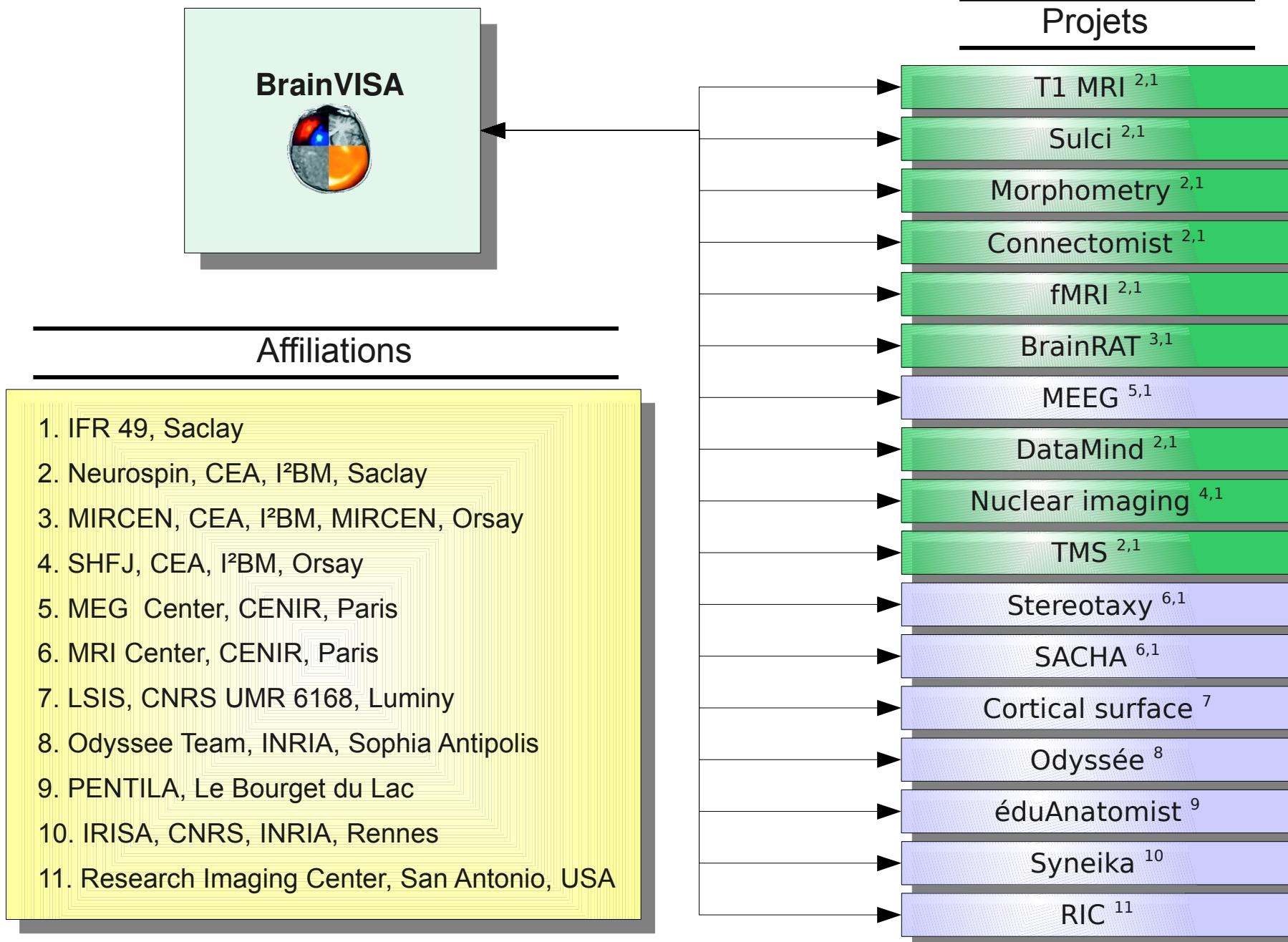
# Overview of BrainVISA architecture



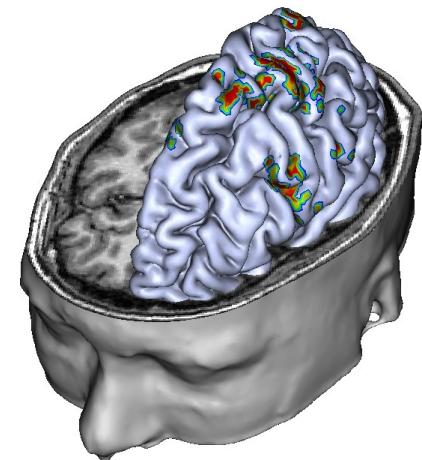
# Resource sharing with BrainVISA



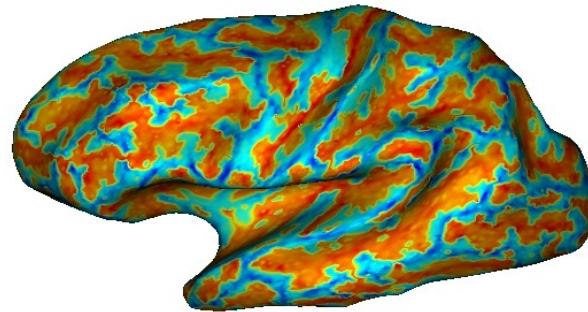
# Some projects based on BrainVISA tools



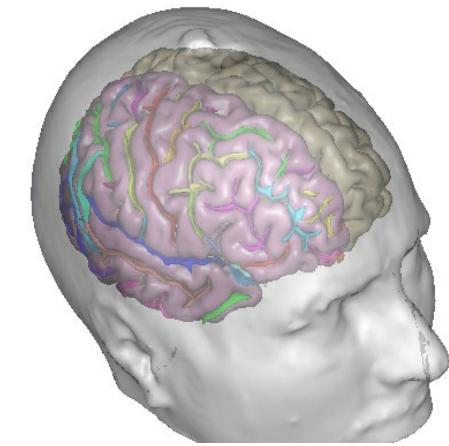
# BrainVISA toolboxes (1)



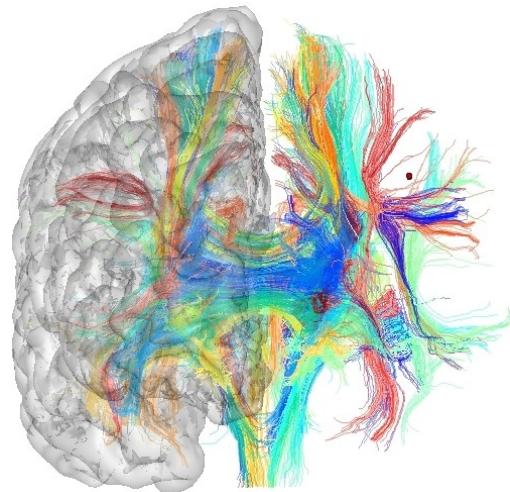
multimodal interactive  
visualization



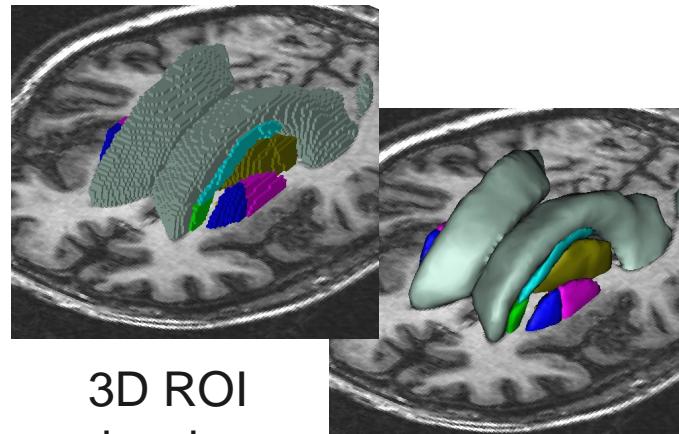
surface analysis



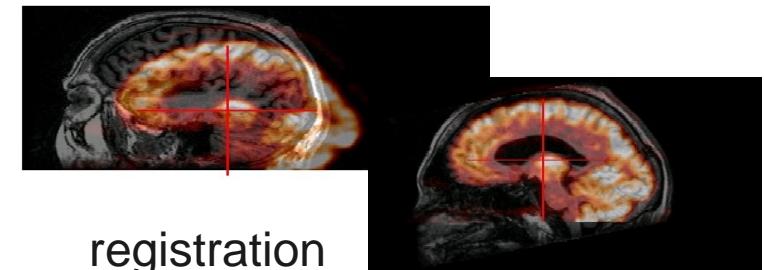
anatomy segmentation  
and sulci identification



