Retour sur les Jdev 2015
Un peu de données, un peu de calcul

T3 Atelier 6
Julia pour vos calculs scientifiques intensifs

T3 GT 5
Différentes méthodes d'**optimisation** d'un code **Python**

T3 Atelier 8
Python, apprentissage statistique et analyse de données pour la modélisation prédictive

T8 GT 3
Validation et vérification de son logiciel scientifique : analyse, développement, exploitation

T8 Atelier 2
Programmation **GPU** moderne. Cuda est-il encore compliqué en 2015 ?

T8 Atelier 5
Structuration des données avec **HDF5**

Contacts : Y. Rogez(IPAG) & L. Tavard (LGGE)

julia , pour vos calculs scientifiques intensif

→ Implémentation :

- Langage de programmation récent (2013) et toujours en développement destiné au calcul scientifique (version 1.0)
- Open source (licence MIT)
- Mode interactif et compilé (à la volée)

→ Alternative/complément aux langages R/Matlab /Python

→ Un mode parallèle (par défaut se lance sur 1CPU)

→ Doc dispo sur le site des JDEV :

pdf complet + TP :
https://github.com/mcanouil/Presentation/tree/master/JULIA_TP
Tuto : http://julialang.org/learning/
Tips : https://github.com/JuliaLang/julia/tree/master/contrib

- **Scikit (http://scikits.appspot.com/scikits)**

Un ensemble de librairies spécialisées basées sur SciPy

- **Learn (http://scikit-learn.org/stable/)**

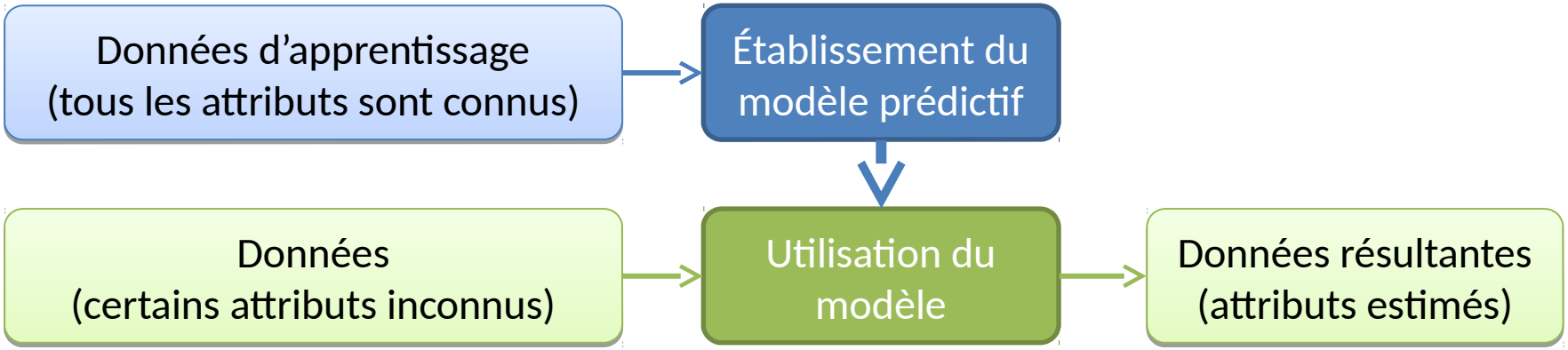
Pour le machine learning et data mining

- **Doc présente sur le site des JDEV :**

support du tutorial : https://github.com/ogrisel/parallel_ml_tutorial

Doc de scikit-learn : https://scikit-learn.org

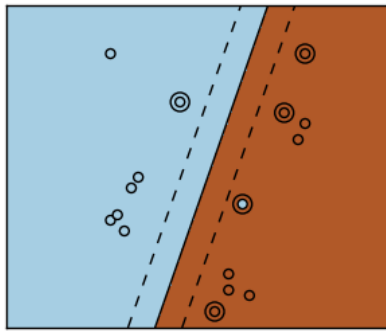
Tuto complémentaire écosystème Python : https://scipy-lectures.github.io/

