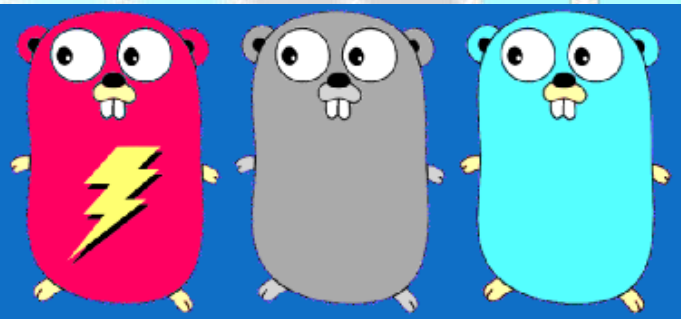
UNIVERSITÉ DE VERSAILLES ST-QUENTIN-EN-YVELINES

université PARIS-SACLAY

Génie Logiciel pour le Calcul Scientifique



#2

29/11/2024

jean-michel.batto@cea.fr

cea

https://gogs.eldarsoft.com/M2_IHPS

TD#1 prise en main Docker / MPI

- docker pull jmbatto/m2chps-mpi41

- Git clone https://github.com/jmbatto/master-mpi.git
- ➔ permet de récupérer le docker-compose
- Ensuite il faut aller dans le dossier mpi41-debian-bullseye et y mettre le bon dossier ssh

Docker pour windows (W10)

- Dans powershell : wsl –install ➔ WSL2 (par défaut)
- Installer Docker Desktop Windows ➔ https://docs.docker.com/desktop/windows/install/

Autorisation SSH du client MPI

- Git clone de l'espace de cours ➔ contient le dossier ssh ➔ les clés SSH
- Pour que le client MPI puisse accéder aux ressources via ssh, il faut les autorisations !

```
$HOME/.ssh
-rw-r--r--   authorized_keys
-rw-------   id_rsa
-rw-r--r--   id_rsa.pub
-rw-r--r--   known_hosts
```

Création des nœuds, connexion

INTERACTIF

- D'abord créer un swarm

```
docker swarm init
```

- Puis y attacher un réseau

```
docker network create --driver=overlay
  --attachable yml_mpinet
```

- Dans mpi41-debian-bullseye faire

```
docker-compose up --scale mpihead=1 --
  scale mpinode=2 -d
```

➔ utilise le fichier docker-compose.yml

Se connecter à un shell docker (utiliser le bash)

Compilation ‘élémentaire’ 2/2

INTERACTIF

- On souhaite « pratiquer » MPI ➔ utiliser mpicc + lmpi

hello_world.cc

```
$mpirun --mca orte_base_help_aggregate 0 --mca
  btl_tcp_if_include 10.0.2.0/24 -n 2 -host
  10.0.2.28,10.0.2.27 hello_world
Hello world from processor 1f9914291b09, rank 0 out of
  2 processors
Hello world from processor f2057a67f137, rank 1 out of
  2 processors
$ mpicc mpi_any_source.cc -o any_source -lmpi
$ mpirun --mca orte_base_help_aggregate 0 --mca
  btl_tcp_if_include 10.0.2.0/24 -n 2 -host
  10.0.2.28,10.0.2.27 any_source
[MPI process 0] I send value 12345.
[MPI process 1] I received value 12345, from rank 0.
```

Les benchmarks // Docker

INTERACTIF

Analyse du code Python

```
mpiuser@120dfa60a718:~$ mpirun --mca
  orte_base_help_aggregate 0 --mca
  btl_tcp_if_include 10.0.2.0/24 -n 2 -host
  10.0.2.24,10.0.2.19 python3
  ./mpi4py_benchmarks/all_tests.py
```

- Running 2 parallel MPI processes
- # MPI Matrix action on a vector, 20 iterations of size 10000
- # Duration [s] Throughput [#/s]
- 1.403 14.26
- 1.374 14.56
- 1.286 15.55
- 1.329 15.05

all_tests.py // MPI Matrix action on a vector

INTERACTIF

```
while counter < iter:
        comm.Barrier()                    ### Start stopwatch ###
        t_start = MPI.Wtime()
        for t in xrange(20):
            my_new_vec = np.inner(my_M, vec)
            comm.Allgather(
                [my_new_vec, MPI.DOUBLE],
                [vec, MPI.DOUBLE]
            )
        comm.Barrier()
        t_diff = MPI.Wtime() - t_start    ### Stop stopwatch
if myid == 0:
        print ('%-10.3f%20.2f' % (t_diff, bs/t_diff))

        counter += bs
```

Fichier : al_test.py

INTERACTIF

On s'intéresse au code (wget http://mvapich.cse.ohio-state.edu/download/mvapich/osu-micro-benchmarks-7.3.tar.gz)

configure puis

/osu-micro-benchmarks-7.3/mpi/pt2pt/osu_bibw.c

Extrait du code…

```
if (myid == 0) {
 double tmp = size / 1e6 * options.iterations * window_size;
 fprintf(stdout, "%-*d%*.*f\n", 10, size, FIELD_WIDTH,
 FLOAT_PRECISION, tmp / t_total);
 fflush(stdout); }
```

➔question : comment comparer des benchmarks en Python et en C ?

```
310     if (myid == 0) {
311         if (options.omb_enable_ddt) {
312             tmp_total = omb_ddt_transmit_size / 1e6 * options.iterations *
313                 window_size * 2;
314         } else {
315             tmp_total = size / 1e6 * options.iterations * window_size * 2;
316         }
317         fprintf(stdout, "%-*d", 10, size);
318         if (options.validate) {
319             fprintf(stdout, "%*.*f%*s", FIELD_WIDTH, FLOAT_PRECISION,
320                     tmp_total / t_total, FIELD_WIDTH,
321                     VALIDATION_STATUS(errors));
```

Facteurs agissants sur les benchmarks

- Connaitre la capacité de la VM Docker

INTERACTIF

```
docker stats --no-stream --format "table {{.Name}}\t{{.Container}}\t{{.MemUsage}}"
```

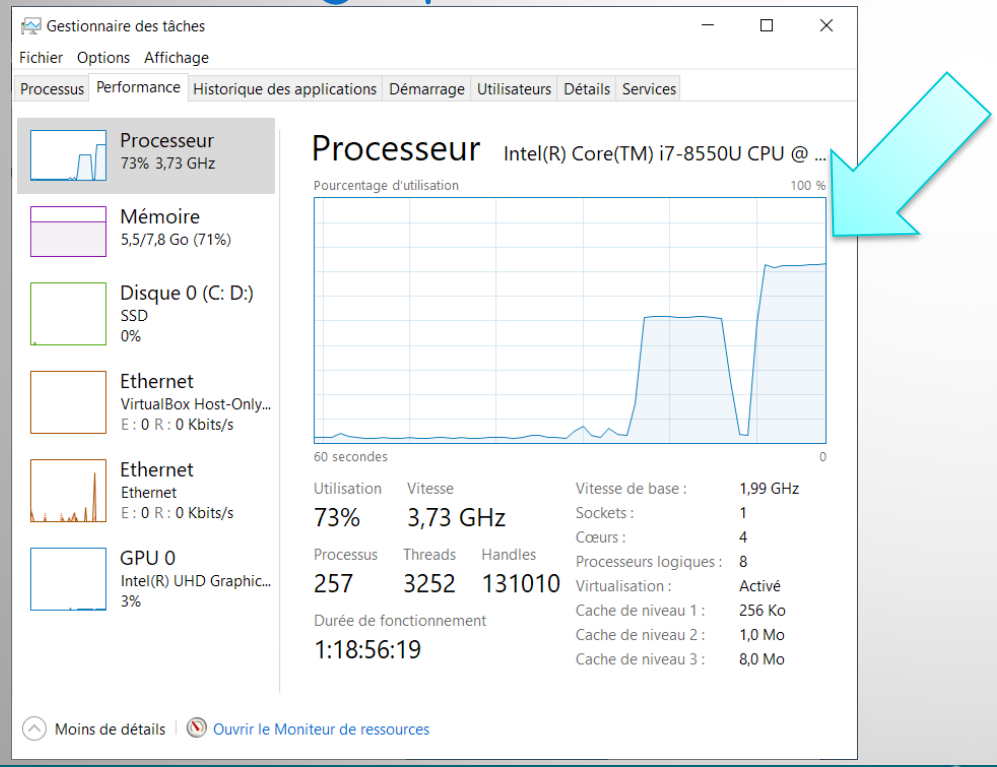
```
root@vps_eldarsoft:~# docker stats --no-stream --format "table {{.Name}}\t{{.Container}}\t{{.MemUsage}}"
NAME                CONTAINER           MEM USAGE / LIMIT
gogs                428f4d97c4a1        352.8MiB / 1.899GiB
traefik             7cce703b64a8        22.41MiB / 1.899GiB
root@vps_eldarsoft:~#
```

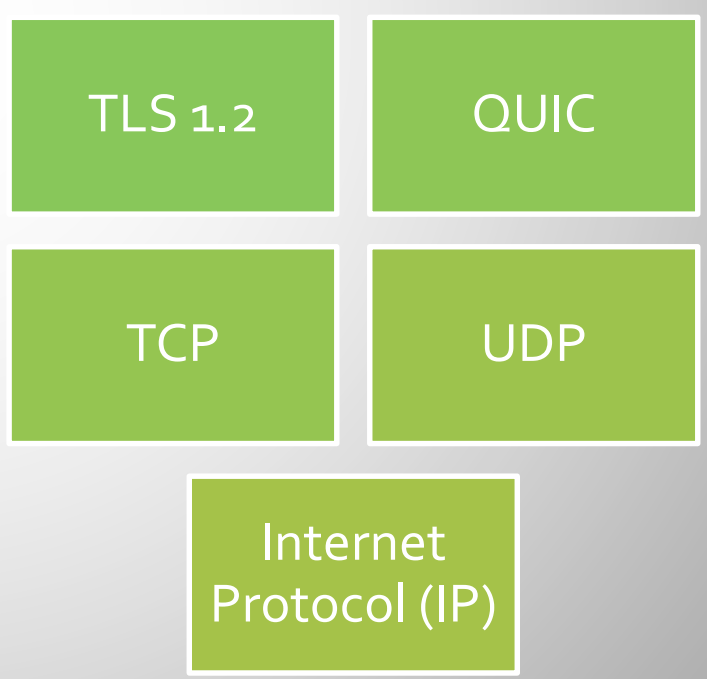
```
NAME                    CONTAINER           MEM USAGE / LIMIT
telegrafmonitor-auto    f19b09e8cd2b        11.43MiB / 14.33GiB
pure-ftpd               50da42229bd9        2.738MiB / 14.33GiB
jugydemo                93d1b26824ea        5.902MiB / 14.33GiB
jugy_web2               d23ceeb0a02a        9.828MiB / 14.33GiB
proxyApiSVMS            cd9f4b43fea7        2.707MiB / 14.33GiB
apiSVMS                 8c06a1152108        18MiB / 14.33GiB
redmine-menlosystems    f53ba3caa5d6        490.5MiB / 14.33GiB
traefik                 dfc54d4ab4da        21.88MiB / 14.33GiB
reoges_web              c0c5a071dd12        10.81MiB / 14.33GiB
chronograf              d1a698395cd3        12.58MiB / 14.33GiB
telegraf-auto           e7452b5722d6        26.61MiB / 14.33GiB
chronograf-poisson      ac14cbeb59dd        17.4MiB / 14.33GiB
kapacitor-auto          e8d7bc45a584        33.67MiB / 14.33GiB
phpmyadmin              060b9b29b02f        83.9MiB / 14.33GiB
influxdb                1fa90bc9f49a        1.261GiB / 14.33GiB
gogs                    69a2c6cfe67f        99.53MiB / 14.33GiB
mariadb                 edb0c20bdd20        7.556GiB / 14.33GiB
```