DTI analysis and  
fiber tracking



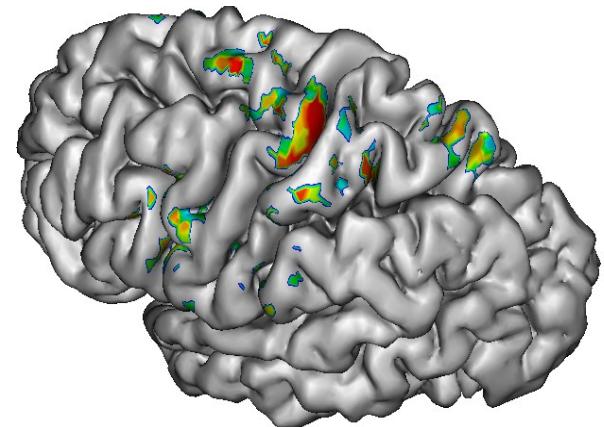
3D ROI  
drawing



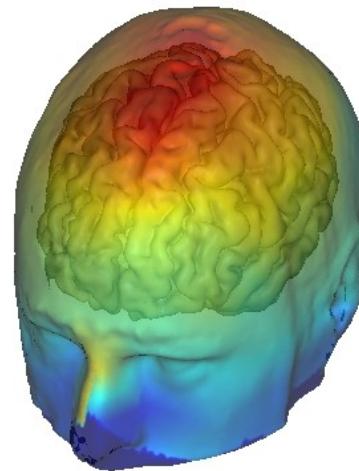
registration

# BrainVISA toolboxes (2)

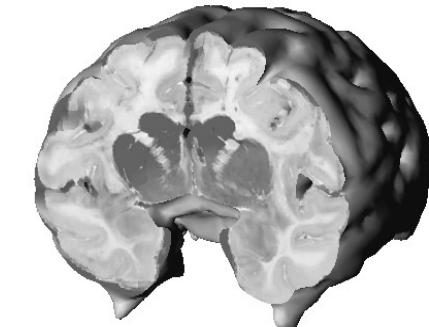
---



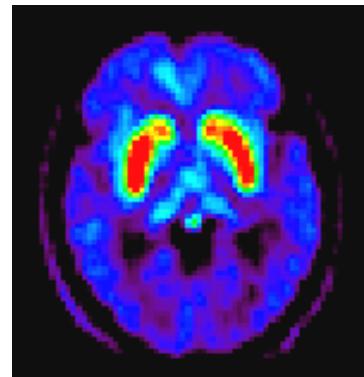
functionnal MRI



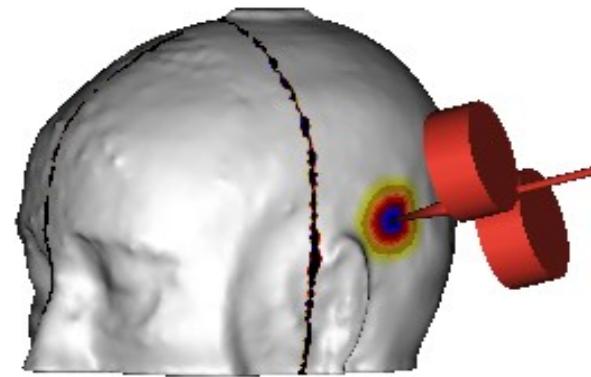
MEG / EEG



histology



nuclear imaging



TMS

# BrainVISA web site: <http://brainvisa.info>

The screenshot shows the BrainVISA/Anatomist Home Page in a Konqueror browser window. The title bar reads "BrainVISA/Anatomist Home Page - Konqueror". The menu bar includes "Document", "Édition", "Affichage", "Aller", "Signets", "Outils", "Configuration", "Fenêtre", and "Aide". The toolbar contains various icons for file operations like Open, Save, Print, and Search. The URL bar shows "http://brainvisa.info/". The main content area features a logo of a brain with a colorful, segmented appearance. The title "BrainVISA / Anatomist" is displayed prominently, followed by "(français)" with a French flag icon. Below the title is a navigation menu with tabs: BRAINVISA, ANATOMIST, NEWS, DOWNLOAD (highlighted in pink), PUBLICATIONS, FAQ, CONTACT, and FORUM. To the left of the text is a cartoon illustration of a brain with arms and legs, pointing towards the text. The text describes BrainVISA as a software for image processing, mentioning its assembly-line nature and command-line interface. It also highlights the distribution of a toolbox for MR image segmentation, listing specific outputs like morphometry, surface meshes, and cortical fold graphs. A sequence of images at the bottom illustrates the workflow from a raw brain scan to various processed outputs. A footer note mentions the "Anatomist" visualization software.

**BrainVISA / Anatomist**  
(français)

**BRAINVISA ANATOMIST NEWS DOWNLOAD PUBLICATIONS FAQ CONTACT FORUM**

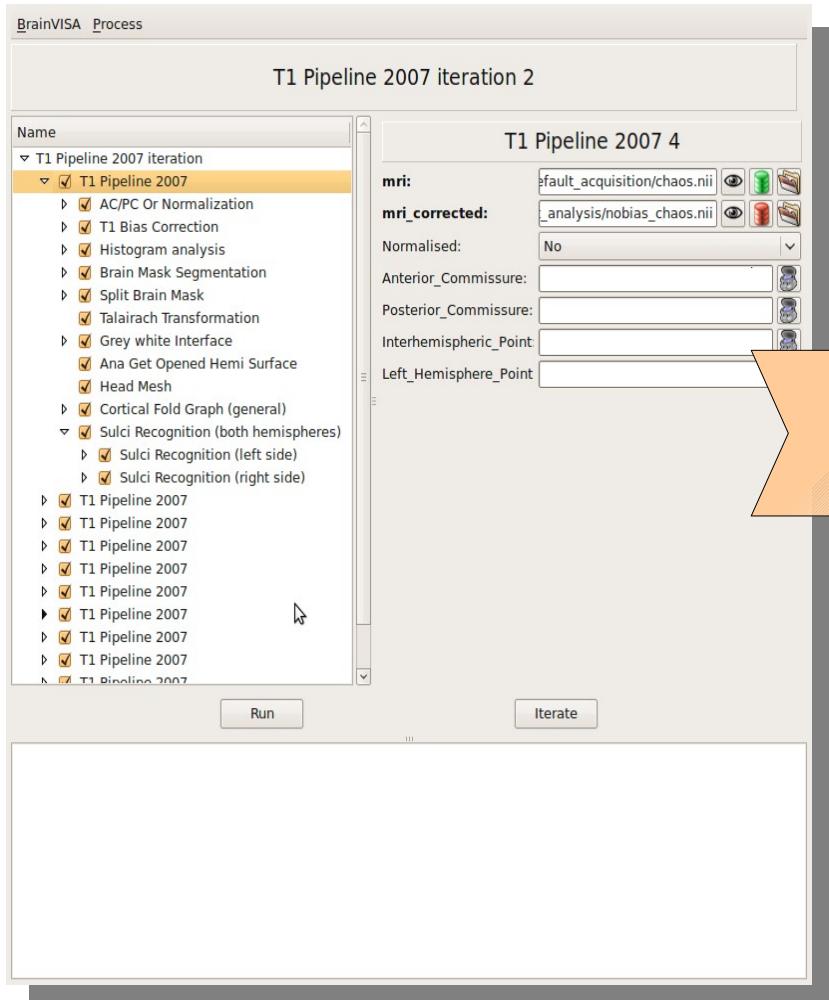
*BrainVISA* is a software, which embodies an image processing factory. A simple control panel allows the user to trigger some sequences of treatments on series of images. These treatments are performed by calls to command lines provided by different laboratories. These command lines, hence, are the building blocks on which are built the assembly lines of the factory.

*BrainVISA* is distributed with a toolbox of building blocks dedicated to the segmentation of T1-weighted MR images. The product of the main assembly line made up from this toolbox is the following: grey/white classification for Voxel Based Morphometry, Meshes of each hemisphere surface for visualization purpose, Spherical meshes of each hemisphere white matter surface, a graph of the cortical folds, a labeling of the cortical folds according to a nomenclature of the main sulci:

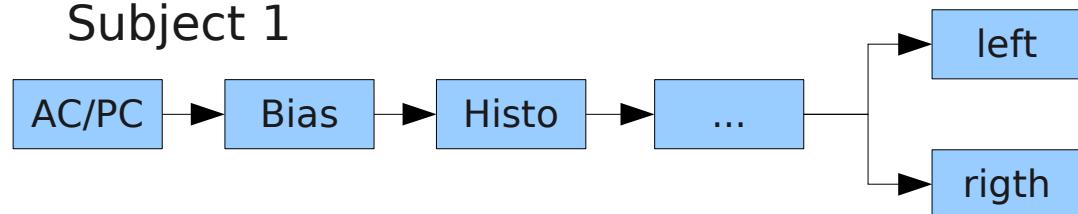
A glimpse of the package content is proposed in [BrainVISA help pages](#).  
[About BrainVISA](#).

*Anatomist* is a visualization software, which main originality is a generic module dedicated to structural

# What is BrainVISA parallelization ?



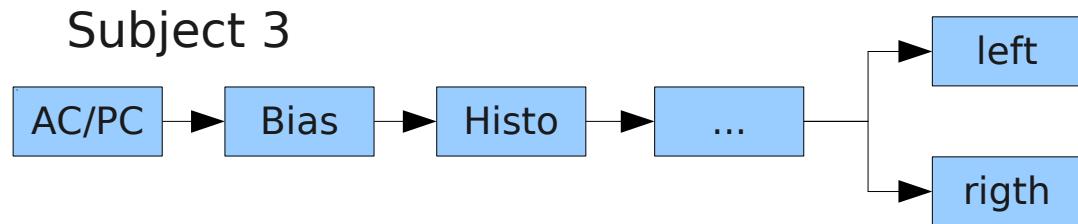
Subject 1



Subject 2



Subject 3



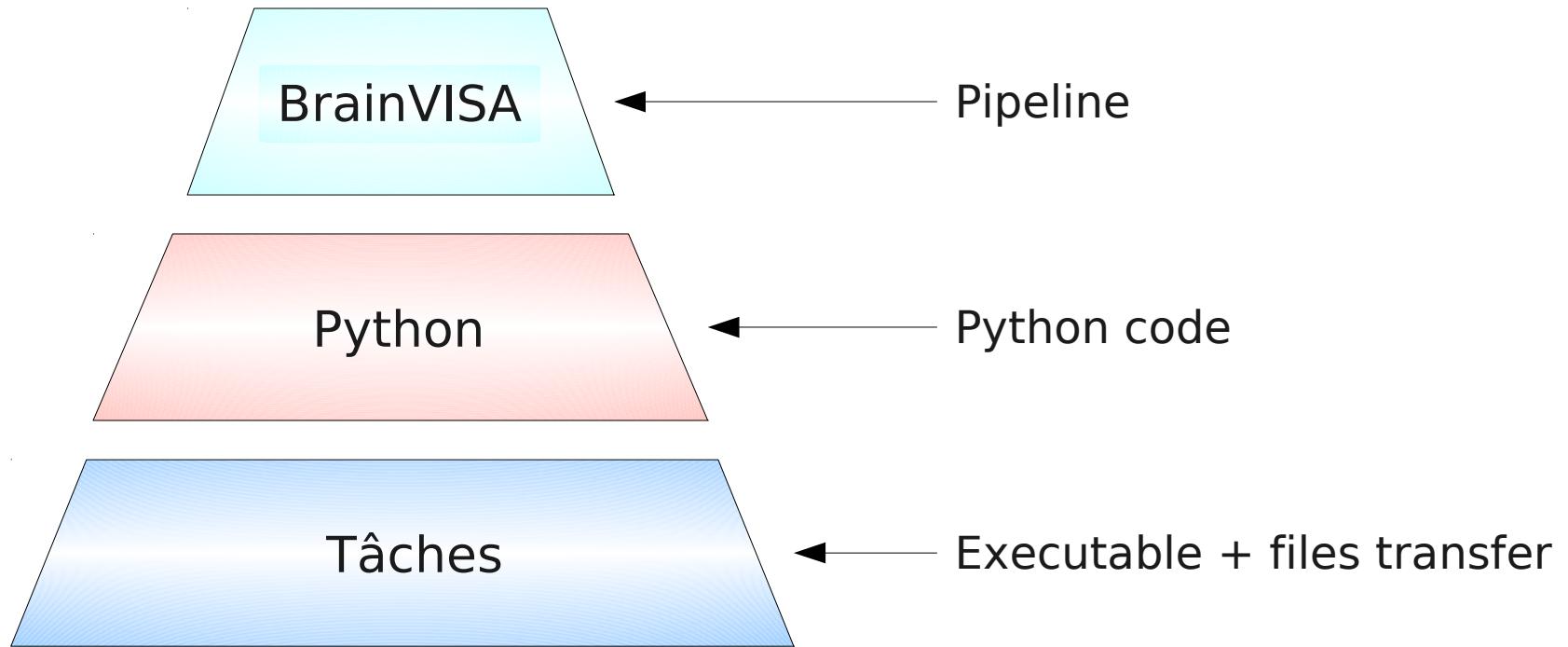
Subject 4



▪  
▪  
▪

# Three parallelization layers

---



# Parallélisation de tâches

---

▶ Objectif :

Exploiter au mieux les ressources disponibles pour minimiser le temps d'exécution d'un ensemble de tâches.

