Le couplage visualisation / simulation avec ParaView

Alexandre Ancel

ancel@math.unistra.fr

Centre de Modélisation et de Simulation de Strasbourg (Cemosis),
Institut de Recherche Mathématique Avancée (IRMA),
Université de Strasbourg

2 juillet 2015

Plan de la présentation

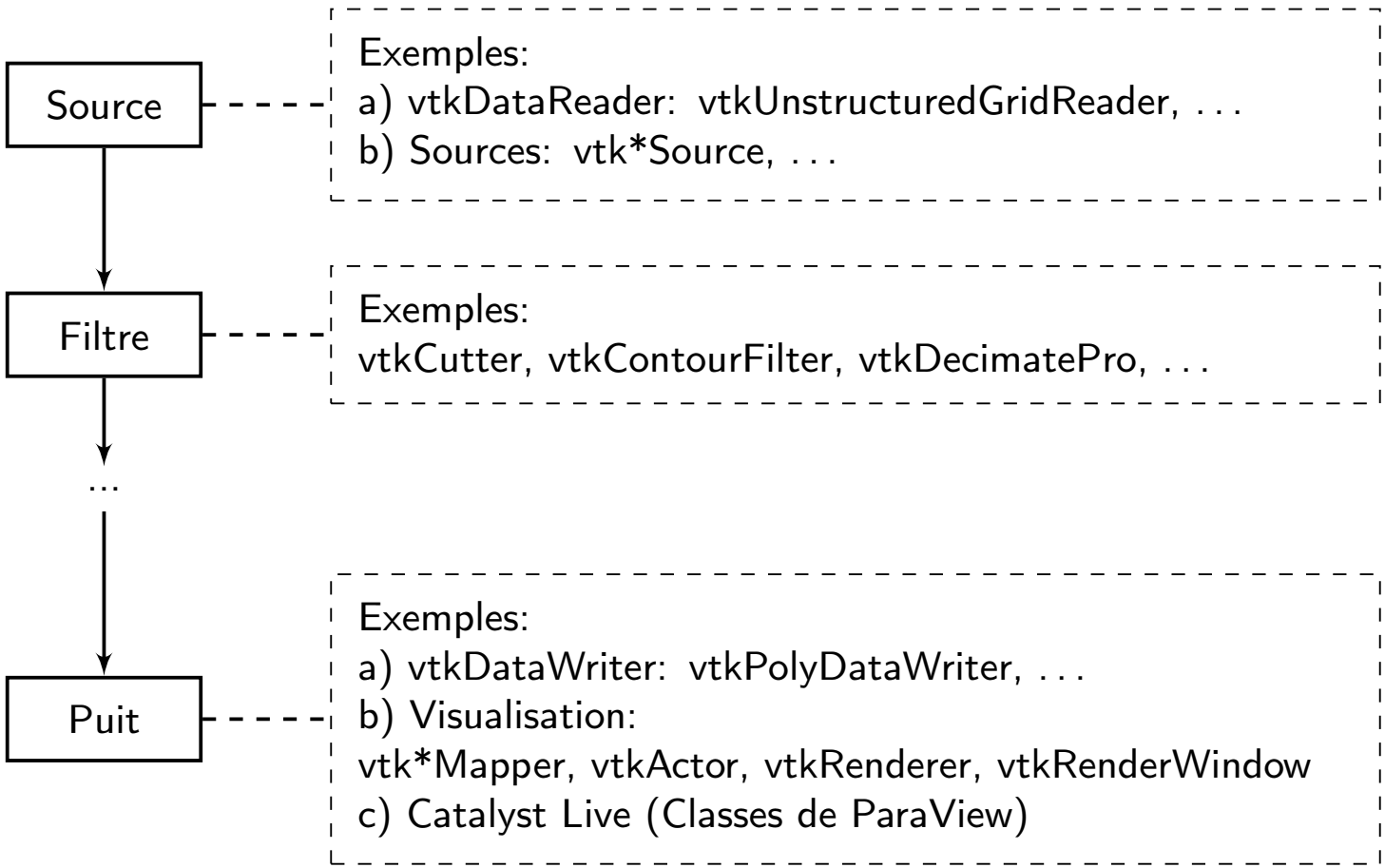
Plan

ParaView

- ParaView
  - Logiciel de visualisation scientifique open-source et gratuit
  - Depuis 2000, Kitware et plusieurs laboratoires nationaux des Etats Unis (Los Alamos, Sandia, Livermore)
  - Parallèle (MPI)