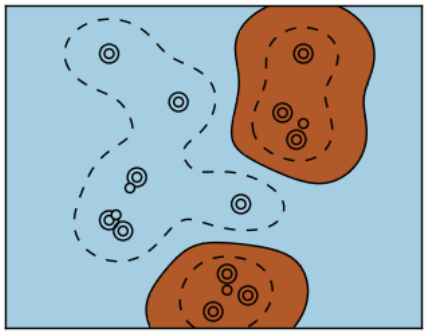


scikits learn
machine learning in Python

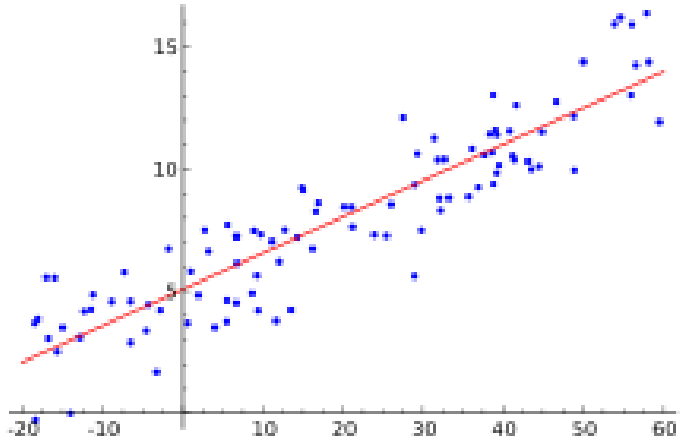
→ “Supervised learning”

Classification: classifier les nouvelles données dans des catégories, ou sous-ensembles (attribut inconnu = catégorie d’appartenance)





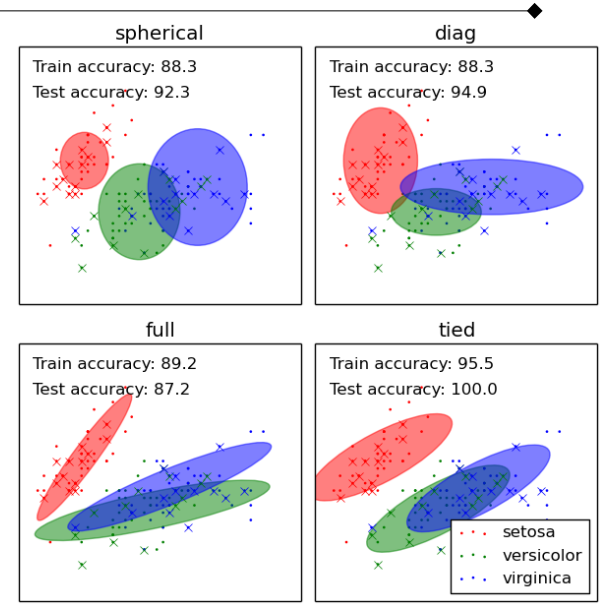
Régression: établissement de règles numériques pour estimer les valeurs des attributs manquants

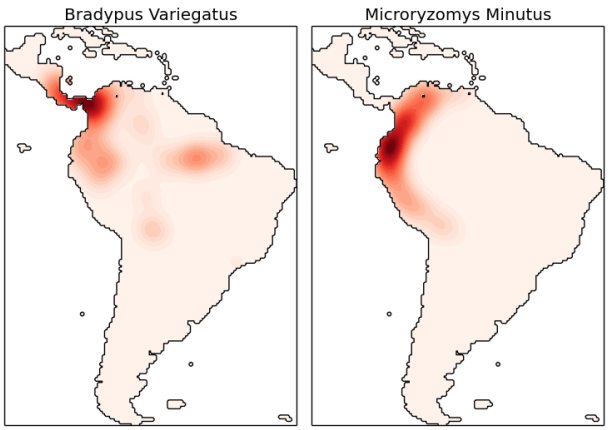


scikits learn
machine learning in Python

→ « Unsupervised learning »

Ici on a plus de données complètes. Ces outils permettent de générer des groupement (clustering) de données ou des visualisations statistiques (density estimation) en analysant les similitudes entre les attributs des données

