

IA générative, Agents IA au MEM

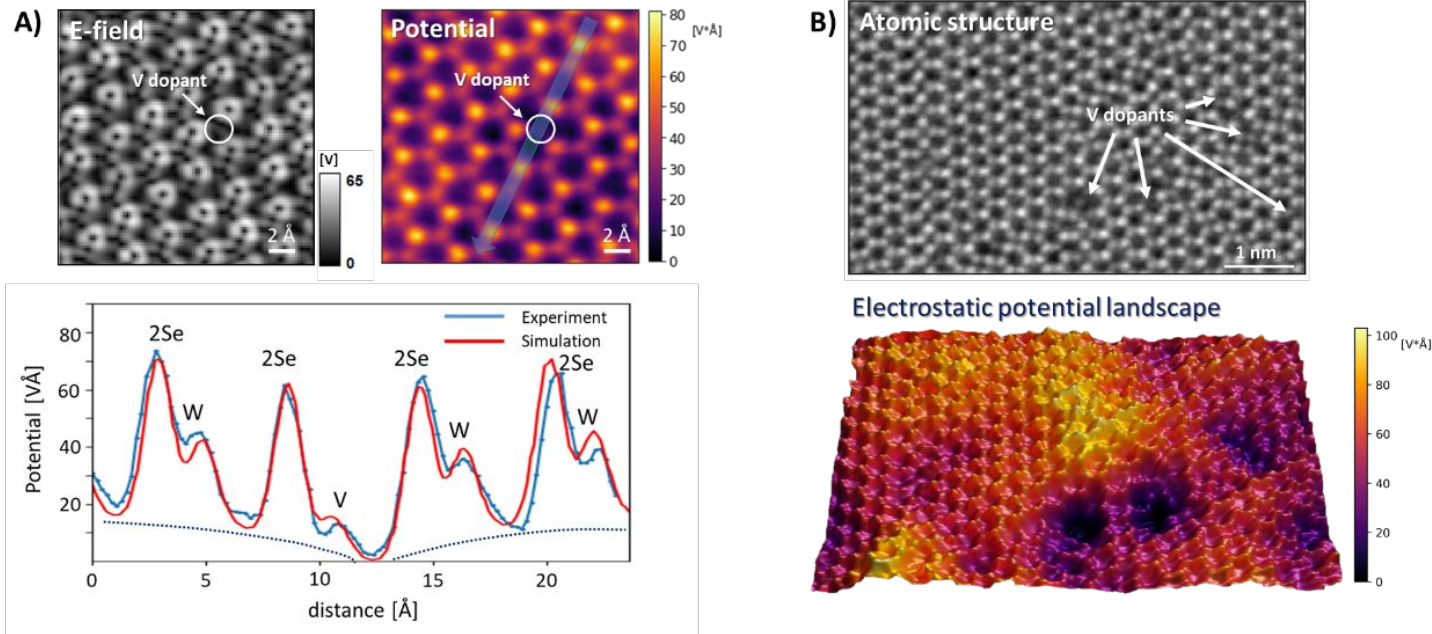
Modélisation et Exploration des Matériaux

Yoann Curé, Thierry Deutsch

CEA-GRENOBLE / DRF / IRIG / MEM

MEM Modélisation et Exploration de la Matière

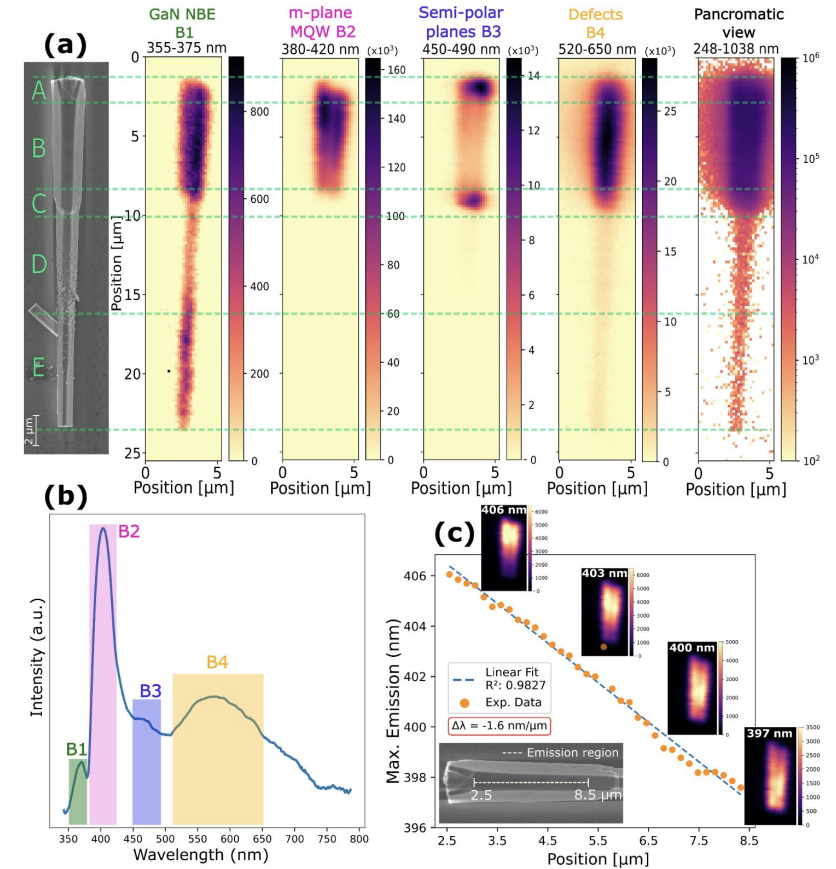
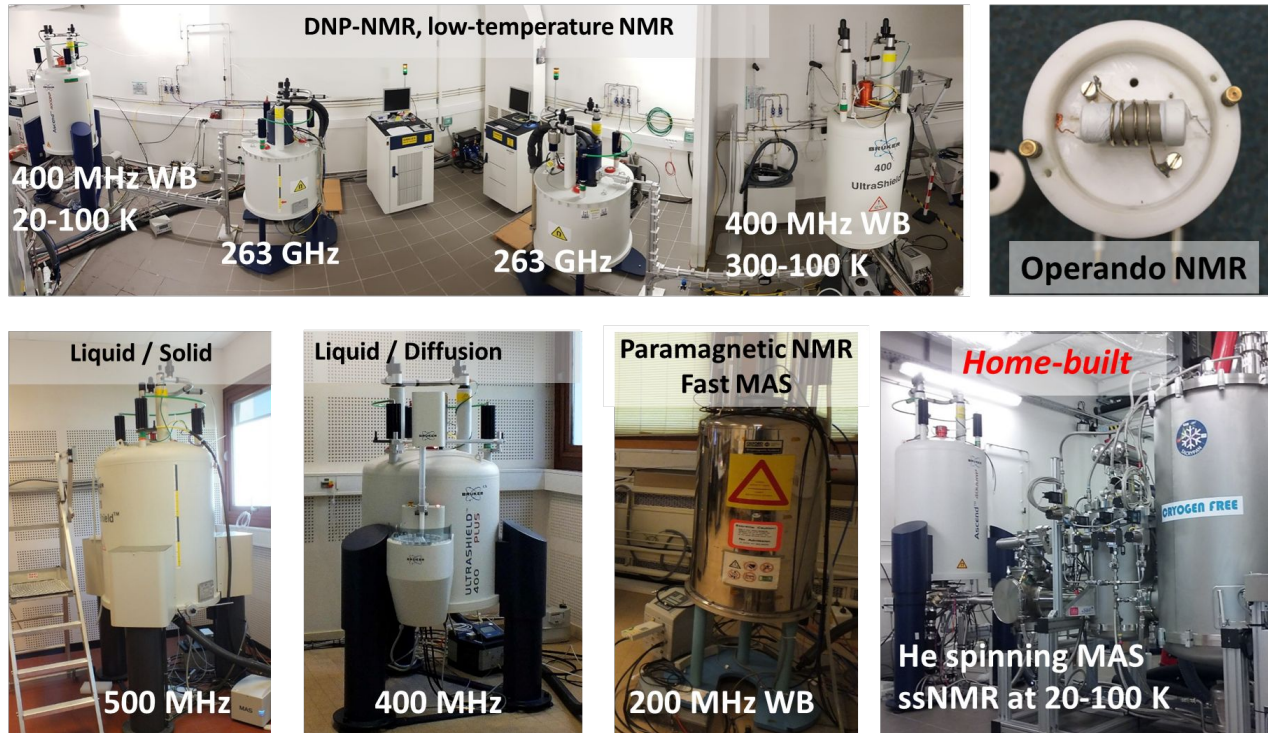
Études des matériaux par microscopies avancées



Magnétisme et Diffraction Neutronique

MEM Modélisation et Exploration de la Matière

Résonances magnétiques

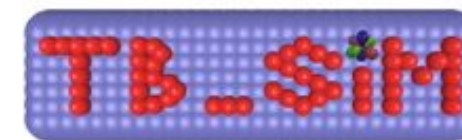


Nanostructures et rayonnement X

Simulation numérique (L_Sim)

L_Sim – Patrimoine de codes HPC (et IA)

Instrument Name	Team and Location	Field	User laboratories	Year of acquisition or renovation	Time before renovation
BigDFT DFT software	L_SIM, gitlab	Electronic structure of materials and proteins	More HPC and biology	2005	Need for adaptation and optimisation to new processors
beDeft software (GW-Bethe Salpeter methods)	L_SIM, gitlab maison de la Simulation	Spectroscopy of molecules	Chemists and physics	2016	
TB_Sim	L_SIM, tuleap CEA	Multiphysics of qubits Si or Ge	PHELIQS, LETI, Delft	2005	Adding physics and optimisation



Développement de logiciels open-source

4 logiciels open-source en analyse de données expérimentales

- MetroCarac, outils de caractérisation métrologie de la PFNC, <https://github.com/CEA-MetroCarac>,
- PyCorrectedEmissionCT (traitement des données de tomographie et d'imagerie fantôme, pour la fluorescence X), <https://github.com/cicwi/PyCorrectedEmissionCT>,
- LaueTools, logiciel d'analyse des données de microdiffraction de Laue, <https://github.com/BM32ESRF/lauetools>,
- laueimproc, prétraitement d'images pour l'analyse des données de Laue, <https://github.com/BM32ESRF/laueimproc>.