Autres facteurs de performance

INTERACTIF

- Le disque (si il y a des I/O)
- Le réseau (influence si UDP vs TCP) (ie QUIC)
- Binding (static vs late)
- Charge processeur





- TD1 : individuel
- http://mvapich.cse.ohio-state.edu/benchmarks/
  - wget http://mvapich.cse.ohio-state.edu/download/mvapich/osu-micro-benchmarks-7.3.tar.gz
  - ➔installation, compilation, test, benchmarks (C vs Python) (vous pouvez comparer les 2 versions de Python avec le C (tableau avec 3 colonnes)
  - ➔Mettre dans la Forge Gogs un rapport du TD1 (**1 page** PDF – présente un benchmark *MPI Bi-Directional Bandwidth Test* osu_bibw dans un tableau avec une introduction qui donne le contexte, avec le nom de l'étudiant et la date) : **pour le 15 décembre 2024 minuit**

Création des noeuds

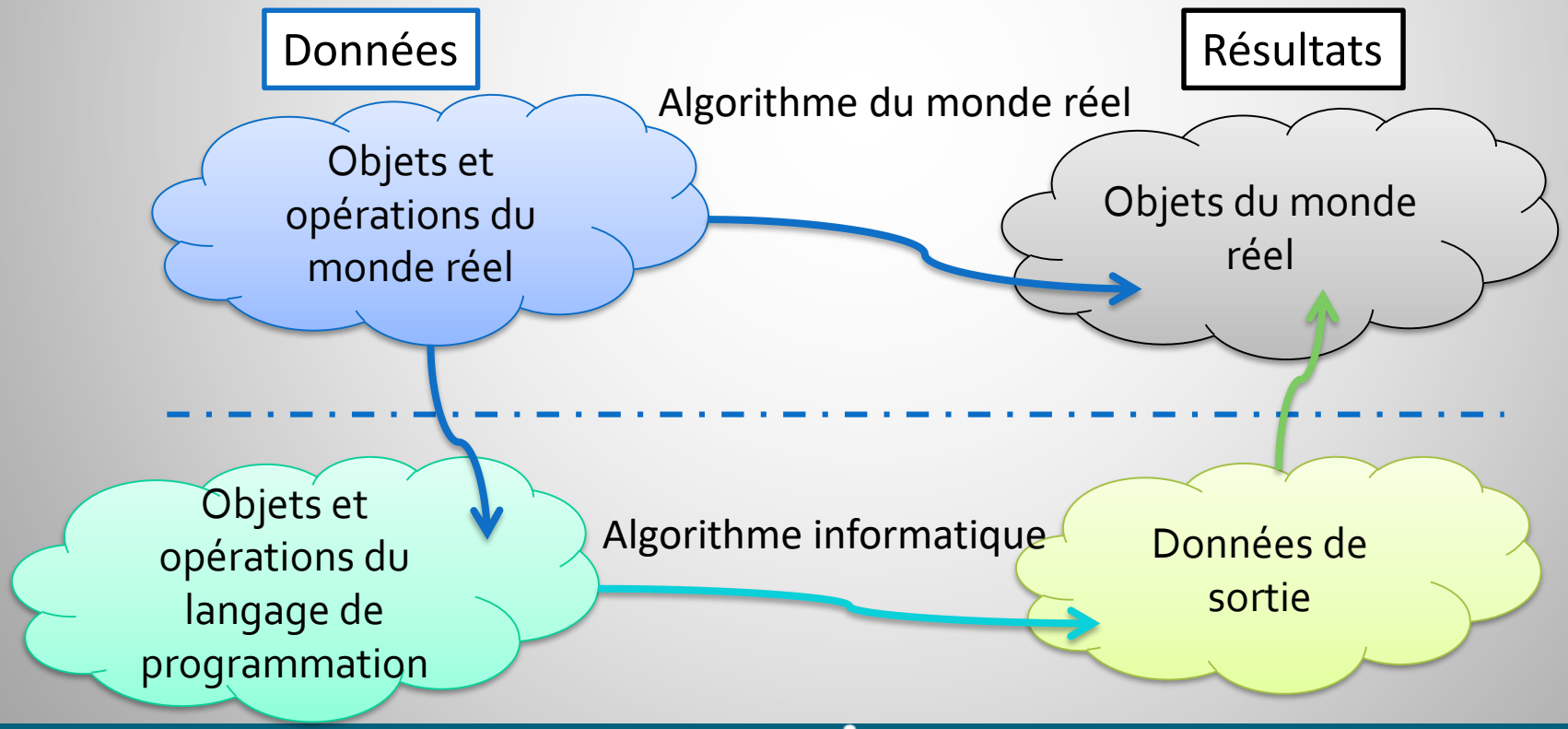
- `docker build -t jmbatto/m2chps-mpi41` ➔ permet de construire l'image Docker

A ne pas faire car cela prend plus de temps de construire que de télécharger.

- `docker-compose scale mpihead=1 mpinode=9` ➔ utilise le fichier docker-compose.yml

La POO 1/2

- La POO = Programmation Orientée Objet
- Il s'agit d'associer une abstraction informatique parallèle à l'espace du problème

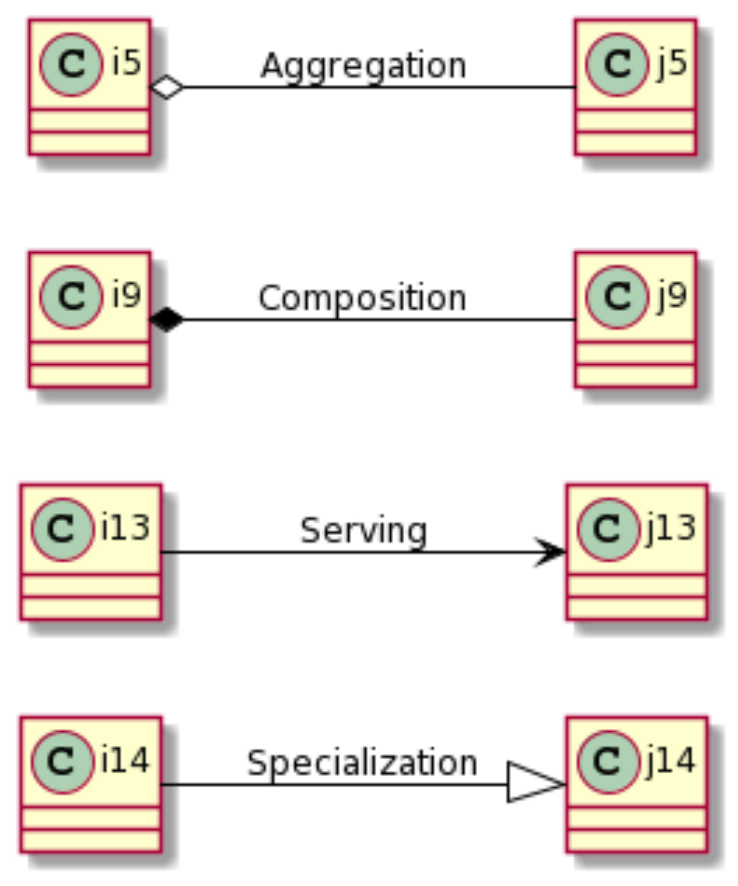


POO 2/2

- Démarche POO
  - Identifier les objets et leurs attributs.
  - Identifier les opérations qui affectent chaque objet et les opérations que chaque objet doit déclencher.
  - Etablir la visibilité de chaque objet vis-à-vis des autres objets.
  - Etablir l'interface de chaque objet.
  - Implémenter chaque objet.

Communiquer

UML : un outil de représentation graphique



Fichier : PlantUML_Language_Reference_Guide_fr.pdf

http://www.plantuml.com/plantuml/uml

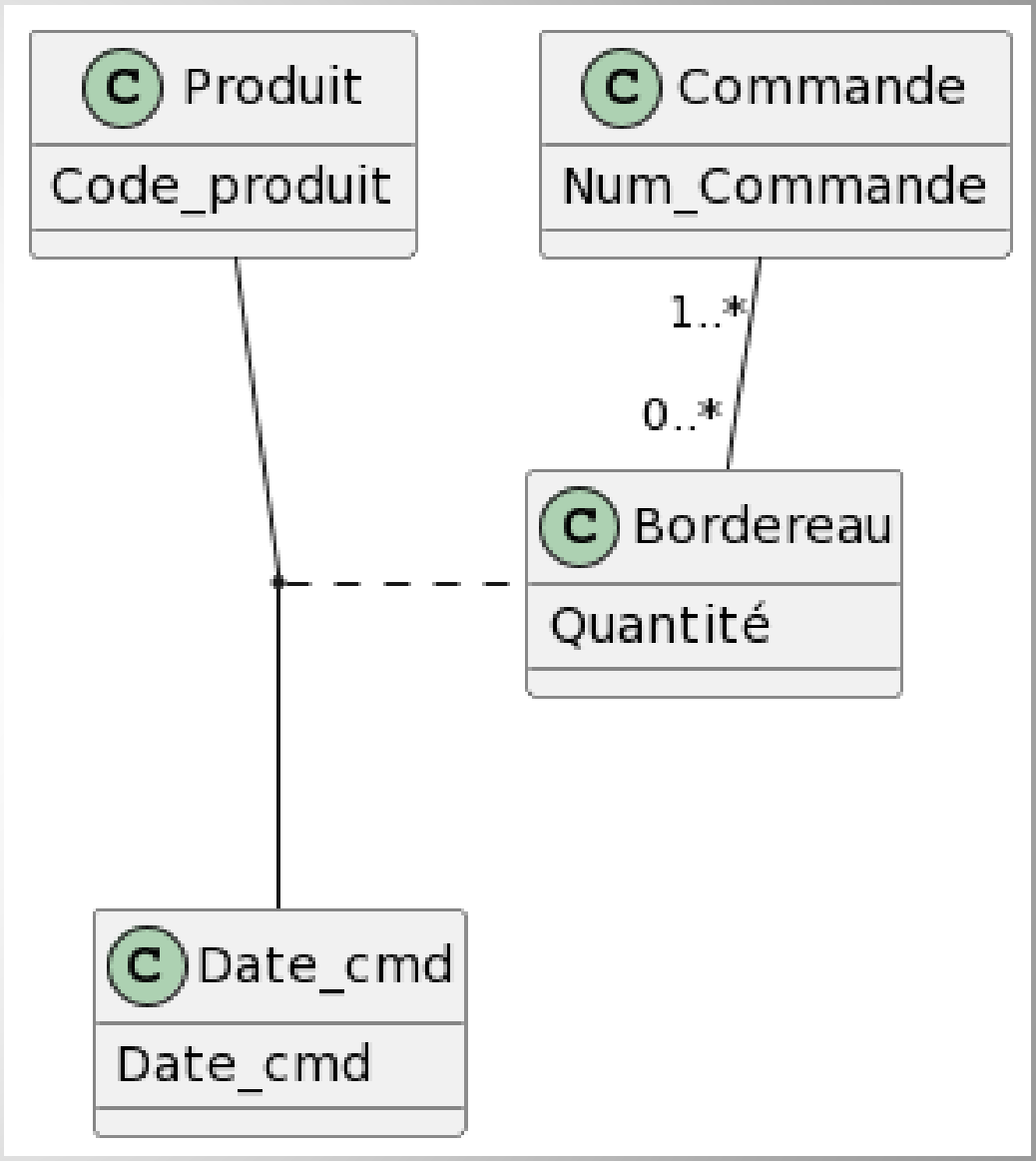
UML

- Unified Modeling Language
  - Langage de modélisation unifié : résultat de l'entente de 3 leaders de la modélisation orientée objet (Grady Booch, Jeff Rumbaugh, Ivar Jacobson) 1997.
  - Pose de manière propre :
    - Le Quoi (les objets) avec le diagramme d'instances (d'objets)
    - Le Comment (les interactions entre objets et utilisateurs) avec le diagramme d'activités, le diagramme d'états, le diagramme de séquence, le diagramme de communication
    - La Fabrication avec les diagrammes de paquetages et les diagrammes de classes