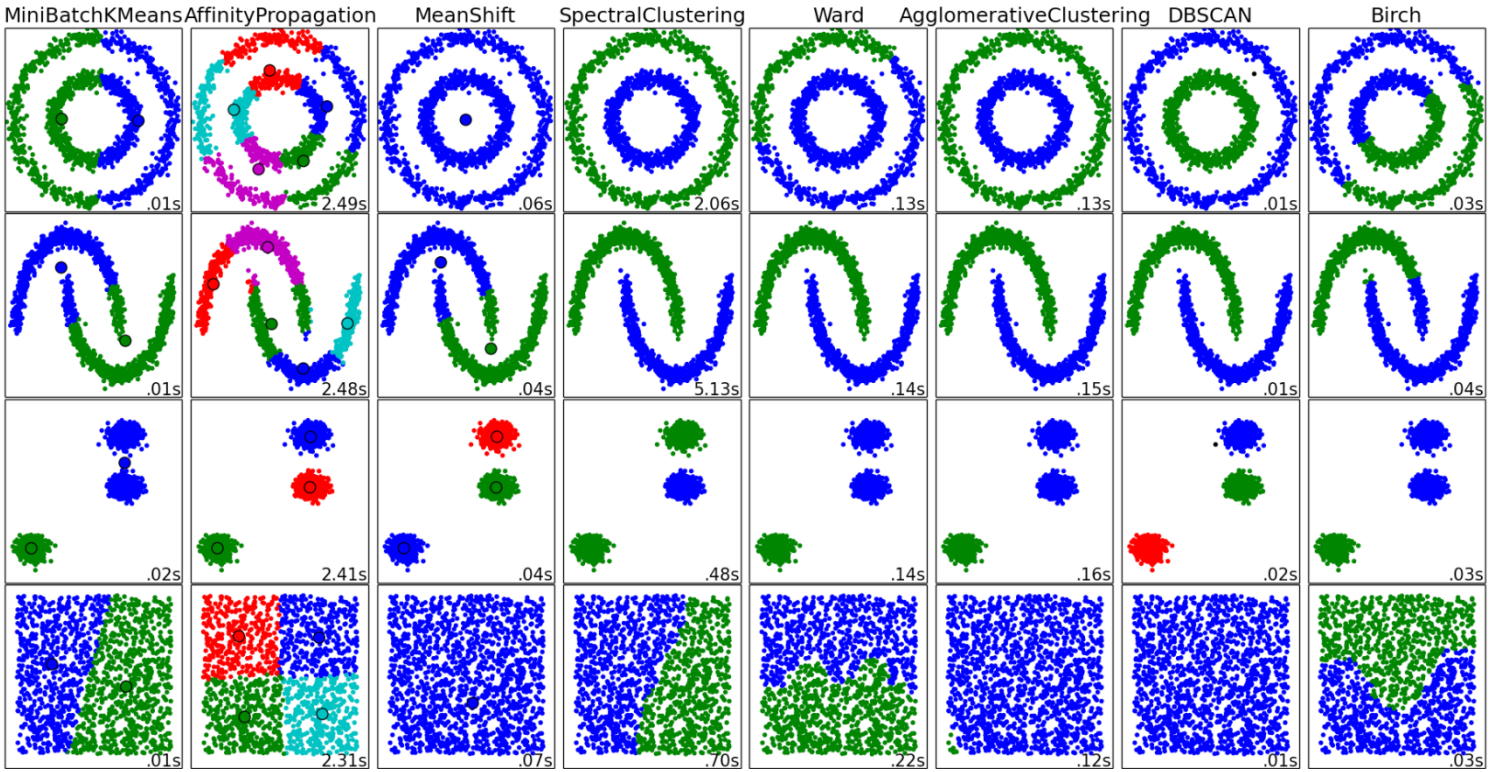




scikits learn
machine learning in Python

Beaucoup d'algorithmes disponibles

Choix par la documentation et évaluation par tests



Python: optimisations

- Utiliser la vectorisation des opérations avec numpy

- Compiler
  - - avec Cython : Mixer du C/C++ avec du code python
    - Ou avec Numba : Compiler du python avec les optimisations C/C++

- Paralléliser avec la lib multiprocessing (pools de tâches)

- Calculer sur GPU
    - PyCuda (inclure du code Cuda dans python)
    - Numba (peut compiler du code python vers du Cuda)
    - Remplacer BLAS par nvBLAS dans numpy (uniquement pour les opération matricielles) http://scelementary.com/2015/04/09/nvidia-nvblas-in-numpy.html

Retour sur les Jdev 2015
Un peu de données, un peu de calcul

T3 Atelier 8
Python, apprentissage statistique et analyse de données pour la modélisation prédictive

T3 GT 5
Différentes méthodes d'**optimisation** d'un code **Python**

T8 Atelier 6
Julia pour vos calculs scientifiques intensifs

T8 Atelier 2
Programmation **GPU** moderne. Cuda est-il encore compliqué en 2015 ?

