TER

Simulation d'application dynamiques pour plateformes de calculs hautes performances

Steven QUINITO MASNADA

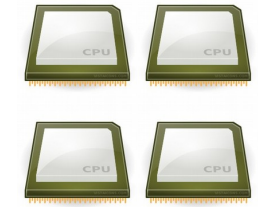
Équipe MESCAL/LIG
Sous la direction d'A. Legrand

Grenoble, 8 Juin 2015

Topic

Architecture et standards HPC

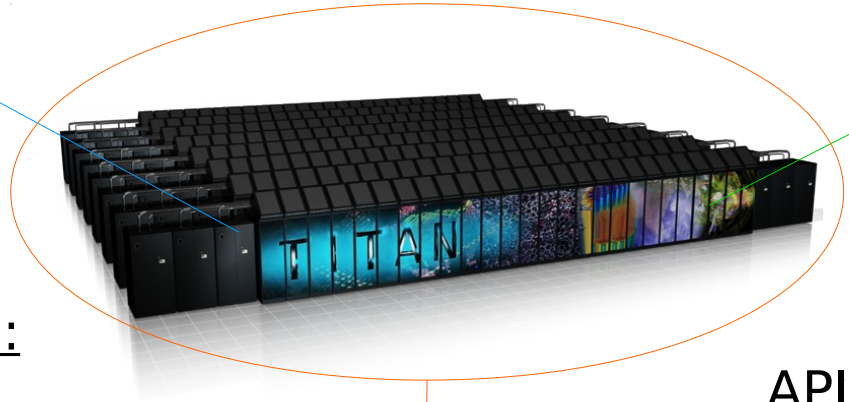
Multicœurs



OpenMP

API multithread :

- De plus haut niveau que PThread
- Permet d'exploiter les architectures multicœurs
- Facilite le découpage des traitements



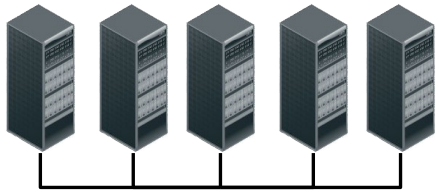
Hybride

NVIDIA CUDA

API de caculs sur GPU :

- Exécution de noyaux de calcul
- Transferts entre la RAM des GPUs et des CPUs
- Spécifier calculs GPU
- Synchroniser CPUs - GPUs

Clusters



MPI

API de communication :

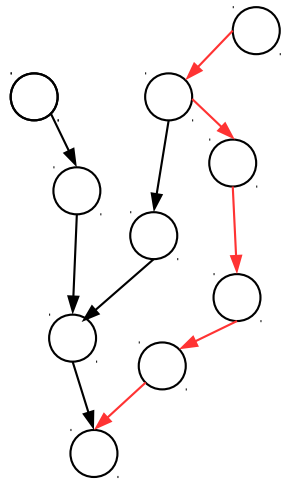
- De plus haut niveau que les sockets
- Mécanismes de communication supplémentaires (exemple broadcast)

Limites des approches classiques

- Utiliser plusieurs paradigmes $\rightsquigarrow$ programmation complexe
- Exemple pour exploiter efficacement un GPU sur un seul noeud:
  - transférer données du CPU au GPU,
  - lancer le calcul sur le GPU
  - gérer synchronisation pendant attente résultat
  - occuper CPU
  - récupérer résultat.
- Et avec plusieurs noeuds/cores/GPUs ?
  - Statique, système réglé comme une horloge $\rightsquigarrow$ pas scalable.
  - Solution: Dynamique mais très difficile avec APIs classiques.

Nouvelle approche: paradigme de tâches

- Nouvelle abstraction: les tâches
  - Plus besoin de se soucier de la ressource sur laquelle le traitement est effectué
  - Exprimer calcul en terme graphe de tâches $\rightsquigarrow$ plus de souplesse et une meilleure portabilité



- Librairie StarPU:
  - Un runtime développé au LaBRI (RUNTIME/STORM).
  - Graphe de dépendances $\rightsquigarrow$ ordonnancemment dynamique et opportuniste
  - Première version pour système hybrique et récémment StarPU MPI
- Problèmatique : Performances difficiles à évaluer car exécution du flot de contrôle non déterministe
  - Configuration runtime, heuristique d'ordonnancement
  - Configuration application, découpage en tâches