  - Offre un ensemble de logiciels pour la visualisation scientifique : ParaView, pvserver, pvdataserver/pvrenderserver, pvbatch, ...
  - Basé sur la librairie VTK (infographie, traitement d'image, visualisation)
- Quelques alternatives
  - EnSight : source code non accessible, gratuit avec limite ou payant
  - VisIt : open-source et gratuit (libsim pour l'in-situ)
  - Gmsh : open-source et gratuit

VTK

- VTK : open-source, orienté objet, pipeline de flot de données (DAG)

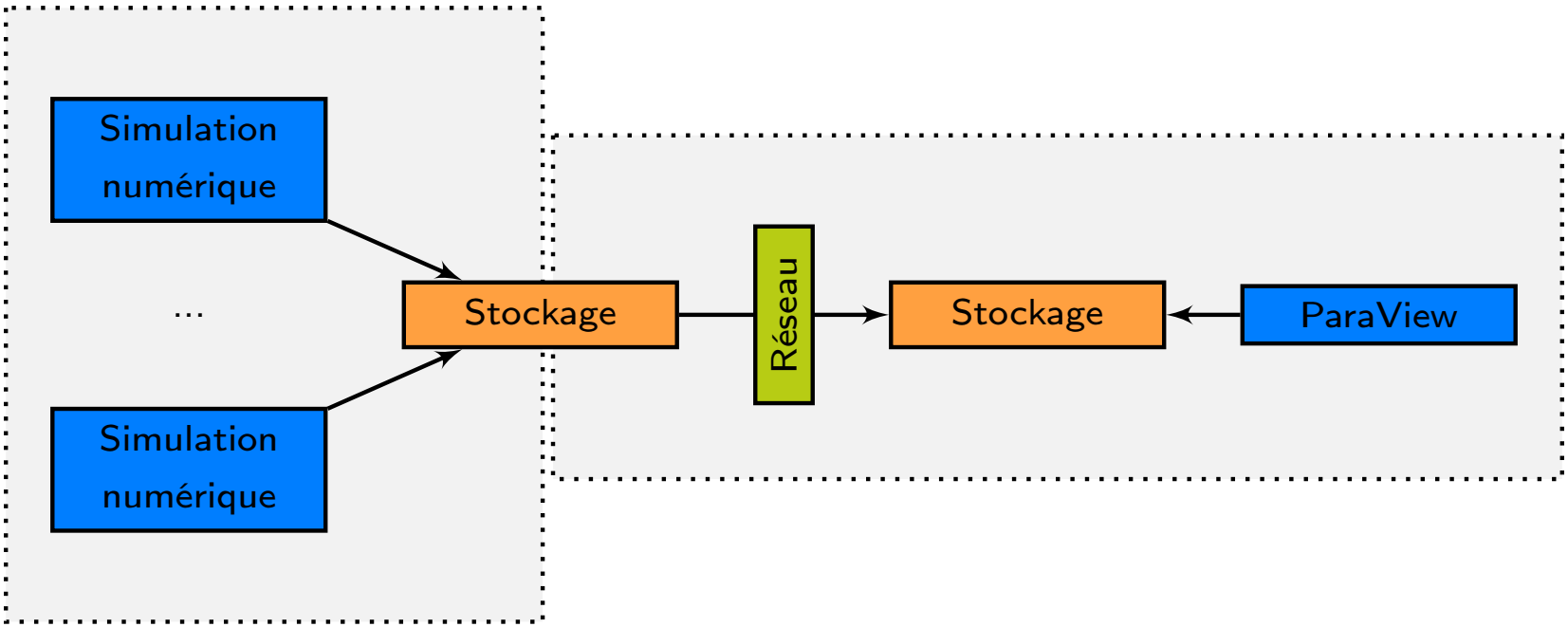


Plan

Comment visualiser ses données ?