T8 GT 3
Validation et vérification de son logiciel scientifique : analyse, développement, exploitation

T8 Atelier 5
Structuration des données avec **HDF5**

Calcul GPU : Intro

- Graphics Processing Units : circuit intégré présent sur les cartes graphiques qui assurent les fonctions de calcul de l'affichage
- Maintenant utilisé pour faire du calcul hautement parallèle (GPGPU : General Purpose GPU)
- Plusieurs constructeurs Nvidia, AMD, Intel

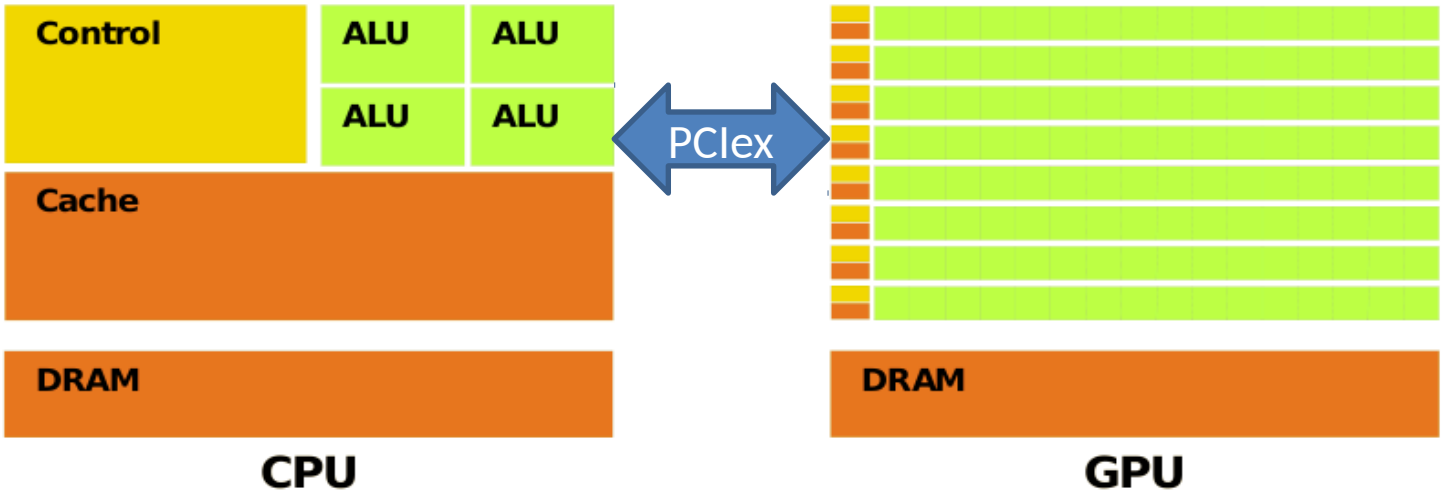
GPU : Architecture

Sur un GPU :
Plusieurs multiprocesseurs (MP) avec chacun:

