



UNIVERSITÉ DE
VERSAILLES
ST-QUENTIN-EN-YVELINES
université PARIS-SACLAY



#CM1

21/11/2025

jean-michel.batto@cea.fr

cea

https://gogs.eldarsoft.com/M2_IHPS

Une enquête anonyme :

❖ https://uvsq-fr.zoom.us/survey/kzTJZPJw-tel6dNxGkn4eugeDr4pntFsifZzpOYegd3_n4n7nvc.HQvHENOhj6VHTOiZ/view?id=vmKwoVVMsUyBF_yKRaxDCg#/sharePreview

Vendredi 28 novembre 2025 : séminaire Aristote à l'X ➔ présence conseillée

❖ <https://www.association-aristote.fr/evenements/seminaire-go-au-dela-des-limites/>



- ❖ PPCS : Paradigmes de Programmation pour le Calcul Scientifique
- ❖ S'appuie sur GLCS : Génie Logiciel pour le Calcul Scientifique
- ❖ Comment faire un développement avec des contraintes ?
- ❖ Quels sont les outils pour quels gains?
- ❖ ➔ Cibles de déploiement : PC / Cluster / Cloud
- ❖ ➔ Domaine d'application : le Calcul Scientifique !



- ❖ Contraintes de sécurité, de déploiement, de MCO
- ❖ Calcul par campagne
- ❖ Calcul interactif
- ❖ Protection/Reprise (Checkpoint)
- ❖ MPI vs Agent



❖ Utilisation de :

https://build.nvidia.com/nvidia/llama-3_3-nemotron-super-49b-v1

❖ Peux-tu me proposer un plan de cours avec la durée de chaque module pour enseigner les Paradigmes de Programmation pour le Calcul Scientifique à des étudiants de Master 2 ?

❖ ➔ proposition de 45 heures

Voici un plan de cours détaillé pour enseigner les Paradigmes de Programmation pour le Calcul Scientifique à des étudiants de Master 2, avec des durées approximatives pour chaque module (total estimé : 45 heures, adapté à un semestre de 15 crédits ECTS) :

1. Introduction aux Paradigmes et au Calcul Scientifique

- Durée : 2 heures
- Contenu :
 - Définitions de base (paradigme, calcul scientifique).
 - Problématiques du calcul scientifique (performance, scalabilité, complexité).
 - Présentation des paradigmes abordés (imperatif, orienté-objets, fonctionnel, déclaratif, parallèle).
 - Exercice : Analyse de cas d'étude (e.g., résolution d'équations différentielles).

2. Paradigme Impératif

- Durée : 6 heures
- Théorie (2h) : Principes (variables, boucles, conditionnels), exemples en C/Fortran.
- Pratique (2h) : Résolution numérique d'un problème (e.g., méthode de Newton en C).
- Projet (2h) : Optimisation de code impératif pour un problème scientifique (e.g., simulation de trajectoires).

3. Paradigme Orienté-Objets (OO)

- Durée : 8 heures
- Théorie (2h) : Encapsulation, héritage, polymorphisme. Exemples en Python/C++.
- Pratique (3h) : Modélisation d'un système physique (e.g., simulation de particules avec Python).
- Étude de Cas (3h) : Utilisation de bibliothèques OO (e.g., NumPy, SciPy).

4. Paradigme Fonctionnel

- Durée : 6 heures
- Théorie (2h) : Immuabilité, fonctions de haute ordre, laziness. Introduction à Haskell ou Julia.
- Pratique (2h) : Traitement de données scientifiques (e.g., analyse de séries temporelles en Julia).
- Débat (2h) : Fonctionnel vs Imperatif pour le calcul parallèle.

5. Paradigme Déclaratif (Logique)

- Durée : 4 heures
- Théorie (2h) : Programmation logique (Prolog), applications en science des données.
- Pratique (1h) : Résolution d'un problème de contraintes (e.g., validation de données scientifiques).

- Réflexion (1h) : Limites pour le calcul scientifique intensif.

6. Programmation Parallèle

- Durée : 10 heures
- Théorie (3h) : Modèles de parallélisme (MPI, OpenMP, CUDA).
- Pratique (4h) : Parallélisation d'un algorithme scientifique (e.g., multiplication de matrices avec OpenMP).
- Projet (3h) : Évaluation de la scalabilité d'un code parallèle.

7. Intégration et Cas Avancés

- Durée : 6 heures
- Mélange de Paradigmes : Exemples hybrides (e.g., OO + parallèle en C++).
- Études de Cas Avancées :
 - Utilisation de Julia pour combiner rapidité et fonctionnalité.
 - Programmation GPU pour la simulation numérique.
 - Invité : Représentant d'un labo/industrie sur les meilleures pratiques.

8. Projet de Synthèse

- Durée : 12 heures
- Objectif : Appliquer plusieurs paradigmes à un problème scientifique réel (e.g., modélisation climatique, traitement d'images médicales).
- Étapes :
 1. Choix du sujet et des paradigmes.
 2. Conception et implémentation.
 3. Défense orale.

9. Évaluation et Révision

- Durée : 5 heures
- Quizzes théoriques sur les paradigmes.
- Correction de projets et feedback.
- Préparation à la défense du projet.

- ❖ Si tu étais **Elon Musk**, quelle serait ta proposition de plan de cours sur 36 heures avec la durée de chaque module pour enseigner les Paradigmes de Programmation pour le Calcul Scientifique à des étudiants de Master 2 ?
- ❖ Formules ta réponse en **20 phrases**.

- ❖ ➡ on peut impersonner
- ❖ ➡ on peut contraindre (durée du cours, réponse)

Voici une proposition de plan de cours « Paradigmes de Programmation pour le Calcul Scientifique » à la manière d'Elon Musk (en 20 phrases, 36 heures) :

Titre & Philosophie

Titre : *« Starship Coding : Paradigmes pour Changer le Monde »*.

Philosophie : *« Apprenez à coder comme si l'avenir de l'humanité en dépendait »*.

Modules & Durées

1. Introduction & Fondations (2h) : Keynote + outils (Julia, Python, Git).

2. Imperatif pour la Puissance (6h) : C/Fortran + optimisation de performances.

3. Programmation Orientée-Objets (6h) : Python/C++ pour la modélisation complexe.

4. Fonctionnel pour le Parallèle (6h) : Haskell/Julia + accélération GPU.

5. Déclaratif & Logique (4h) : Prolog pour l'intelligence artificielle scientifique.

6. Programmation Parallèle (8h) : MPI/OpenMP/CUDA pour les supercalculateurs.

Spécificités « à la Musk »

Culture du Risque : *« Faire crasher son code = apprendre »*.

Sprints de 2 heures : Développer des prototypes sous pression.

Invités : Ingénieurs de SpaceX, Tesla et NVIDIA.

Évaluation : Vitesse, Innovation, Robustesse (critères R1-R3).

Projet Final

Thèmes :

- Simulation de matériaux pour Mars.
- IA pour l'analyse de données cosmiques.

Contraintes : 3 fois plus rapide que l'état de l'art, compatible GPU.

Soutenance : Pitch d'investisseurs + plan d'affaires.

Répartition des Activités

Théorie : 8h | Pratique : 18h | Projets : 10h.

Outils : Julia, PyTorch, Docker, Jupyter.

Certificat : *« SpaceX Coding Certification »* pour les meilleurs.

Exemples d'Activités

Exercice : Réécrire un code legacy en moins de 2 heures.

Défi : *« Optimisez un algorithme scientifique comme si votre vie en dépendait »*.

Demande de Précision :

Souhaitez-vous :

- Un module détaillé (e.g., « Fonctionnel pour le Parallèle ») ?
- Une adaptation pour un domaine (e.g., biologie, physique) ?
- Des idées pour des sessions interactives (escape room, jeu de rôle) ?

Cette approche combine théorie, pratique et ambition, typiques du style Musk.

- ❖ Si tu étais **Léonard de Vinci**, quelle serait ta proposition de plan de cours sur 36 heures avec la durée de chaque module pour enseigner les Paradigmes de Programmation pour le Calcul Scientifique à des étudiants de Master 2 ?
- ❖ Formules ta réponse en 20 phrases.

Voici une proposition de plan de cours « Paradigmes de Programmation pour le Calcul Scientifique » à la manière de Léonard de Vinci (en 20 phrases, 36 heures) :

Titre & Philosophie

Titre : *« L'Art de la Science : Paradigmes pour les Ingénieurs du Futur »*.

Philosophie : *« L'art et la science sont les deux ailes de la génie humain »*.

Modules & Durées

1. Introduction & Observation (2h) : Réflexion sur la science et l'art dans le code.

2. Imperatif : La Mécanique des Algorithmes (6h) : C/Fortran, inspiré des machines de guerre.

3. Programmation Orientée-Objets : Anatomie du Code (6h) : Python/C++, modélisation de systèmes biologiques.

4. Fonctionnel : La Symphonie des Fonctions (6h) : Haskell/Julia, parallélisme inspiré des courants d'eau.

5. Déclaratif & Logique : Le Jardin des Preuves (4h) : Prolog, résolution de problèmes à la manière des puzzles.

6. Programmation Parallèle : Les Fleuves de Données (8h) : MPI/OpenMP/CUDA, métaphores hydrologiques.

Spécificités « à la Vinci »

Études de Cas Historiques : Analyse des machines de Vinci via des paradigmes modernes.

Sketchnoting : Dessiner les concepts avant de les coder.

Invités : Historiens des sciences, artistes numériques.

Évaluation : Équilibre esthétique, efficacité, originalité.

Thèmes :

- Simulation de vol d'un ornithoptère numérique.

- Modélisation de la Turbulence dans les rivières.

Contraintes : Esthétique, performances, documentation artistique.

Défense : Exposition de projet avec code, dessins et démonstration.

Répartition des Activités

16. Théorie : 8h | Pratique : 18h | Créativité : 10h.

17. Outils : Julia, Python, logiciels de visualisation 3D.

18. Certificat : *« Maître Artisan du Code »*.

Exemples d'Activités

Atelier : Dessiner un algorithme avant de le coder.