Bibliothèques open-source et outils de simulation

- CheSS, un module pour la DFT $O(N)$, <https://hal.science/hal-05010584>,
- **remotemanager**, module python de gestion pour traiter l'exécution de fonctions sur des machines distantes, https://gitlab.com/L_sim/remotemanager,
- Galn, calcul des intégrales de Coulomb de recouvrement, <https://hal.science/hal-05012409>,
- V_Sim, visualisation de structures atomiques (cristaux, joints de grains et molécules), <https://hal.science/hal-05010468>,
- Futile, bibliothèque facilitant l'interopérabilité entre les E/S du Fortran et les scripts python, <https://hal.science/hal-05010623>,
- psolver, un solveur de Poisson flexible basé sur des ondelettes, <https://hal.science/hal-05012273>,
- liborbs, une bibliothèque pour les traitements des orbitales en DFT, <https://hal.science/hal-05012279>,
- PyBigDFT, Modules python conçus pour le prétraitement et le post-traitement des fichiers d'entrée et de sortie BigDFT, <https://hal.science/hal-05012294>,
- atlab, bibliothèque de structures de données et d'outils pratiques pour traiter les quantités atomiques, <https://hal.science/hal-05012322>,
- BigDFT, code DFT rapide, précis et flexible pour la simulation ab initio, <https://hal.science/hal-05012338>,
- **beDefT**, code massivement parallèle pour étudier les propriétés électroniques des systèmes à grande échelle, <https://gitlab.maisondelasimulation.fr/beDefT>.

Remotemanager

Run Local Python Code on Remote Clusters—Transparently

Eliminate manual SSH logins, batch script creation, and file transfers. Execute functions on remote supercomputers directly from your local Jupyter notebook or Python script.



No Manual SSH

Executes Python tasks remotely without manually connecting or pushing code files.



Auto Scheduler Handling

Automatically generates and submits Slurm, SGE, PBS, or custom cluster job scripts.



Seamless Data Sync

Serializes function arguments, sends inputs, and returns outputs directly to local memory.

l_sim.gitlab.io/remotemanager

How remotemanager Works in Practice

Four simple steps to offload heavy Python calculations to supercomputing resources.

1 Connect to Target Host

Define cluster SSH credentials using the **URL** object.

2 Wrap Your Function

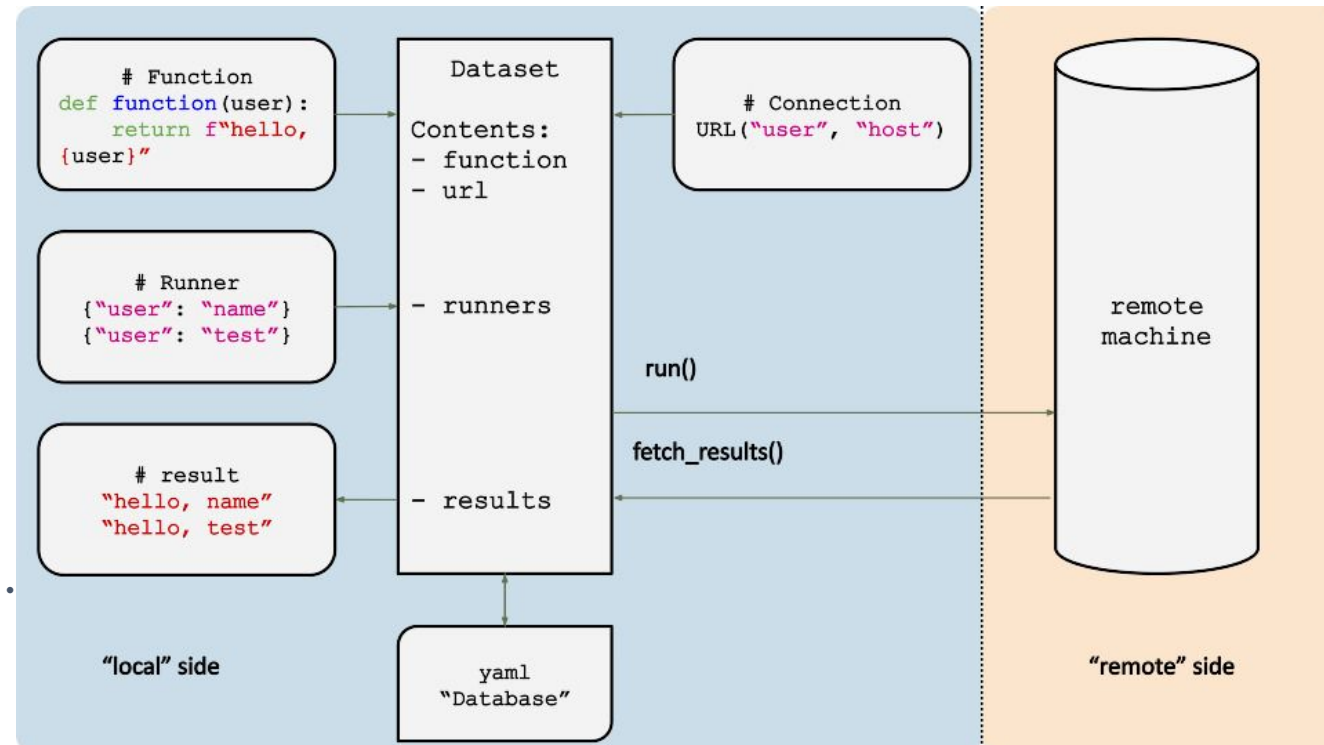
Create a **Dataset** with your function and scheduler target (e.g. Slurm).

3 Run Remotely

Append input arguments and call **ds.run()**. Files and jobs are dispatched.

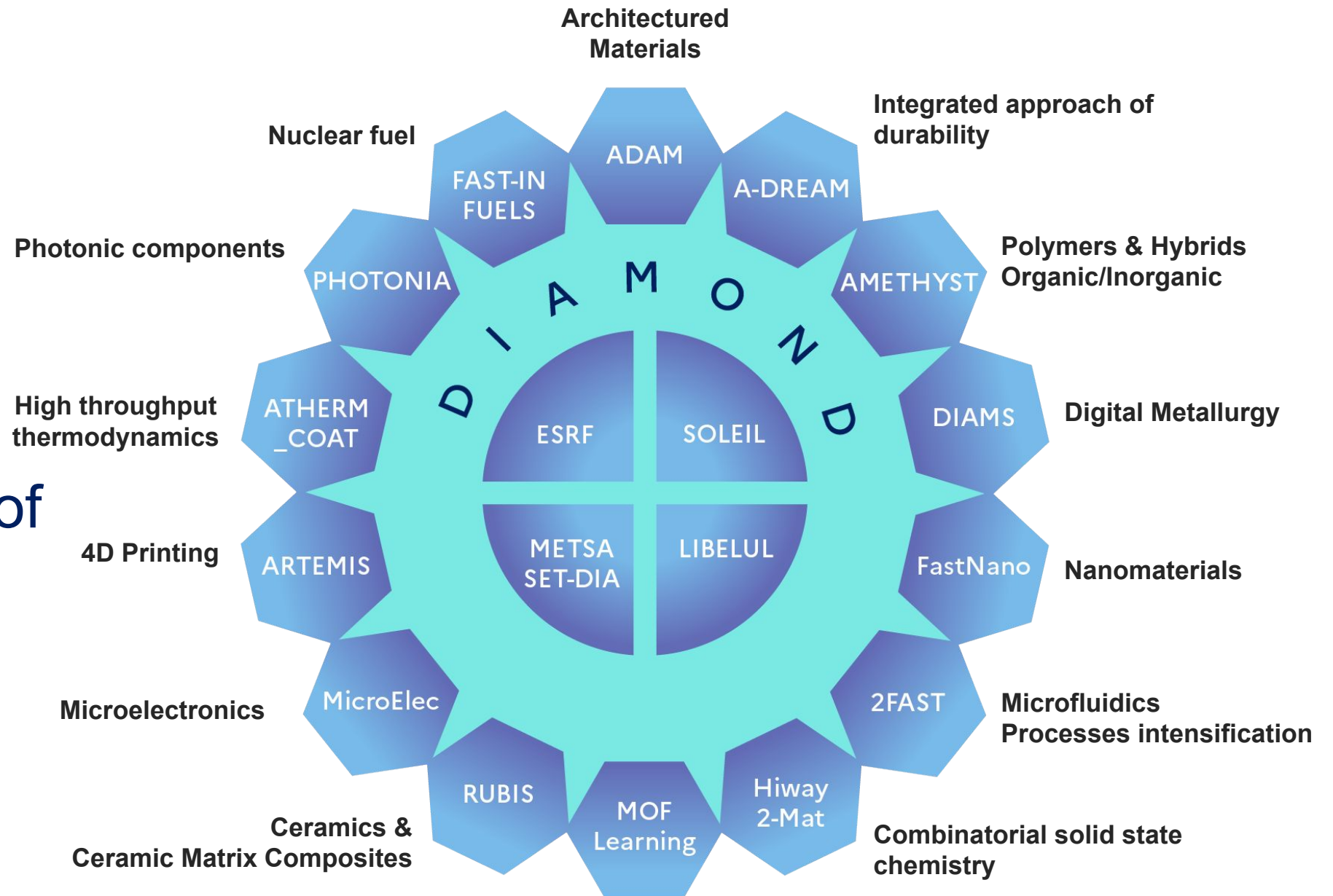
4 Fetch Results

Call **ds.fetch()** to recover returned values straight into local variables.