Diagramme de classes

http://www.plantuml.com/plantuml/uml

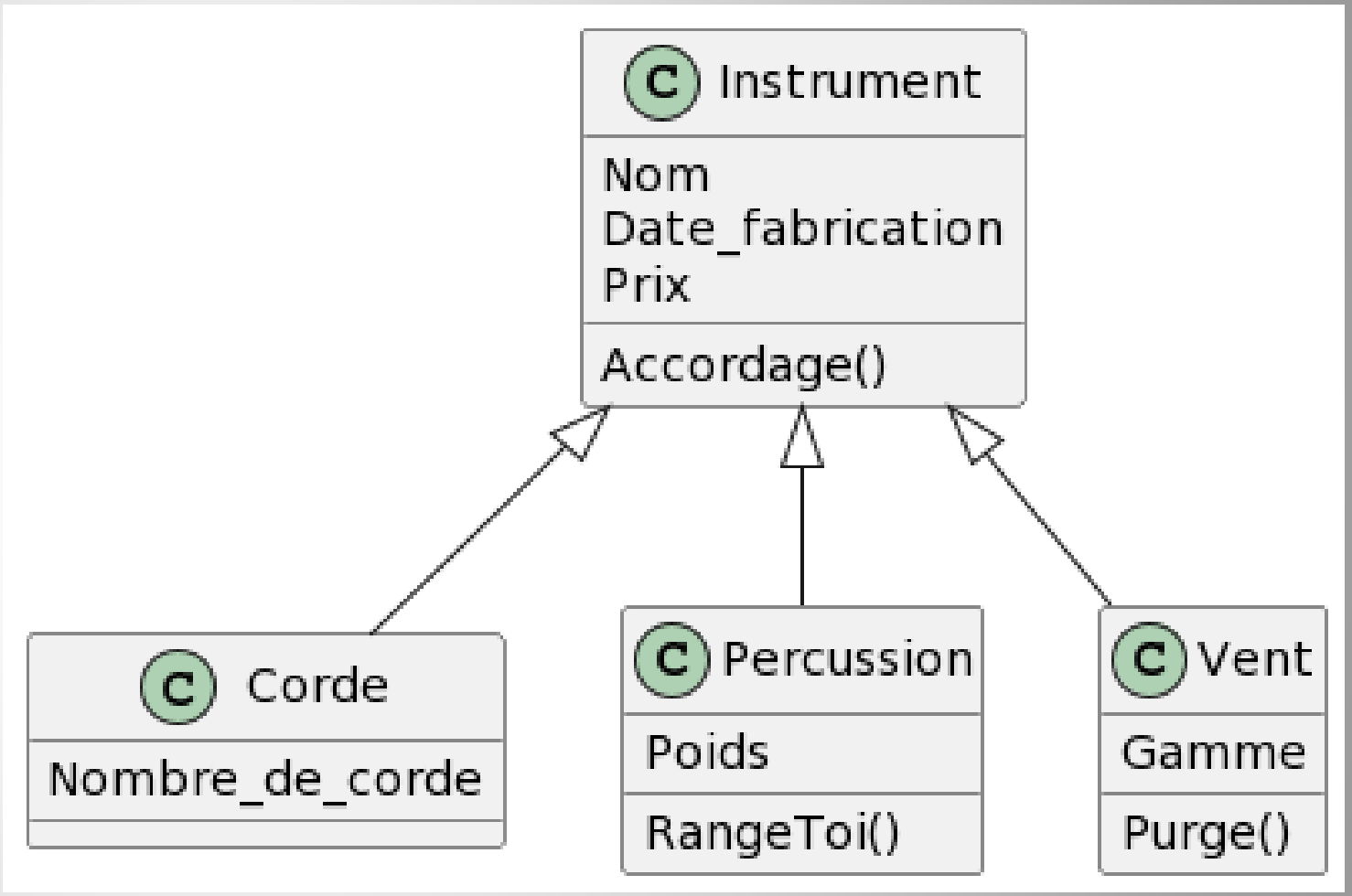
```
@startuml
class Commande {
Num_Commande
}
class Date_cmd {
Date_cmd
}
class Produit {
Code_produit
}
class Bordereau {
Quantité
}
' arité = 2
Commande "1..*" -down- "0..*" Bordereau
' Une classe ternaire / arité = 3
(Produit, Date_cmd) .. Bordereau
@enduml
```



Une classe peut avoir une arité

Héritage

```
' commentaire
@startuml
class Instrument {
Nom
Date_fabrication
Prix
Accordage()
}
class Corde {
Nombre_de_corde
}
class Percussion {
Poids
RangeToi()
}
class Vent {
Gamme
Purge()
}
Instrument <|--Vent
Instrument <|--Percussion
Instrument <|--Corde
@enduml
```



Corde spécialise Instrument

Composition / agrégation

```
' comment
@startuml
class Type_Véhicule {
PayeTaxe()
}
class Véhicule {
Nombre_de_Passager
}
class Moteur {
Puissance
}
Moteur "1" *-right- "plusieurs" Véhicule : contient
Type_Véhicule "1" o-- "plusieurs" Véhicule : connait/\nagrégation
@enduml
```

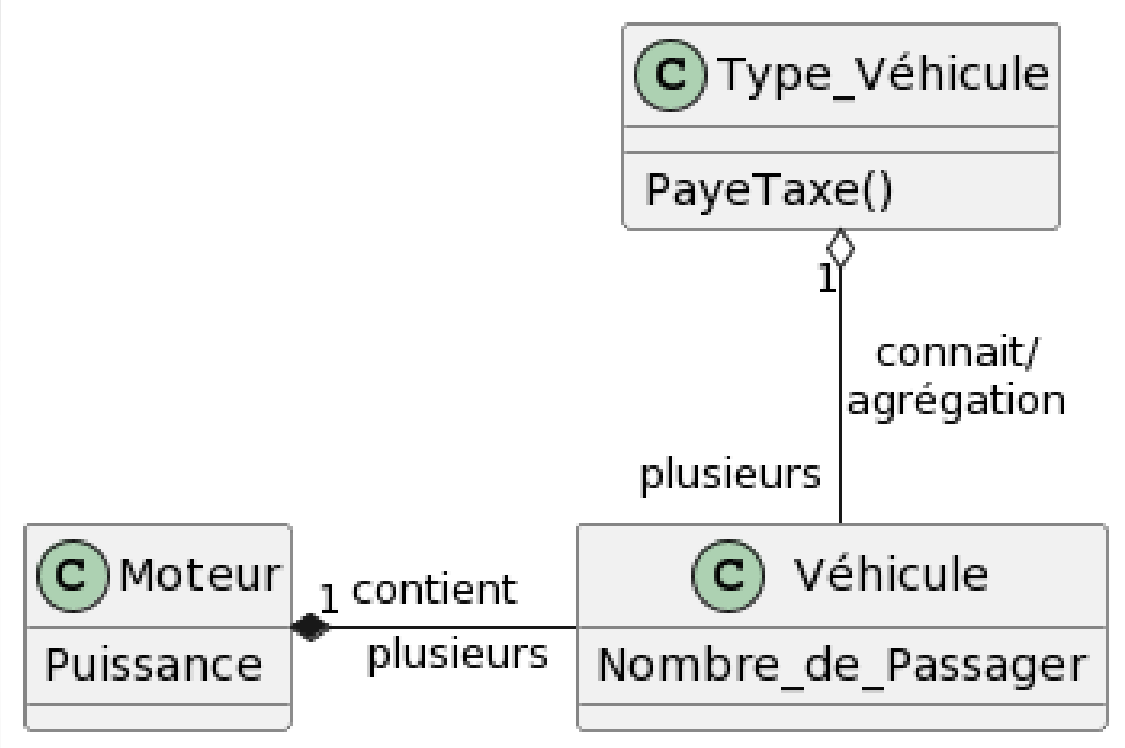
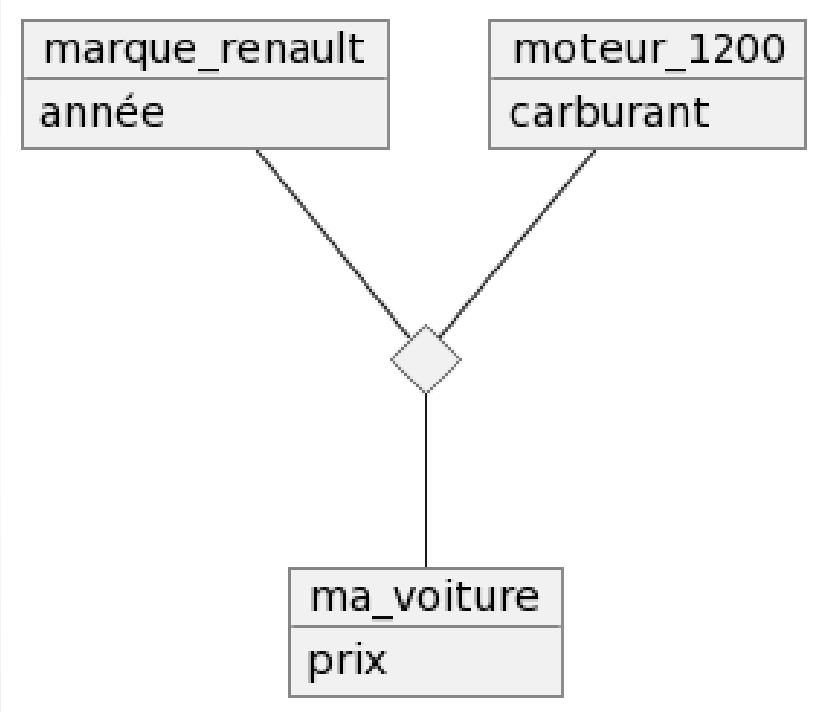


Diagramme d'objets

```
@startuml
object marque_renault
marque_renault : année
object moteur_1200
moteur_1200 : carburant
diamond dia
object ma_voiture
ma_voiture : prix
marque_renault -- dia
moteur_1200 -- dia
dia -- ma_voiture
@enduml
```



meta/llama-3.1-70b-instruct

INTERACTIF

- https://build.nvidia.com/meta/llama-3_1-70b-instruct
  - donne moi le code pour PlantUML qui décrit la relation entre une objet de type marque_renault avec un attribut année, un objet de type moteur_1200 avec un attribut carburant et une instance de ma_voiture composition des deux précédents.

```
@startuml
class MarqueRenault {
 année : int
}
class Moteur1200 {
 carburant : string
}
class MaVoiture {
}
MaVoiture --* MarqueRenault : "possède une"
MaVoiture --* Moteur1200 : "possède un"
@enduml
```