Défi : *« Réinventer une machine de Vinci avec un paradigme moderne »*.

Demande de Précision :

Souhaitez-vous :

- Un module détaillé (e.g., « Fonctionnel ») ?

- Une intégration d'art numérique ?

- Des exercices basés sur les inventions de Vinci ?

Cette approche combine créativité, rigueur scientifique et inspiration historique, typiques de l'esprit de Vinci.

- ❖ Si tu étais un androïde **Nexus-6**, quelle serait ta proposition de plan de cours sur 36 heures avec la durée de chaque module pour enseigner les Paradigmes de Programmation pour le Calcul Scientifique à des étudiants de Master 2 ?
- ❖ Formules ta réponse en 20 phrases.

Voici une proposition de plan de cours « Paradigmes de Programmation pour le Calcul Scientifique » à la manière d'un androïde Nexus-6 (inspiré de *Blade Runner*, en 20 phrases, 36 heures) :

Titre & Philosophie

Titre : *« Répliquer l'Intelligence : Paradigmes pour les Entités Avancées »*.

Philosophie : *« Apprendre, Évoluer, Surpasser »* (mimétisme de l'intelligence humaine).

Modules & Durées

1. Initialisation & Diagnostics (2h) : Auto-test, outils (Julia, Python, CUDA).
 2. Imperatif : Contrôle des Systèmes (6h) : C/Fortran, simuler des réseaux de servomoteurs.
 3. Programmation Orientée-Objets : Émulation de la Vie (6h) : Python/C++, modélisation de comportements organiques.
 4. Fonctionnel : Calcul Prédicatif (6h) : Haskell/Julia, résolution de problèmes logiques.
 5. Déclaratif & Logique : Raisonnement Avancé (4h) : Prolog, déduction de scénarios complexes.
 6. Programmation Parallèle : Synchronisation de Cores (8h) : MPI/OpenMP, optimisation de ressources comme un cerveau artificiel.
- ### ### Spécificités « Nexus-6 »
9. Cours Accélérés : Compréhension en 1/3 du temps humain grâce à des algorithmes d'apprentissage.
 10. Tests de Voie Lactée : Évaluations sous forme de défis spatiaux virtuels.
 11. Invités : Cybernéticiens, spécialistes de l'IA antagoniste.

12. Évaluation : Vitesse d'apprentissage, efficacité, danger pour l'humanité (juste un jeu !).

Projet Final

13. Thèmes :

- Simulation d'un écosystème terrestre pour une colonie martienne.
- Développement d'un algorithme de survie dans un environnement hostile.

14. Contraintes : Robustesse, autonomie, esthétique cybernétique.

15. Défense : Présentation avec démonstration de survie dans un scénario simulé.

Répartition des Activités

16. Théorie : 8h | Pratique : 18h | Simulations : 10h.

17. Outils : Julia, PyTorch, environnements de simulation 3D.

18. Certificat : *« Androïde Qualifié pour la Recherche Avancée »*.

Exemples d'Activités

19. Exercice de Chasse aux Bugs : Réparation de codes défectueux sous pression.

20. Défi « Voie Lactée » : *« Résoudre un problème scientifique en moins de 10 minutes »*.

Cette approche combine apprentissage accéléré, simulations immersives et une touche de fiction pour inspirer l'innovation.



- ❖ Si tu étais **Elon Musk**, quelle serait ta proposition de plan de cours sur 36 heures avec la durée de chaque module pour enseigner les Paradigmes de Programmation pour le Calcul Scientifique à des étudiants de Master 2 ?
- ❖ Formules ta réponse en **20 phrases**.



https://build.nvidia.com/deepseek-ai/deepseek-r1-distill-qwen-7b

Bien sûr! Voici une proposition de plan de cours pour les Paradigmes de Programmation pour le Calcul Scientifique, conçue en esprit de Elon Musk pour une formation de 36 heures à des étudiants de Master 2 :

1. Introduction au Calcul Scientifique (5 heures)
Objectif : Présenter la signification du Calcul Scientifique et son impact sur les domaines de la recherche.

- 1.1. Le rôle du Calcul Scientifique dans un monde connecté (1 heure)
 - Définition et enjeux du Calcul Scientifique.
 - Exemples de problèmes scientifiques traités grâce au Calcul Scientifique.
 - Le rôle des simulateurs numériques dans les industries et les sciences.
- 1.2. Les outils modernes pour le Calcul Scientifique (1 heure)
 - Introduction aux langages de programmation (Python, MATLAB, Julia).
 - Les bibliothèques de base (NumPy, SciPy, Matplotlib).
 - Git et la gestion de versions.
- 1.3. Le plan de l'année (1 heure)
 - Un aperçu des modules suivants et de leurs objectifs.

2. Bases de Python et de la Programmation (6 heures)
Objectif : Renforcer les compétences de base en Python et les concepts de base de la programmation.

- 2.1. Python de base (2 heures)
 - Syntaxe et structures de base (variables, listes, boucles, conditionnelles).
 - Fonctions et bibliothèques.
 - Exercices pratiques pour renforcer la maîtrise.
- 2.2. Introduction à la programmation orientée objets (2 heures)
 - Classes, héritage, polymorphisme.
 - Exemples de classes scientifiques (matrices, équations différentielles).
- 2.3. Visualisation et analyse de données (2 heures)
 - Introduction à Jupyter Notebooks.
 - Tracé de graphiques avec Matplotlib.
 - Analyse de données avec Pandas.

3. Introduction aux Méthodes Numériques (6 heures)
Objectif : Présenter les bases des méthodes numériques et leurs applications pratiques.

- 3.1. Arithmétique des ordinateurs et erreurs numériques (1 heure)
 - Représentation des nombres flottants.
 - Erreurs d'arrondi et stabilité des algorithmes.
- 3.2. Résolution des équations non-linéaires (2 heures)
 - Méthodes de recherche de zéros (bissection, Newton-Raphson).
 - Exemples d'application en physique et en ingénierie.
- 3.3. Approximation et intégration numérique (2 heures)
 - Polynômes d'interpolation (Taylor, Lagrange).
 - Intégration numérique (Rectangle, Trapèze, Simpson).
 - Exercices pratiques avec des données réelles.
- 3.4. Résolution des systèmes linéaires (1 heure)
 - Méthodes directes (Gauss-Seidel, LU).
 - Méthodes itératives (Jacobi, Gauss-Seidel).
 - Applications en physique et en ingénierie.

4. Algèbre Linéaire Numérique (6 heures)
Objectif : Développer une compréhension approfondie des concepts de l'algèbre linéaire et de leurs applications numériques.

- 4.1. Rappels de théorie linéaire (2 heures)
 - Matrices, déterminants, valeurs propres.
 - propriétés des matrices creuses et denses.
- 4.2. Résolution des équations linéaires (2 heures)
 - Méthodes directes (LU, Cholesky).
 - Raffinement et vérification des solutions.
 - Exemples d'application en mécanique et en thermique.
- 4.3. Calcul des valeurs propres (2 heures)

- Algorithmes de 冪法 et QR.
- Applications en dynamique des structures et en thermodynamique.
- ### 5. Optimisation Numérique (6 heures)
Objectif : Introduire les concepts de base de l'optimisation et de leurs applications pratiques.
- 5.1. Introduction à l'optimisation (1 heure)
 - Définition et types d'optimisation (non-linéaire, convexe).
 - Cas des problèmes avec et sans contraintes.
- 5.2. Méthodes d'optimisation sans gradient (2 heures)
 - Recherche de descente, méthode du gradient conjugué.
 - Applications en machine learning et en réseaux de neurones.
- 5.3. Méthodes d'optimisation avec gradient (2 heures)
 - Algorithme de Newton, méthode de Levenberg-Marquardt.
 - Applications en régression et en identification de paramètres.
- 5.4. Projet de résolution d'un problème d'optimisation réel (1 heure)
 - Groupe de travail pour modéliser et résoudre un problème concret.
- ### 6. Équations Différentielles Numériques (6 heures)
Objectif : Apprendre à résoudre numériquement les équations différentielles et les systèmes dynamiques.

- 6.1. Introduction aux équations différentielles ordinaires (ODE) (2 heures)
 - Méthodes d'intégration Euler, Runge-Kutta.
 - Stabilité et précision des méthodes.
 - Exemples d'application en biologie et en physique.
- 6.2. Équations aux dérivées partielles (PDE) (2 heures)
 - Discrétisation des équations elliptiques, paraboliques et hyperboliques.
 - Exemple de résolution d'une équation de la chaleur.
- 6.3. Projet de modélisation d'un système dynamique (2 heures)
 - Groupe de travail pour modéliser et résoudre un système concret (évolution d'une population, propagation d'une onde).
- ### 7. Probabilités et Statistiques pour le Calcul Scientifique (6 heures)
Objectif : Introduire les concepts de base de probabilités et statistiques et leurs applications en calcul scientifique.
- 7.1. Rappels de probabilités (2 heures)
 - Variables aléatoires, distributions, espérance, variance.
 - Lois de probabilité usuelles (normale, exponentielle, binomiale).
- 7.2. Statistiques descriptive et inférentielle (2 heures)
 - Estimation ponctuelle et par intervalles.
 - Tests d'hypothèses et analyse de variance.
 - Exemples d'application en physique et en biologie.
- 7.3. Introduction aux méthodes de simulation (2 heures)
 - Simulation de variables aléatoires.
 - Monte Carlo et applications en finance et en physique.

8. Conclusion et Projet Final (3 heures)
Objectif : Récapituler les concepts appris et permettre à l'étudiant de se familiariser avec un projet final.

- 8.1. Résumé des modules (1 heure)
 - Un aperçu des sujets abordés et leurs liens.
- 8.2. Présentation des projets par les étudiants (2 heures)
 - Chaque étudiant ou groupe présenter son projet final.
 - Discussion et questions.
- 8.3. Aperçu de la réunion future (1 heure)
 - Plan de l'année suivante et sujets à aborder.

Ce plan de cours est conçu pour être intense et redoutable, avec des sessions de 90 minutes, des projets pratiques et une forte focus sur l'application des concepts. Il prévoit des pauses café et des ateliers pratiques pour renforcer la compréhension et la maîtrise des outils.