Évaluation des performances en HPC: grandes approches

Test sur systèmes réels

- Exécution réelle sur la plateforme cible $\rightsquigarrow$ coûteux, difficulté d'accès
- Exécution non déterministe nécessite de réaliser beaucoup d'expériences $\rightsquigarrow$ extrapolations difficiles

Simulation : Généralités

- Utilisation de modèles pour prédire comportements
- Permet s'affranchir de la plateforme $\rightsquigarrow$ peu coûteux
- Extrapolation simplifiée / simulation très rapide

Simulation: 2 possibilités

Simulation par rejeu de trace

Pas adapté ici car flot de contrôle non déterministe

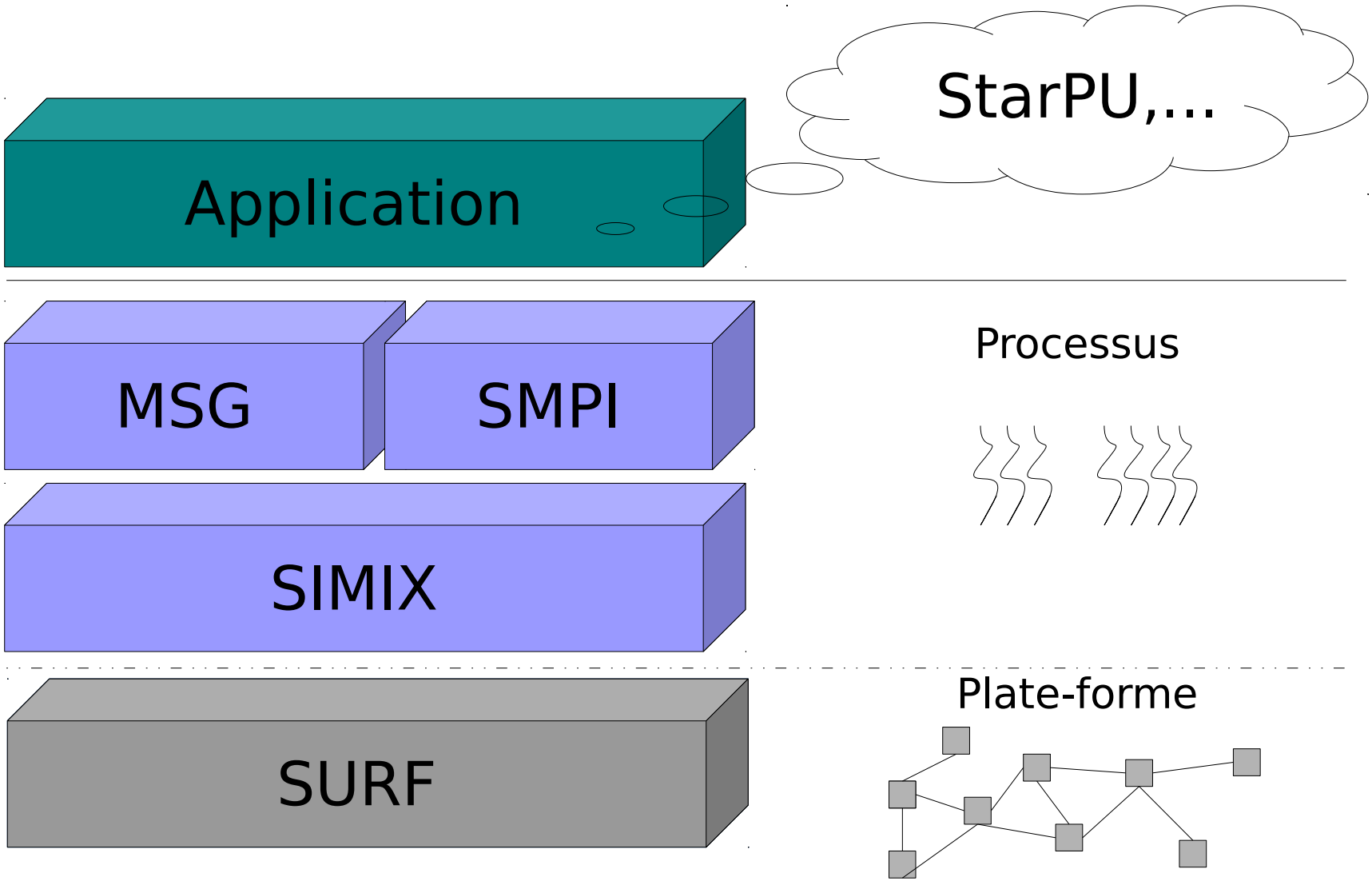
Hybride simulation / émulation

- Simulation réaliste $\rightsquigarrow$ exécuter vrai code
- Simuler plate-forme et OS.
- Émuler de l'application.