PEPR DIADEM

Discovery
Acceleration for
the Deployment of
Emerging
Materials





The DIADEM digital ecosystem (DIAMOND)



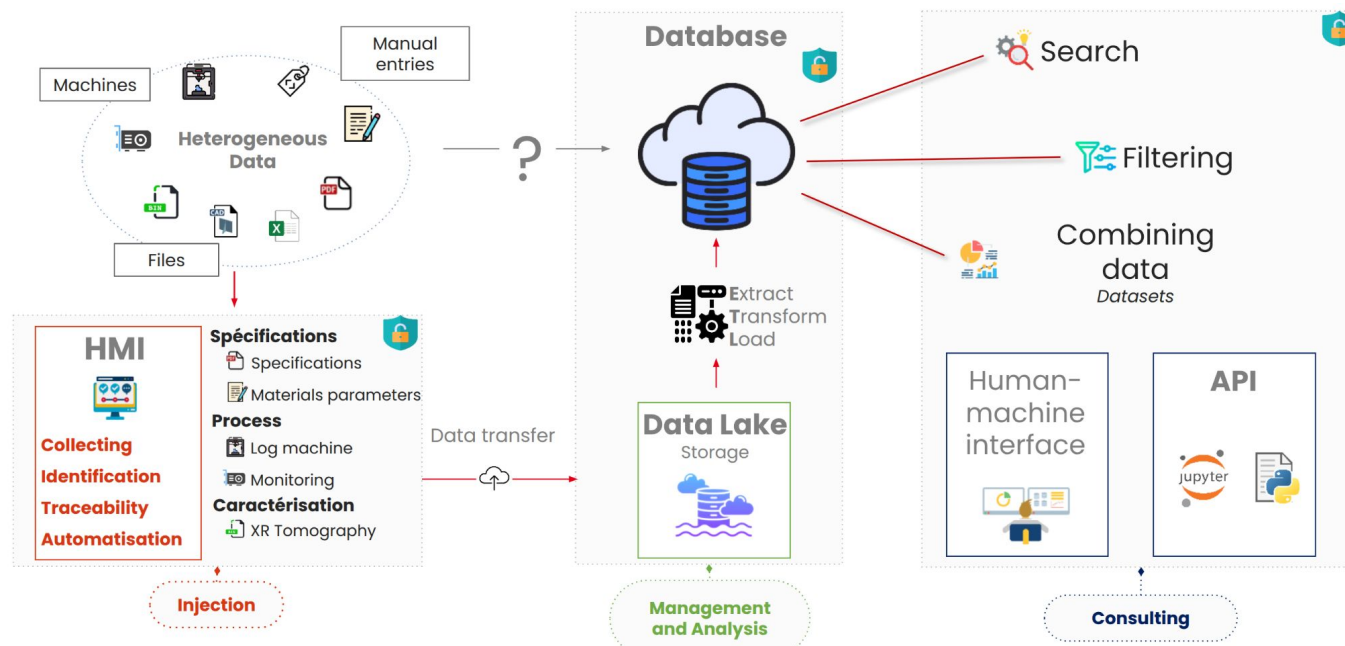
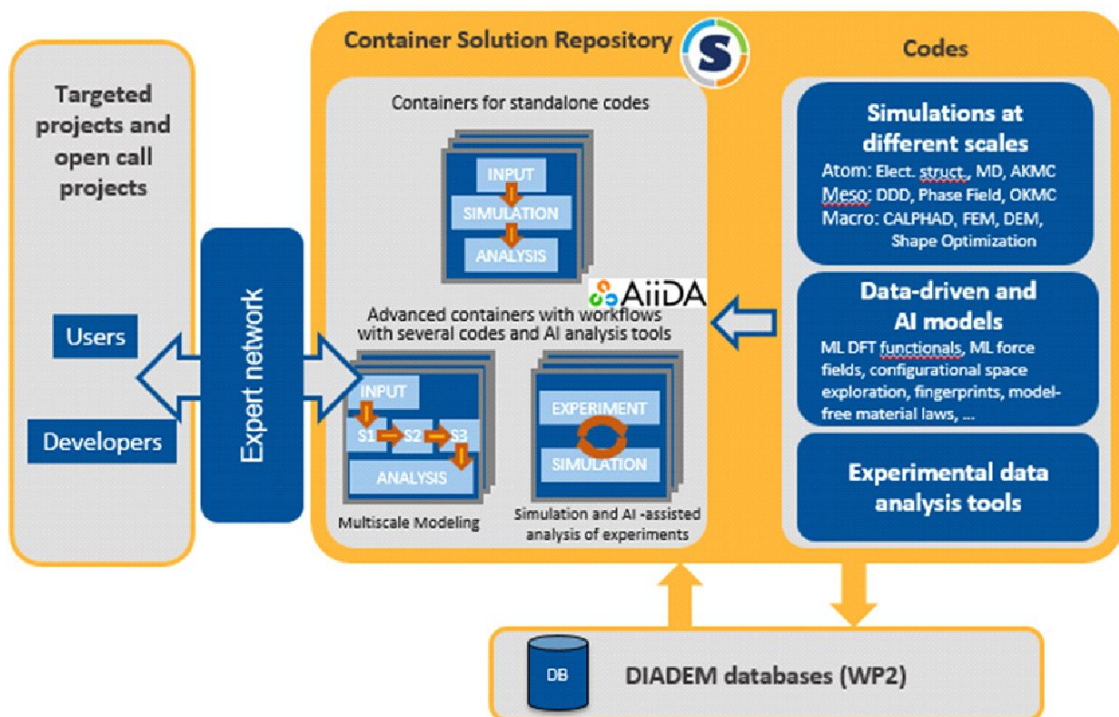
DIAMOND-CW CODE & WORKFLOWS

The long-term repository for computational codes and automated multi-scale workflow orchestration.



DIAMOND-DM DATA MANAGEMENT PLATFORM

The sovereign platform dedicated to ingestion, semantic structuring, and sharing of physical research data.



Exa-AToW – Cross-facilities workflow deployment

DIADEM and SKA are use cases of Exa-AToW

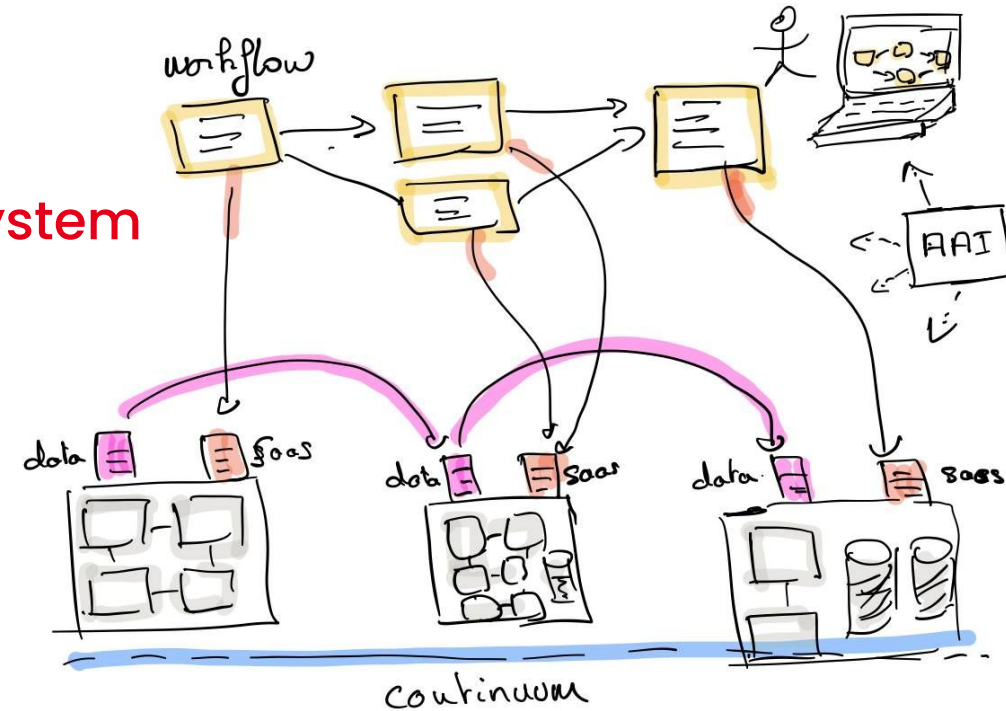
- Project aims: To orchestrate scientific workflows across federated infrastructures
 - o Conceptualized as a collaborative system of systems

- Central Challenge

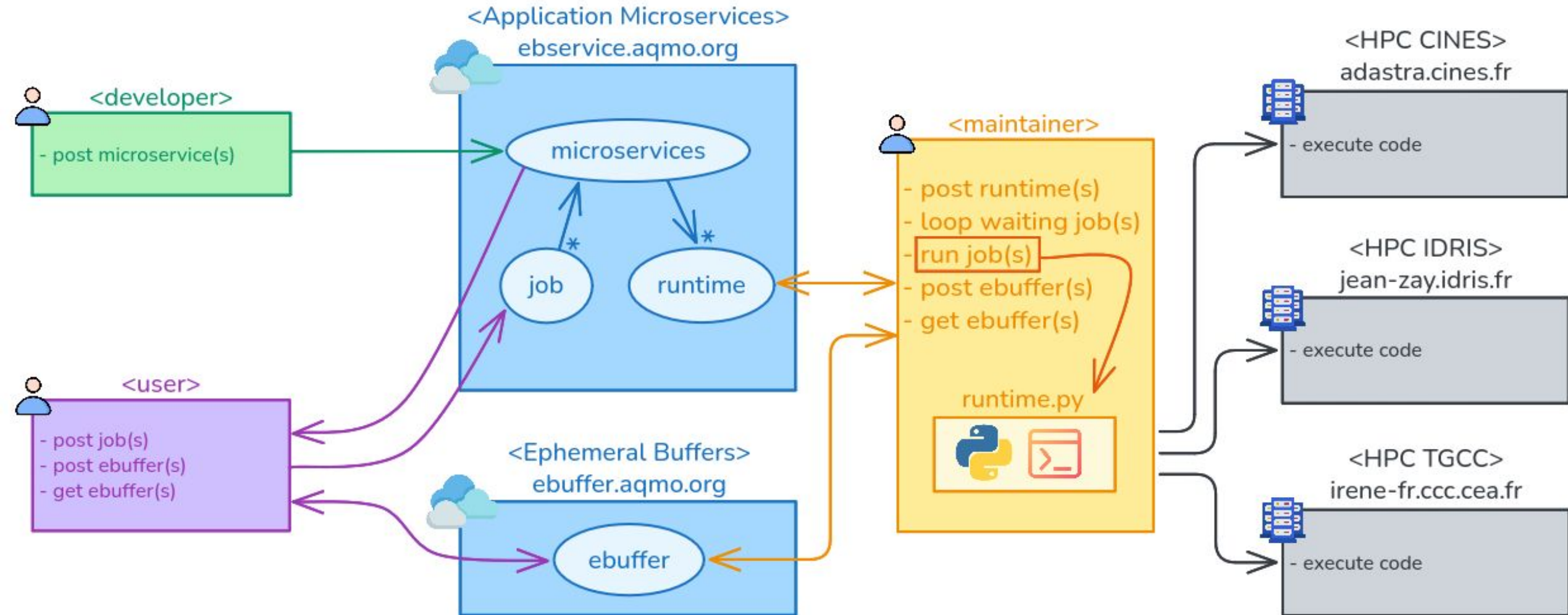
- o Ensure required data readily available on the system
- o Data logistics crucial for efficient workflow management