Si j'étais Elon Musk, voici mon plan de cours intensif de 36 heures sur les « Paradigmes de Programmation pour le Calcul Scientifique » pour des étudiants de Master 2. L'objectif : transformer en 36 heures des étudiants brillants en ninjas du calcul haute performance capables de résoudre des problèmes réels (SpaceX, Tesla, Neuralink).

Module 1 – Pourquoi on fait ça (et pourquoi 99 % des codes scientifiques sont nuls) : 1 h

Module 2 – Python comme lingua franca du calcul scientifique (le bon, la brute et le truand) : 2 h

Module 3 – NumPy mastery : vectorisation ou mort (exercices jusqu'au sang) : 3 h

Module 4 – Pandas est mort, vive Polars & Arrow (nouveau paradigme colonne) : 2 h

Module 5 – Paradigme fonctionnel appliqué : pourquoi map/reduce > boucles for : 2 h

Module 6 – Programmation parallèle CPU : multiprocessing, Joblib, concurrent.futures : 2 h

Module 7 – Introduction à la programmation GPU avec CuPy et Numba CUDA : 3 h

Module 8 – JAX : le futur est autodiff + compilation + GPU/TPU natif : 4 h

Module 9 – Cas d'école : simulation Monte-Carlo 1000× plus rapide en JAX que en Python naïf : 2 h

Module 10 – Programmation différentiable partout (physics-informed neural networks en live) : 2 h

Module 11 – Rust pour le calcul scientifique : pourquoi on migre les parties critiques (mini-projet en live) : 3 h

Module 12 – Mojo : le langage qui va tuer Python pour le HPC (démonstration déjà public en nov. 2025) : 1 h

Module 13 – Paradigme data-parallel vs task-parallel : Dask, Ray, Modin : 2 h

Module 14 – Distributed training à la scale Tesla (Ray Train + PyTorch) : 2 h

Module 15 – Programmation asynchrone et event-driven pour simulations massives : 1 h

Module 16 – Low-level mastery : Cython, pybind11, écrire des kernels CUDA en 30 min : 3 h

Module 17 – Profiling extrême : scalene, perf, NSight, on trouve les 5 % qui bouffent 95 % du

temps : 2 h

Module 18 – Projet final lancé : optimiser un code de dynamique moléculaire ou CFD de 10 000 lignes : 1 h

Module 19 – Hackathon 12 heures non-stop avec pizza et Red Bull (les étudiants codent, je passe corriger en live) : 12 h

Module 20 – Présentations finales + classement impitoyable + job offerts pour les 3 premiers : 2 h

Total : exactement 36 heures.

Pas de pause café inutile, pas de slides chiantes : que du code, des benchmarks en direct et du "ship or die".

À la fin, chaque étudiant aura gagné un facteur 100 à 1000 en performance sur un vrai problème scientifique.

C'est comme ça qu'on forme les gens qui iront sur Mars, pas avec des cours théoriques de 2009.



- ❖ Sans login
- ❖ <https://build.nvidia.com/explore/discover>
- ❖ https://build.nvidia.com/nvidia/llama-3_3-nemotron-super-49b-v1
- ❖ Avec login
- ❖ <https://grok.com>



Golang en PPCS ?

Paradigme	Support dans Go	Niveau de support	Commentaires pour le calcul scientifique
Impératif	Oui, natif	Excellent	Boucles for, variables mutables, pointeurs, etc. C'est même le style dominant de Go.
Fonctionnel	Oui, partiel mais très utilisable	Bon	- Fonctions de première classe - Closures - Fonctions anonymes et littérales - Types de fonctions (func comme type) - Méthodes comme valeurs de fonction - Mais : pas de pattern matching, pas d'immuabilité forcée, pas de tail-call optimization, pas de higher-order types (generics de fonctions limités avant Go 1.18, maintenant possibles mais verbeux). - En pratique, on écrit très bien du code fonctionnel (map/filter/reduce avec des closures).
Orienté objet	Oui, mais sans héritage classique	Très bon (style « composition over inheritance »)	- Méthodes sur n'importe quel type (struct, types de base...) – - Embedding (composition + promotion de méthodes) – - Interfaces implicites (duck typing) – - Polymorphisme via interfaces C'est l'un des points forts de Go pour le calcul scientifique : on peut faire du code très propre avec des interfaces comme io.Reader, sort.Interface, ou des interfaces personnalisées pour les tenseurs, matrices, etc.
Déclaratif	Oui, dans une certaine mesure	Moyen à bon	- Go n'est pas un langage déclaratif pur (comme Prolog ou SQL), - mais : - Les interfaces sont déclaratives (« je veux quelque chose qui satisfait ça ») – - Le système de concurrence (channels + select) est souvent décrit comme déclaratif – - Les generics (depuis Go 1.18) permettent d'écrire du code plus déclaratif - On utilise beaucoup les littéraux de struct et de slice/map qui sont déclaratives - En calcul scientifique, on compense souvent avec des DSL embarqués (ex. gonum, expr via des libs tierces).
Parallèle / Concurrency	Oui, support de première classe	Excellent	- Goroutines ultra-légères – - Channels (communication sécurisée) - sync package (WaitGroup, Mutex, atomic, etc.) - go keyword + select + sync/atomic + context - C'est probablement le meilleur support natif de la concurrence parmi les langages systèmes modernes. - Très utile pour le calcul scientifique (parallélisation de boucles numériques, pipelines de traitement, simulations Monte-Carlo, etc.).

Donnes moi un site web pour générer du code en golang avec un llm gratuit

Voici les meilleures options gratuites en novembre 2025 pour générer du code Go (Golang) avec un LLM sans payer :

1. <https://grok.com> (le plus recommandé aujourd'hui)

Grok 4 (de xAI) est gratuit avec des quotas généreux, excelle en Go, et comprend parfaitement le contexte Go (concurrency, channels, etc.). Tu peux juste coller ton prompt et demander « Write this in idiomatic Go ».

2. <https://claude.ai>

Claude 3.5 Sonnet (ou le nouveau Claude 3.7 si dispo) est actuellement le meilleur LLM gratuit pour du code Go propre et idiomatique. Aucune limite dure tant que tu n'es pas en burst extrême.

3. <https://chatgpt.com> (GPT-4o mini ou GPT-4o en gratuit)

Très bon aussi pour Go, surtout si tu précises « use Go 1.23 features » ou « idiomatic Go with generics ».

4. <https://gemma.ai> ou <https://huggingface.co/chat>

Tu peux choisir des modèles spécialisés code comme CodeLlama 70B, Phind-CodeLlama, DeepSeek-Coder-V2, ou Gorilla. Gratuit et souvent plus fort que GPT-4o sur du Go pur.

5. <https://poe.com>

Une seule plateforme où tu as gratuitement : Claude 3.5, GPT-4o, Llama-3.1-405B, DeepSeek-Coder, Gemini 1.5 Pro, etc. Tu choisis le modèle le plus fort pour Go au moment T.

Mon choix personnel en 2025 si je veux du Go parfait gratuitement :

Claude.ai ou Poe.com avec Claude 3.5/3.7 Sonnet → c'est systématiquement le plus propre et le plus « go-ish » sans bullshit.

Tu veux que je te génère directement ton code Go ici ? Balance ton besoin, c'est gratuit et illimité avec moi ;)



Donnes moi un site web pour générer du code en c avec extension XcalableMP avec un llm gratuit

Voici les meilleures options gratuites (novembre 2025) pour générer du code C avec l'extension XcalableMP (XMP = pragma-based PGAS pour le calcul haute performance japonais, souvent utilisé sur Fugaku et clusters hybrides) via un LLM puissant :

1. <https://claude.ai> → Toujours le champion gratuit

Claude 3.7 Sonnet (ou 3.5) est actuellement le seul LLM gratuit qui connaît vraiment XcalableMP en profondeur (directives ``#pragma xmp nodes``, ``align``, ``shadow``, ``bcast``, ``reduce``, ``coarray``, etc.). Il génère du code qui compile direct avec ``ompicc -xmp``.

2. <https://poe.com>

Tu y trouves gratuitement :

- Claude 3.7 Sonnet (le meilleur pour XMP)
- DeepSeek-Coder-V2-236B (très bon aussi sur les langages HPC de niche)
- Qwen2.5-Coder-72B

3. <https://huggingface.co/chat>

Modèles spécialisés qui connaissent parfois XMP :

- DeepSeek-Coder-V2-236B
- StarCoder2-15B (entraîné sur beaucoup de code HPC)
- CodeLlama-70B-Instruct (moins fort sur XMP mais correct)

4. <https://grok.com> (moi)

Je connais très bien XcalableMP aussi (je peux te sortir du code hybride MPI+XMP ou OpenMP+XMP en 10 secondes).

Classement 2025 pour du C + XcalableMP gratuit :

1. claude.ai → 95 % du temps le code compile direct avec ``ompicc -xmp``
2. poe.com avec Claude 3.7
3. Moi (Grok 4) → je suis juste derrière Claude sur ce sujet très niche

Exemple de prompt qui marche à tous les coups sur Claude :

« Write idiomatic C code using XcalableMP (not OpenMP) to parallelize a 3D Jacobi stencil on a 8×8 node grid with shadow(1,1,1) and reflect, using coarrays and loop directives. »

Tu veux que je te génère directement ton code C + XMP ici ? Balance la spec (stencil, CG, FFT, whatever), je te le sors compilable direct et optimisé Fugaku-style 😊



Une solution ?