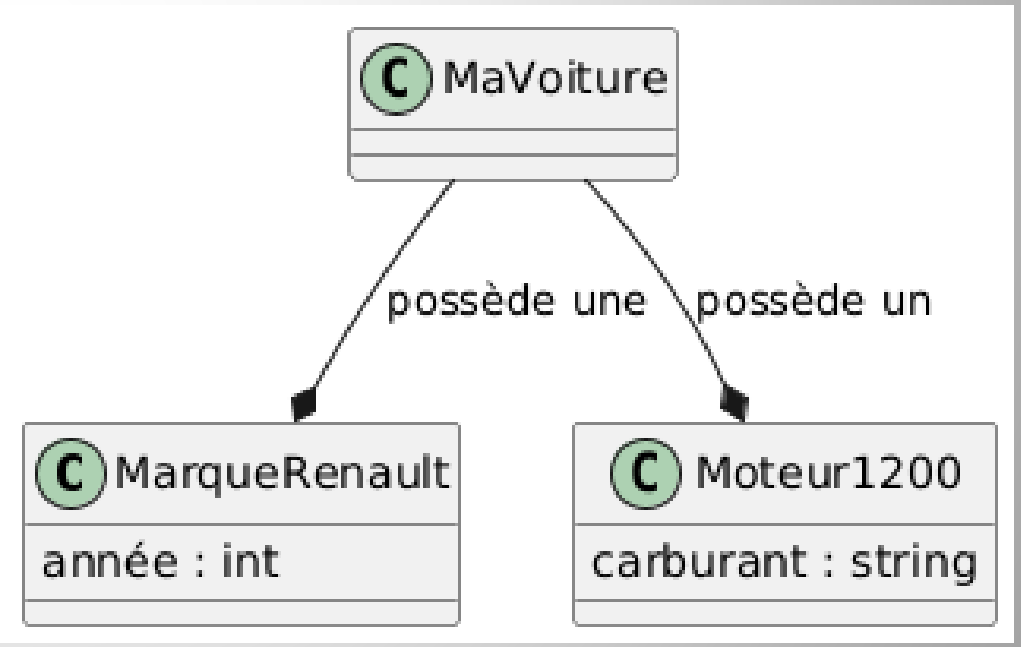
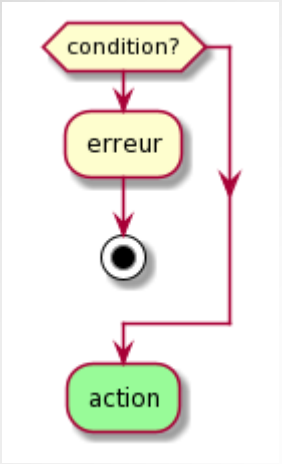


Diagramme d'activités

```
@startuml
if (condition?) then
:erreur;
stop
endif
#palegreen:action;
@enduml
```




```
@startuml
start
while (données disponibles?)
#palegreen:lecture données;
:traite données;
endwhile
stop
@enduml
```

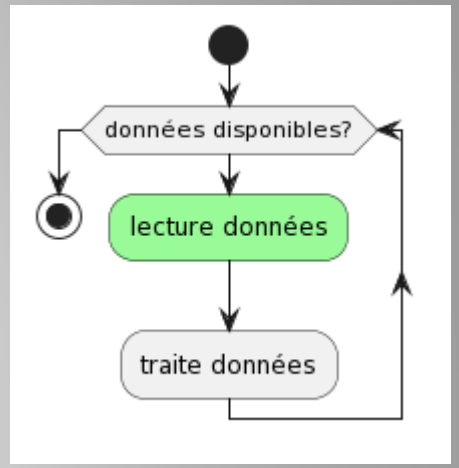
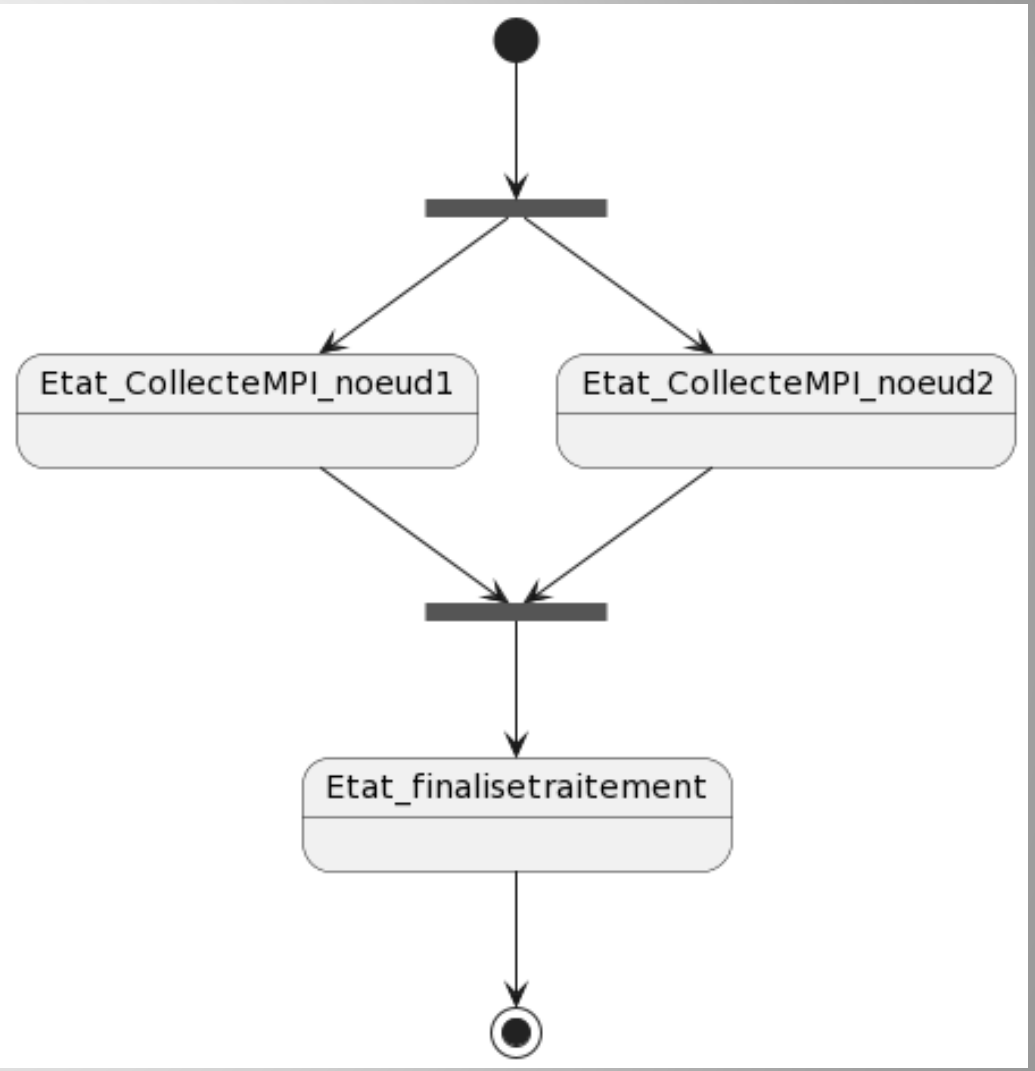


Diagramme d'états

```
@startuml
state fork_state <<fork>>
[*] --> fork_state
fork_state --> Etat_CollecteMPI_noeud1
fork_state -->Etat_CollecteMPI_noeud2
state join_state <<join>>
Etat_CollecteMPI_noeud1--> join_state
Etat_CollecteMPI_noeud2--> join_state
join_state --> Etat_finalisetraitement
Etat_finalisetraitement--> [*]
@enduml
```



meta/llama-3.1-70b-instruct

INTERACTIF

Peux-tu écrire en code PlantUML le diagramme d'états qui représente la synchronisation de deux threads de traitement MPI Etat_CollecteMPI_noeud1 et Etat_CollecteMPI_noeud2. Ceux-ci se divisent en deux branches à partir d'un état de fourche, puis se synchronisent à un état de jointure nommé Etat_finalisetraitement avant de se terminer à un état terminal

```
@startuml
state fork_state <<fork>>
[*] --> fork_state
fork_state --> Etat_CollecteMPI_noeud1
fork_state --> Etat_CollecteMPI_noeud2
state join_state <<join>>
Etat_CollecteMPI_noeud1 --> join_state
Etat_CollecteMPI_noeud2 --> join_state
join_state --> Etat_finalisetraitement
Etat_finalisetraitement --> [*]
@enduml
```

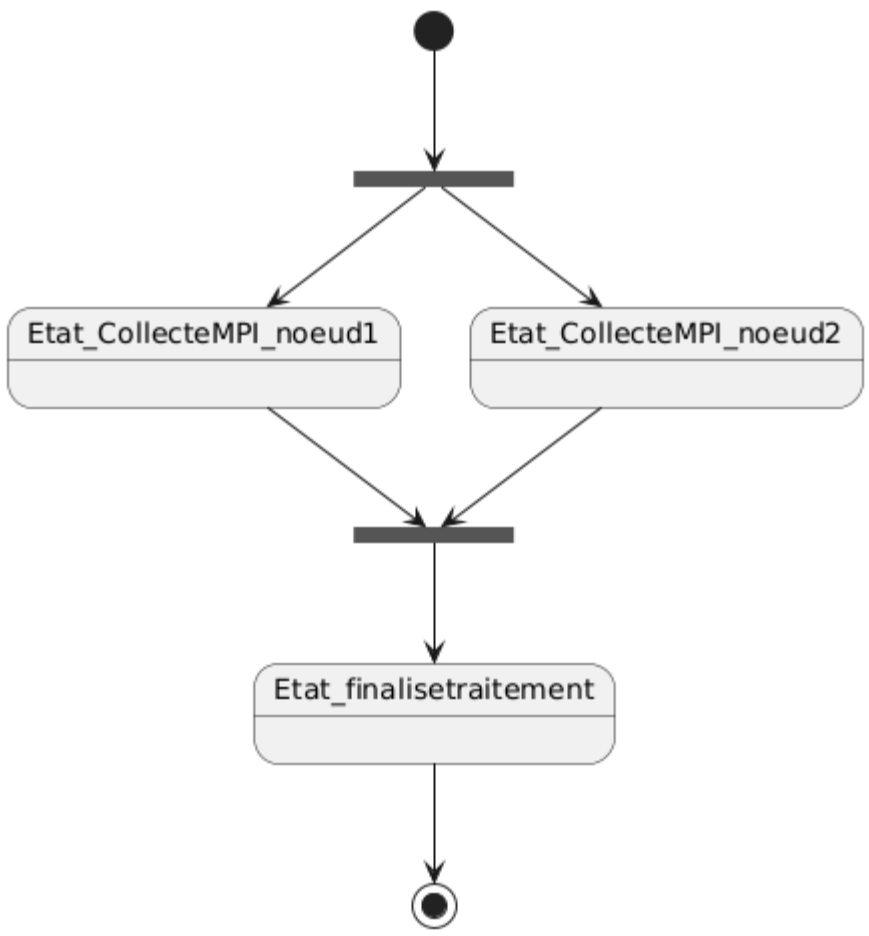


Diagramme d'états

```
@startuml
state fork_state <<fork>>
[*] --> fork_state
fork_state --> Etat_CollecteMPI_noeud1 : thread1
fork_state -->Etat_CollecteMPI_noeud2 : thread2
state join_state <<join>>
Etat_CollecteMPI_noeud1 --> panic : erreur
Etat_CollecteMPI_noeud2 --> panic : erreur
Etat_CollecteMPI_noeud1--> join_state : fin thread1
Etat_CollecteMPI_noeud2--> join_state : fin thread2
join_state --> Etat_finalisetraitement
Etat_finalisetraitement--> [*]
@enduml
```