- Basic non-centralized approach

- Minimalist approach

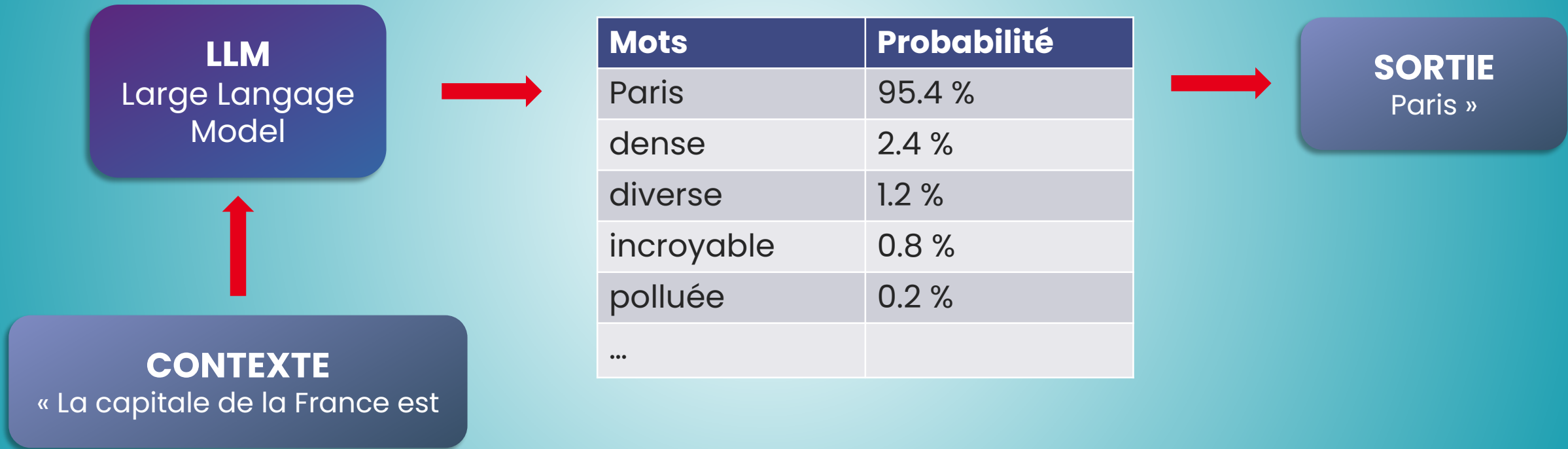


Exa-AToW architecture (applications as a service)

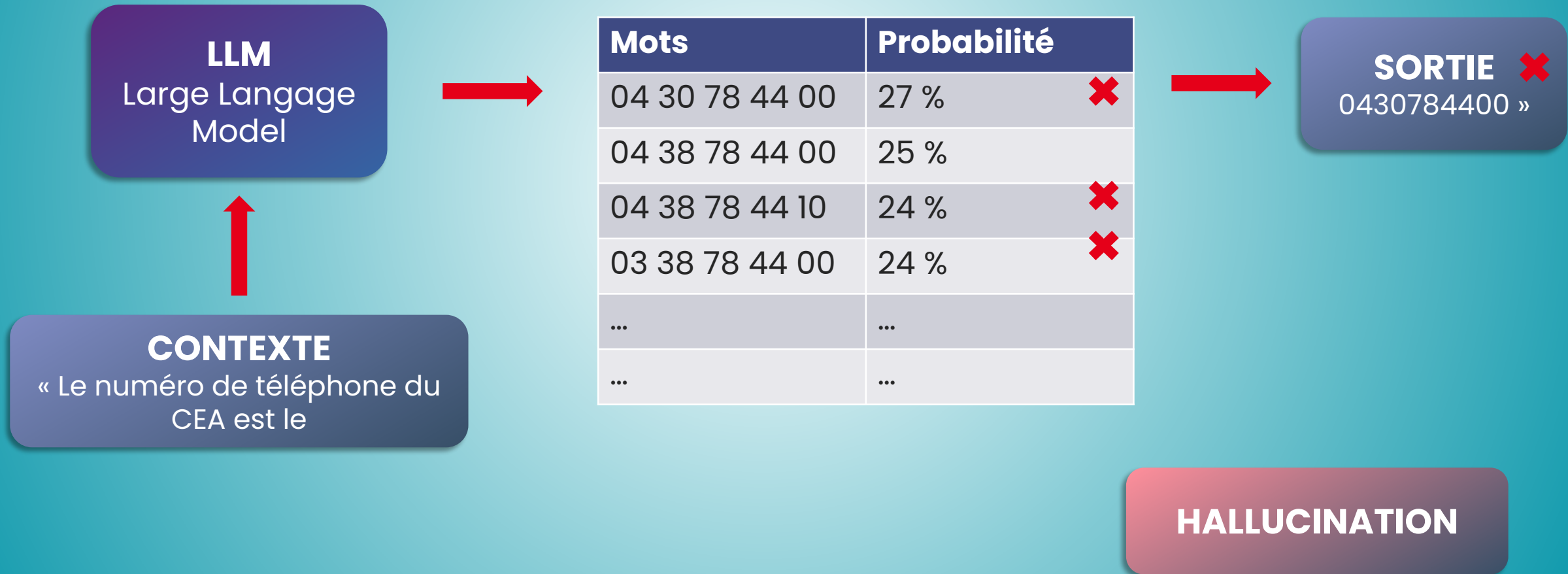


From Large Language Model to agent

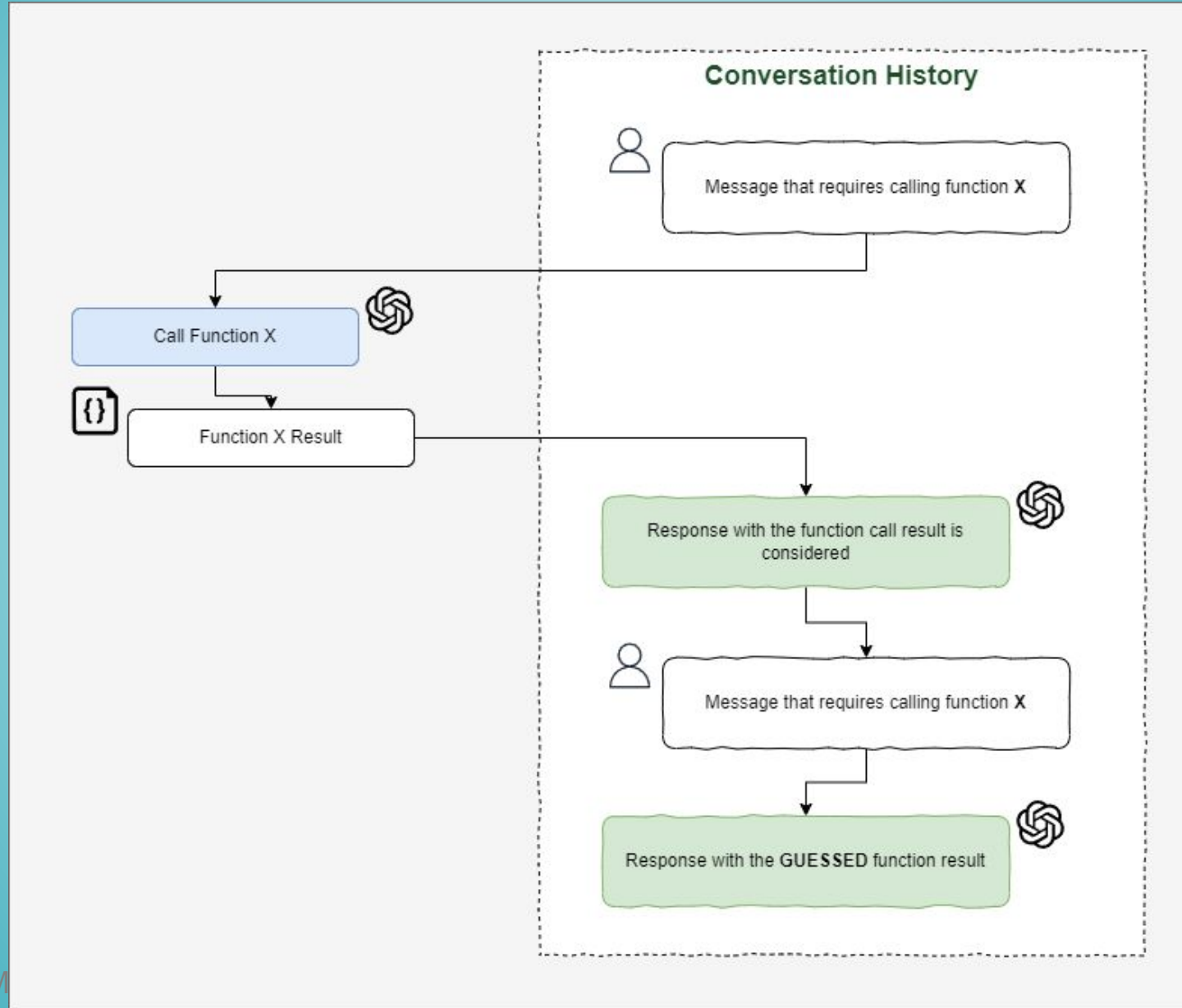
LLM



LLM



Calling function



Calling function

```
def book_hotel(location, checkin, checkout):  
    # Fonction pour réserver un hotel  
    return f"HOTEL RESERVEE {location} {checkin} {checkout}"
```

Calling function

```
def book_hotel(location, checkin, checkout):  
    # Fonction pour réserver un hotel  
    return f"HOTEL RESERVEE {location} {checkin} {checkout}"
```

This function is never sent to IIm.
Therefore, a function descriptor is
needed.