▶ Tâche (Job) :

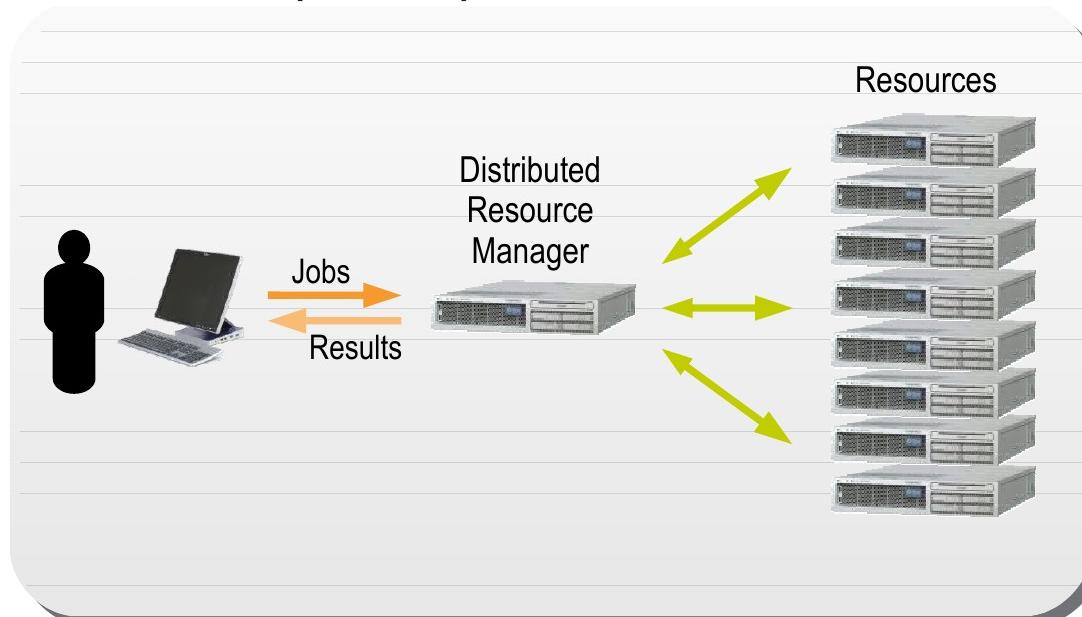
- executable + paramètres + données
- exécution unique en un temps fini

▶ Deux niveaux de granularité pour la parallélisation :

- exécution de plusieurs tâches indépendantes (par exemple un unique programme appliqué à un ensemble de données )
- programmation parallèle
  - ▶ mémoire partagée (1 machine, N coeurs): threads, OpenMP, etc.
  - ▶ Mémoire distribuée (M machines, N coeurs): MPI, etc.

# Utilisation des ressources par les chercheurs

- ▶ 1 machine (plusieurs coeurs)
  - ▶ système d'exploitation
- ▶ Plusieurs machines
  - ▶ ssh, implémentation MPI ...
- ▶ Plusieurs utilisateurs
  - ▶ accords/arrangements nécessaires
- ▶ Plus de machines, plus d'utilisateurs & plus de tâches
  - ▶ Systèmes de gestion de ressources distribuées (DRMS)

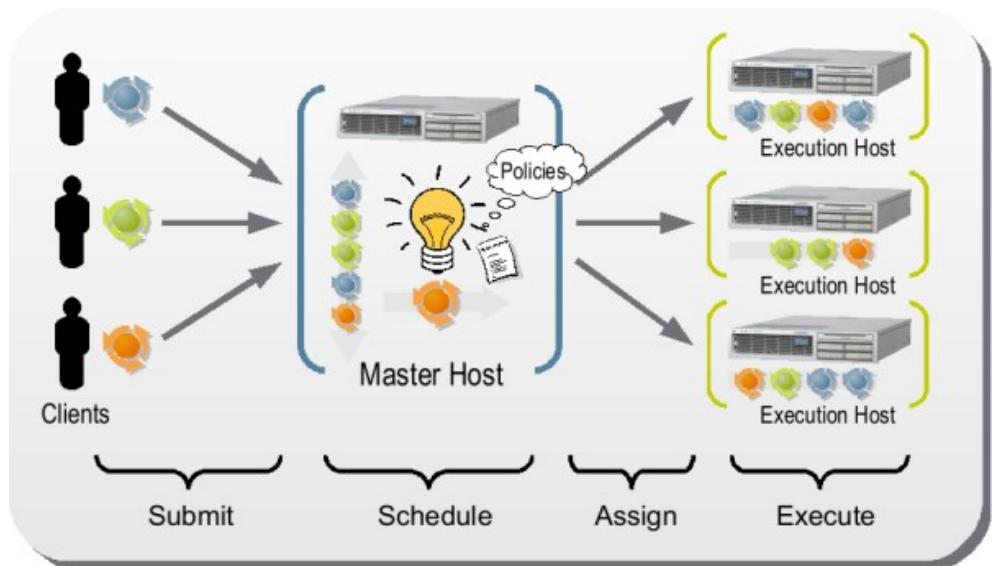
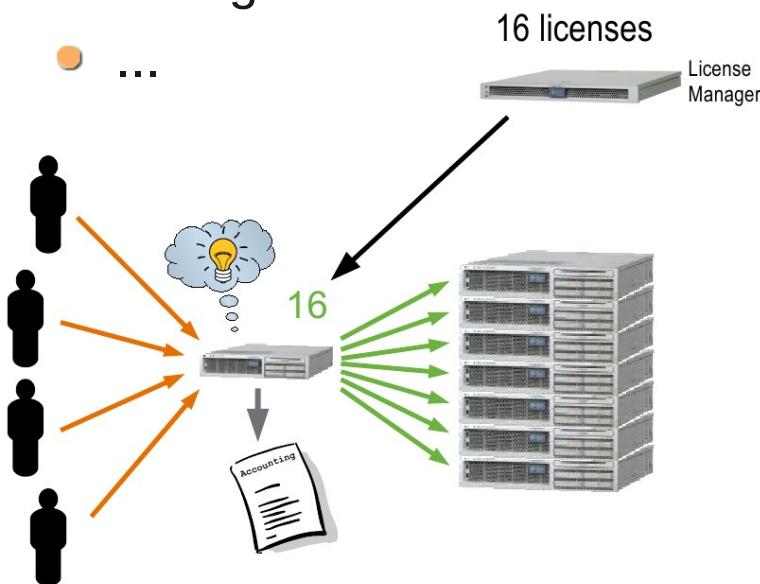


Gestion de la distribution des tâches :

- Pour optimiser l'utilisation des ressources
- Selon une politique d'utilisation
- De manière transparente pour l'utilisateur

# Distributed resource Management System

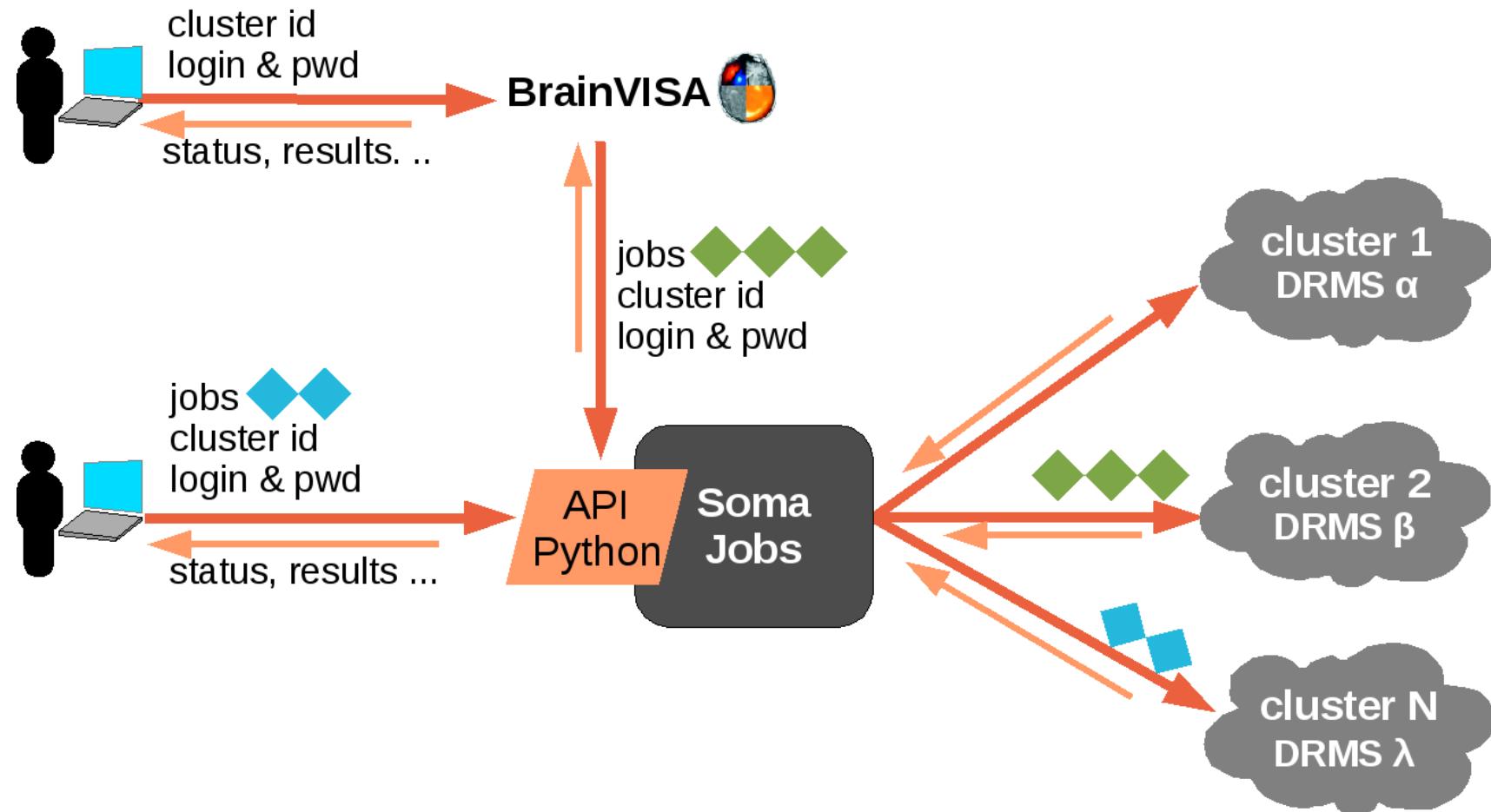
- ▶ Solutions pour l'exploitation des ressources distribuées :
  - systèmes de facturation
  - gestion des licences logicielles
  - gestion de ressources hétérogènes
  - politique d'utilisation hautement configurable



- ▶ Quelques exemples :
  - Sun Grid Engine
  - Condor
  - LSF =>
    - ▶ CCRT platine & titane
  - Torque/PBS
  - Cluster DSV, CCRT cesium