Approches classiques de visualisation de données : Post-mortem

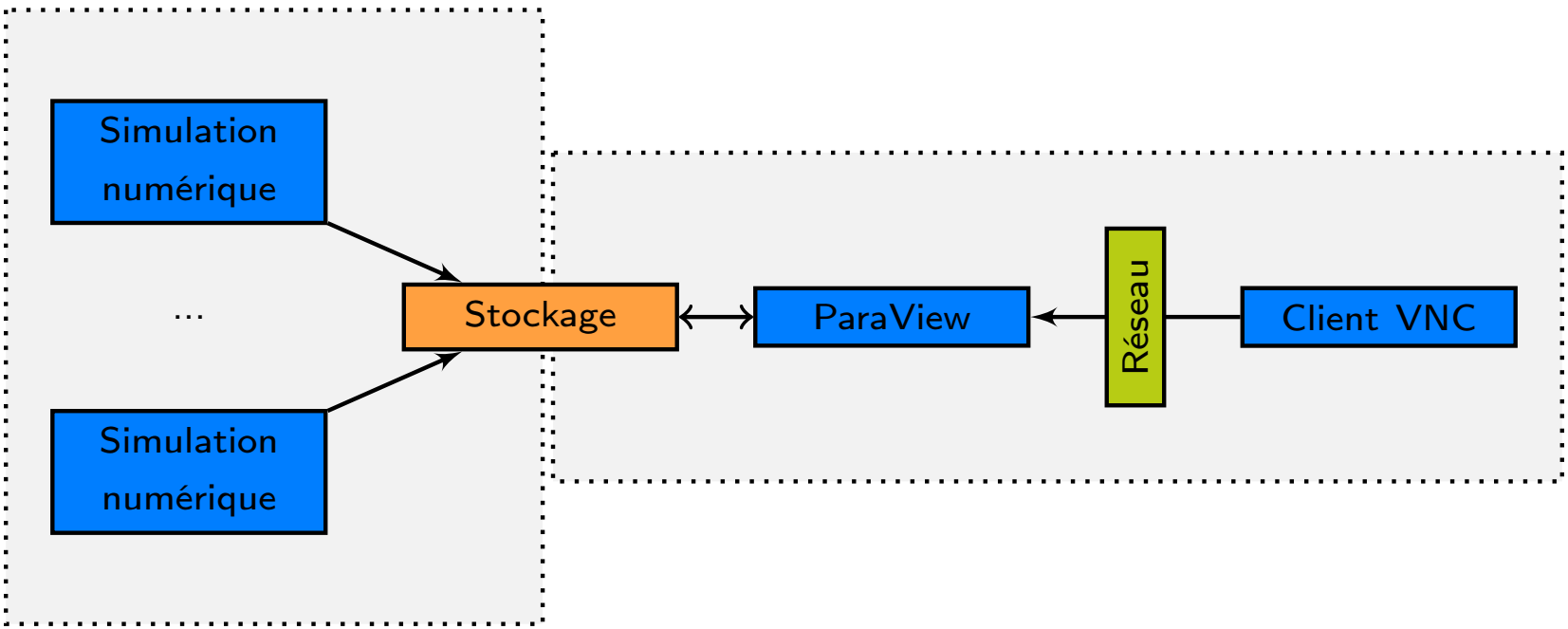
- Copie
  - Copie des données sur la machine locale
  - Exploration locale avec ParaView



Comment visualiser ses données ?