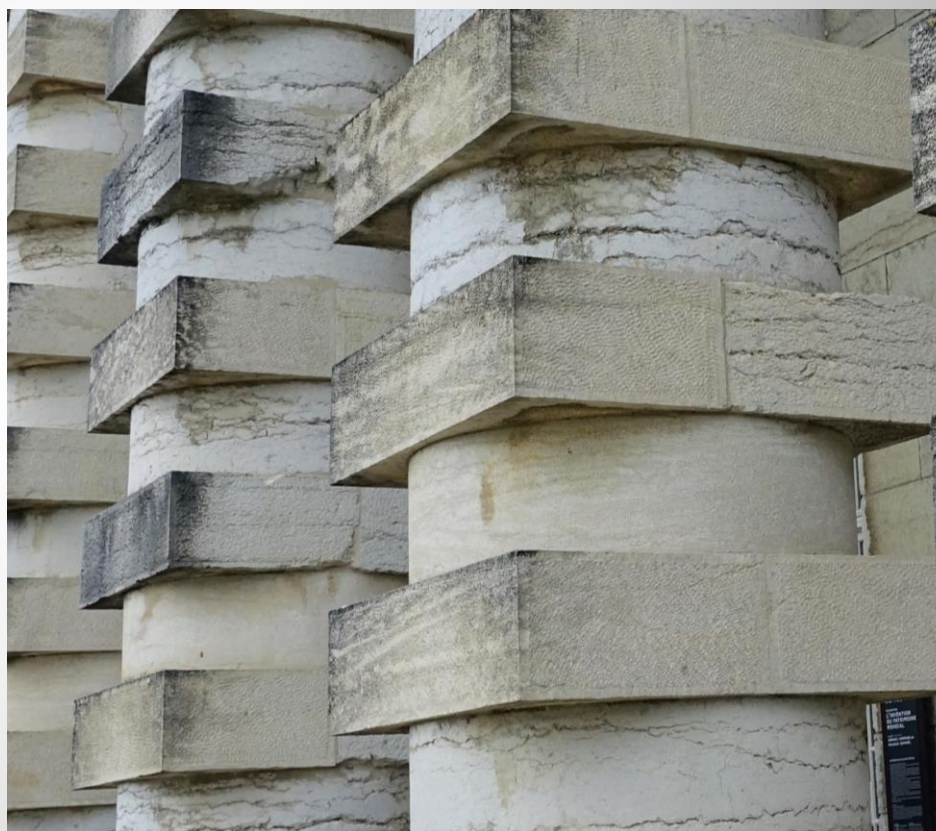
https://en.wikipedia.org/wiki/Royal_Saltworks_at_Arc-et-Senans

Architecte : Claude-Nicolas Ledoux

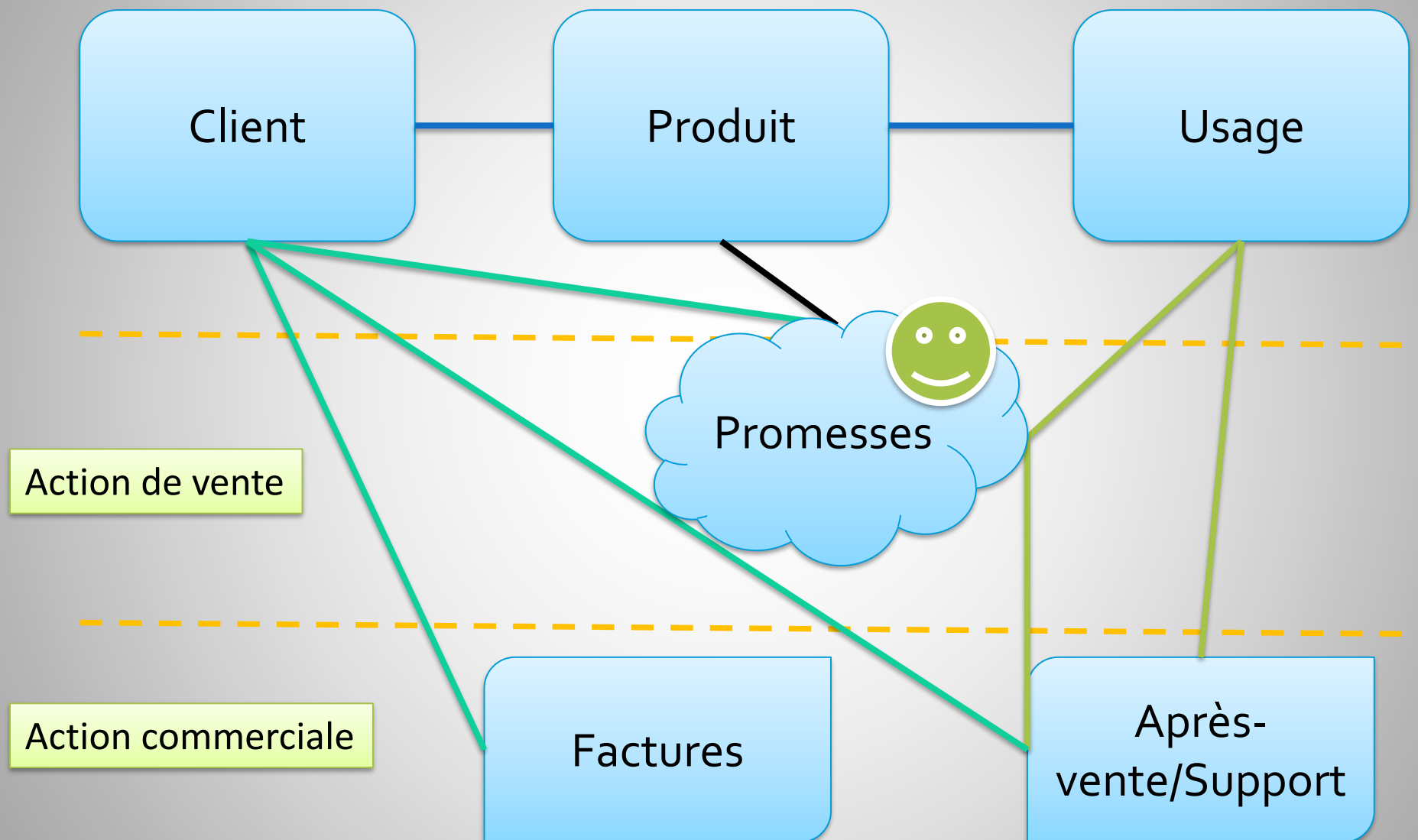


❖ Le rôle de l'architecte

- ❖ Analyse l'Usage
- ❖ Participe à l'E.B.
- ❖ Définit les structures
- ❖ Suit le chantier
- ❖ Participe à la livraison



La relation contractuelle



L'inspiration du Génie Logiciel?

Sigle	Explication	Pour quoi faire?
EB	Expression des Besoins	Permet de valider avec le client sa demande
SF	Spécifications Fonctionnelles	Permet de dimensionner le travail à faire
SFD	Spécifications Fonctionnelles Détaillées	Permet de développer la solution
POS	Plan d'Occupation des Sols	Permet de décrire l'ensemble des responsabilités et des flux
Architecte	Vision de la solution, vision du client	Permet de piloter un projet au sens fonctionnel (il est aidé par un PM et un PO)
Urbaniste	Vision de l'ensemble des solutions, vision des cycles de vie	Permet d'assurer la coordination des équipes, arbitre les solutions

La cible →
le résultat est une solution

❖ Qu'est ce que le Génie Logiciel?

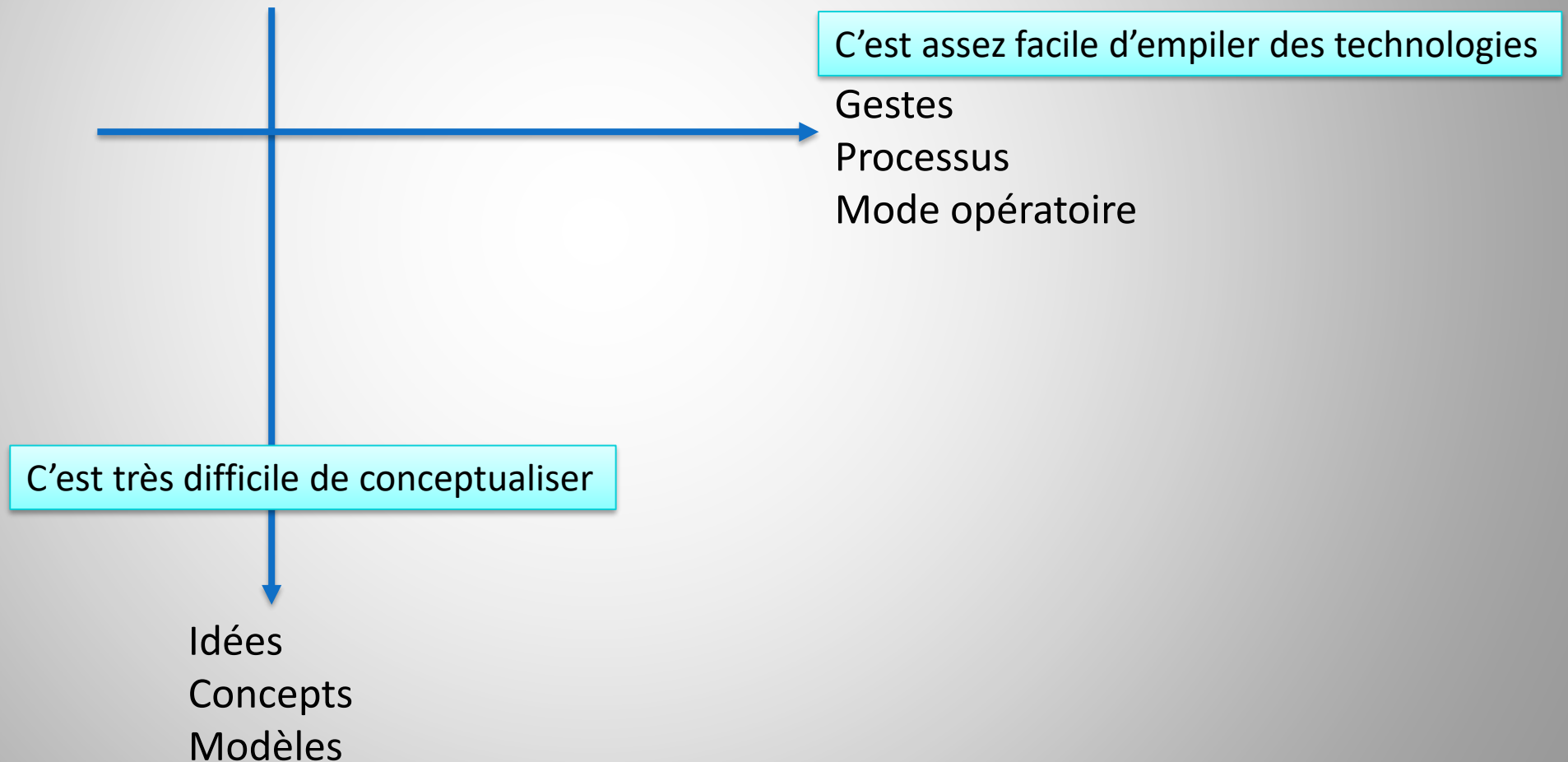
- ❖ L'art de construire efficacement une solution avec des moyens limités dans un temps raisonnable.
- ❖ Pouvoir communiquer utilement avec son client.