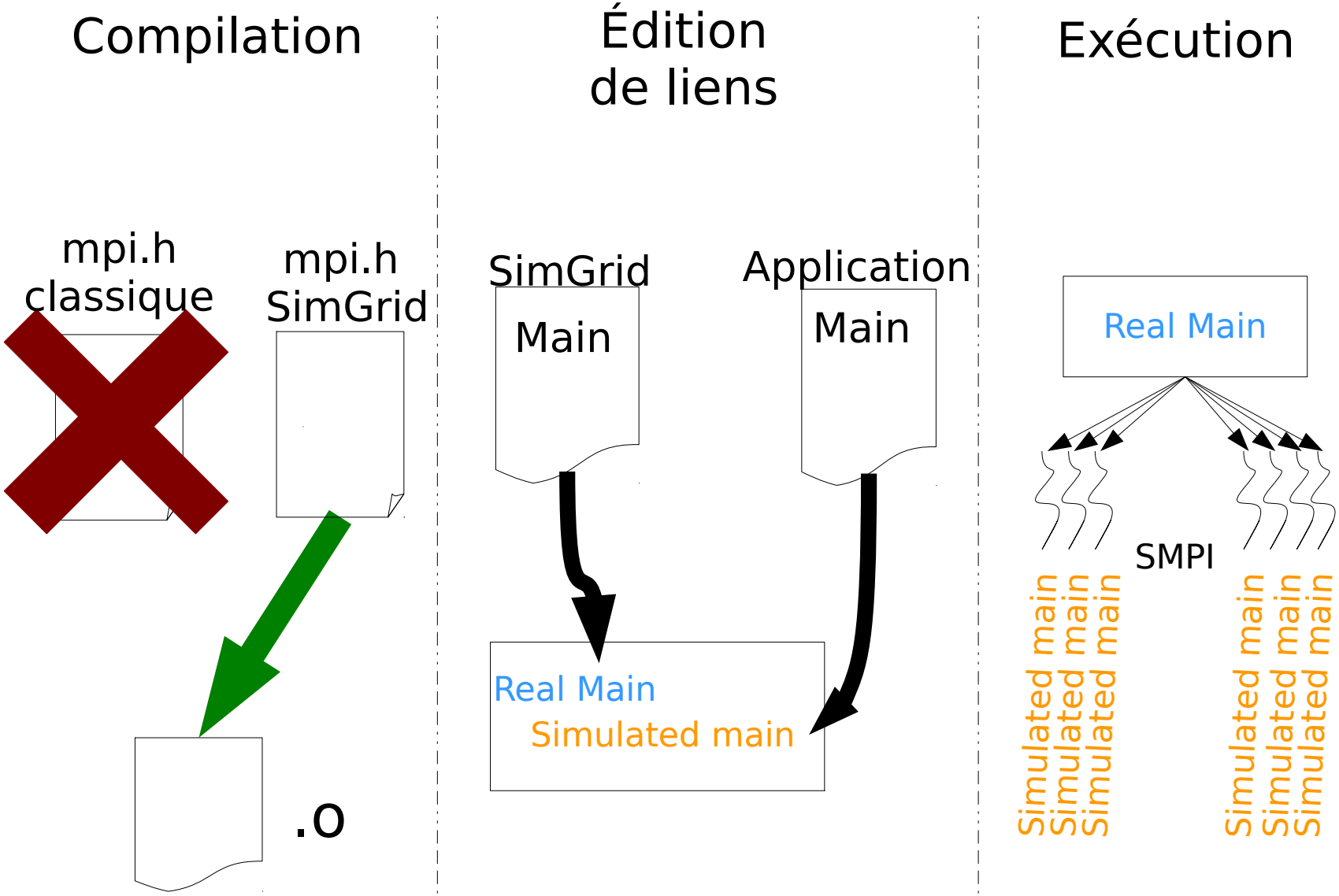
SimGrid simulateur de systèmes distribués, grilles de calculs, systèmes peer to peer et cloud.
StarPU SimGrid, approche simulation / émulation mais avec un seul noeud.