Approches classiques de visualisation de données : Post-mortem

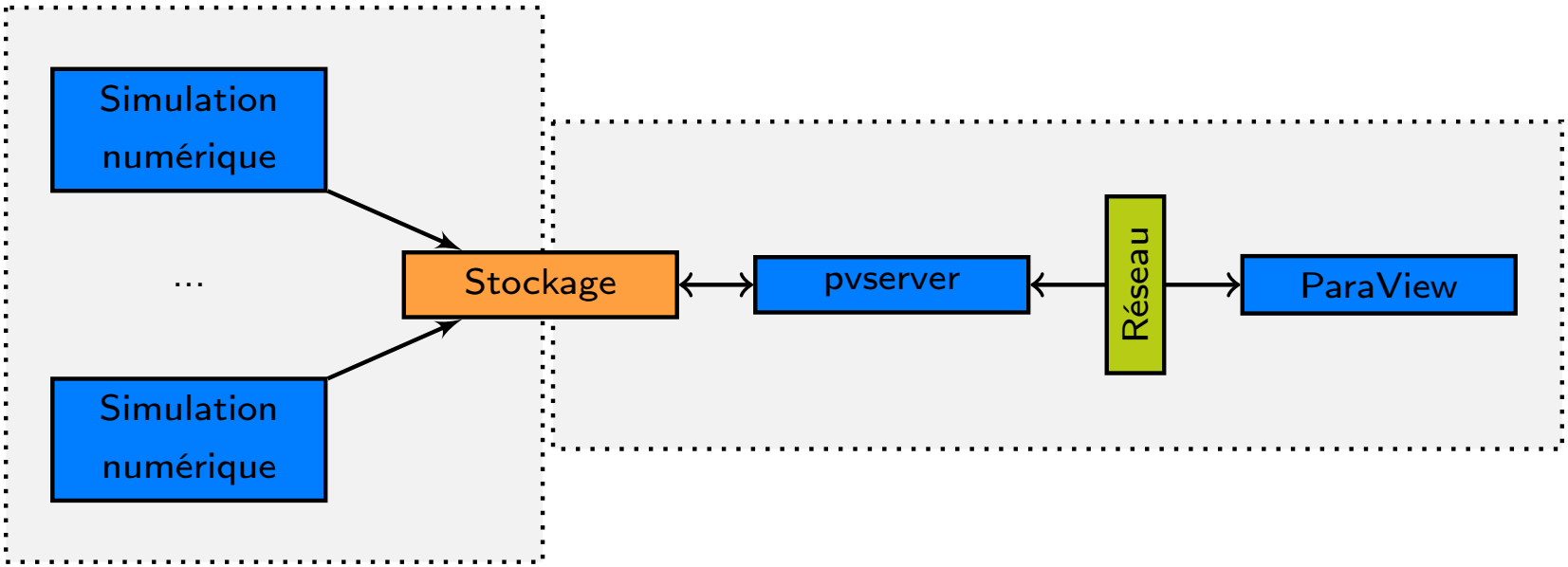
- VirtualGL/VNC
  - Les données restent sur le serveur
  - Lancement du ParaView sur le serveur (avec Virtualgl par exemple)
  - Lancement d'un serveur VNC sur le serveur
  - Connexion au serveur VNC et intéraction



Comment visualiser ses données ?

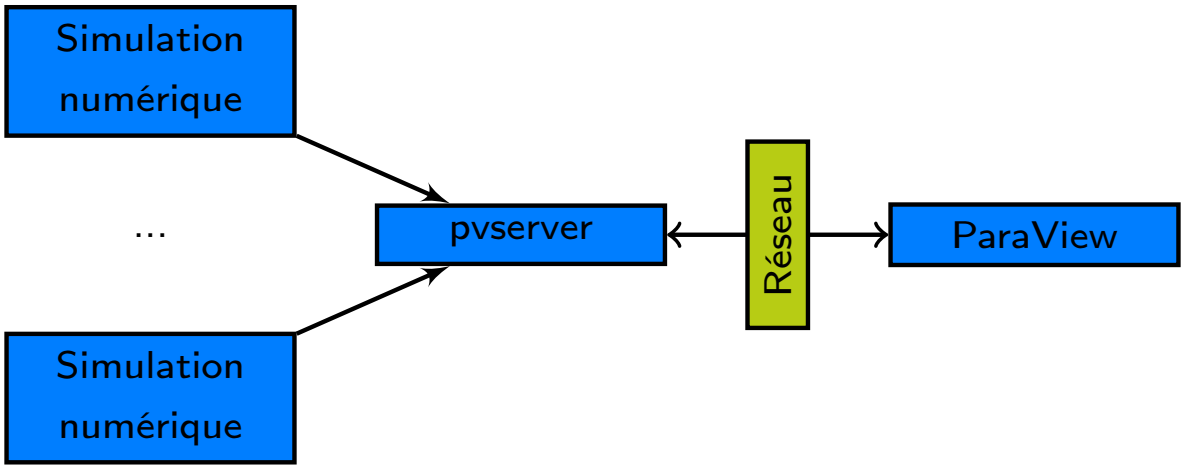
Approches classiques de visualisation de données : Post-mortem