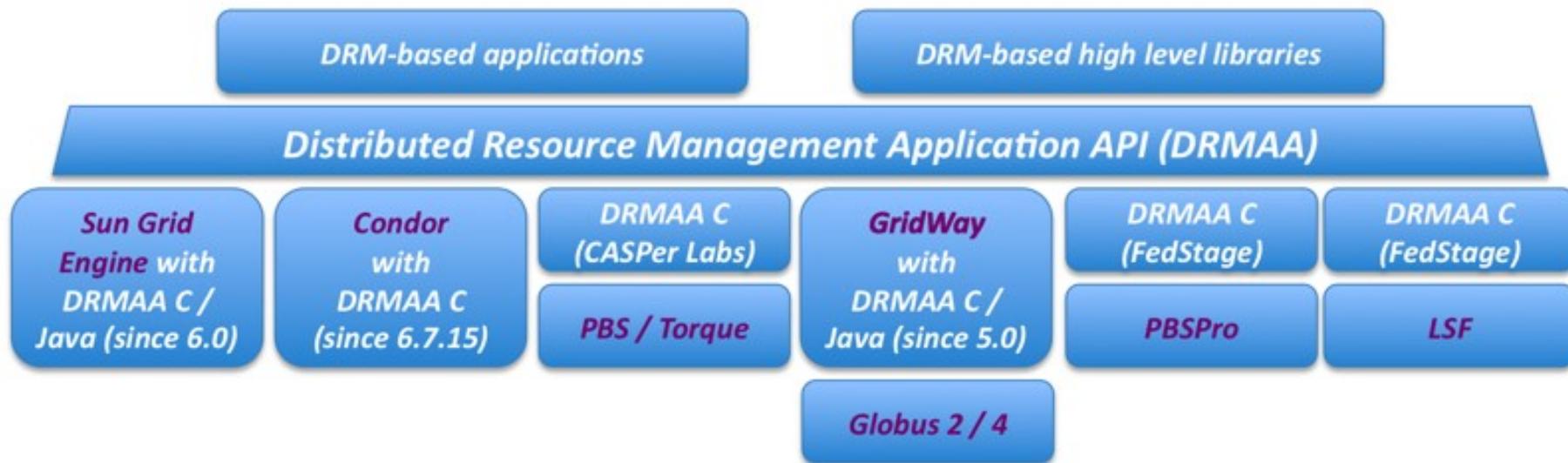
# Objectifs du projet

- ▶ API simple et unique
- ▶ Accès distant transparent
- ▶ Gestion des transferts de fichier
- ▶ Gestion des déconnexions
- ▶ Ajout aisément de l'accès à un nouveau cluster



# Existant => DRMAA

- ▶ Distributed Resource Management Application API:
  - API unique pour soumettre et contrôler des tâches sur les différents DRMS
  - DRMAA working group
    - ▶ développement des spécifications
    - ▶ travail en cours sur DRMAA v2.0
  - Implémentation de sources variés principalement en C :



# Ce que DRMAA ne résout pas

DRMAA et ses implémentations :

✓ API unique pour l'accès aux DRMS

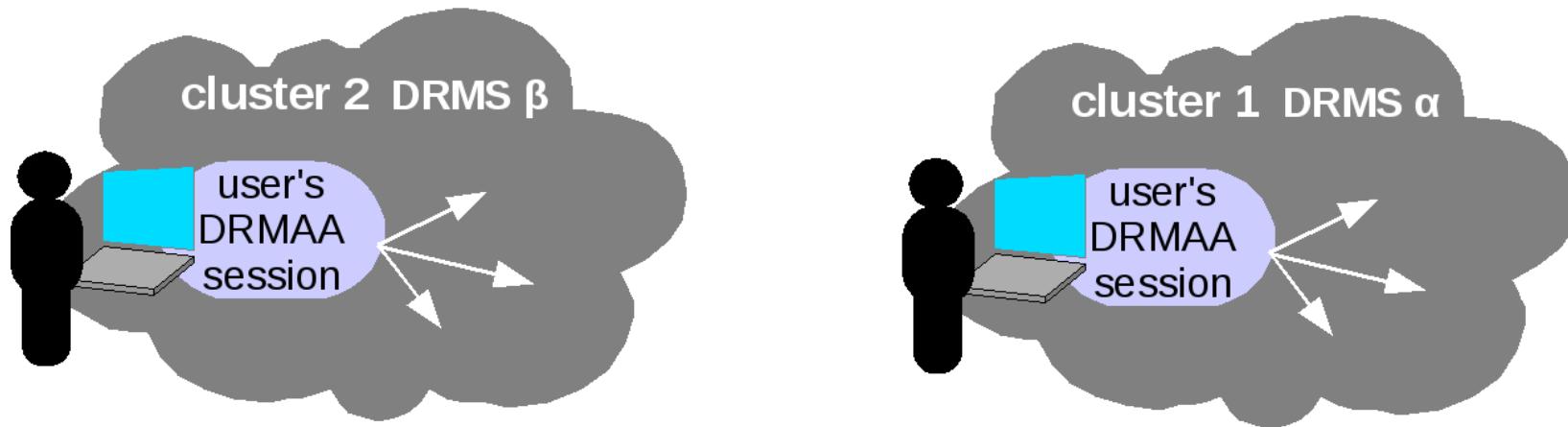
- ▶ soumission des tâches
- ▶ contrôle => suspendre, supprimer
- ▶ suivi => statut des tâches

✗ Pas d'utilisation possible à distance / pas de transfert de fichier

✗ Aucune sauvegarde des données

✗ Ne gère pas les déconnexions (selon les implémentations)

✗ API compliquée



# Solution développée : API Python

- ▶ Connection
    - constructeur
      - ▶ resource\_id,
      - ▶ login
      - ▶ password,
      - ▶ (config\_file)
  - ▶ Soumission de tâches
    - submit
  - ▶ Suivi des tâches
    - status
    - exitInformation
    - stdoutReadLine
    - stderrReadLine
  - ▶ Contrôle des tâches
    - wait
    - stop
    - restart
    - kill
  - ▶ Transferts de fichiers
    - sendFile
    - registerFileTransfer
    - retrieveFile
    - cancelTransfer
  - ▶ Données utilisateur
    - jobs
    - jobInformation
    - transfers
    - transferInformation
- undetermined  
**queued\_active**  
system\_on\_hold  
user\_on\_hold  
user\_system\_on\_hold  
**running**  
system\_suspended  
user\_suspended  
user\_system\_suspended  
**done**  
**failed**
- Returned value + Exit status:*  
**aborted**  
**finished\_regularly**  
finished\_signal  
finished\_unclear\_condition  
**killed\_by\_user**

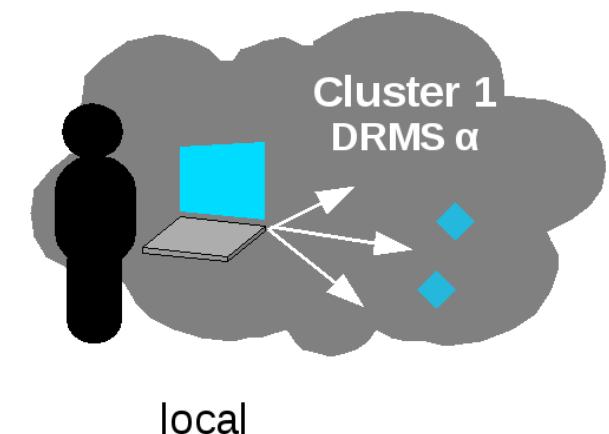
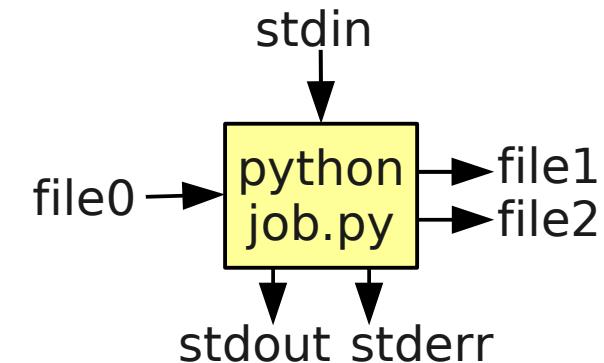
# Méthode de soumission de tâches

---

- ▶ `def submit( self,  
              command,  
              referenced_input_files = None,  
              referenced_output_files = None,  
              stdin = None,  
              join_stderrot = False,  
              disposal_timeout = 168,  
              name_description = None,  
              stdout_path = None,  
              stderr_path = None,  
              working_directory = None)`

# Soumission de tâches sans transfert de fichier

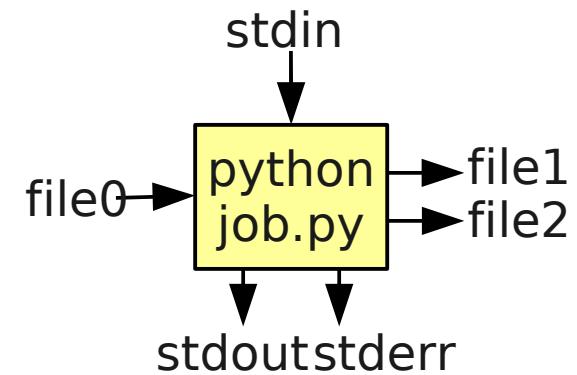
```
▶ j.submit(  
    command = ["python", "mypath/job.py"],  
    referenced_input_files = None,  
    referenced_output_files = None,  
    stdin = "mypath/stdin",  
    stdout_path = "mypath/stdout",  
    stderr_path = "mypath/stderr",  
    disposal_timeout = 48,  
    join_stderrot = False,  
    name_description = "my python job",  
    working_directory = "mypath")
```



# Soumission de tâches avec transfert de fichier

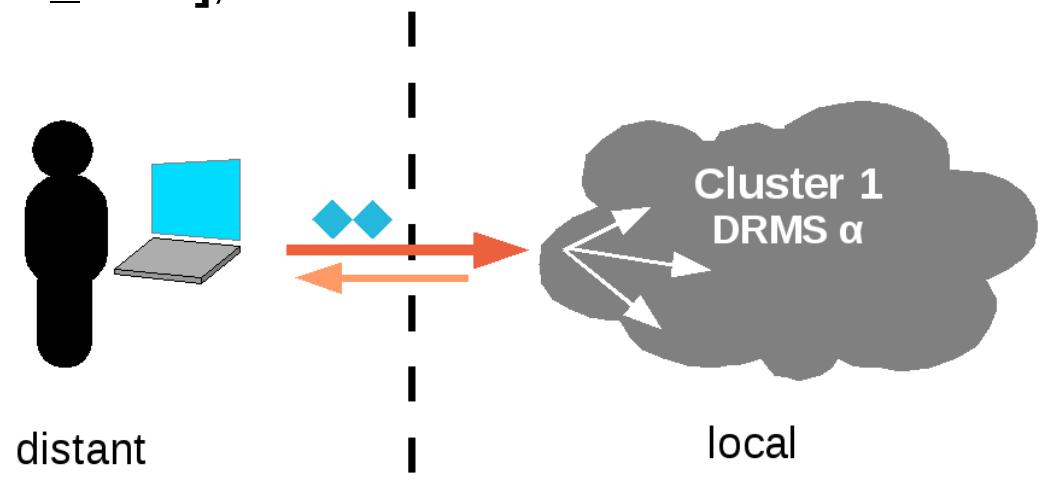
```
file0_local = j.sendFile("mypath/file0", 168)
script_local = j.sendFile("mypath/job.py", 168)
stdin_local = j.sendFile("mypath/stdin", 168)

file1_local = j.registerFileTransfer("mypath/file1", 168)
file2_local = j.registerFileTransfer("mypath/file2", 168)
```