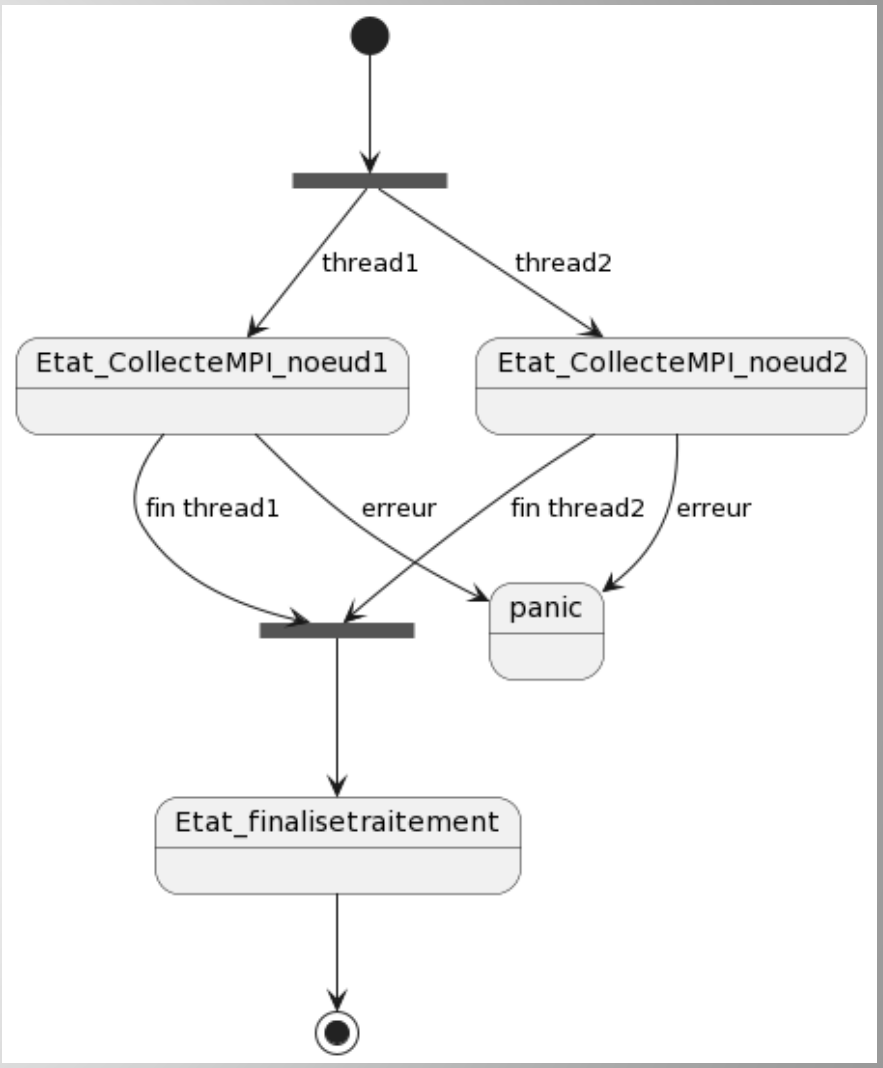
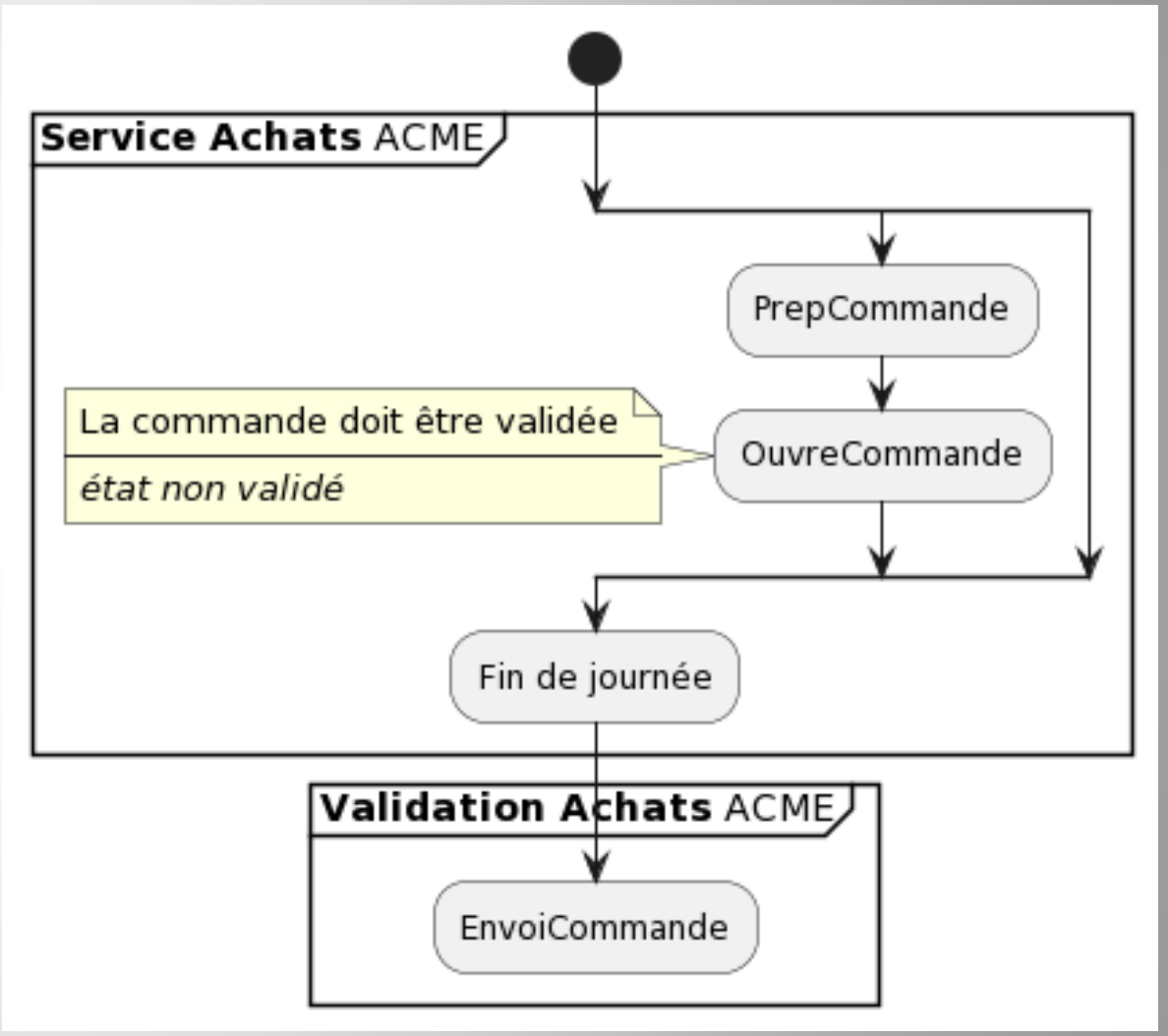


Diagramme d'activités

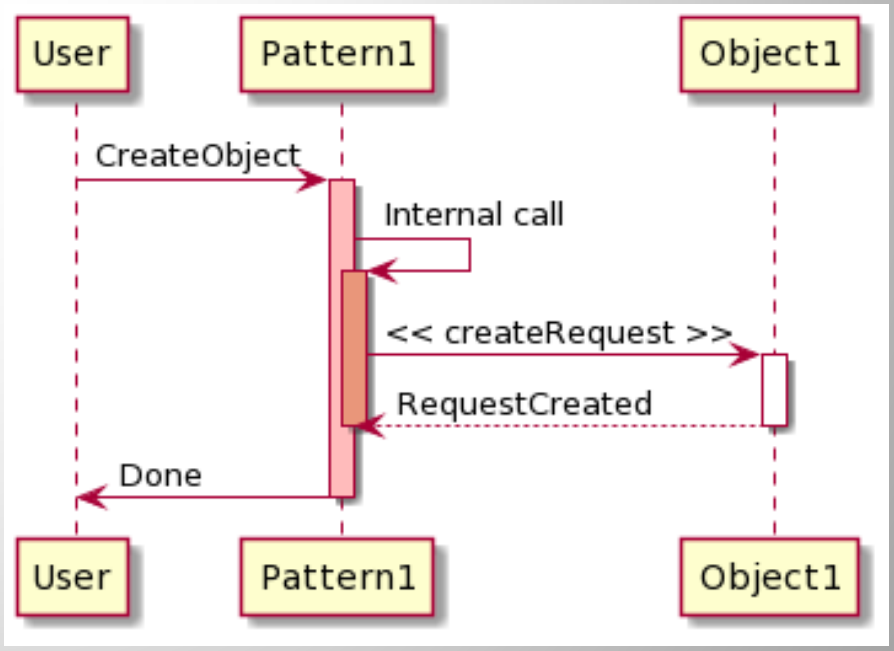
```
@startuml
start
partition "**Service Achats** ACME" {
split
:PrepCommande;
:OuvreCommande;
note left
La commande doit être validée
----
//état non validé//
end note
split again
-[hidden]->
end split
:Fin de journée;
}
partition "**Validation Achats** ACME" {
:EnvoiCommande;
}
@enduml
```



UML – diagramme de séquence

http://www.plantuml.com/plantuml/uml/RP91pe8m3CVlVOectva3lq88-IGadZGQ7w2E7jlHSiIVNoOoiyWEMzFrrt_xBqSbfrDdOPZVzhG3YU9bPOV-mgJ08wcoapJ3dMDIFbnlR1J8QFiavV2FVtNzFnzO4n6x5n-GHKFMRi4LkVkPRDTBQE3L5xKKCynpDEylJloOU5GiYj0dbC4Q997wlraMQA5XN_2R2tmqYbSZmvmWtYmjMPbRmK0