Topic

SimGrid: Généralités

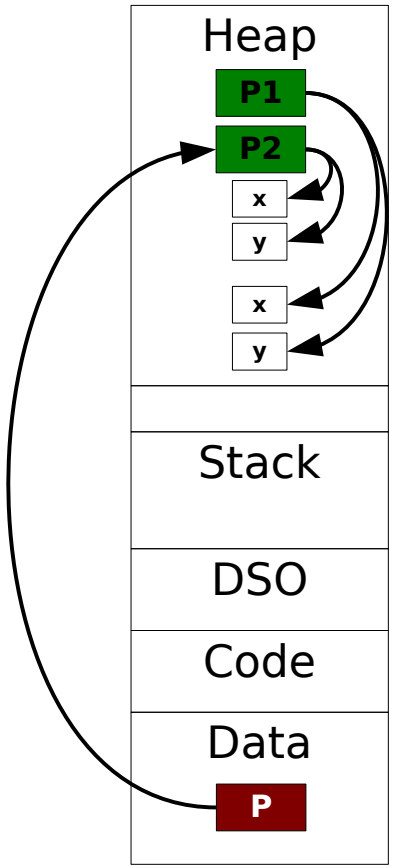


Construction de l'application MPI simulée



Privatisation du segment data

- Dans SimGrid les processus sont modélisés par des threads $\rightsquigarrow$ espace adressage partagé.
- MPI environnment à mémoire distribuée.
- Mécanisme de privatisation: ségment virtuel (mmap)



StarPU MSG

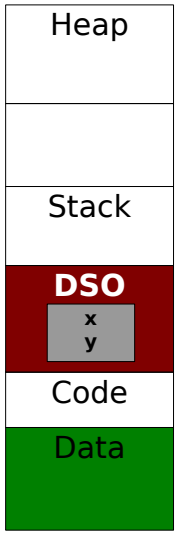
- Première version StarPU $\rightsquigarrow$ hybride
- Basé sur MSG car modèle de performance plus proche (communications, environnement mémoire partagé), CPUs GPUs.
- Émulation: le vrai code de l'ordonnanceur de StarPU et de l'application sont exécutés
- Simulation: calculs, allocations mémoire des tâches, transfert CUDA.

StarPU SMPI: Difficultés de mise en oeuvre

- Besoin de 2 modèles de programmation différents à la fois:
  - StarPU intra noeuds: mémoire partagée $\rightsquigarrow$ MSG, partage
  - StarPU inter noeuds: mémoire distribuée $\rightsquigarrow$ SMPI, privatisation
- Besoin de 2 modèles de performance différents à la fois:
  - StarPU intra noeuds: CPU-GPU $\rightsquigarrow$ MSG ad hoc
  - StarPU inter noeuds: réseau $\rightsquigarrow$ SMPI

  Besoin de modifications un peu complexes dans SURF $\rightsquigarrow$ pas dans le cadre de ce stage.
- MSG et SMPI normalement pas utilisés ensemble $\rightsquigarrow$ initialiser correctement les 2.

- Problème des bibliothèques dynamiques.



Topic

Techniques et étude de l'existant

Prise en main

- Dépôt git submobules:
  - StarPU SMPI:
    - SimGrid
    - StarPU
- Suivi:
  - Cahier de laboratoire org mode github.
- Compréhension:
  - Documentation.
  - SimgGrid = 106 350 lignes de codes.
  - StarPU = 172 251 lignes de codes.
  - "Code mining" et vérifications: GDB, Valgrind.