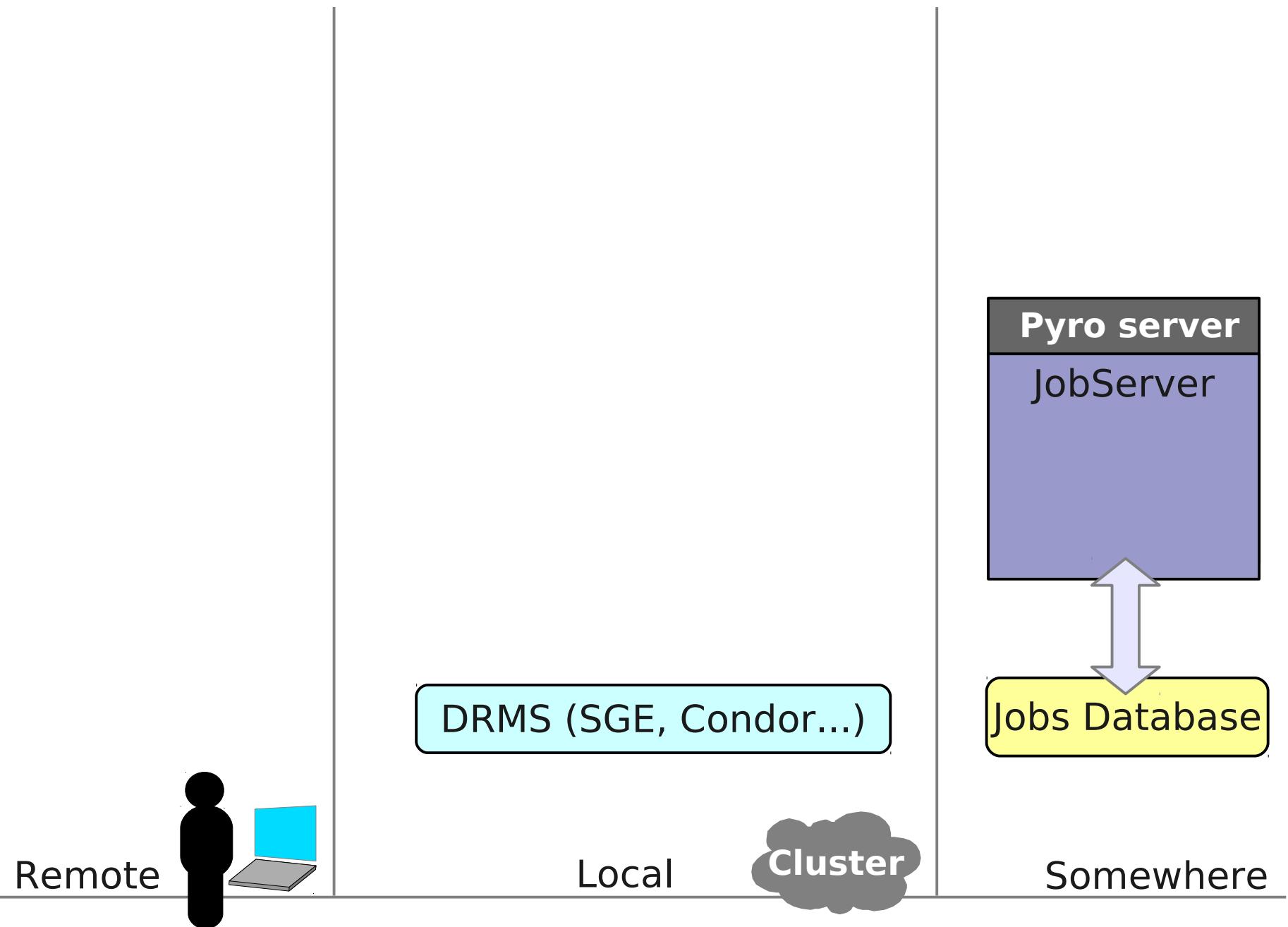


```
jobid = j.submit(  
    command = ["python", script_local],  
    referenced_input_files =[file0_local, script_local, stdin_local],  
    referenced_output_files =[file1_local, file2_local],  
    stdin = stdin_local,  
    join_stderrot = False,  
    disposal_timeout = 168,  
    name_description = "my python job")
```

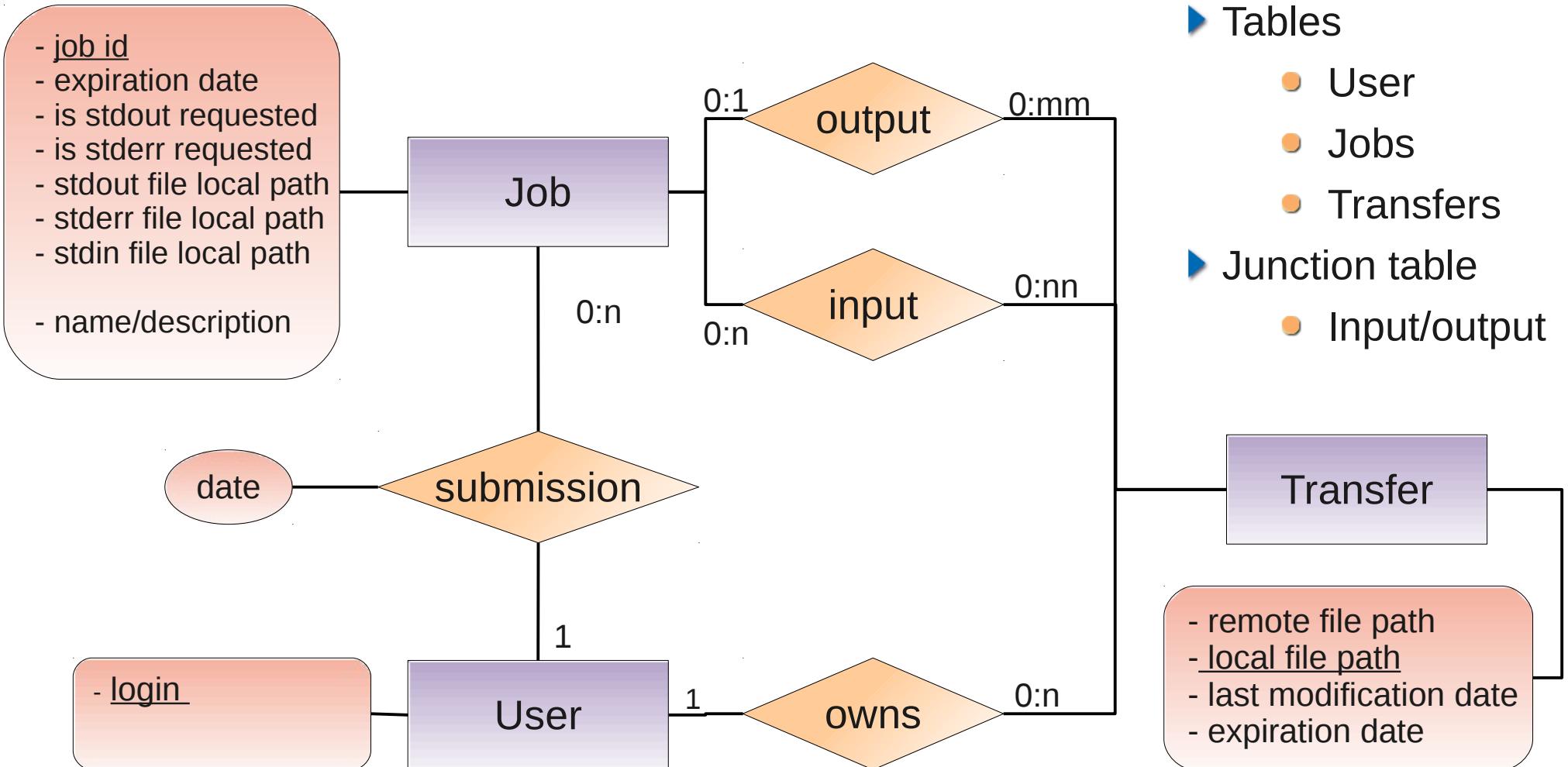
```
j.wait(jobid)  
j.retrieveFile(file1_local)  
j.retrieveFile(file2_local)
```