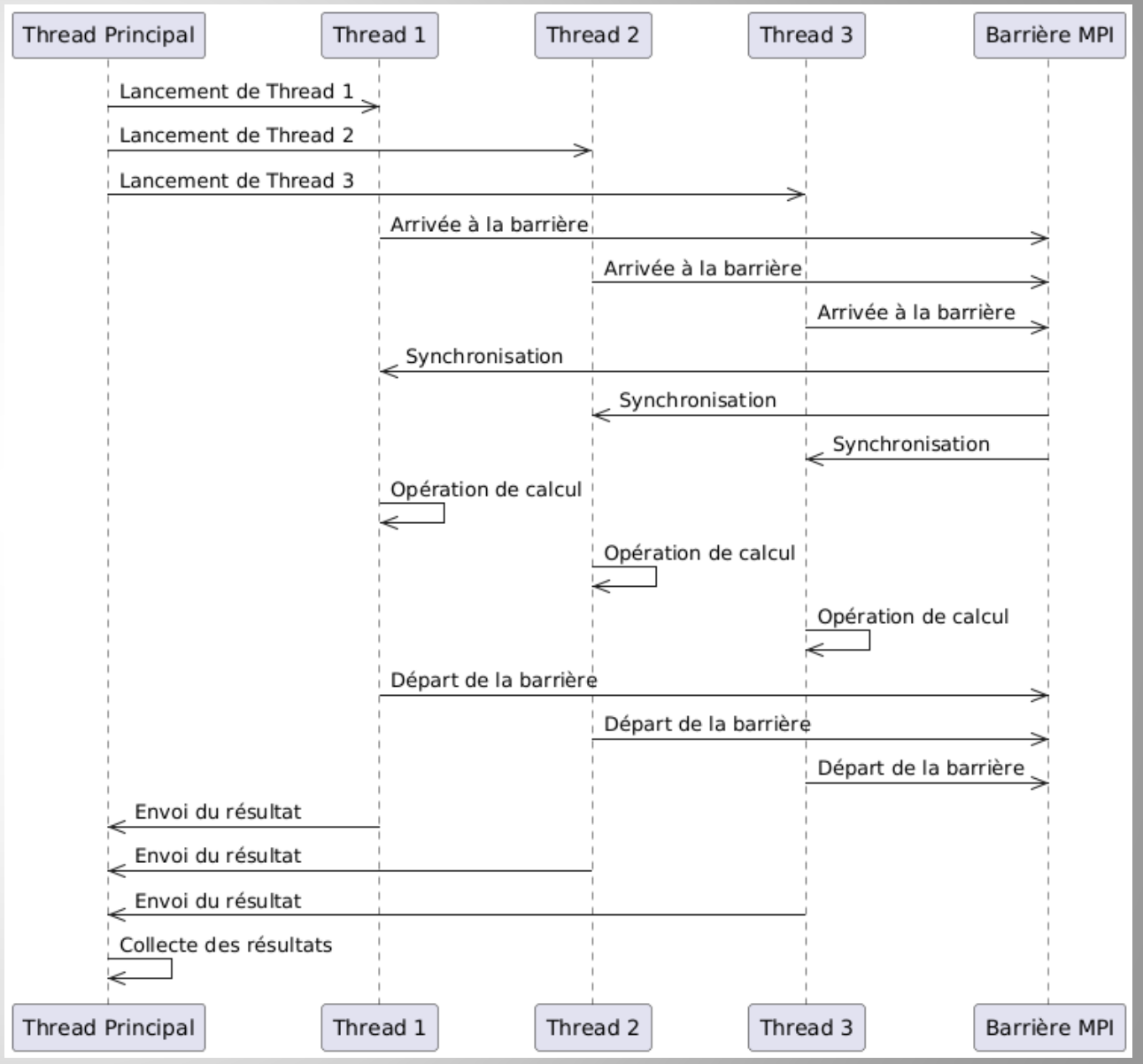
```
@startuml
participant User
User -> Pattern1: CreateObject
activate Pattern1 #FFBBBB
Pattern1-> Pattern1: Internal call
activate Pattern1 #DarkSalmon
Pattern1 -> Object1 : << createRequest >>
activate Object1
Object1 --> Pattern1: RequestCreated
deactivate Object1
deactivate Pattern1
Pattern1 -> User: Done
deactivate Pattern1
@enduml
```



meta/llama-3.1-70b-instruct

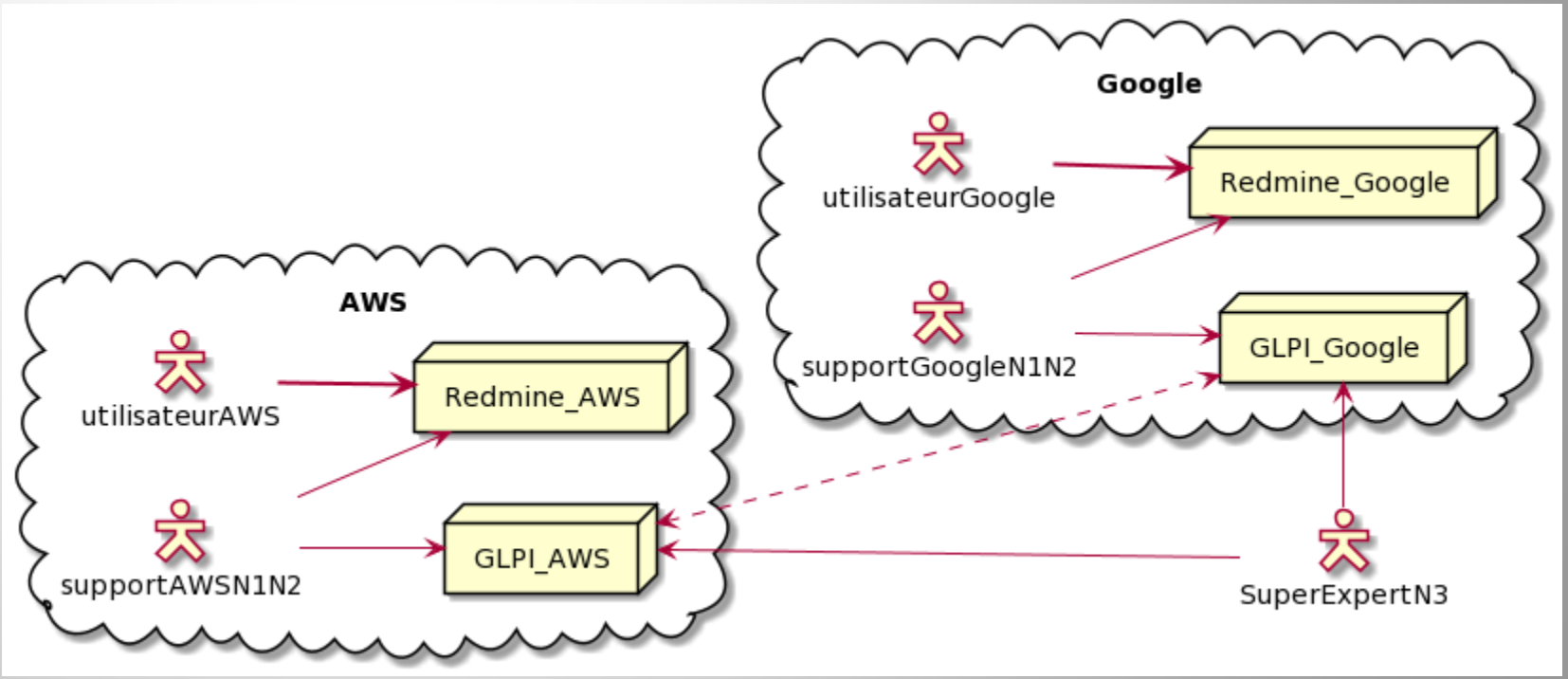
INTERACTIF

- Peux-tu écrire en PlantUML le diagramme de séquence qui démontre la synchronisation MPI sur une barrière MPI entre 3 threads qui exécutent une opération de calcul ? Le résultat des 3 threads est collecté par un thread principal en charge du lancement des 3 threads.

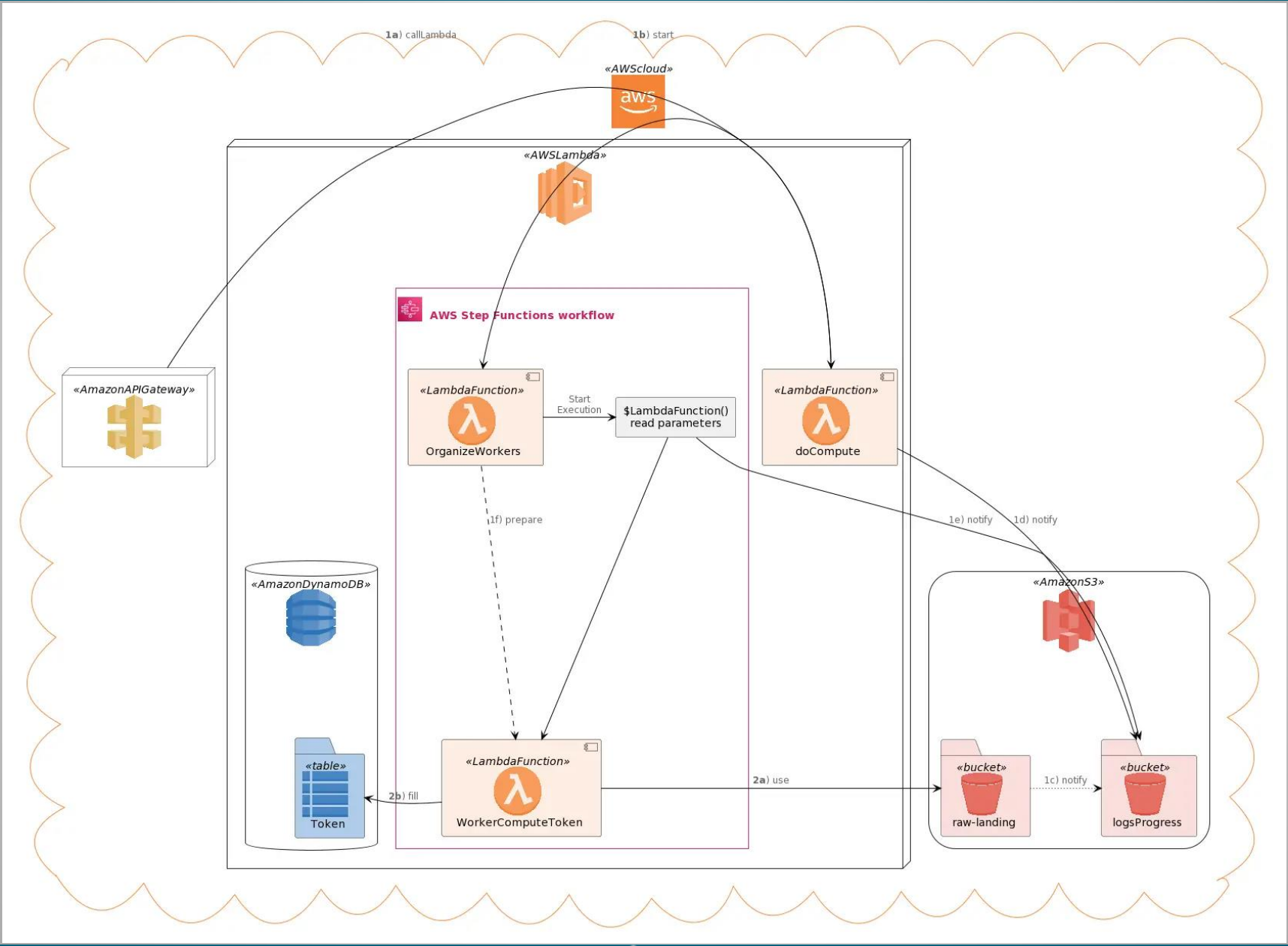


UML : aider à communiquer

```
@startuml
skinparam actorStyle Hollow
left to right direction
actor SuperExpertN3
cloud "Google" {
actor utilisateurGoogle
actor supportGoogleN1N2
node "Redmine_Google"
node "GLPI_Google"
utilisateurGoogle -[bold]->Redmine_Google
supportGoogleN1N2 --> Redmine_Google
supportGoogleN1N2 --> GLPI_Google
}
cloud "AWS" {
actor utilisateurAWS
actor supportAWSN1N2
node "Redmine_AWS"
node "GLPI_AWS"
utilisateurAWS -[bold]->Redmine_AWS
supportAWSN1N2 --> Redmine_AWS
supportAWSN1N2 --> GLPI_AWS
}
GLPI_AWS <-.-> GLPI_Google
SuperExpertN3 -left-> GLPI_AWS
SuperExpertN3 -left-> GLPI_Google
@enduml
```