- pvserver et pvdataserver/pvrenderserver
  - Les données restent sur le serveur
  - Lancement d'un pvserver (pvdataserver/pvrenderserver) sur le(s) serveur(s)
  - Connexion au(x) pvserver (pvdataserver/pvrenderserveur) et intéraction
  - Autre :
    - Verifier que le DISPLAY soit bien accessible sur le pvserver ou pvrenderserver Données rendues soit localement, soit sur le serveur
    - Possibilité de configurer le rendu local/distant en fonction de la taille des données : ParaView Settings → Render View → Remote Render Threshold



Comment visualiser ses données ?

- Approche In-situ :
  Visualiser les données dès qu'elles sont générées par la simulation



ParaView

- Principale motivation pour l'In-situ :
  - Augmentation sans relâche de la puissance de calcul
    ⇒ Permet de lancer des simulations avec des précisions croissantes
    ⇒ Augmentation de la taille des données en sortie
  - Le stockage des données à un coût à la fois en temps d'entrée-sorties et financier ⇒ Intégrer la phase de post-processing dans la simulation
- Avantages :
  - S'interface avec n'importe quel pipeline VTK
  - Réduire, voire éviter, les écritures sur disque
  - Raccourcir le pipeline de la simulation à la visualisation :
    ⇒ Pas de nécessité de copier des données depuis un cluster distant ...
    ⇒ Permet le monitoring de la simulation
  - Possibilité de modifier les données sauvegardées/envoyées depuis la simulation :
    Elagage, extraction d'iso-surface(s), extraction des informations pertinentes de la simulation ...
- Inconvénients :
  - Les données peuvent transiter par le réseau : Bande passante !

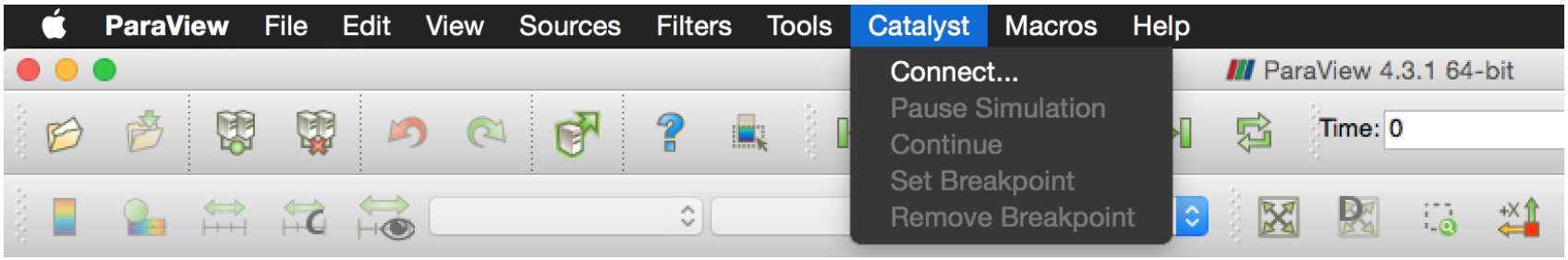
Plan

Point de départ

- Approche :
  - Approche développeur
  - Utilisation de MPI pour les simulations
- Vous avez une application qui :
  - Génère un maillage (potentiellement variable en fonction du temps)
  - Génère des champs de valeurs pour le maillage (scalaires/vecteurs/tenseurs par sommet/simplexe/hypercube/polygone)
  - Potentiellement peut écrire ses données au format VTK
- Exemple avec Feel++ :
  - Librairie Open-source en C++11, https://github.com/feelpp/feelpp
  - Résolution d'équation à dérivées partielles
  - Syntaxe proche de la syntaxe mathématique
  - Parallèle (MPI)
  - Plusieurs backends d'écriture de données : EnSight Gold, HDF5 ... VTK