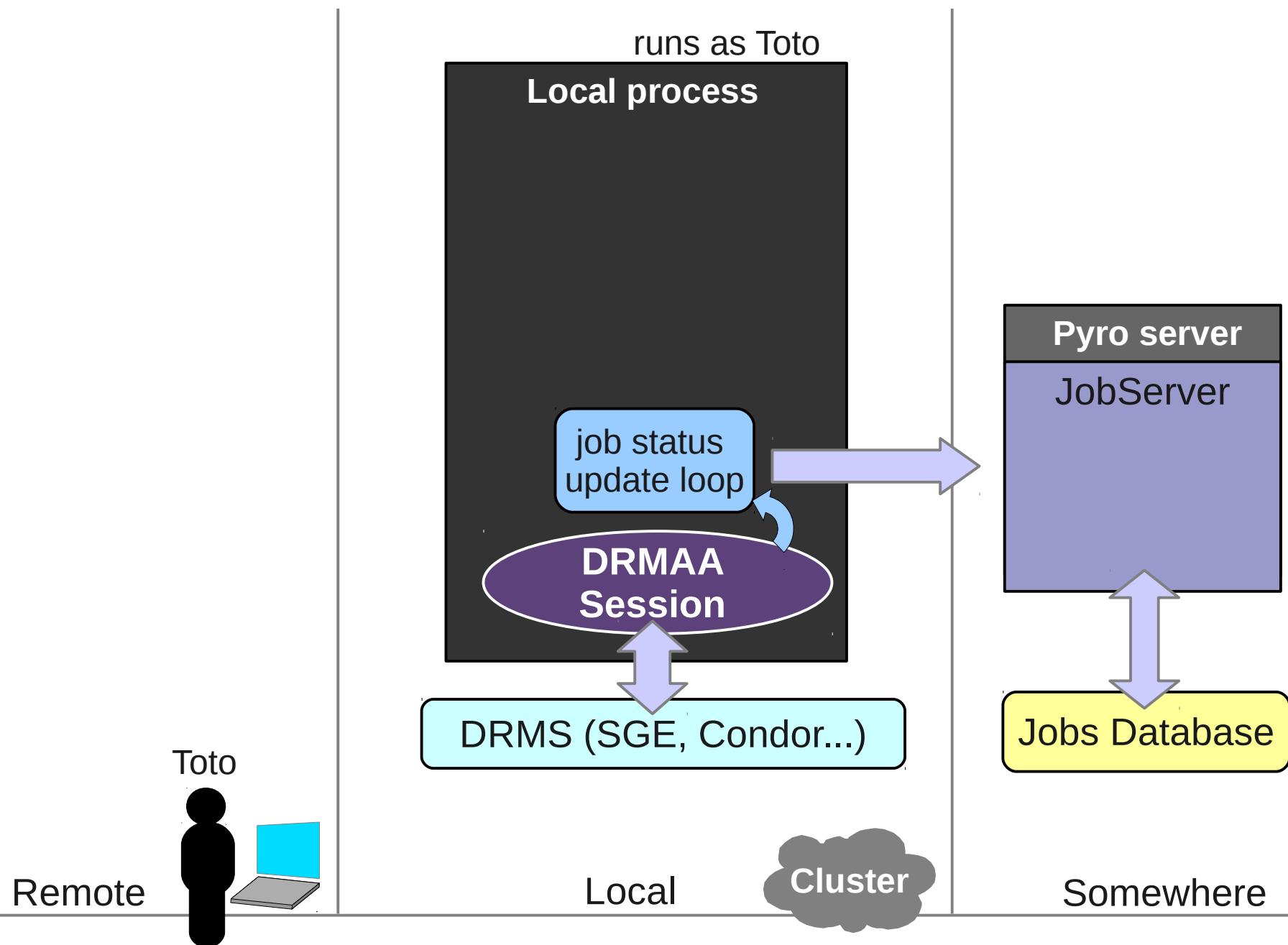
# Solution développée - Architecture



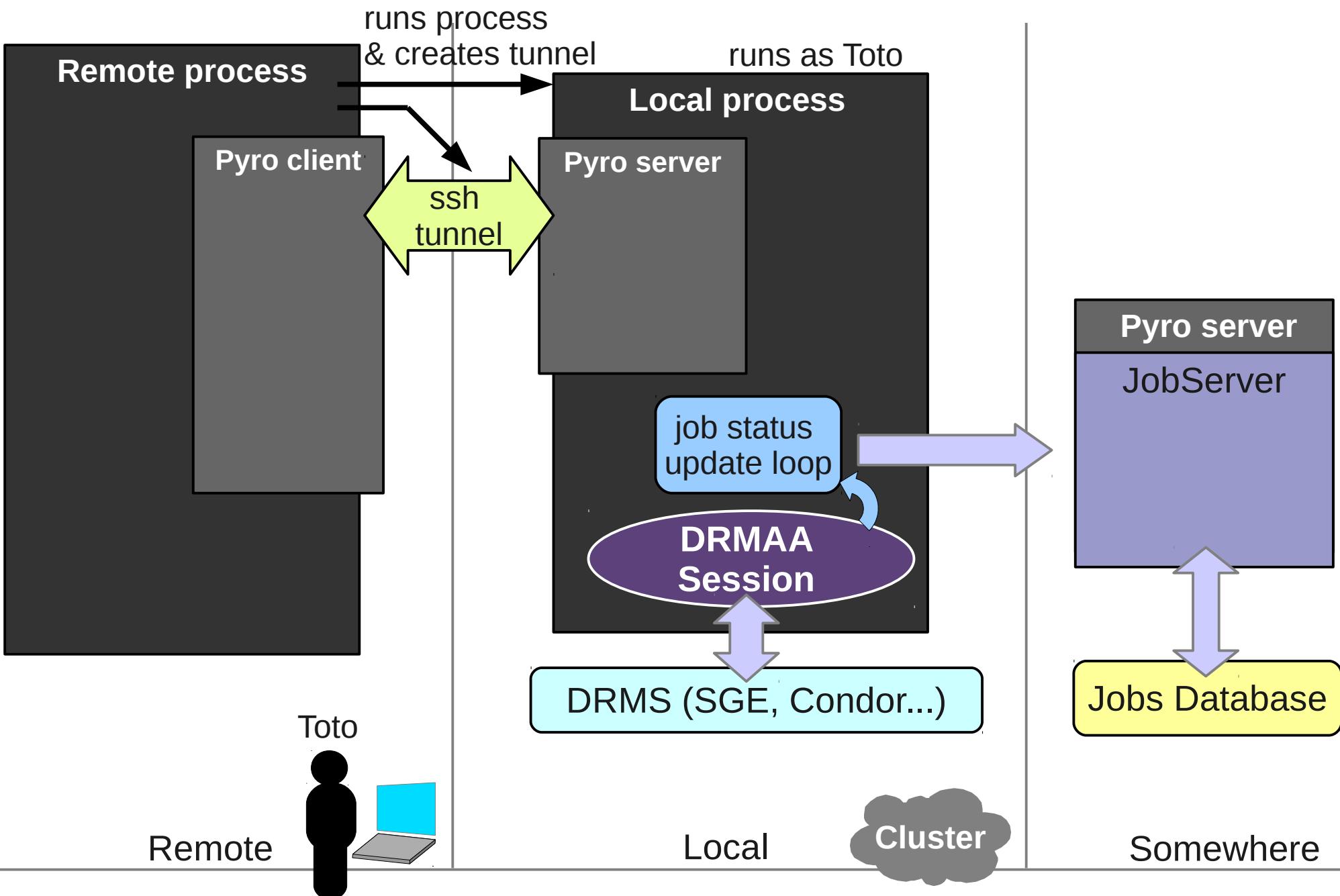
# Solution développée - Base de donnée



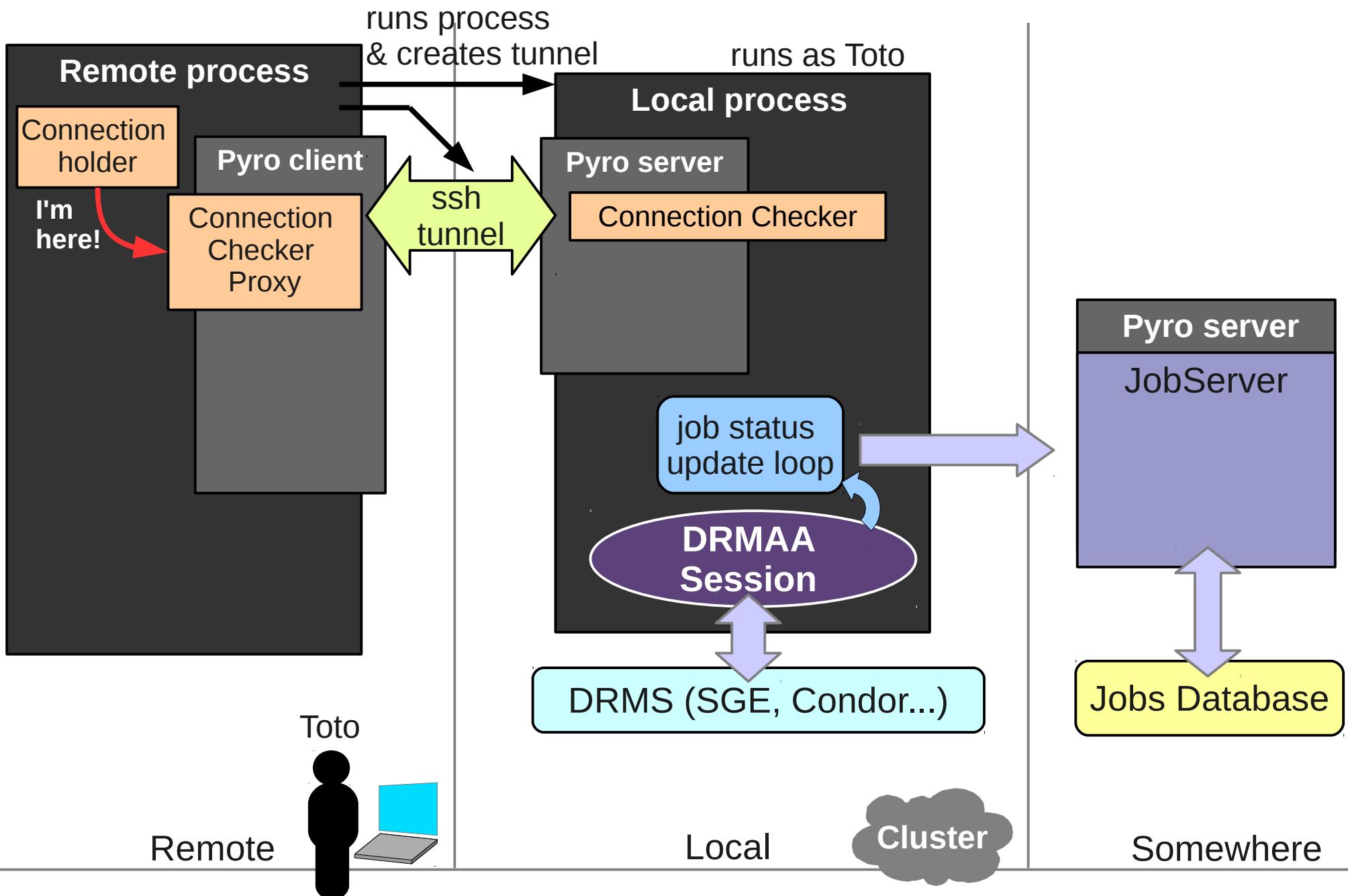
# Solution développée - Architecture



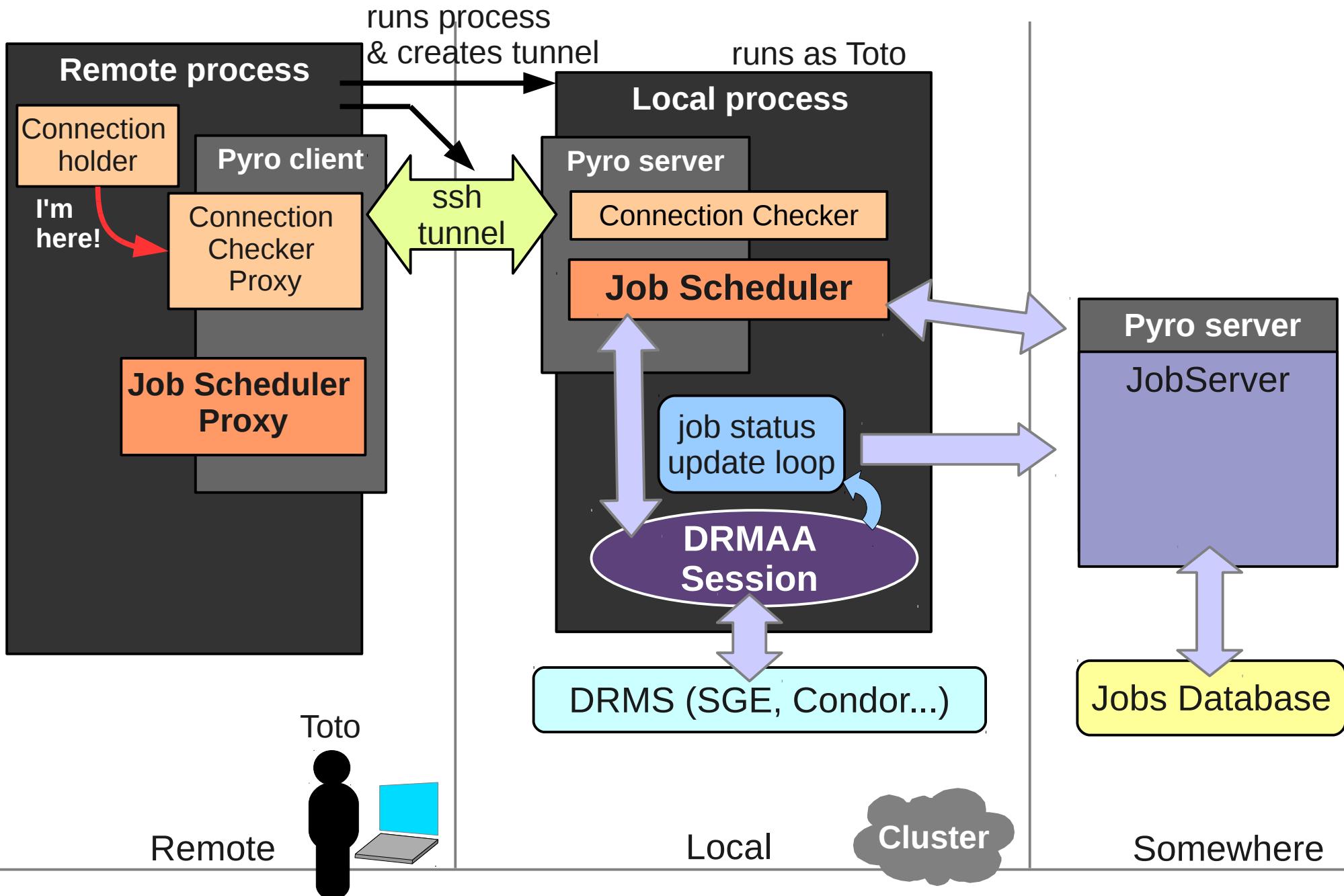
# Solution développée - Architecture



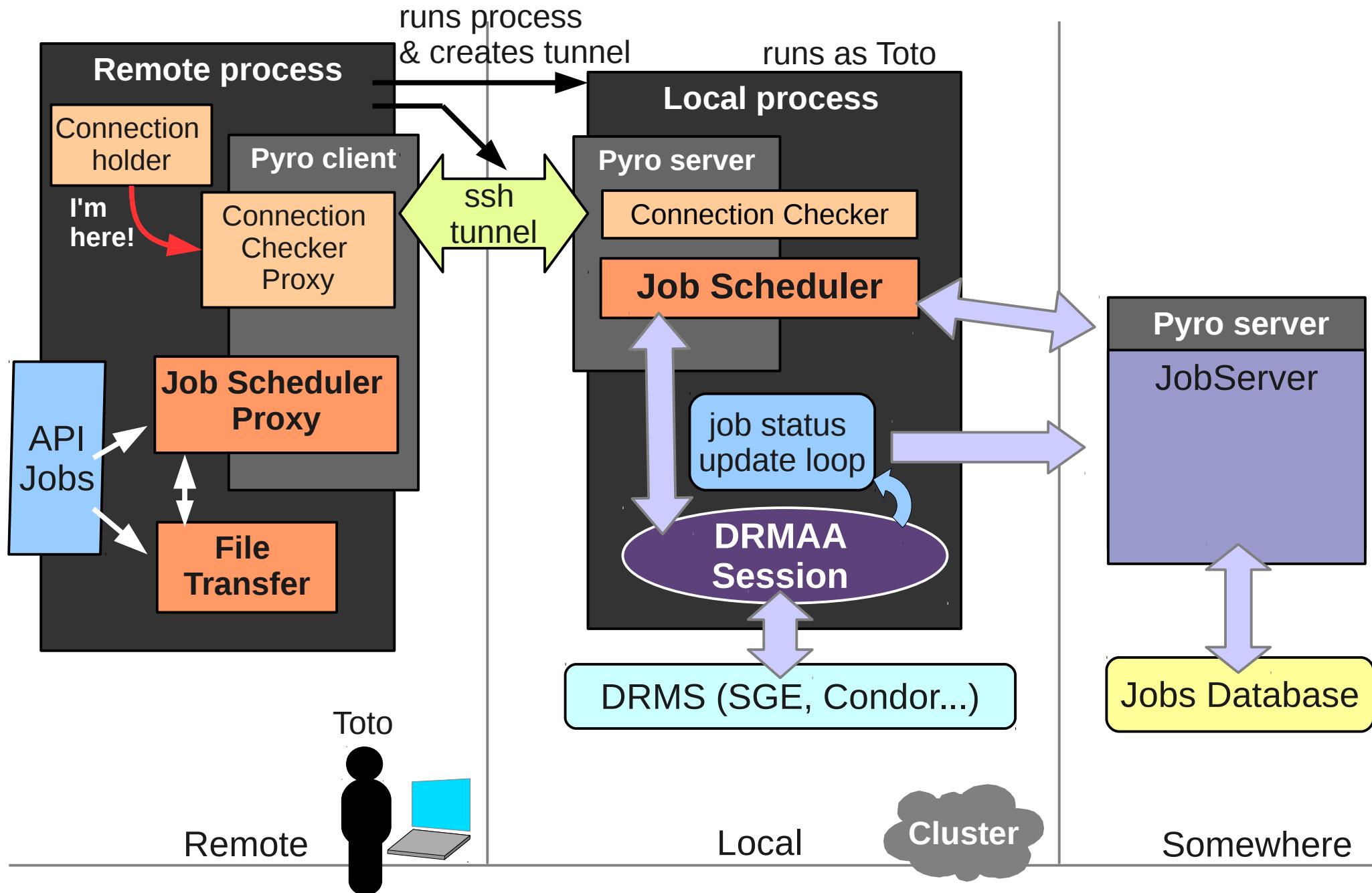
# Solution développée - Architecture



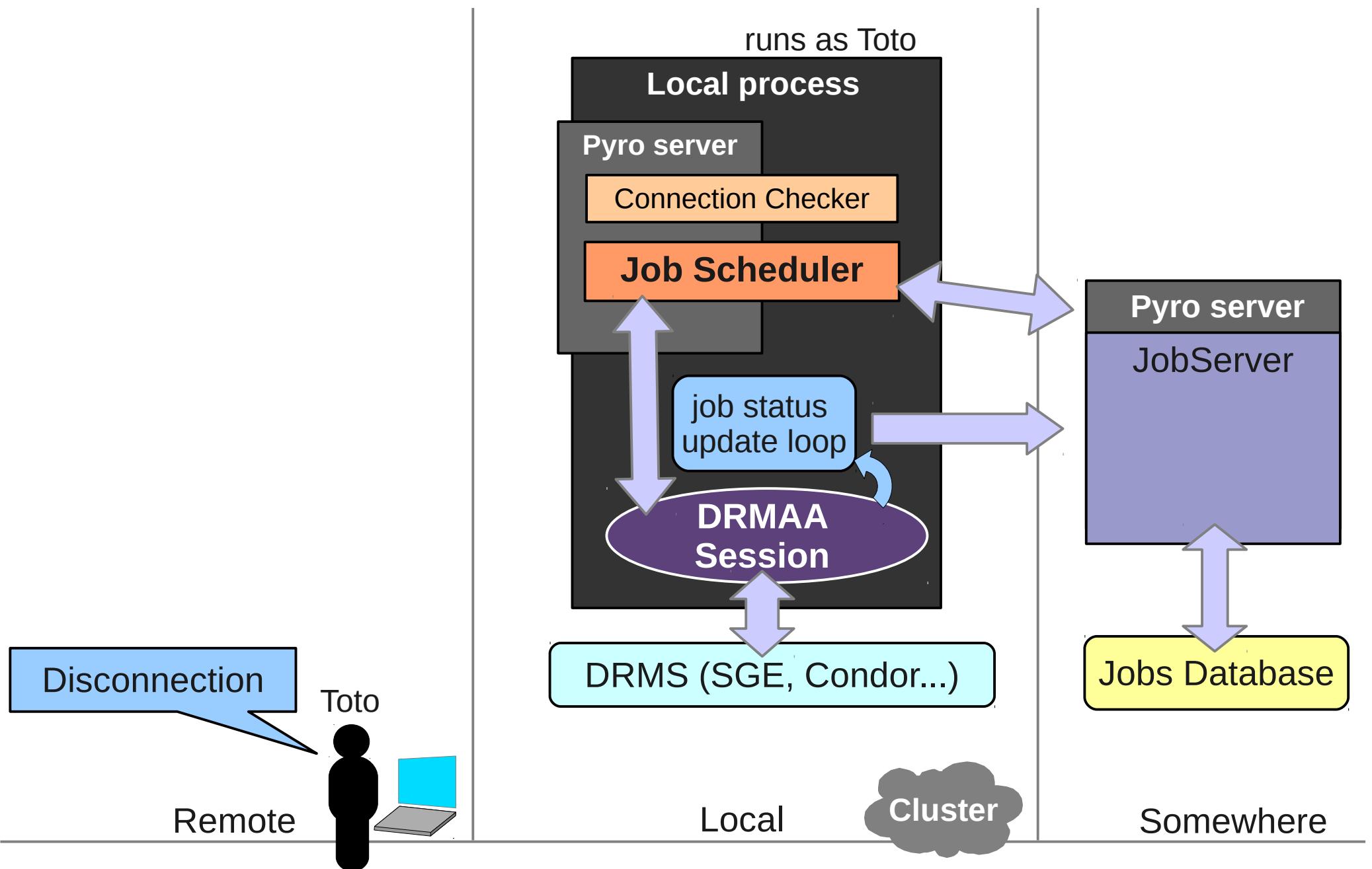
# Solution développée - Architecture



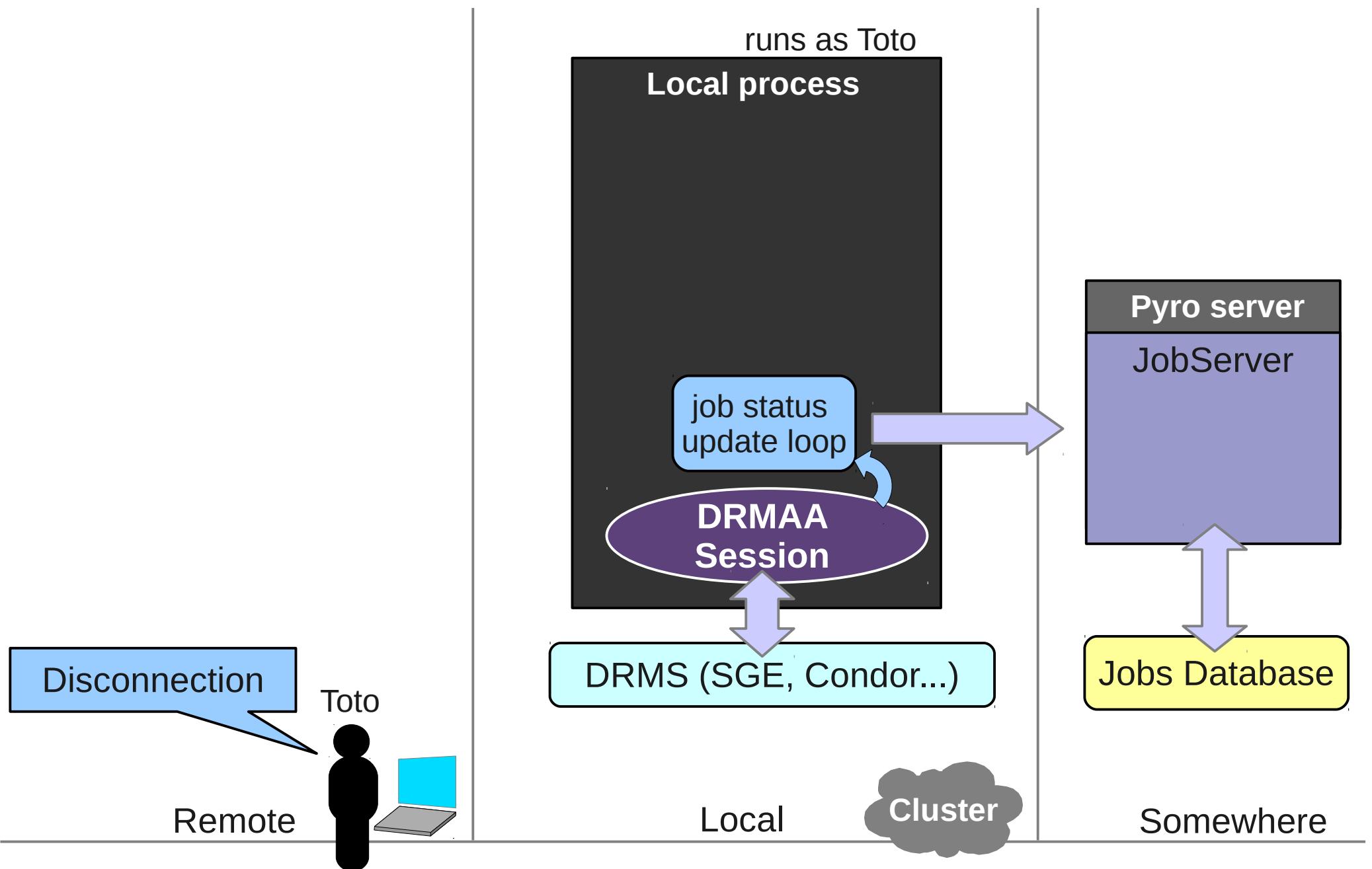