Respecter une iconographie (ici AWS)



PlantUML

```
@startuml

!define AWSPuml https://raw.githubusercontent.com/awslabs/aws-icons-for-plantuml/v13.1/dist
!include AWSPuml/AWSCommon.puml
!include AWSPuml/Groups/all.puml
!include AWSPuml/ApplicationIntegration/StepFunctions.puml
!includeurl <aws/common.puml>
!includeurl <aws/ApplicationServices/AmazonAPIGateway/AmazonAPIGateway.puml>
!includeurl <aws/Compute/AWSLambda/AWSLambda.puml>
!includeurl <aws/Compute/AWSLambda/LambdaFunction/LambdaFunction.puml>
!includeurl <aws/Database/AmazonDynamoDB/AmazonDynamoDB.puml>
!includeurl <aws/Database/AmazonDynamoDB/table/table.puml>
!includeurl <aws/General/AWScloud/AWScloud.puml>
!includeurl <aws/General/client/client.puml>
!includeurl <aws/General/user/user.puml>
!includeurl <aws/SDKs/JavaScript/JavaScript.puml>
!includeurl <aws/Storage/AmazonS3/AmazonS3.puml>
!includeurl <aws/Storage/AmazonS3/bucket/bucket.puml>

skinparam componentArrowColor Black
skinparam componentBackgroundColor White
skinparam nodeBackgroundColor White
skinparam agentBackgroundColor White
skinparam artifactBackgroundColor White
```

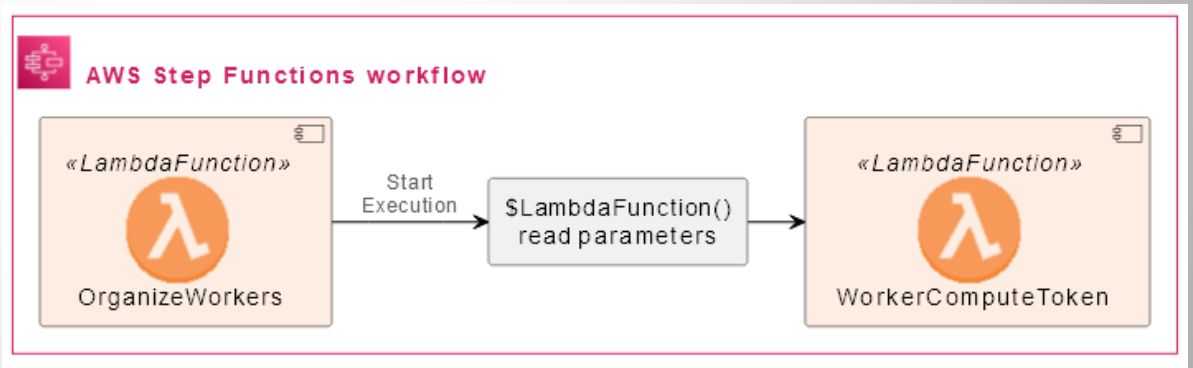
Paramétrage Pour inclure les icônes AWS

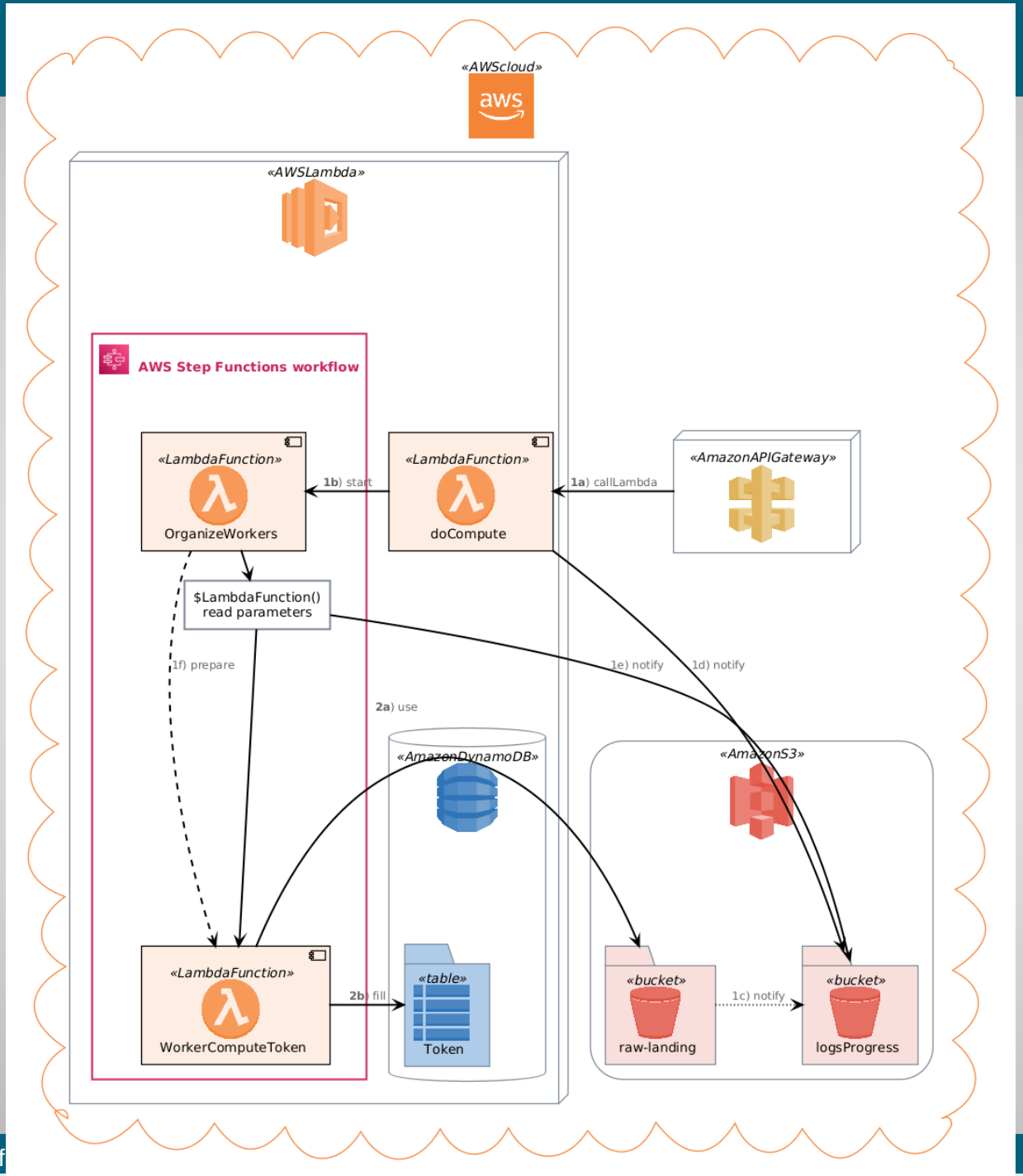
Rendu

PlantUML

```
AWSCLOUD(aws) {
    AMAZONS3(s3) {
        BUCKET(raw,raw-landing)
        BUCKET(logs,logsProgress)
    }
    AMAZONAPIGATEWAY(api)
    AWSLAMBDA(lambda) {
        LAMBDAFUNCTION(doCp,doCompute)
      StepFunctionsWorkflowGroup(sfw) {
      LAMBDAFUNCTION(oWk,OrganizeWorkers)
      LAMBDAFUNCTION(sfw1,WorkerComputeToken)
      rectangle "$LambdaFunction()\nread parameters" as sfw0
    }
    AMAZONDYNAMODB(dynamo) {
        TABLE(Token,Token)
    }
     oWk -> sfw0: Start\nExecution
     sfw0 -> sfw1
    }
}
api -l-> doCp:**1a**) callLambda
doCp -l-> oWk:**1b**) start
raw ~> logs :1c) notify
doCp --> logs:1d) notify
sfw0 --> logs:1e) notify
oWk ..> sfw1:1f) prepare
sfw1 -r-> raw:**2a**) use
sfw1 -r-> Token:**2b**) fill
@enduml
```

-l-> : left
-u-> : up
-d-> : down
-u-> : up

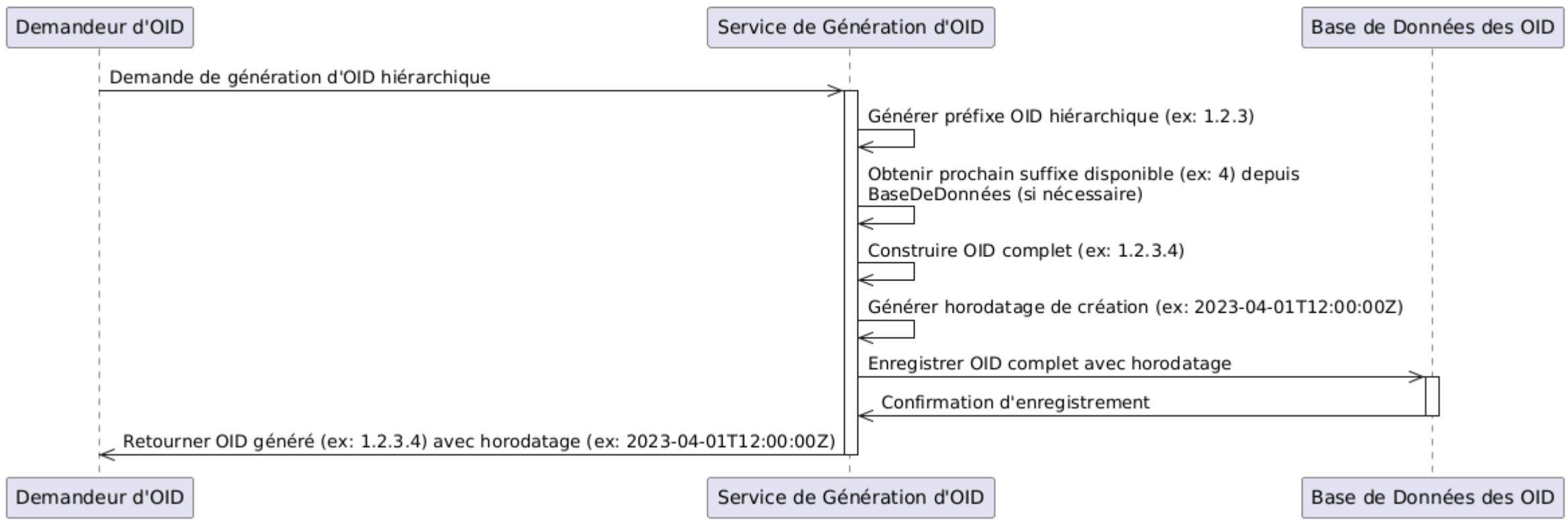



«AWScloud»
aws
«AWSLambda»
AWS Step Functions workflow
«LambdaFunction»
OrganizeWorkers
1b) start
«LambdaFunction»
doCompute
1a) callLambda
«AmazonAPIGateway»
$LambdaFunction()
read parameters
1f) prepare
1e) notify
1d) notify
2a) use
«AmazonDynamoDB»
«AmazonS3»
«LambdaFunction»
WorkerComputeToken
2b) fill
«table»
Token
«bucket»
raw-landing
1c) notify
«bucket»
logsProgress