- Un cache
- Une unite de controle
- Plusieurs coeurs

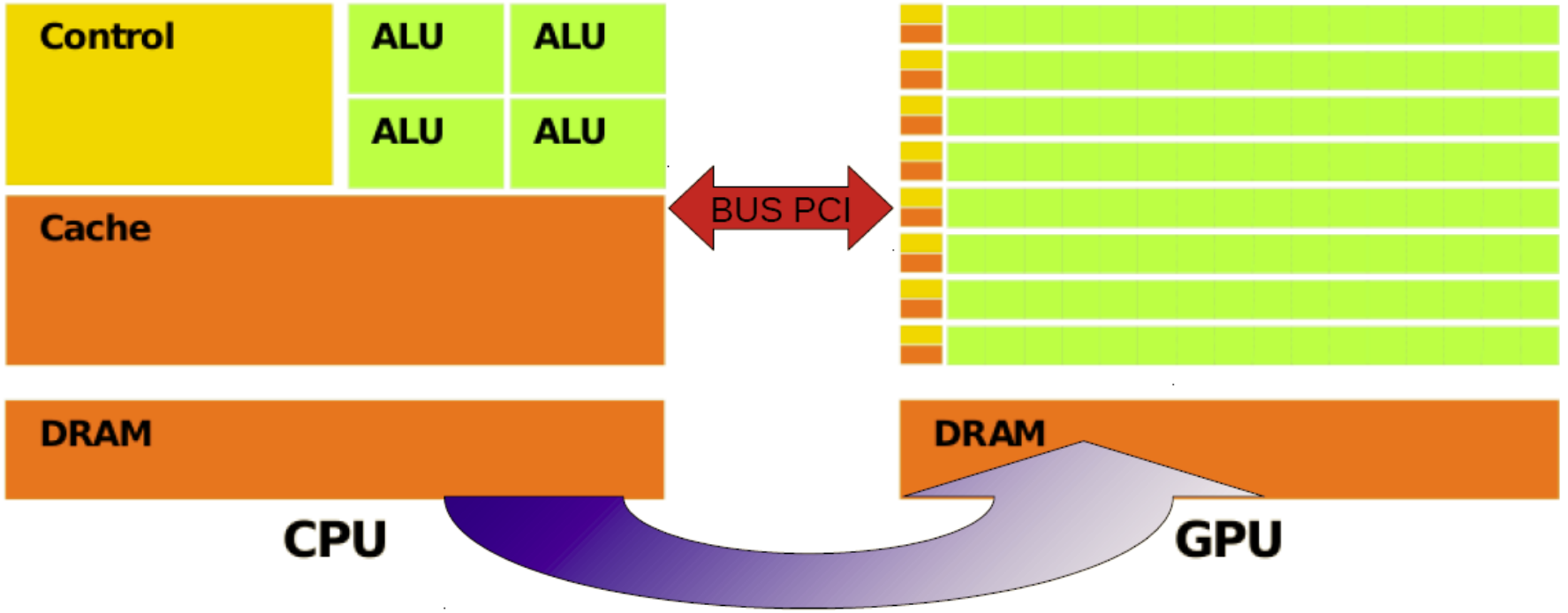
Memoire vive partagée entre tous les MP

Exemple Nvidia Tesla K20 (type dispo sur CIMENT/Froggy: 9 x 2 x K20):
4,8 Go de memoire vive partagée
13 MP, 192 coeurs/MP (2496 coeurs au total)