Calling function

```
def book_hotel(location, checkin, checkout):  
    # Fonction pour réserver un hotel  
    return f"HOTEL RESERVEE {location} {checkin} {checkout}"
```

Function descriptor

```
{  
  "name": "book_hotel",  
  "description": "Book a hotel for given dates",  
  "parameters": {  
    "type": "object",  
    "properties": {  
      "location": {  
        "type": "string",  
        "description": "The city"  
      },  
      "checkin": {  
        "type": "string",  
        "description": "Check-in date"  
      },  
      "checkout": {  
        "type": "string",  
        "description": "Check-out date"  
      }  
    },  
    "required": ["location", "checkin", "checkout"]  
  }  
}
```

Calling function

```
messages = [  
    {"role": "system", "content": " You are an assistant"},  
    {"role": "user", "content": " I would like to book a hotel"}  
]
```

```
response = openai.chat.completions.create(  
    model="gpt-4o",  
    messages=messages,  
    functions=[descriptor function],  
    function_call="auto",  
    stream=True,  
)
```

Calling function

Available functions : `book_hotel`

→ All required parameters ?

→ No

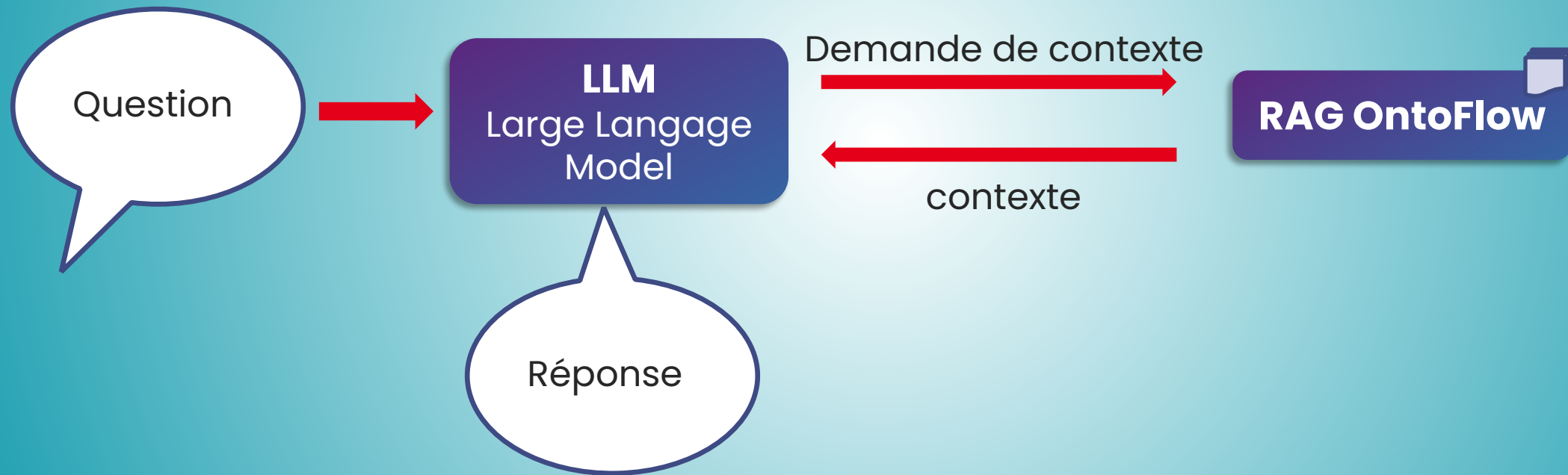
Ask a question to complete all required parameters

Response → 'Where would you like to book a hotel, and for which dates?'

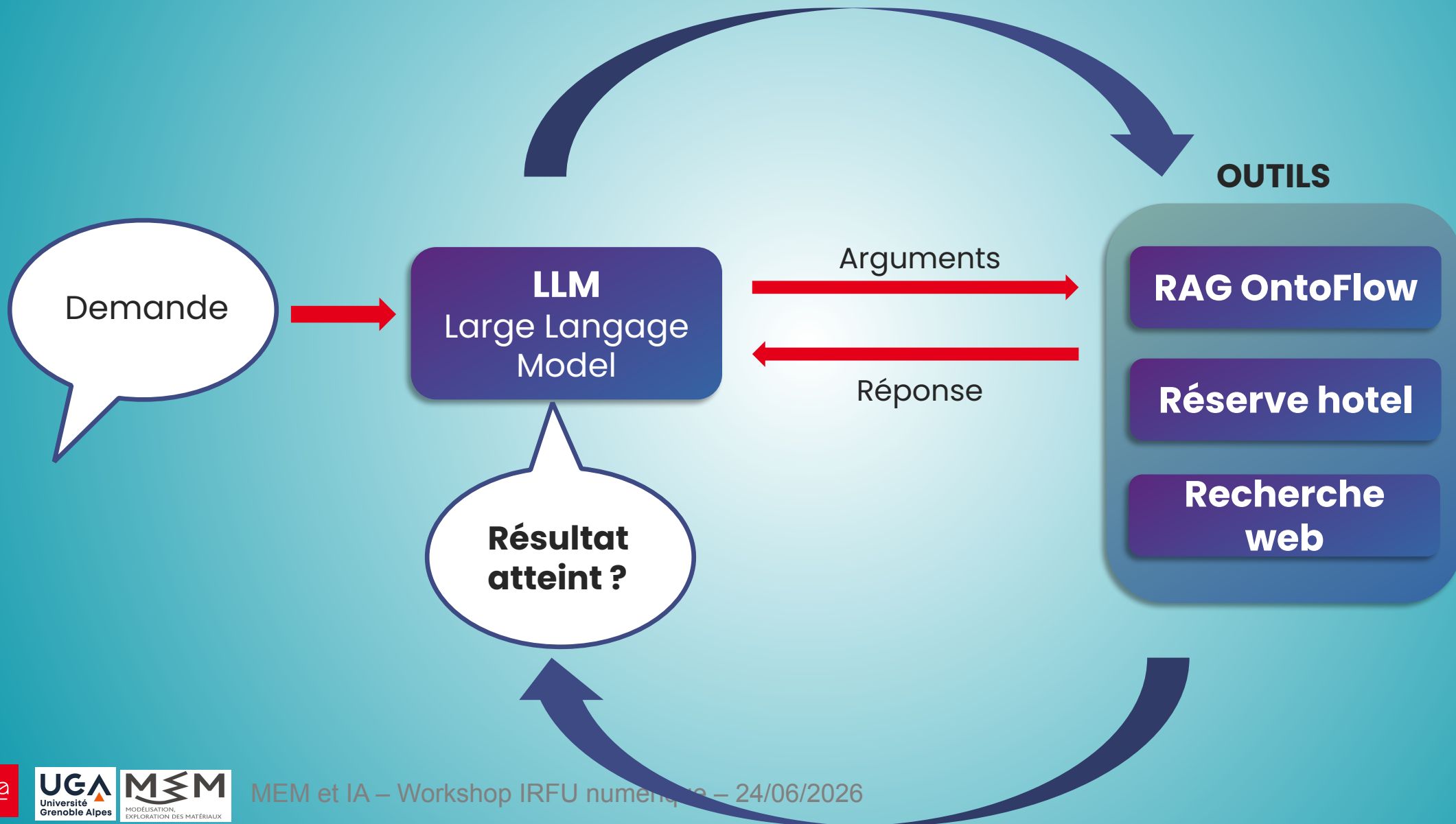
→ Yes

Execute function `book_hotel` with parameters

Agent IA



Agent IA



Agent IA

The Think → Act → Observe loop

Agent IA



Agent IA

The Core Reasoning System



What is the capital of France? -> Paris
Static knowledge

Agent IA

The connected problem solver (Think + Act)



What's the weather like in Paris?

Call tools 'get_weather' with argument 'Paris'

Make a good sentence to response

Agent IA

The strategic problem solver (Think + Act + Observe)



Find a good coffee shop halfway between my office at 1600 Amphitheatre Parkway, Mountain View, and my client's office at 1 Market St, San Francisco.

1. Think: "I must first find the halfway point."

- **Act:** Call the **Maps** tool with both addresses.

- **Observe:** "The halfway point is Millbrae, CA."

2. Think: "Now I must find coffee shops in Millbrae. The user asked for 'good' ones, so I will search for places with a 4-star rating or higher."

- **Act:** Call the **places** tool with **query="coffee shop in Millbrae, CA", min_rating=4.0**. (This is context engineering - it automatically created a new, focused search query from the previous step's output).

- **Observe:** "The search returns 'Millbrae Coffee' and 'The Daily Grind'."

3. Think: "I will synthesize these results and present them to the user."

Agent IA

The rise of collaborative multi-agent systems (Think + Act + Observe) x N



Agents treat other agents as tools. Imagine a "Project Manager" agent receiving a "Mission": "Launch our new 'Solaris' headphones."

1. Delegates to MarketResearchAgent: "Analyze competitor pricing for noise-canceling headphones. Return a summary document by tomorrow."

2. Delegates to MarketingAgent: "Draft three versions of a press release using the 'Solaris' product spec sheet as context."

3. Delegates to WebDevAgent: "Generate the new product page HTML based on the attached design mockups."

When ?