Petit comparatif IA

INTERACTIF

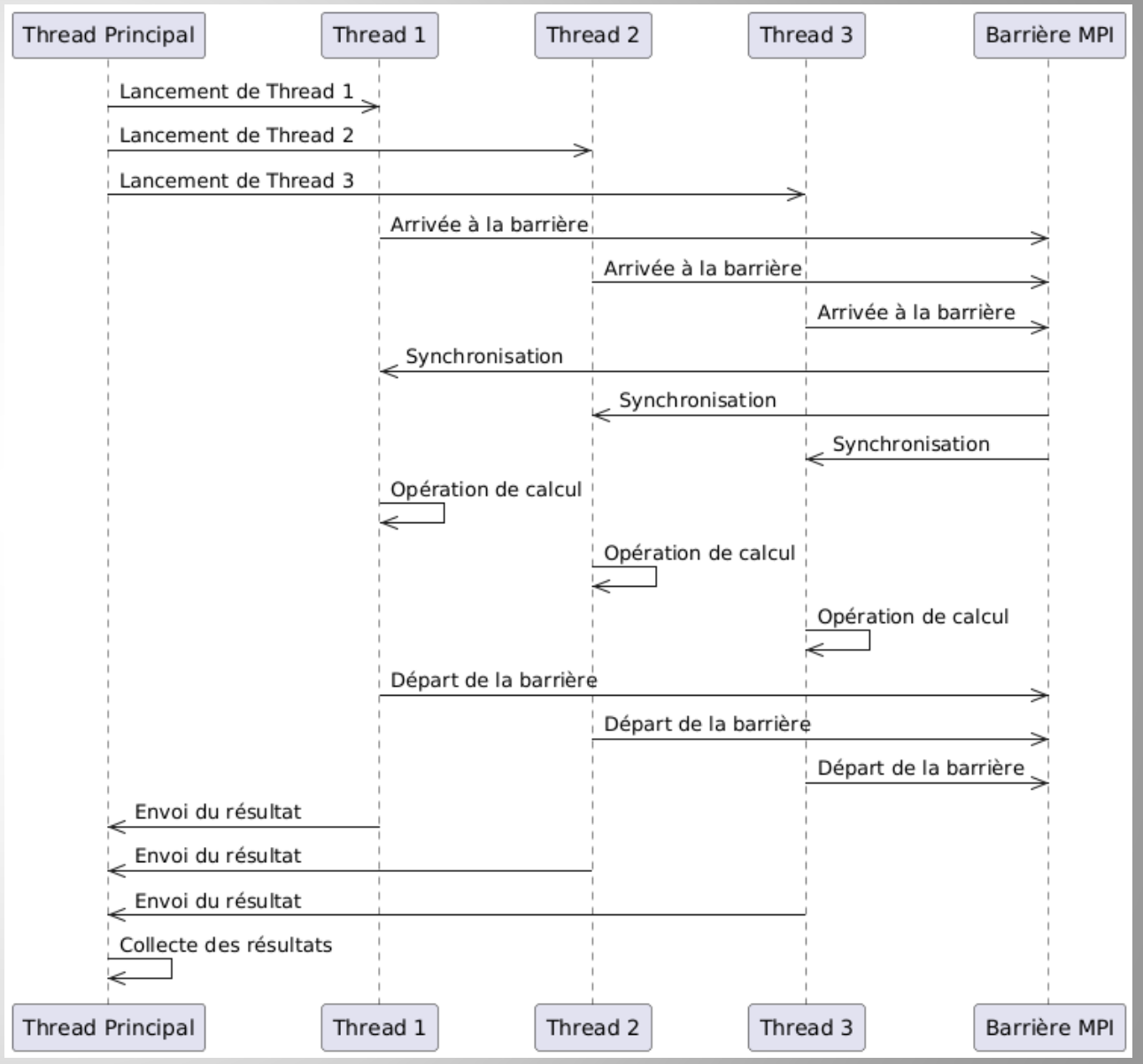
- Peux-tu m'écrire le script PlantUML pour décrire un service de génération d'OID hiérarchique avec horodate de la génération ?
- ➔meta/llama-3.1-70b-instruct
- ➔nvidia/llama-3.1-nemotron-70b-instruct https://build.nvidia.com/nvidia/llama-3_1-nemotron-70b-instruct



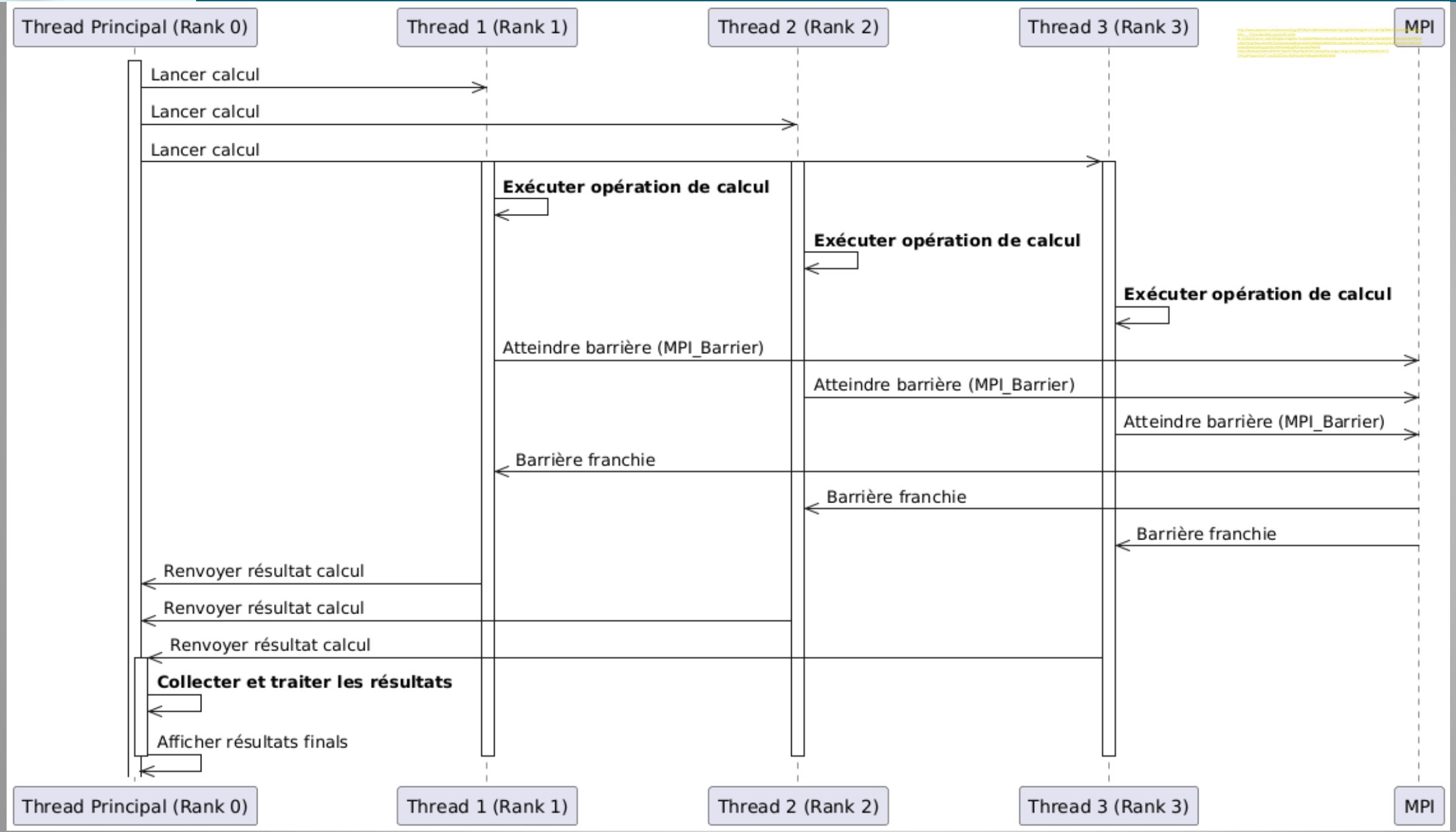
meta/llama-3.1-70b-instruct

INTERACTIF

- Peux-tu écrire en PlantUML le diagramme de séquence qui démontre la synchronisation MPI sur une barrière MPI entre 3 threads qui exécutent une opération de calcul ? Le résultat des 3 threads est collecté par un thread principal en charge du lancement des 3 threads.



nvidia/llama-3.1-nemotron-70b-instruct



L'IA peut faire des erreurs
Par ex, manque balise end note

CSP – Aout 1978 – Tony Hoare

Fichier : tony_hoare-CSP1978.pdf

Programming Techniques
S. L. Graham, R. L. Rivest Editors

Communicating Sequential Processes

C.A.R. Hoare
The Queen's University
Belfast, Northern Ireland

This paper suggests that input and output are basic primitives of programming and that parallel composition of communicating sequential processes is a fundamental program structuring method. When combined with a development of Dijkstra's guarded command, these concepts are surprisingly versatile. Their use is illustrated by sample solutions of a variety of familiar programming exercises.

Key Words and Phrases: programming, programming languages, programming primitives, program structures, parallel programming, concurrency, input, output, guarded commands, nondeterminacy, coroutines, procedures, multiple entries, multiple exits, classes, data representations, recursion, conditional critical regions, monitors, iterative arrays
CR Categories: 4.20, 4.22, 4.32

grams, three basic constructs have received widespread recognition and use: A repetitive construct (e.g. the **while** loop), an alternative construct (e.g. the conditional **if..then..else**), and normal sequential program composition (often denoted by a semicolon). Less agreement has been reached about the design of other important program structures, and many suggestions have been made: Subroutines (Fortran), procedures (Algol 60 [15]), entries (PL/I), coroutines (UNIX [17]), classes (SIMULA 67 [5]), processes and monitors (Concurrent Pascal [2]), clusters (CLU [13]), forms (ALPHARD [19]), actors (Hewitt [1]).

The traditional stored program digital computer has been designed primarily for deterministic execution of a single sequential program. Where the desire for greater speed has led to the introduction of parallelism, every attempt has been made to disguise this fact from the programmer, either by hardware itself (as in the multiple function units of the CDC 6600) or by the software (as in an I/O control package, or a multiprogrammed operating system). However, developments of processor technology suggest that a multiprocessor machine, constructed from a number of similar self-contained processors (each with its own store), may become more powerful, capacious, reliable, and economical than a machine which is disguised as a monoprocessor.

In order to use such a machine effectively on a single task, the component processors must be able to communicate and to synchronize with each other. Many methods of achieving this have been proposed. A widely adopted method of communication is by inspection and updating of a common store (as in Algol 68 [18], PL/I, and many machine codes). However, this can create severe problems in the construction of correct programs and it may lead to expense (e.g. crossbar switches) and

Questions

- ➔Qui est l'auteur et quelles sont ses qualités/reconnaissances à la date de l'article?
- ➔Quels sont les problèmes adressés par l'article?
- ➔Quelles sont les solutions?