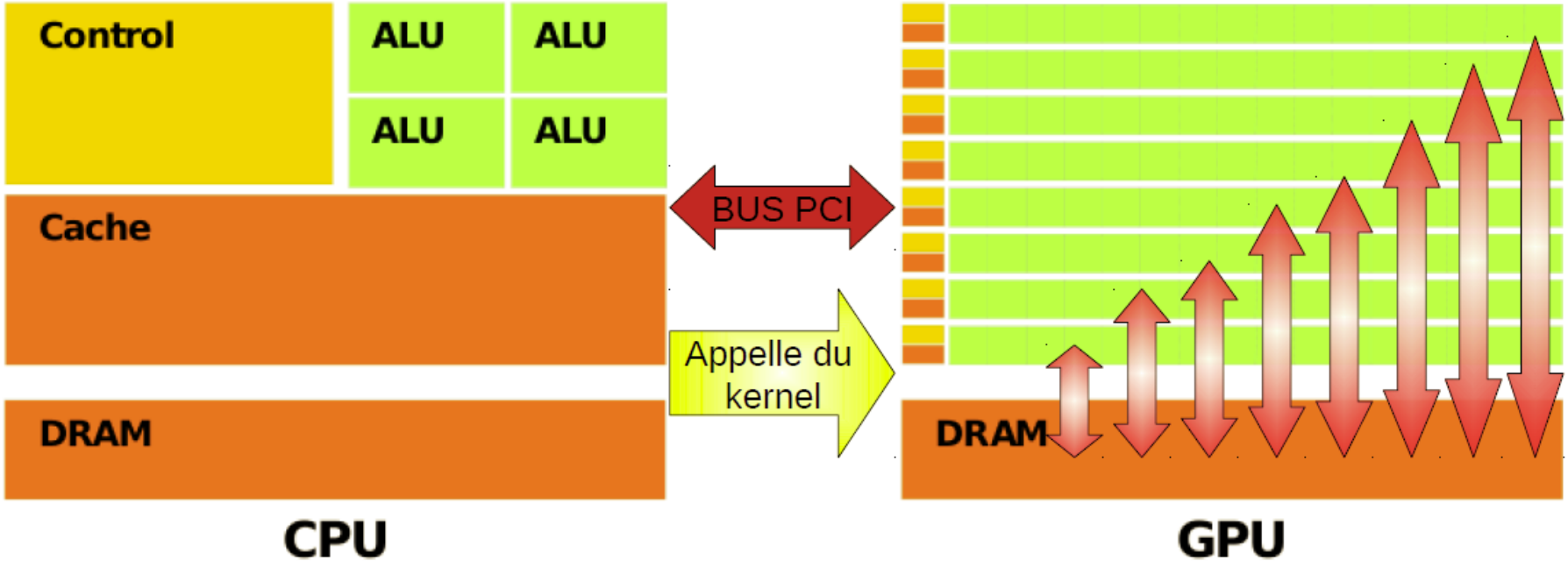
GPU: Fonctionnement global

1°) Copie des données d'entrée



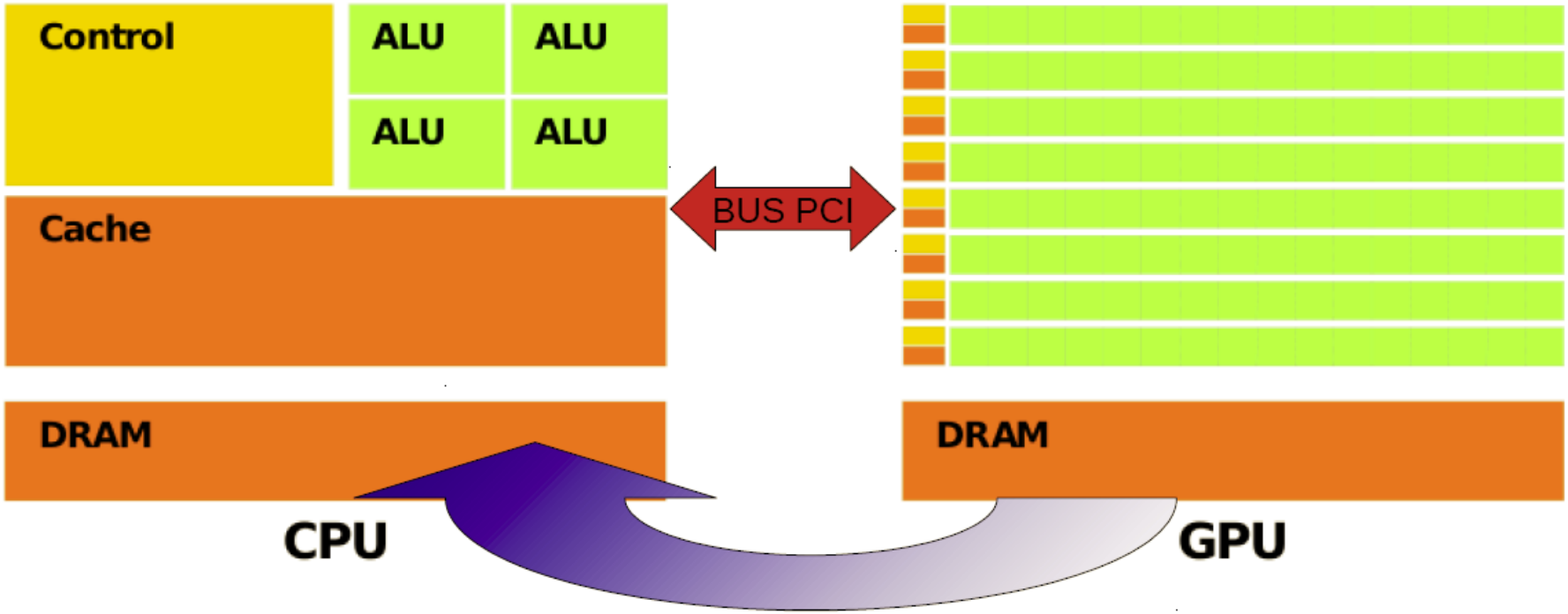
GPU: Fonctionnement global

2°) Exécution des kernels parallèles



GPU: Fonctionnement global

3°) Copie des données résultats



GPU: Comment programmer ?

- Coder le programme soi-même:
  - Cuda C/C++ (Nvidia)
  - OpenCL (générique?)
  - Directives OpenACC, OpenHMPP, OpenMP 4.0 (à base de #pragma en C)

- Utiliser des bibliothèques Cuda (Nvidia):
  - BLAS (Algèbres linéaire)
  - CuFFT (Transformée de Fourrier rapide)
  - …
- Ou même en python
  - PyCuda
  - nvBLAS

GPU: Comment programmer ?