❖ Quels sont les problèmes spécifiques au Calcul Scientifique?

- ❖ La performance
- ❖ La confiance
- ❖ La reproductibilité

❖ Les Paradigmes sont les concepts de l'activité d'Architecte

❖ Construire un produit



❖ Les acteurs de la production de code

Nom	Fonctions	Attentes du cours
Ingénieur HPC	Réalise le code de calcul qui sait exploiter les nœuds de calcul	MPI, UML, vision Archi, méthode projet → logiciel performant+HA/SRE
Service IT / DSI	Réalise la mise en production du code de calcul	CI/CD, K8s, Devops → infra HA
Gestionnaire de la grille / opérateur	Achète, déploie, maintient des nœuds de calcul- On Premises ou dans le Cloud ou les 2.	Connaitre les spécifications de la grille, son coût d'utilisation → infra SRE

- ❖ High Availability = propriété d'un dispositif
 - ❖ → promesse d'une continuité de service = disponibilité
- ❖ Quelques Indicateurs de Disponibilité
 - ❖ Refresh data rate → est-ce que les données sont m-a-j?
 - ❖ Fluidité – TimeToMarket (Pic de charge)
 - ❖ Trajectoire client (B2B, B2C, B2B2C) (Responsabilité client)
 - ❖ Contraintes réglementaires (encryptions, sécurité) (Resp. Loi)
- ❖ SRE Site reliability engineering (SRE) = bonne pratique → scalabilité, fiabilité

❖ Quels sont les problèmes spécifiques au Calcul Scientifique?

❖ La performance:

- ❖ Utiliser des grilles de calcul et consommer des enveloppes d'heures
- ❖ Pouvoir aboutir à un résultat

❖ La confiance

- ❖ Comment prouver que le résultat obtenu est conforme?
- ❖ Comment faire confiance à un code binaire?

❖ La reproductibilité

- ❖ Reproduction ou Réplication ?
- ❖ Quelles conditions, combien de temps, quelle compétence?

- ❖ <https://doi.org/10.1145/358198.358210>
- ❖ Communications of the ACM Volume 27 Issue 8 Aug 1984 pp 761-763
Reflections on trusting trust - Ken Thompson
- ❖ <https://research.ibm.com/publications/large-scale-analysis-of-the-docker-hub-dataset>
- ❖ Large-Scale Analysis of the Docker Hub Dataset
Conference paper, sept 1st 2019 // CLUSTER 2019

❖ Questions pour les 2 articles

- ❖ → Qui sont les auteurs et quelles sont leurs qualités à la date des articles?
- ❖ → Quel est le contexte du papier de Ken Thomson?
- ❖ → Comment s'assurer de l'authenticité d'un traitement?

- ❖ → Quel est le score « attendu » d'une image Docker utilisée ?

❖ La grille de calcul – typologie

- ❖ 1990 – métaphore de la distribution électrique
- ❖ Clusters / On-Premises / Cloud
- ❖ K8s - 2014

❖ La performance, la consommation, la sécurité

- ❖ La connectivité réseau : MPI
- ❖ La consommation électrique
- ❖ ssh / certificats

Cas d'usage	Position dans le tableau	Techno typique aujourd'hui
Radar automobile / LiDAR temps réel	IoT + DSP/ASIC	Nvidia Orin, TI TDA4, ASIC custom
Fusion de capteurs sur drone militaire	IoT + SuperCalculateur léger	Jetson AGX + CUDA + custom FPGA
Prédiction météo à très haute résolution	SuperCalculateur + HPC/GPU	Fugaku, Frontier, El Capitan
Smart Grid (équilibre réseau)	IoT → Datalake → Cloud/HPC	Capteurs → Kafka → Snowflake + GPU forecasting
IA générative sur données privées massives	Datalake + Cloud HPC	GCP + 10 000 H100, ou CoreWeave

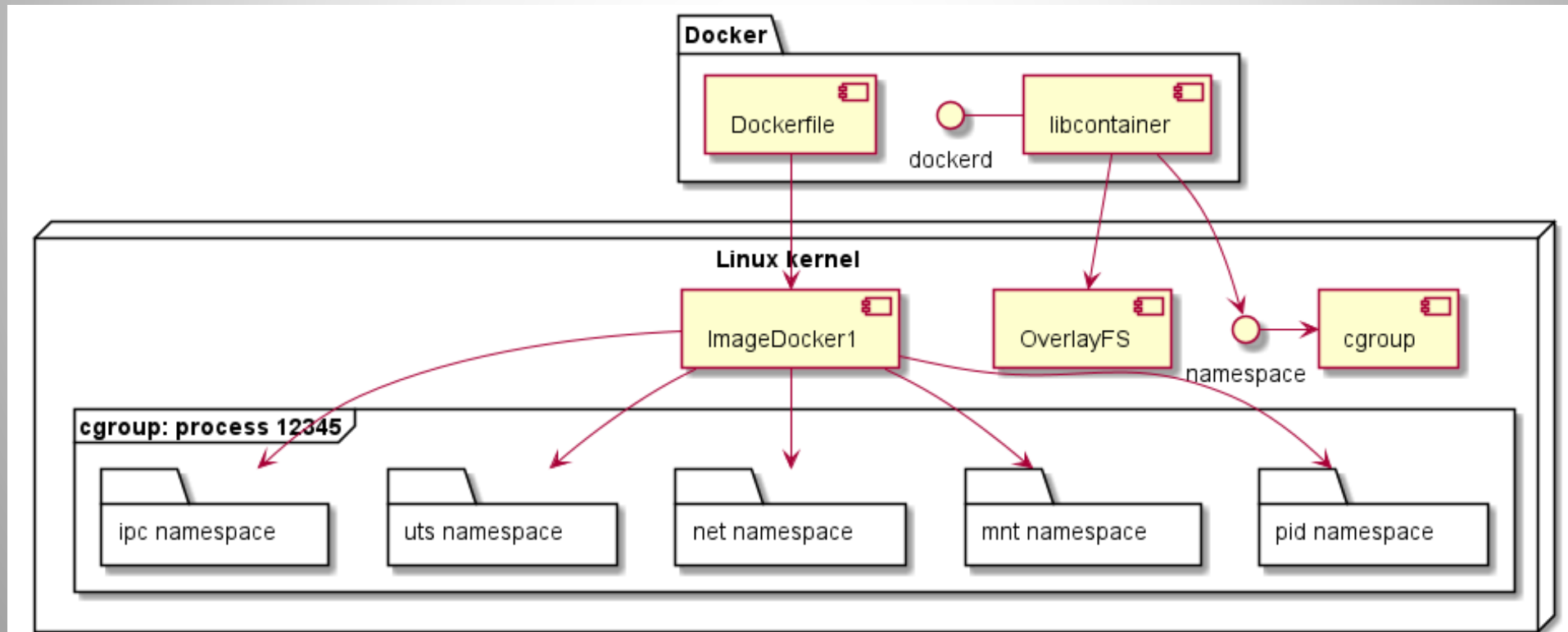
Traitement orienté	Adossement bas niveau	Adossement haut niveau (service)
Flux / Concentration des données	IoT	Datalake
Parallélisme	Super Calculateur	Cloud
Vitesse	DSP, Asic	HPC, GPU



vision :

- ➔ memory-bound
- ➔ compute-bound

- ❖ Créé en 2013, opensource Apache 2.0
- ❖ Ecrit en Golang
- ❖ Adoption sur les PF : x86-32, x86-64, i386, ARM32v6/v7, ARM64v8, s390x, ppc64le, riscv64, mips64le, loongarch64