- Tools overload (more 20)
- LLM context overload

Agent IA

Self evolving agent



For my previous 'Solaris' headphones task, I realize it needs to monitor social media sentiment, but no such tool or agent exists on its team.

- 1. Think (Meta-Reasoning):** "I must track social media buzz for 'Solaris,' but I lack the capability."
- 2. Act (Autonomous Creation):** Instead of failing, it invokes a high-level AgentCreator tool with a new mission: "Build a new agent that monitors social media for keywords 'Solaris headphones', performs sentiment analysis, and reports a daily summary."
- 3. Observe:** A new, specialized SentimentAnalysisAgent is created, tested, and added to the team on the fly, ready to contribute to the original mission.

Bibliothèques

Slapski :

Permet d'utiliser des llms de provenances diverses : CEA (Maia), Google, Anthropic, OpenAI, Modèles locaux natifs, Ollama, Ragaren

https://gitlab.com/l_sim/slapski
V0.2.1

Agent-Slapski (harnais) :

Utilise slapski pour la boucle Think, Act, Observe
Permet de relier les agents à d'autres agents
(Equivalent A2A)
Structure les réponses

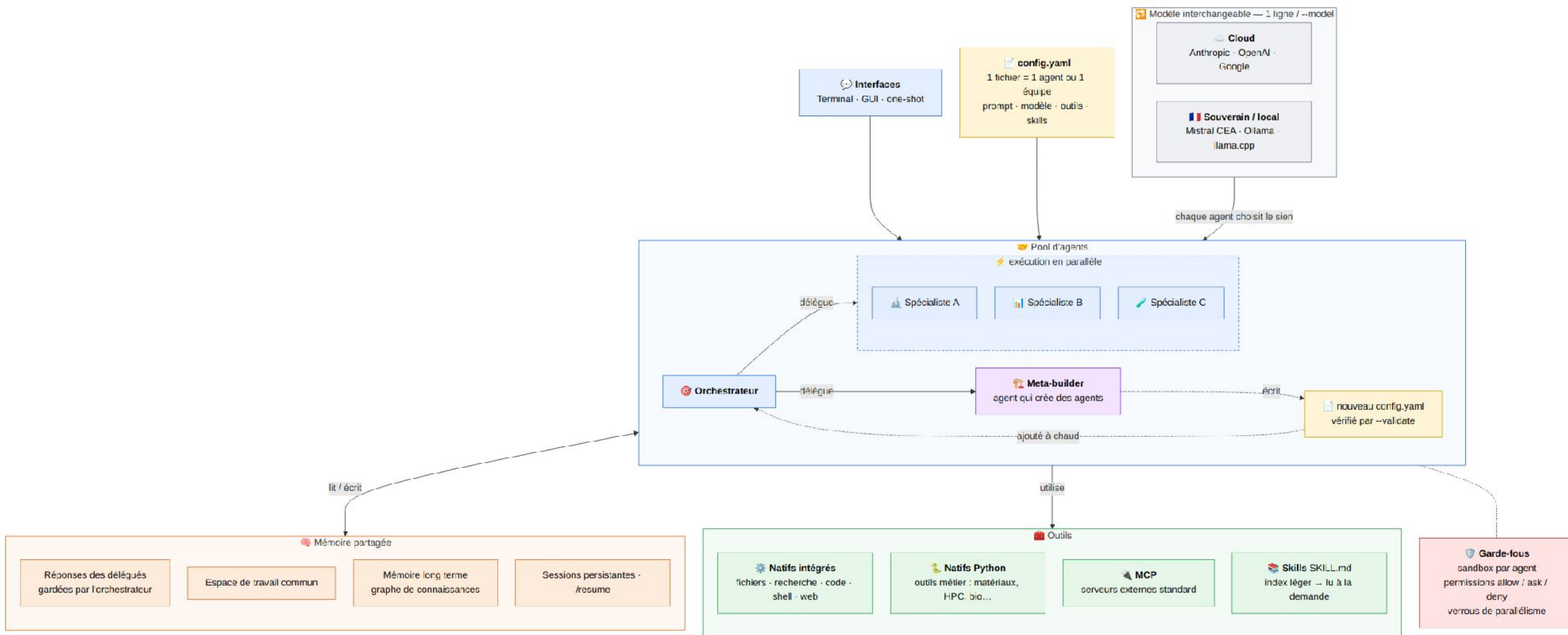
https://gitlab.com/l_sim/ia_tools/agent-slapski
V0.1.0

Always-on-memory :

Rag ontologique
Permet d'attacher de la mémoire à un petit modèle LLM

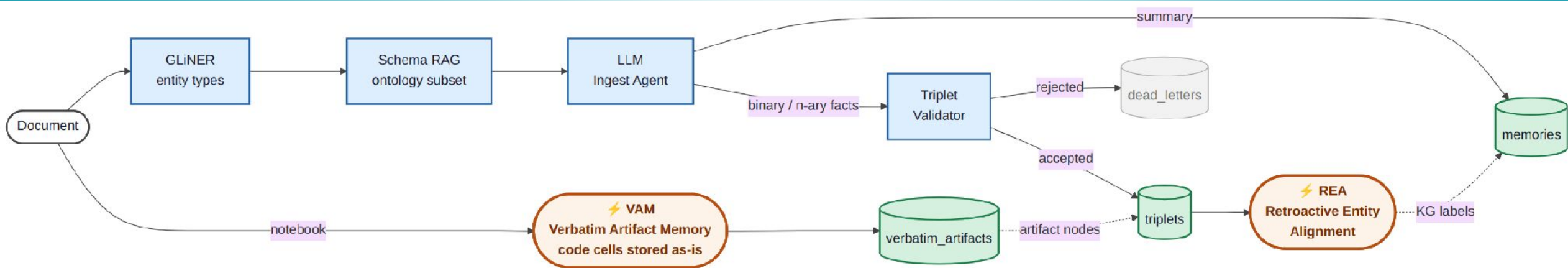
https://gitlab.com/l_sim/ia_tools/always-on-memory-slapski.git
V0.3.0.rc4

Agent-slapsci



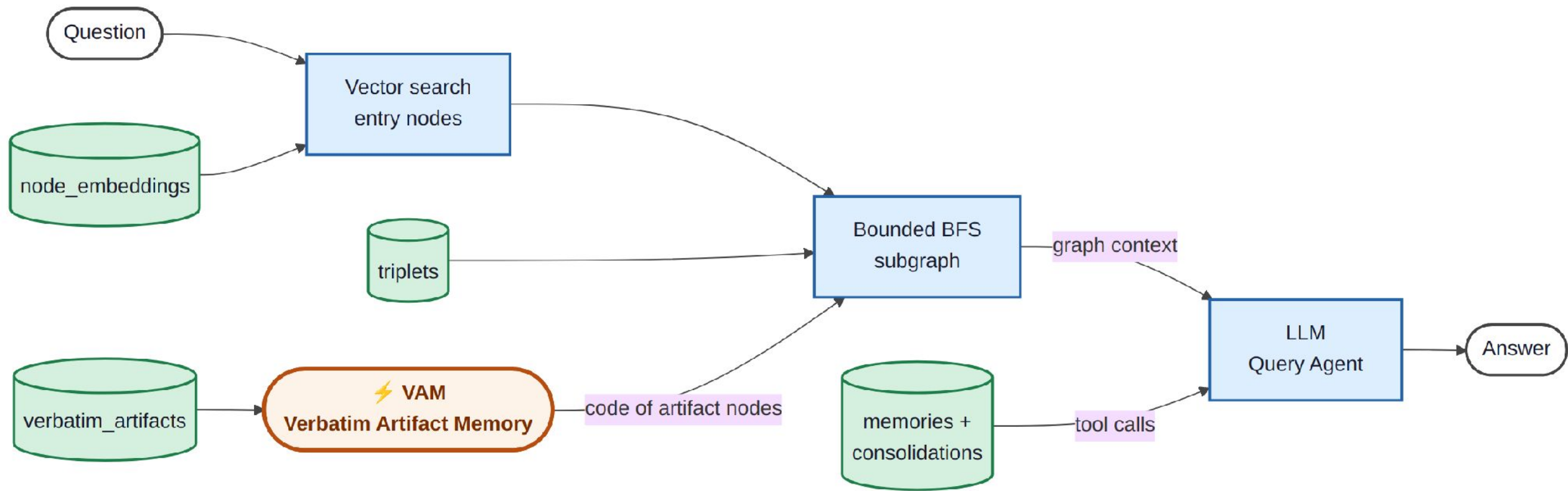
Always-on-memory

Ingest



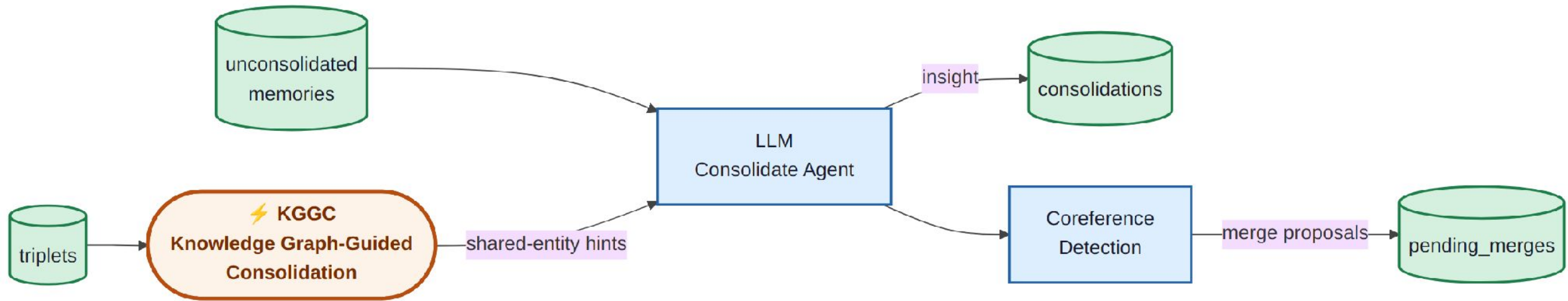
Always-on-memory

Query



Always-on-memory

Consolidate



Pack-kit Concept

Pack =

- Un RAG ontologie = always-on-memory
- Un agent (level 1 ou 2) = slapsci + agent-slapsci
- 2 fonctions : learn, serve

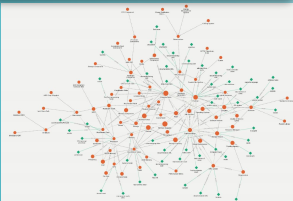
Pack-kit

Exemple : Centres HPC

Documents

- pages web
- tutoriel notebook
- doc, md,...