- Exemple minimaliste

Exécuté sur le CPU

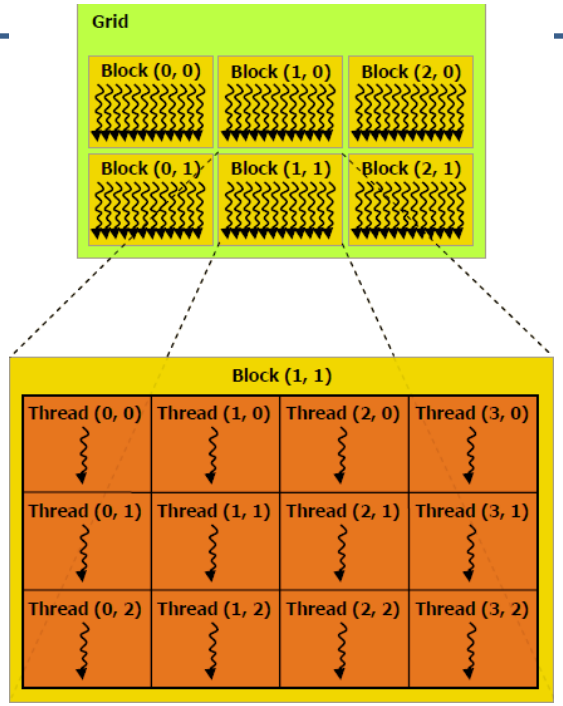
```
int main (void)
{
        type var ;                      // Variable sur l'host
        type d_var ;                    // Variable sur le device
        type N ;                        // Nombre d'éléments
        int size = N*sizeof( type ) ;   // Taille
        ...
        // Allocation de mémoire sur l'host
        var = (type*)malloc( size );
        //Allocation de mémoire sur le device
        cudaMalloc( (void**)&d_var, size );

        // Lancement de kernel sur le device, d_var en paramètre
        kernel<<<1,1>>>(d_var) ;
        // Copie de d_var vers l'host
        cudaMemcpy(var, d_var, size, cudaMemcpyDeviceToHost);
        ...
        // Libération de la mémoire de l'host
        free(var) ;
        // Libération de la mémoire du device
        cudaFree(d_var) ;

        return 0;
}
```

Exécuté sur le GPU

```
__global__ void kernel(type *var1)
{
        *var1 = ...
}
```



GPU: Debug et profiling

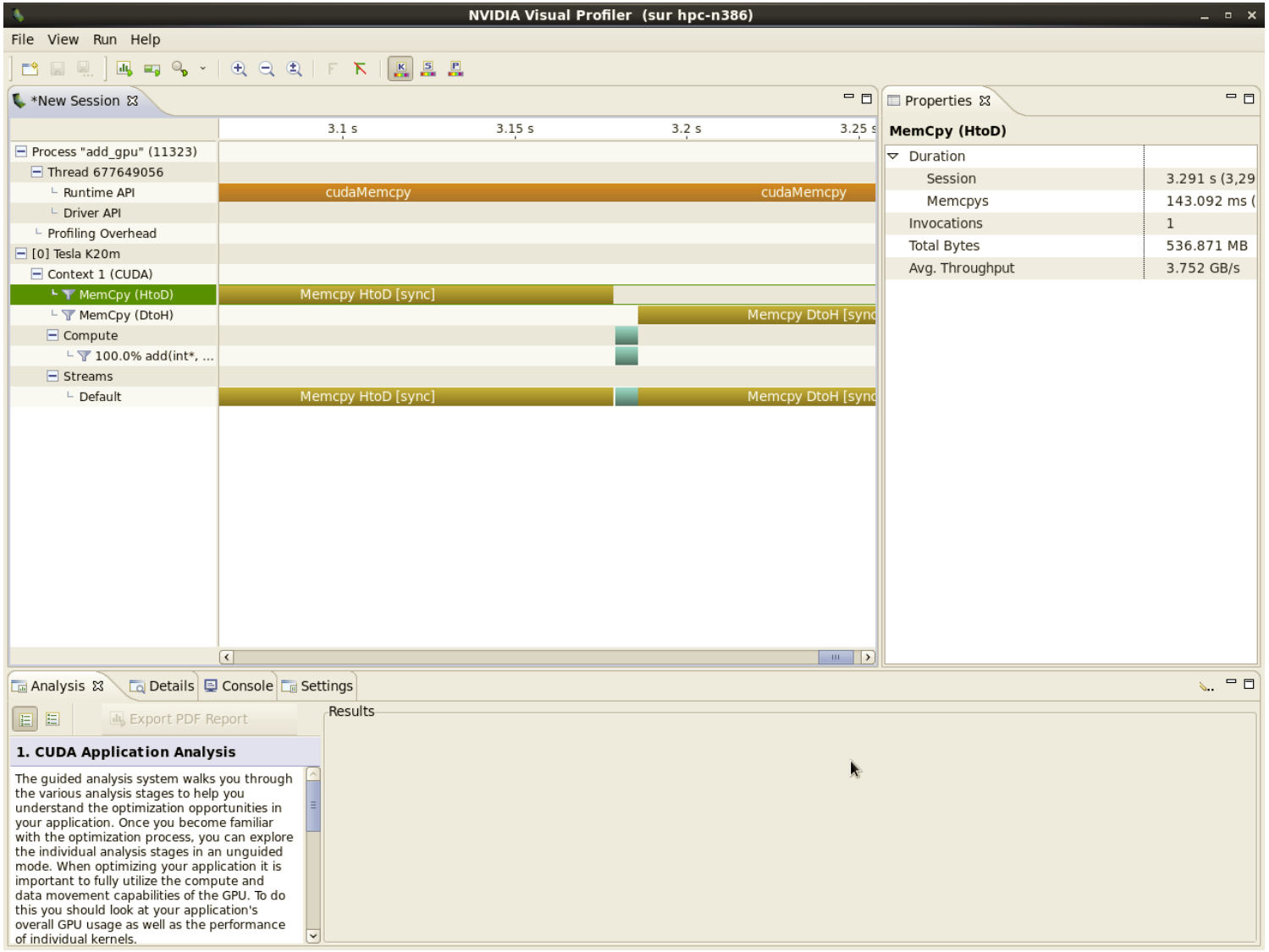
- On compile avec nvcc (utilise le compilateur C/C++ par défaut pour le code CPU)