# Solution développée - Architecture



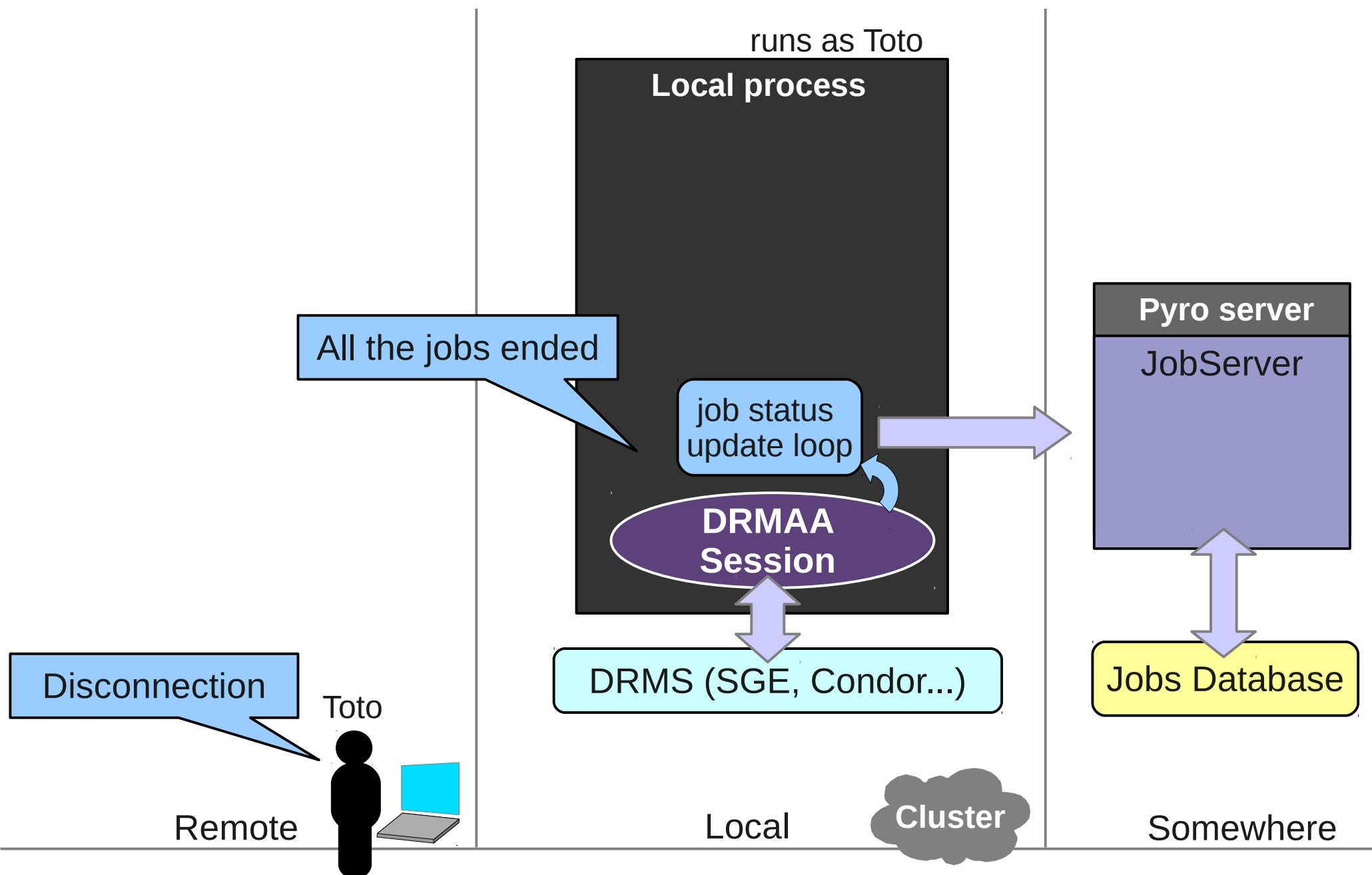
# Solution développée - Architecture



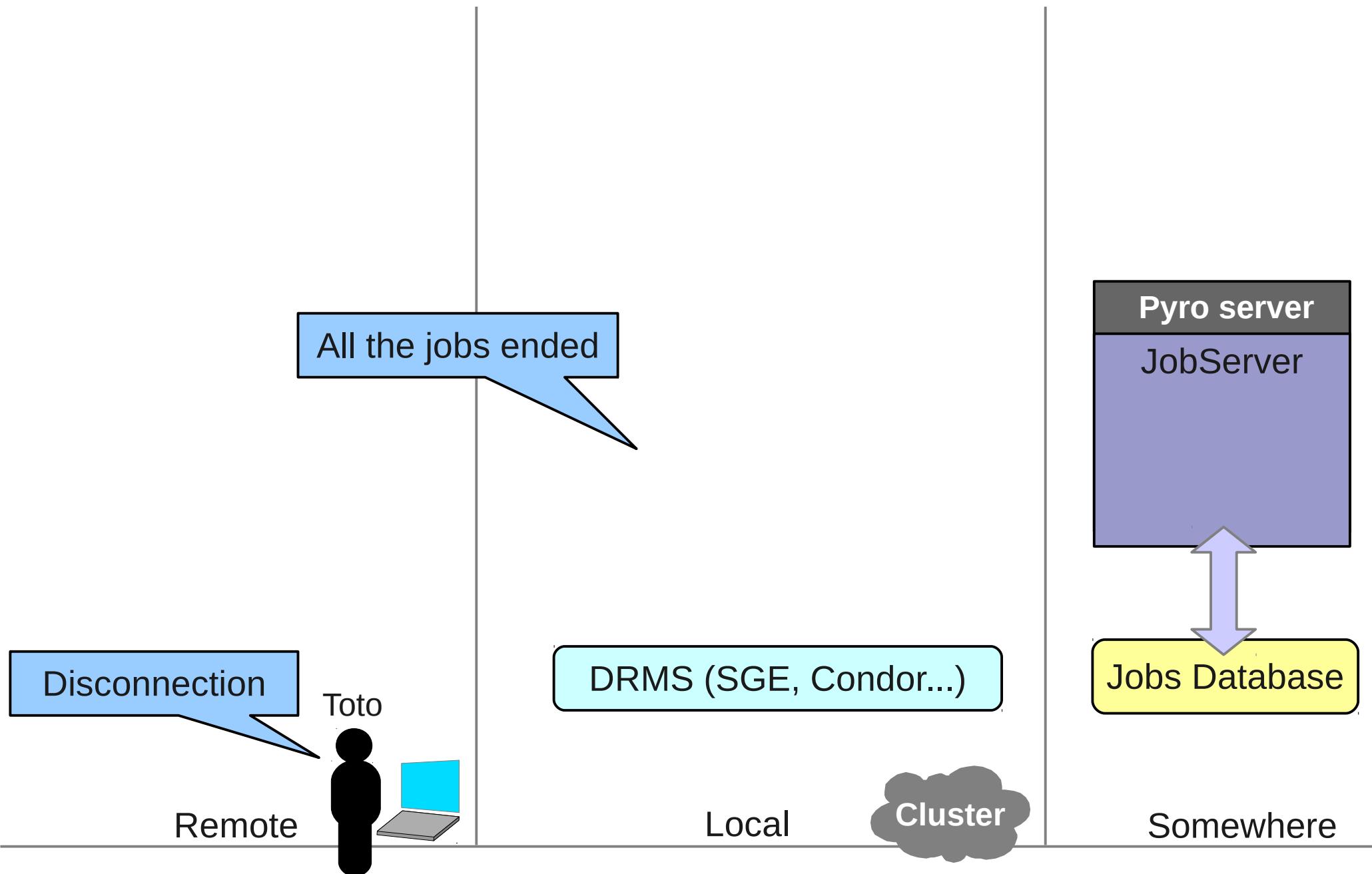
# Solution développée - Architecture



# Solution développée - Architecture



# Solution développée - Architecture



# Conclusion et perspectives

---

- ▶ Travail réalisé
  - Etude de l'existant / Tests / Installations / Conception / Architecture
  - Implémentation de l'API en python
  - Batterie de tests sur le cluster test avec SGE:
    - ✓ API complète en mode local et distant
    - ✓ soumission de jobs parallèles MPI
- ▶ En cours:
  - tests et debugging sur le cluster de test => DRMAA Condor
  - tests et debugging sur le cluster national CEA/DSV => DRMAA PBS/Torque
- ▶ A faire:
  - Amélioration du système de transfert de fichiers
  - Installation et tests au CEA/CCRT => DRMAA LSF & DRMAA PBS/Torque
  - Interface BrainVISA
  - Interface graphique pour le suivi et contrôle des tâches