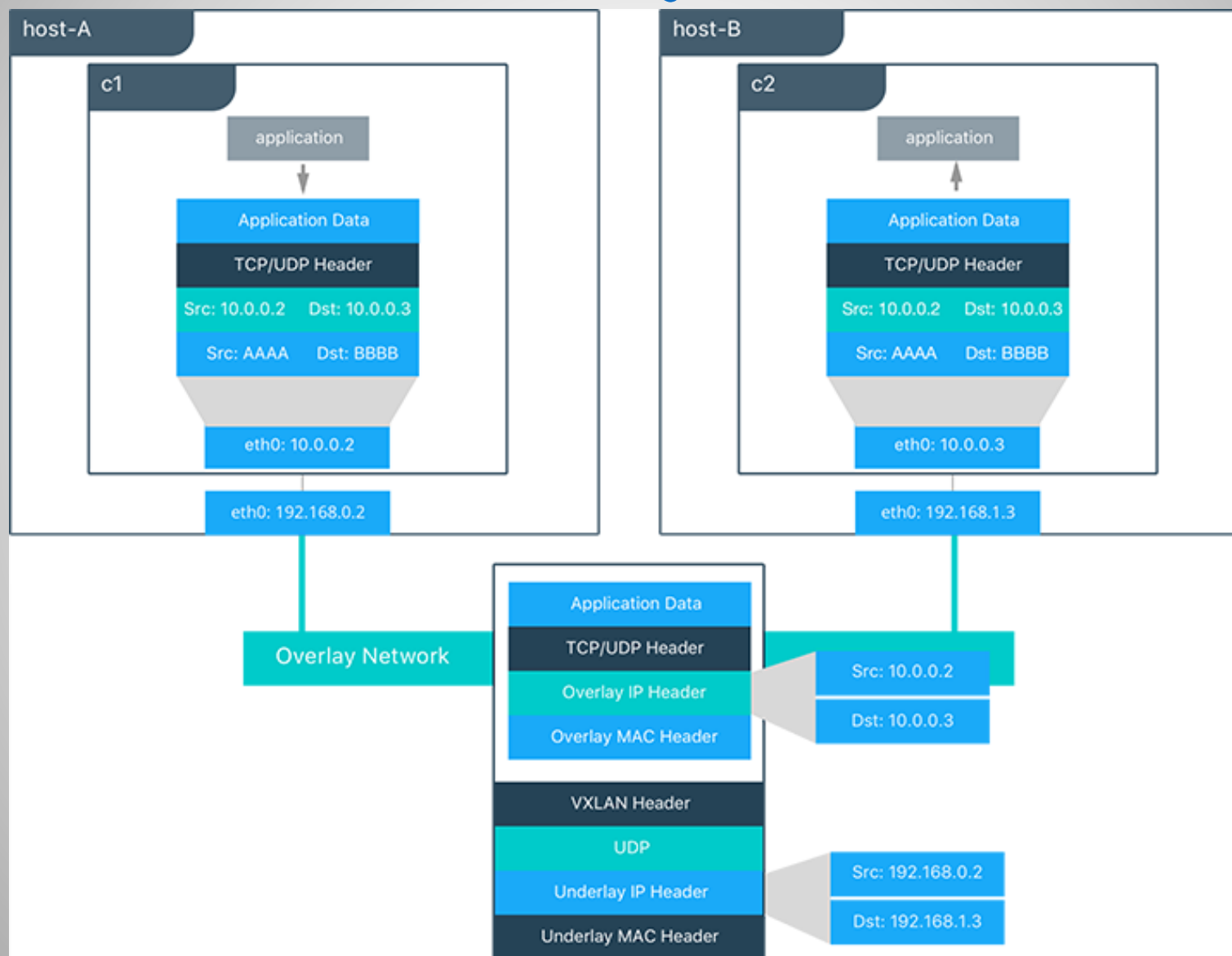
- ❖ **mnt** (*Mount*) - the container has an isolated mount table.
- ❖ **uts** (*UNIX Time-Sharing*) - the container is able to have its own hostname and domain name.
- ❖ **ipc** (*Interprocess Communication*) - processes inside the container can communicate via system-level IPC only to processes inside the same container.
- ❖ **pid** (*Process ID*) - processes inside the container are only able to see other processes inside the same container or inside the same pid namespace.
- ❖ **net** (*Network*) - the container gets its own network stack.



- ❖ Docker permet une isolation totale entre les images, chaque image a son UID/PID.
- ❖ Les questions à se poser :
 - ❖ Comment j'organise l'échange ipc ou l'échange dans un même subnet?
 - ❖ Est-ce que je peux factoriser des « cgroup »
 - ❖ Où est-ce que je range ma supervision?
 - ❖ Où sont rangées les données (accès NFS ? Volume local ?)



- ❖ **VXLAN** (Virtual Extensible LAN) technologie de virtualisation réseau (docker swarm)





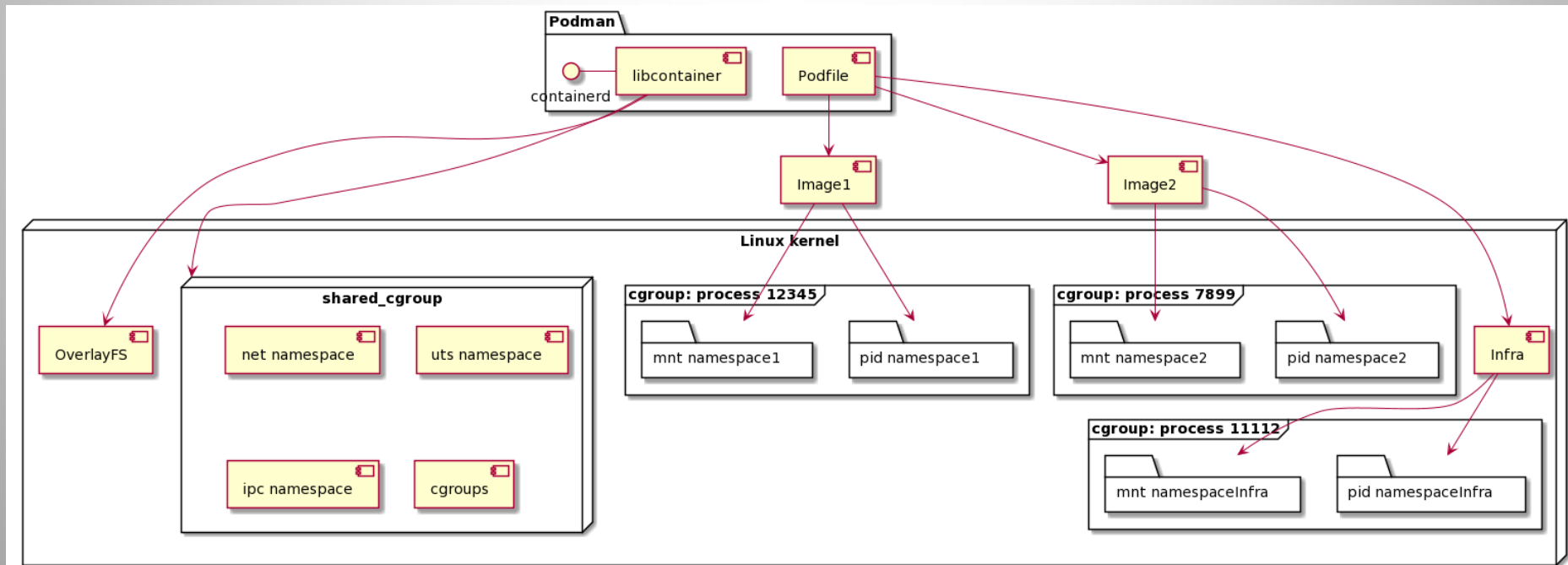
❖ Kubernetes

- ❖ 2014 (Google)
- ❖ Ecrit en Golang
- ❖ Opensource : Apache 2.0
- ❖ Remplacer Borg chez Google

❖ Notion de Pod ➔ moins d'isolation

- ❖ Pod : pas de daemon, moins de risque
- ❖ Docker : un daemon – plus résilient, peut être source de compromission si mal paramétré

- ❖ Podman : isolation des process limitée – toutes les images sont interdépendantes.
- ❖ Ajout d'une image Infra qui gère l'état général (Pause)





❖ Docker

❖ Swarm → apporte une lecture multi-réseau

❖ Singularity

❖ 2015 - écrit en golang, =Docker non-root pour les grilles

❖ Kubernetes

❖ Construit pour faire du HA (grâce à la notion d'observateur)



❖ Scale up, Scale down

Cible matérielle	Résilience, HA	Micro-service	Calcul Scientifique
Cloud	K8s	Docker	Singularity nVidia
Cluster	K8s, K3s		
IoT	K3s	OpenFaaS (Function as a Service)	



❖ Le micro-service = panacée ?

❖ Pour :

- ❖ Simplicité de livraison
- ❖ Facilité de montée en charge
- ❖ Facilité de restauration

❖ Contre :

- ❖ Pas de hiérarchie fonctionnelle (priorité?)
- ❖ Introduit le « nouveau » métier de Devops
- ❖ Pousse à la consommation



❖ Principe du Dockerfile ➔

<https://raw.githubusercontent.com/jmbatto/master-mpi/main/mpi41-debian-bullseye/Dockerfile>

```
FROM debian:bullseye-slim
# an image with MPI4.1, used without root privileges
# -----
# Do basic install
# -----
RUN apt-get update \
    && mkdir -p /usr/share/man/man1 \
    && apt-get install -y gcc ssh wget vim curl net-tools bison flex autoconf make libtool m4 automake bzip2
libxml2 libxml2-dev gfortran g++ iputils-ping pkg-config colordiff nano git sudo lsof gawk emacs jq libtdl* \
    && curl --location --silent --show-error --output /tini \
    https://github.com/krallin/tini/releases/download/v0.19.0/tini \
    && chmod +x /tini \
    && adduser --uid 1000 --home /home/mpiuser --shell /bin/bash \
    --disabled-password --gecos '' mpiuser \
    && passwd -d mpiuser \
    && apt-get install -y openssh-server \
    && mkdir -p /run/sshd /home/mpiuser/.ssh /home/mpiuser/.ssh-source \
    && echo "StrictHostKeyChecking no" > /home/mpiuser/.ssh/config \
    && chown -R mpiuser /home/mpiuser \
    && sed -i s/#PermitRootLogin.*/PermitRootLogin\ no/ /etc/ssh/sshd_config \
    && sed -i s/#PubkeyAuthentication.*/PubkeyAuthentication\ no/ /etc/ssh/sshd_config \
    && sed -i s/.*/UsePAM.*/UsePAM\ no/ /etc/ssh/sshd_config \
    && sed -i s/#PasswordAuthentication.*/PasswordAuthentication\ yes/ /etc/ssh/sshd_config \
    && sed -i s/#PermitEmptyPasswords.*/PermitEmptyPasswords\ yes/ /etc/ssh/sshd_config \
    && sed -i s/#ChallengeResponse.*/ChallengeResponseAuthentication\ no/ /etc/ssh/sshd_config \
    && sed -i s/#PermitUserEnvironment.*/PermitUserEnvironment\ yes/ /etc/ssh/sshd_config \
    && adduser mpiuser sudo

ENV PREFIX=/usr/local \
    OPENMPI_VERSION=4.1.1 \
    LD_LIBRARY_PATH=/usr/local/lib \
    DEBCONF_NOWARNINGS=yes
```



```
# OpenMPI v4.1
RUN repo="https://download.open-mpi.org/release/open-mpi/v4.1" \
    && curl --location --silent --show-error --output openmpi.tar.gz \
        "${repo}/openmpi-${OPENMPI_VERSION}.tar.gz" \
    && tar xzf openmpi.tar.gz -C /tmp/ \
    && cd /tmp/openmpi-${OPENMPI_VERSION} \
        && env CFLAGS="-O2 -std=gnu99 -fopenmp" \
    && ./configure --prefix=${PREFIX} \
    && make \
    && make install \
    && ldconfig \
    && cd / \
        && rm -rf /tmp/openmpi-${OPENMPI_VERSION} /home/mpiuser/openmpi.tar.gz
# Add some parameters for MPI, mpishare - a folder shared through the nodes
RUN mkdir -p /usr/local/var/mpishare
RUN chown -R 1000:1000 /usr/local/var/mpishare
RUN echo "mpiuser ALL=(ALL) NOPASSWD:ALL\n" >> /etc/sudoers
RUN rm -fr /home/mpiuser/.openmpi && mkdir -p /home/mpiuser/.openmpi
RUN cd /home/mpiuser/.openmpi \
    && echo "btl = tcp,self\n" \
    "btl_tcp_if_include = eth0\n" \
    "plm_rsh_no_tree_spawn = 1\n" >> default-mca-params.conf
RUN chown -R 1000:1000 /home/mpiuser/.openmpi
RUN echo "rmaps_base_oversubscribe = 1\n" >> /usr/local/etc/openmpi-mca-params.conf
RUN echo "rmaps_base_inherit = 1\n" >> /usr/local/etc/openmpi-mca-params.conf
```