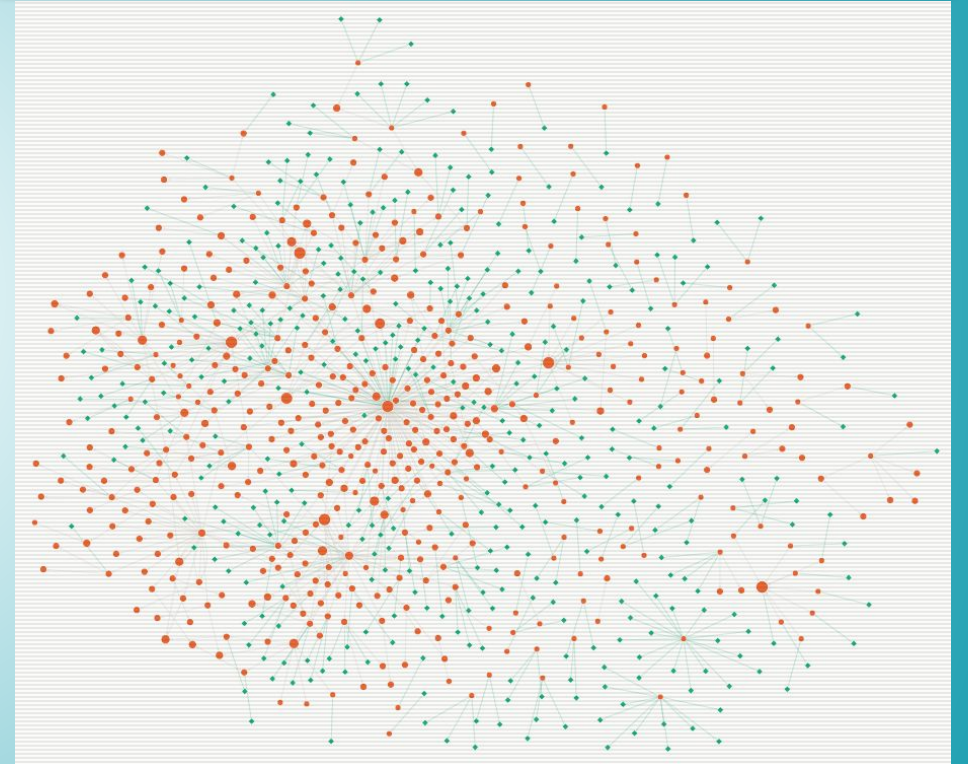
Schéma TTL



Pack-kit : learn

=

Ontologie instanciée, embedding sur noeuds

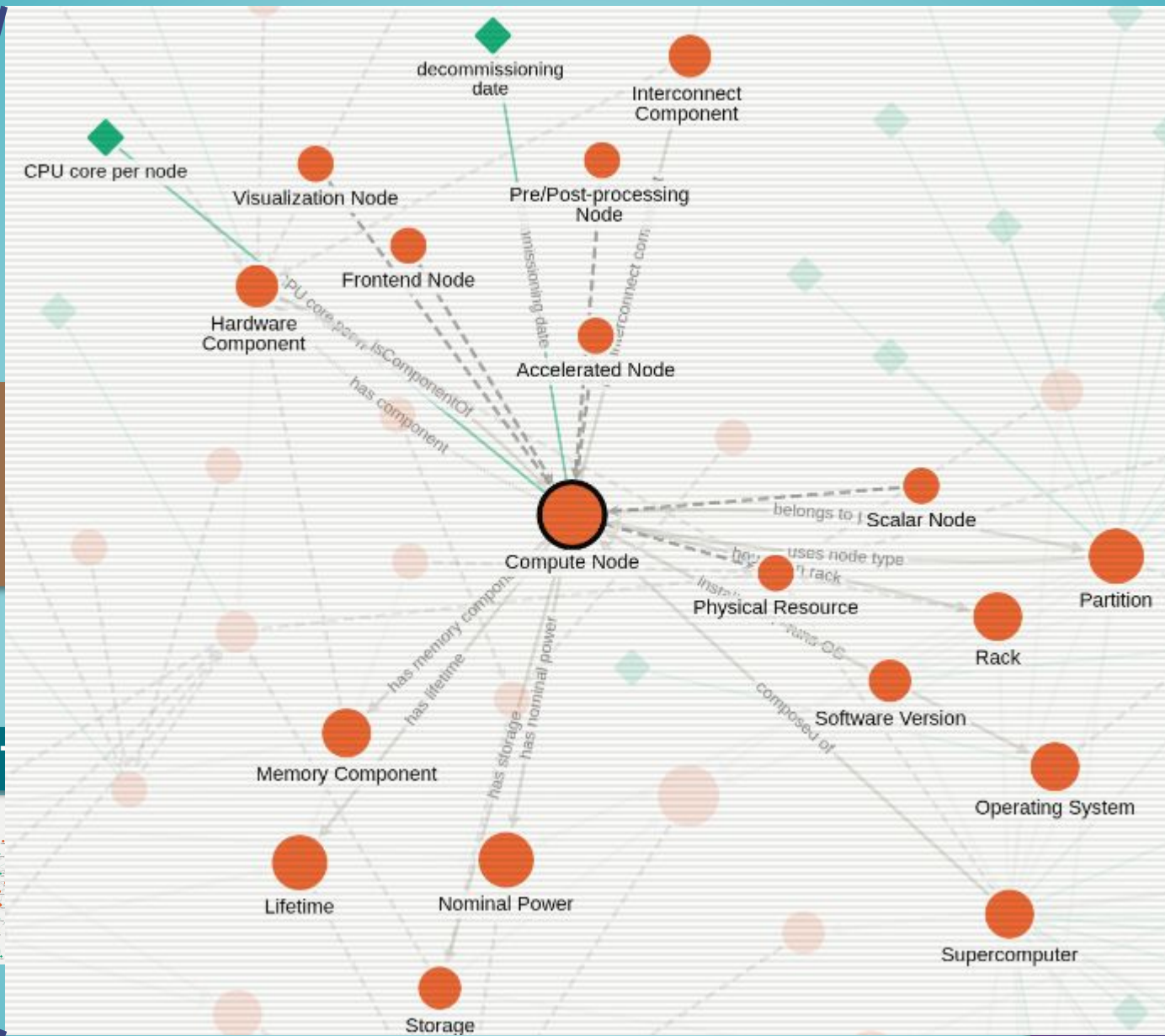
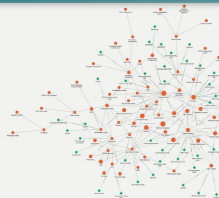


SQLite: fichier .db

Documents

- pages web
- tutoriel notebook
- doc, md,...

Schéma T

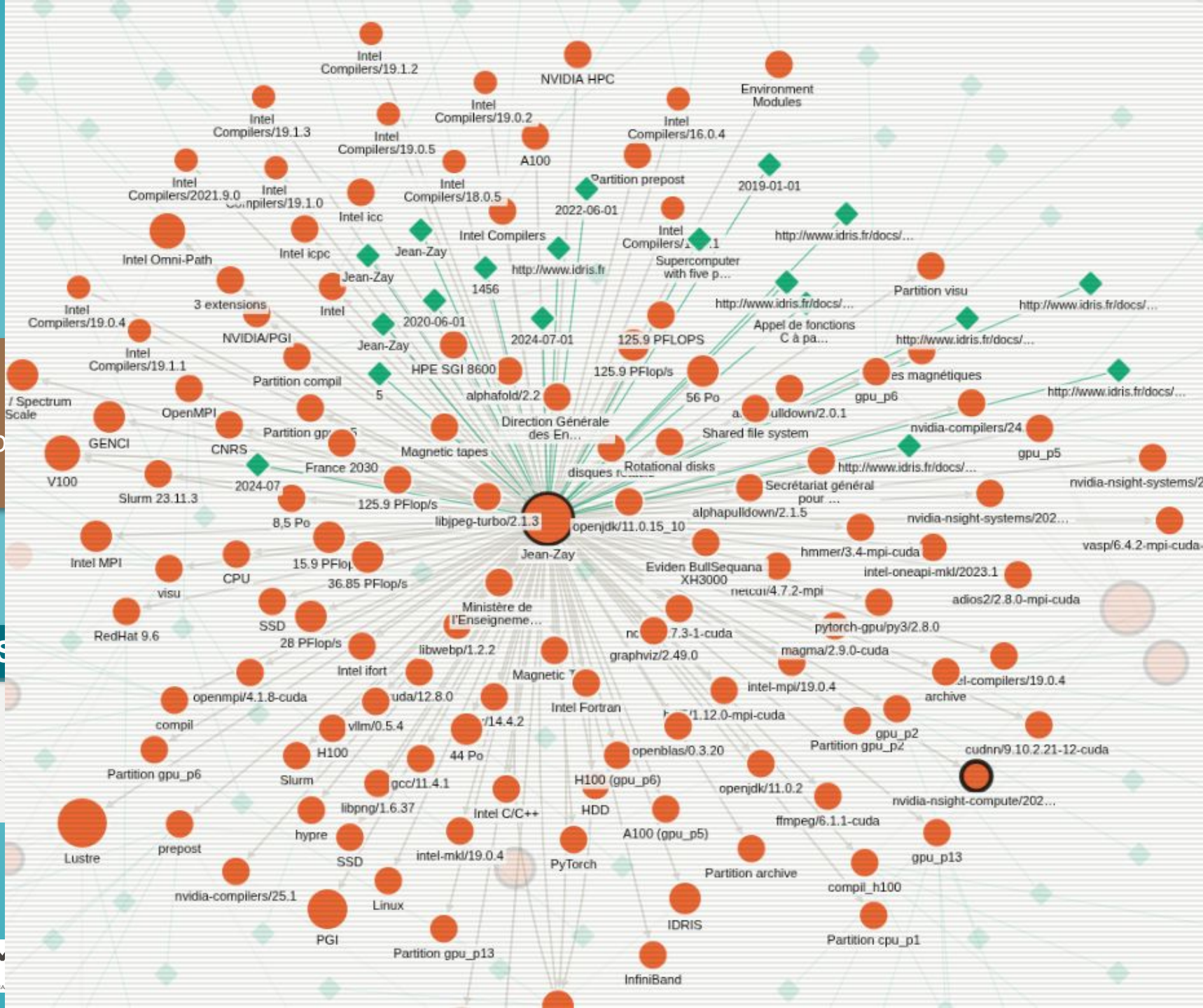


embedding sur noeuds

fichier .db

- pages web
- tutoriel notebook
- doc, md,...