- On peut inclure des infos de débuggage et remonter des timings automatiques pour le profiling
- Possibilité d'ajouter des traces qui seront visibles dans NVVP, surtout pour profiler aussi l'exécution de code CPU

- Outil graphique nvvp
- Outil de monitoring nvidia-smi

- Ces outils sont dispo sur CIMENT/Froggy/nœuds kepler

GPU: Debug et profiling



GPU: Dans quels cas c'est adapté?

- **Relativement peu de RAM** (< 6 Go typiquement) donc sur des sous-ensemble restreints, il faut pouvoir découper le problèmes si possible de manière indépendante

- **Les copies CPU <-> GPU sont couteuses** donc:
  - Il faut pouvoir les limiter en favorisant 1 grosse copie
  - Ou les paralléliser avec les calculs

- Le mieux est d'avoir **bcp de calculs simples** sur « pas trop » de données

- Pour obtenir la meilleure parallélisation des threads, ils faut
  - un kernel avec peu de branchements (if/else) pour que tous fassent les mêmes opérations simultanément
  - Des données organisées de manière coalescente

Retour sur les Jdev 2015
Un peu de données, un peu de calcul

T3 Atelier 8
Python, apprentissage statistique et analyse de données pour la modélisation prédictive

T3 GT 5
Différentes méthodes d'**optimisation** d'un code **Python**

T8 Atelier 6
Julia pour vos calculs scientifiques intensifs

T8 Atelier 2
Programmation **GPU** moderne. Cuda est-il encore compliqué en 2015 ?

T8 GT 3
Validation et vérification de son logiciel scientifique : analyse, développement, exploitation

T8 Atelier 5
Structuration des données avec **HDF5**

Logiciel de calcul: vérification & validation

Faire la différence entre

→ Vérification: adéquation entre les spécifications et les fonctionnalités proposées

→ Validation: les résultats du calcul sont conformes à la physique (au autre science) simulée

Logiciel de calcul: Vérification

- Vérification
  - Tests unitaires et non- régression
  - Gestion des erreurs
  - Il existe pour ça des méthodes et outils maintenant largement connus et utilisés par les développeurs
  - Une bonne couverture de test a un coup non négligeable sur le développement du logiciel

Logiciel de calcul: Validation

- Validation

- Comparaison aux modèles analytiques

Souvent applicable uniquement pour des cas simple, donc on ne valide pas tout

- Comparaison à des méthodes différentes (logiciels existants par exemple)

Les interfaces (échanges de données et cohérence physique des résultats comparés) est souvent complexe

→ Problème: cela demande parfois plus d'énergie de développer les méthodes de validation que le logiciel de calcul lui-même !

Solution: … chacun trouve un compromis acceptable …

Retour sur les Jdev 2015
Un peu de données, un peu de calcul

T3 Atelier 8
Python, apprentissage statistique et analyse de données pour la modélisation prédictive

T3 GT 5
Différentes méthodes d'**optimisation** d'un code **Python**

T8 Atelier 6
Julia pour vos calculs scientifiques intensifs

T8 Atelier 2
Programmation **GPU** moderne. Cuda est-il encore compliqué en 2015 ?

T8 GT 3
Validation et vérification de son logiciel scientifique : analyse, développement, exploitation

T8 Atelier 5
Structuration des données avec **HDF5**

Atelier : Structuration des données avec HDF5

- HDF5 = Hierarchical Data Format 5
- Bibliothèque d'entrée-sortie parallèles utilisant MPI-I/O
  - → optimisation possible MPI-I/O
- Équivalent à READ/WRITE mais
  Portable (inter-plateformes)
- Fonctionnalités avancées
  Description de données (dim, commentaires)
  Compression à la volée
  Compréhensible par bcp d'outils de visu (VisIt, ParaView, …)
- Bindings en Python, C,C++ Fortran