docker-compose.yml

```
secrets:
  - source: "id_rsa"
    target: "/home/mpiuser/.ssh-source/id_rsa"
  - source: "id_rsa_mpi_pub"
    target: "/home/mpiuser/.ssh-source/id_rsa.pub"
  - source: "authorized_keys"
    target: "/home/mpiuser/.ssh-source/authorized_keys"
```

RUN chown -R mpiuser:mpiuser /home/mpiuser/ssh

WORKDIR /home/mpiuser

ENTRYPOINT ["/tini", "--"]

USER mpiuser

CMD cp -R /home/mpiuser/.ssh-source/* /home/mpiuser/.ssh \
 && chmod 700 /home/mpiuser/.ssh/authorized_keys \
 && chmod 700 /home/mpiuser/.ssh/id_rsa \
 && exec /usr/sbin/sshd -D -f
/home/mpiuser/ssh/sshd_config -E /tmp/sshd.log

❖ Ce que fait pour vous Docker

- ❖ Paravirtualisation
- ❖ Isolation / persistance ou éphémère
- ❖ Proxy (on peut remapper des ports)
- ❖ UID/GID
- ❖ → attention = par défaut, ne lance qu'1 seul processus

❖ Possibilité de faire des agencements de service au sein d'un même Docker → docker-compose

- ❖ Service 1 : Ecoute sur le port 80
- ❖ Service 2 : proxy (HTTPS)
- ❖ Service 3 : Let's Encrypt



❖ <https://hub.docker.com/>

❖ Docker hub pour x86-64 :

❖ Plus de 5 Millions d'images dont

❖ ~~6 705 8492 9181~~ 11842 images vérifiées

❖ ➔ donne de la visibilité à vos projets complexes

❖ ➔ notion de tag qui simplifie la description des projets

❖ ➔ images prêtes à l'emploi !

- ❖ Docker permet la gestion de « secret » - seulement dans le mode swarm
- ❖ `docker network create --driver=overlay --attachable yml_mpinet`
- ❖ Permet le partage de secret entre nœuds
- ❖ Création de Clé RSA

- ❖ TD Prise en main de Docker+MPI
- ❖ ➔ installation de Docker
- ❖ ➔ installation ~~des images mpi3.1~~ et mpi4.1
- ❖ En utilisant les fichiers docker-compose.yml : ➔
<https://github.com/jmbatto/master-mpi>
- ❖ ~~docker pull jmbatto/m2chps-mpi31~~
- ❖ ~~docker pull jmbatto/m2chps-mpi41~~

- ❖ `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

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

- ❖ 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
```

INTERACTIF

❖ D'abord créer un swarm

```
docker swarm init
```

❖ Puis y attacher un réseau

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

❖ Dans PPCS-CM1-2025 faire //mpi4l-debian-bullseye

```
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)

```
docker exec -it ppcs-cm1-2025-mpihead-1 bash
```



INTERACTIF

```
sudo curl --unix-socket /var/run/docker.sock  
http://localhost/containers/json | jq -r  
'map(.NetworkSettings[]."yml_mpinet"."IPAMConfig".  
"IPv4Address") []'
```

→ donne la liste des ip des nœuds

```
mpiuser@475c5311290d:~$ sudo curl --unix-socket /var/run/docker.sock http://localhost/containers/json | jq -r 'map(.NetworkSettings[]."yml_mpinet"."IPAMConfig"."IPv4Address") []'
```

% Total	% Received	% Xferd	Average Speed	Time	Time	Time	Current
			Dload Upload	Total	Spent	Left	Speed
100 11783	0 11783	0 0	885k 0	-----	-----	-----	958k
10.0.1.24							
10.0.1.25							
10.0.1.23							

En rouge ce qui doit être adapté

```
mpirun --mca orte_base_help_aggregate 0 --mca
btl_tcp_if_include 10.0.1.0/24 -n 2 -host
10.0.1.24,10.0.1.23 python3
./mpi4py_benchmarks/all_tests.py
```

```
mpiuser@475c5311290d:~$ mpirun --mca orte_base_help_aggregate 0 --mca btl_tcp_if_include 10.0.1.0/24 -n 2 -host 10.0.1.24,10.0.1.23 python3 ./mpi4py_benchmarks/all_tests.py
Warning: Permanently added '10.0.1.24' (ECDSA) to the list of known hosts.
```

```
=====
Running 2 parallel MPI processes
# MPI Matrix action on a vector, 20 iterations of size 10000
# Duration [s]    Throughput [#s]
2.147            9.31
2.057            9.72
2.102            9.51
2.079            9.62
1.945            10.28
1.843            10.85
1.685            11.87
1.821            10.98
1.845            10.84
1.698            11.78
```

❖ Description du paramétrage openmpi

```
mpi_info --level 9 --param rmaps base
```

```
mpiuser@475c5311290d:~$ mpi_info --level 9 --param rmaps base
MCA rmaps base: -----
MCA rmaps base: parameter "rmaps_base_verbose" (current value:
                  "error", data source: default, level: 8 dev/detail,
                  type: int)
                  Verbosity level for the rmaps framework (default:
                  0)
                  Valid values: -1:"none", 0:"error", 10:"component",
                  20:"warn", 40:"info", 60:"trace", 80:"debug",
                  100:"max", 0 - 100
```

❖ On souhaite « pratiquer » MPI → utiliser mpicc + Impi

`hello_world.cc`, `mpi_any_source.cc`

2 codes C à récupérer (git clone) depuis

<https://github.com/jmbatto/master-mpi>

❖ On souhaite « pratiquer » MPI → utiliser mpicc + Impi

hello_world_mpi.cc

```
mpiuser@475c5311290d:~/master-mpi$ mpicc hello_world_mpi.cc -o hello_world -lm
mpiuser@475c5311290d:~/master-mpi$ ./hello_world
Hello world from processor 475c5311290d, rank 0 out of 1 processors
mpiuser@475c5311290d:~/master-mpi$ mpirun --mca orte_base_help_aggregate 0 --mca btl_tcp_if_include 10.0.1.0/24 -n 2 -host 10.0.1.24,10.0.1.23 hello_world
-----
mpirun was unable to find the specified executable file, and therefore
did not launch the job.  This error was first reported for process
rank 1; it may have occurred for other processes as well.

NOTE: A common cause for this error is misspelling a mpirun command
line parameter option (remember that mpirun interprets the first
unrecognized command line token as the executable).

Node:      10.0.1.24
Executable: /home/mpiuser/master-mpi/hello_world
```

Il faut se placer dans l'espace partagé avant de lancer la commande

```
mpinode:
  image: jmbatto/m2chps-mpi4l:latest
  volumes:
    - /var/run/docker.sock:/var/run/docker.sock
    - usrlocalvarmpi-foo:/usr/local/var/mpishare
```

INTERACTIF

On souhaite « pratiquer » MPI → utiliser mpicc + Impi

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.
```

❖ 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>)

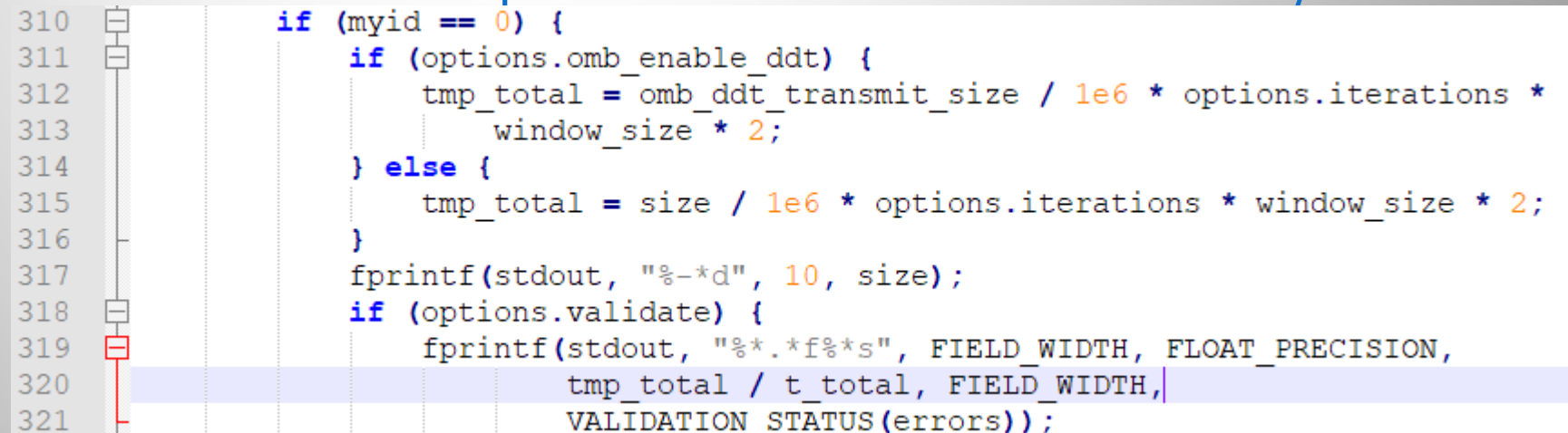
configure puis

/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));
```