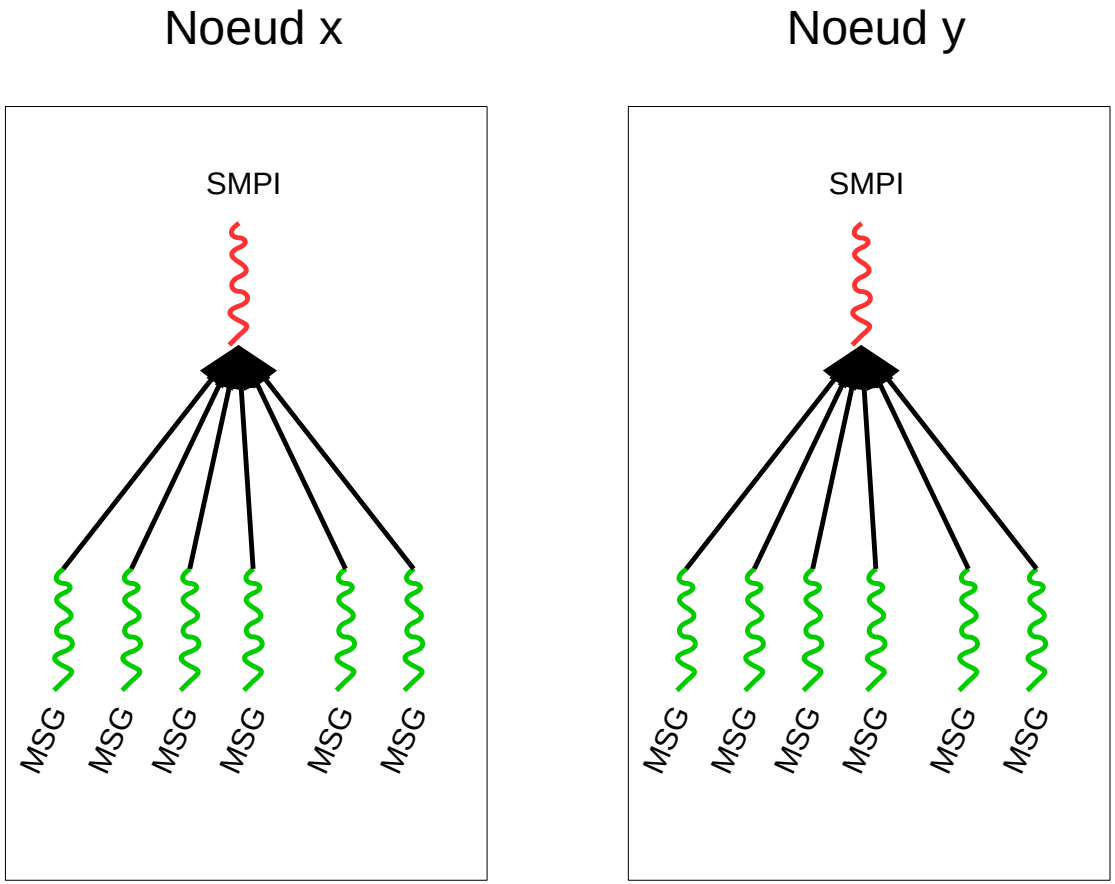
Validation

- Test simple: Modèle simplifié de StarPU MPI $\rightsquigarrow$ isoler problèmes.
- Test StarPU: MPI, Cholesky $\rightsquigarrow$ valider modifications

Topic

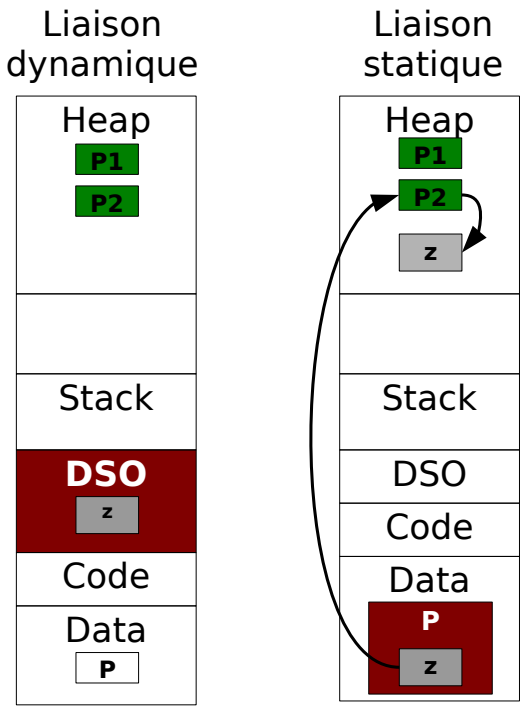
Modification de SimGrid

- Initialisation MSG + SMPI
- Gestion du segment data: les processus MSG créés par un processus SMPI "héritent" du segment de leur père.



Librairie dynamiques et modifications StarPU

- Librairies dynamiques:
  - Utilisation librairies statiques.
- Modification StarPU:
  - Initialisation, car privatisation tardive.



Topic

Pour conclure

Bilan

- StarPU + SimGrid modifié pour simuler StarPU MPI
- Difficulté: apporter modifications minimes dans un code non trivial. Environ 20 lignes sur un total de plus de 270 000

Prochaine étape

- Simulations et mesures avec solveur d'algèbre linéaire
- Vérifications système réel: Grid5000
- stabiliser le prototype (intégrer les modifications aux dépots principaux de StarPU et de SimGrid)

Fin

Merci pour votre attention.