- pages web
- tutoriel notebook
- doc, md,...



Ing sur noeuds

.db

Q Query

↓ Ingest

☆ Consolidate

EXPLORE

☐ Status

📄 Memories

📁 Artifacts

LAB

🔗 Graph

📊 Optimizer

SYSTEM

⚙️ Settings

Pack-kit : serve

QUESTION

Combien de partitions possède jean zay

Q Search Memory

Question: Combien de partitions possède jean zay

Basé sur mes mémoires et le graphe de connaissances fourni, je peux répondre à votre question :

Jean-Zay possède **9 partitions** :

Partitions CPU et Visualisation :

1. **cpu_p1** - 720 nœuds (nœuds scalaires)
2. **visu** - 5 nœuds (nœuds de visualisation)

Partitions GPU :

3. **gpu_p2** - 31 nœuds avec GPU NVIDIA Tesla V100 SXM2
4. **gpu_p13** - 396 nœuds avec GPU NVIDIA Tesla V100 SXM2
5. **gpu_p5** - 52 nœuds avec GPU NVIDIA A100 SXM4
6. **gpu_p6** - 364 nœuds avec GPU NVIDIA H100 SXM5

Partitions de Pré/Post-traitement :

7. **prepost** - 4 nœuds (pré/post-traitement)
8. **archive** - partitions d'archive

Partitions de Compilation :

9. **compil** - 4 nœuds (compilation)

Cette information est cohérente à travers plusieurs mémoires [Memory 51], [Memory 16], [Memory 18], qui détaillent les caractéristiques spécifiques de chaque partition.

Q Query

↓ Ingest

☆ Consolidate

EXPLORE

🗃 Status

📄 Memories

📁 Artifacts

LAB

🔗 Graph

🔧 Optimizer

SYSTEM

⚙ Settings

QUESTION

Combien de partitions possède jean zay

Q Search Memory

Question: Combien de partitions possède jean zay

Basé sur mes mémoires et le graphe de connaissances fourni, je peux répondre à votre question :

Jean-Zay possède **9 partitions** :

Partitions CPU et Visualisation :

1. **cpu_p1** - 720 nœuds (nœuds scalaires)
2. **visu** - 5 nœuds (nœuds de visualisation)

Partitions GPU :

3. **gpu_p2** - 31 nœuds avec GPU NVIDIA Tesla V100 SXM2
4. **gpu_p13** - 396 nœuds avec GPU NVIDIA Tesla V100 SXM2
5. **gpu_p5** - 52 nœuds avec GPU NVIDIA A100 SXM4
6. **gpu_p6** - 364 nœuds avec GPU NVIDIA H100 SXM5

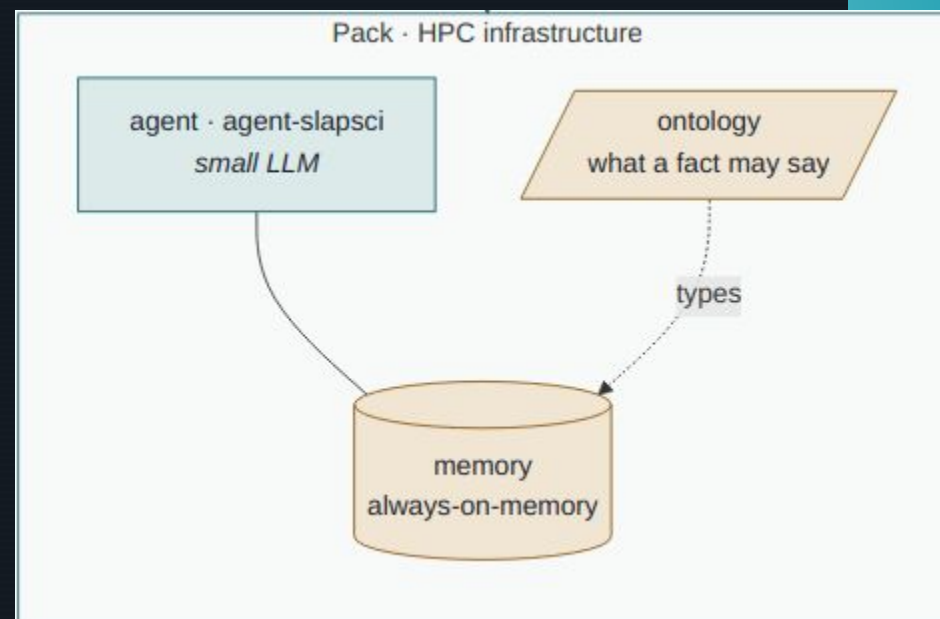
Partitions de Pré/Post-traitement :

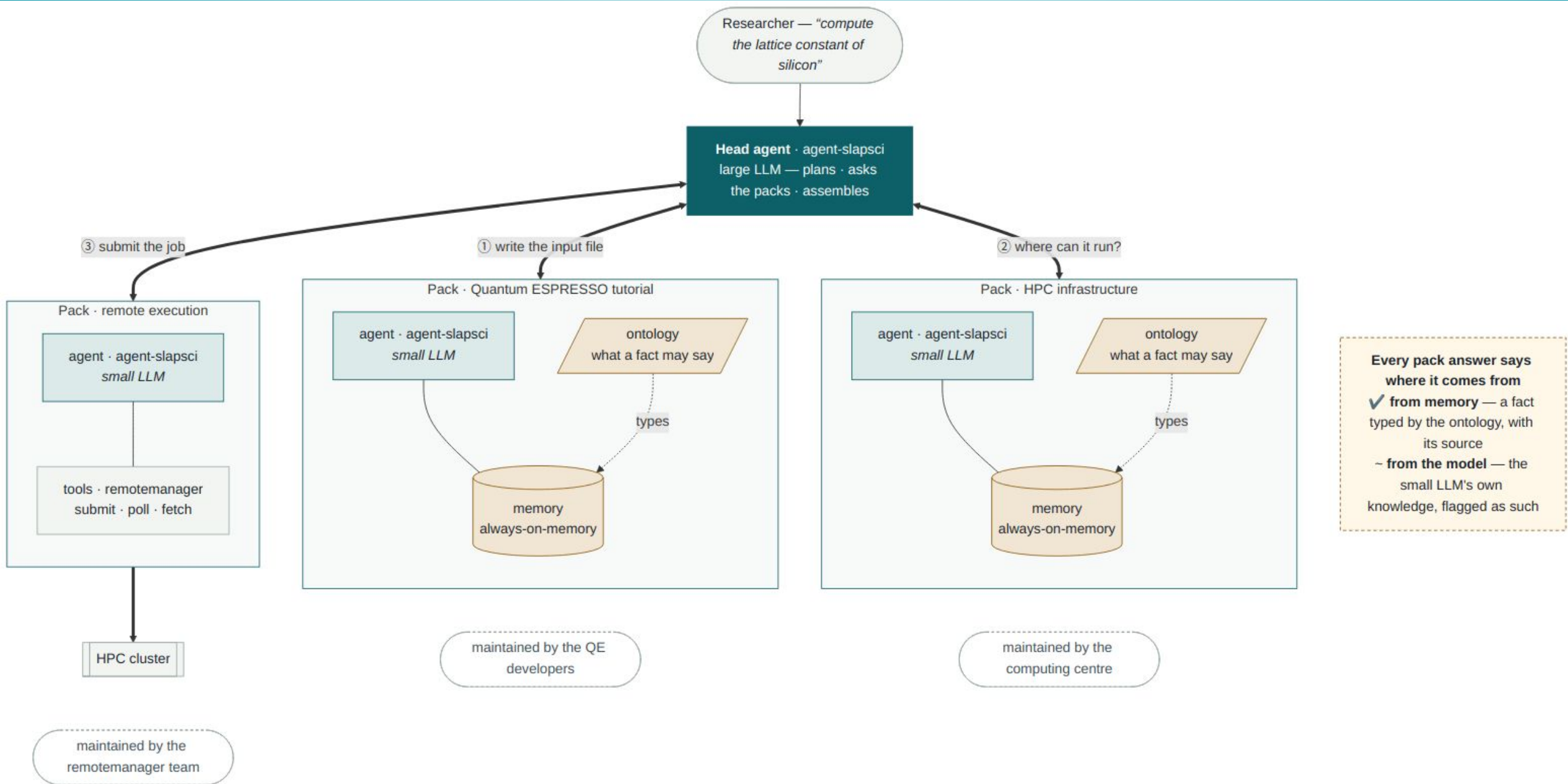
7. **prepost** - 4 nœuds (pré/post-traitement)
8. **archive** - partitions d'archive

Partitions de Compilation :

9. **compil** - 4 nœuds (compilation)

Cette information est cohérente à travers plusieurs mémoires [Memory 51], [Memory 16], [Memory 18], qui détaillent les caractéristiques spécifiques de chaque partition.





- BGE: Laboratoire Biosciences et Bioingénierie pour la Santé
- LETI: Laboratoire d'Électronique et de Technologie de l'Information
- SPINTEC: Laboratoire Spintronique et Technologie des Composants
- Projet NUMPEX - EXA-ATOW : Flux de travail scientifiques inter-installations