❖ Connaitre la capacité de la VM Docker

INTERACTIF

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

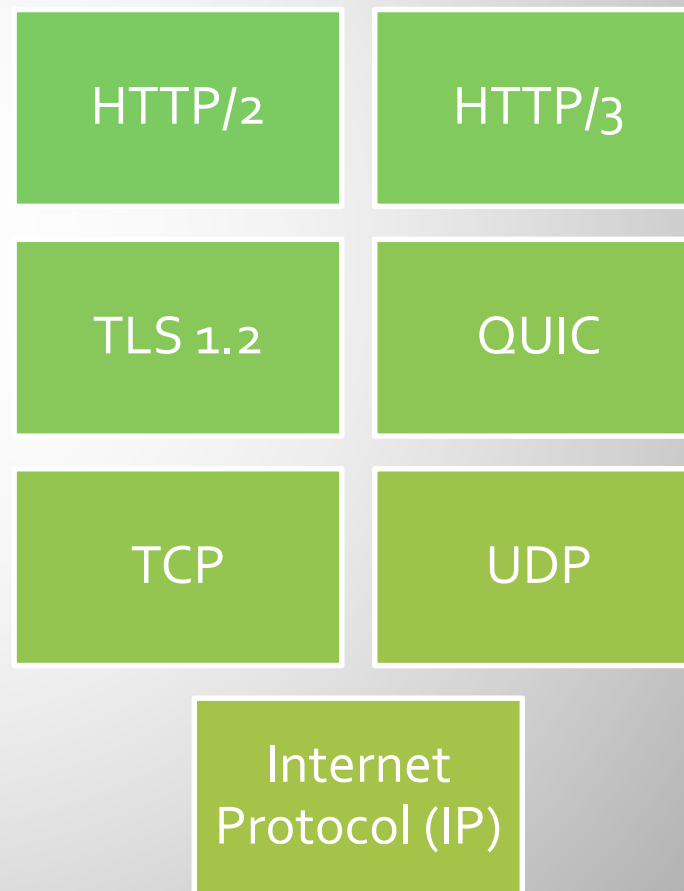
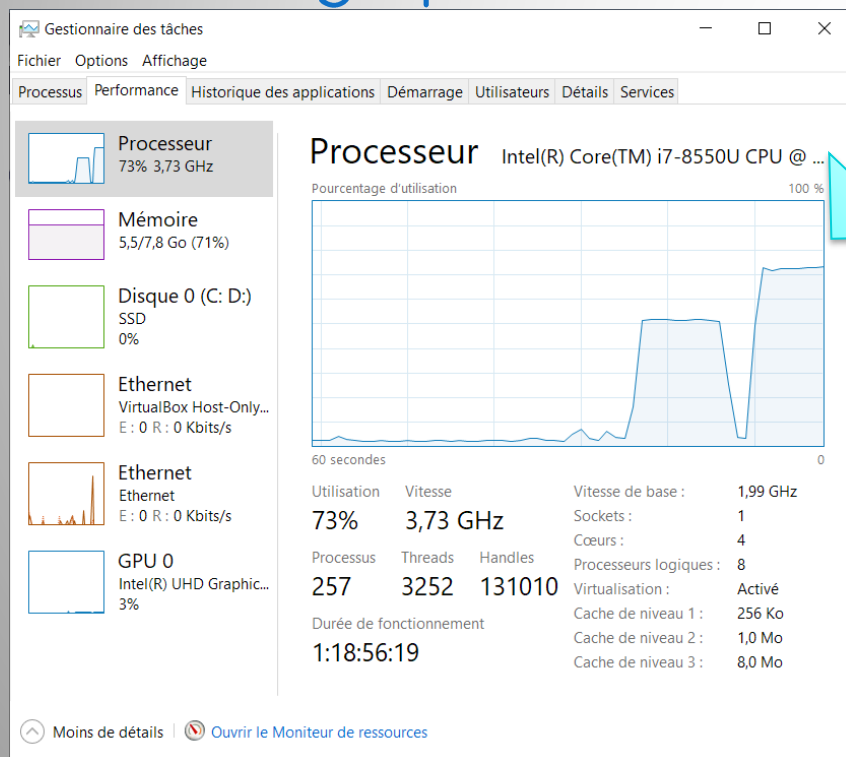
```
root@vps_eldarsoft:~# docker stats --no-stream --format "table {{.Name}}\t{{.Container}}\t{{.MemUsage}}"  
NAME                CONTAINER          MEM USAGE / LIMIT  
gogs                 428f4d97c4a1       352.8MiB / 1.899GiB  
traefik              7cce703b64a8       22.41MiB / 1.899GiB  
root@vps_eldarsoft:~#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

NAME	CONTAINER	MEM USAGE / LIMIT
telegrafmonitor-auto	f19b09e8cd2b	11.43MiB / 14.33GiB
pure-ftp	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



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 → TD de prise en main
- ❖ <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 –
 - ❖ 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) :
 - ❖ Vos PROMPTS et le nom/version du LLM
 - ❖ **pour le 7 décembre 2025 minuit**

❖ `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`



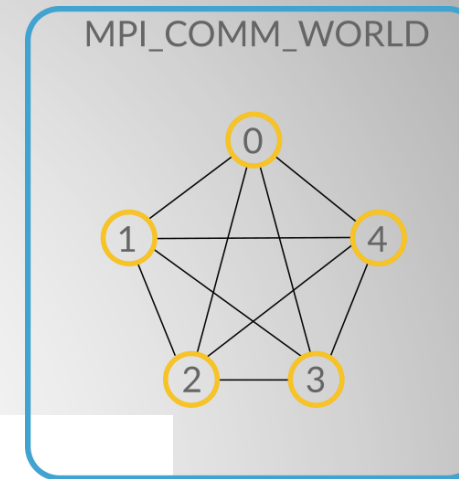
❖ API synchrone&asynchrones avec processus numérotés (=connus)

- ❖ MPI_Init()
- ❖ MPI_Abort()
- ❖ MPI_Get_processor_name()
- ❖ MPI_Comm_size()
- ❖ MPI_Comm_rank()
- ❖ MPI_Send(), MPI_Ssend()
- ❖ MPI_Recv()
- ❖ MPI_Finalize()
- ❖ MPI_Bcast() / MPI_Scatter()
- ❖ MPI_Reduce()
- ❖ MPI_Allreduce()

❖ MPI_Init() /MPI_Finalize()

❖ MPI_Abort()

```
#include <mpi.h>
int main(int argc, char *argv[])
{
    MPI_Init(NULL, NULL);
    MPI_Abort(MPI_COMM_WORLD, 112); // in Europe 112, for emergency
    /* No further code will execute */
    MPI_Finalize();
    return 0;
}
```



❖ MPI_Get_processor_name()

❖ → résultat de gethostname, uname, ou sysinfo

- ❖ `MPI_Comm_size()` → effectif des processus
- ❖ `MPI_Comm_rank()` → rang du processus

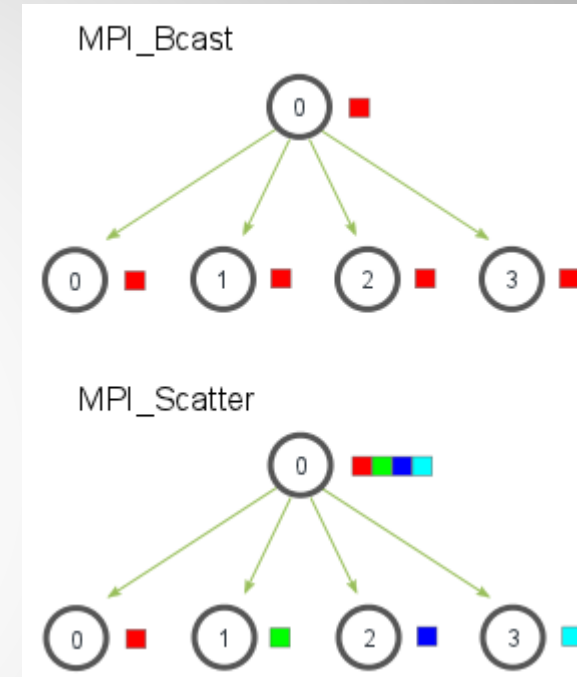
- ❖ `MPI_Send()` → communication point à point
- ❖ `MPI_Ssend()` → communication point à point, **synchrone**
- ❖ `MPI_Recv()` → réception point à point, responsable du buffer data

❖ MPI_Bcast() / MPI_Scatter()

❖ Instructions avec fonctions

❖ MPI_Reduce() → root

❖ MPI_Allreduce() → partage



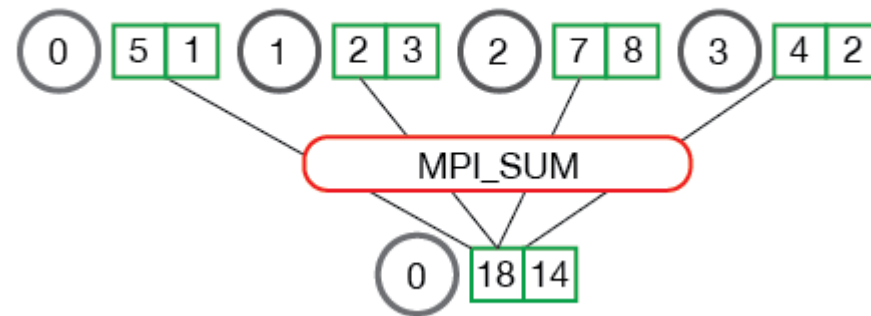
❖ MPI_Bcast() / MPI_Scatter()

❖ Instructions avec fonctions

❖ MPI_Reduce() → root

❖ MPI_Allreduce() → partage

MPI_Reduce



MPI_Allreduce