In-Situ avec ParaView : Catalyst Live

- Implémentation ParaView : Catalyst Live (Depuis la version 4.2)
  - La simulation se connecte à ParaView/pvserver (pas l'inverse)
  - Le code de coprocessing peut être implémenté :
    - soit en Python (C++ "wrappé")
    - ou en C++ (plus complexe)
  - Peu d'impact sur le code
    (mise à part la génération de données VTK, si non implémentée)
  - Un pipeline basique peut être facilement implémenté
- ParaView Catalyst editions :
  - Versions allégées du client ParaView :
    Utilisent moins de mémoire, se concentrent sur la fonctionnalité Catalyst
  - Configurable avec les composants désirés
  - http://www.kitware.com/blog/home/post/606
- Dans l'interface de ParaView :



Integration avec vos codes

- Compilation/Installation :
  - Utiliser cmake pour votre application (facilite l'intégration avec ParaView)
  - Téléchager les sources de ParaView (actuellement, 4.3.1) et compiler :
    - avec Catalyst, MPI et les headers de développement

```
-DPARAVIEW_ENABLE_CATALYST=ON -DPARAVIEW_USE_MPI=ON
-DPARAVIEW_INSTALL_DEVELOPMENT_FILES=ON
```

    - (Optionnel) avec Python

```
-DPARAVIEW_ENABLE_PYTHON=ON
```

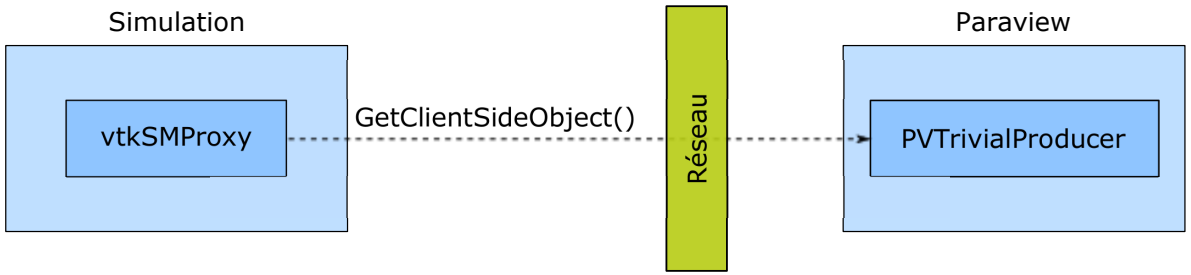
  - Votre application :
    - Compiler et linker votre application avec les headers et librairies de ParaView (Pas de VTK)
    - Typiquement,

```
find_package(ParaView REQUIRED COMPONENTS
    vtkParallelMPI vtkPVCatalyst vtkPVPythonCatalyst)
```

Integration avec vos codes

- Créer et remplir les objets VTK pour gérer vos données :
  - Remplir le type correspondant à vos données
    e.g. vtkPolyData, vtkUnstructuredGrid ...
  - Utiliser une structure composite : vtkMultiBlockDataSet
    - Pour empaqueter les données des différents processus
    - Chaque processeur affecte ses données au block correspondant à son rang
    - Les autres indices resteront NULL
- Si vous utilisez MPI :
  - Par défaut, ParaView utiliser le communicateur MPI_COMM_WORLD
  - Indiquer un communicateur différent avec vtkMPICommunicatorOpaqueComm
- Ajouter le code Catalyst à votre application

- Quelques détails de ce qui se passe sous le capot :
  - Utilise l'API Server-Manager de ParaView : vtkSMSessionProxyManager, vtkLiveInsituLink
  - Proxy : Un objet qui est une interface pour un autre objet.



- Classes importantes :
  - vtkCPProcessor : Gestion du co-processing (AddPipeline(...))
  - vtkCPPipeline : Interface pour l'implémentation de pipelines C++
  - vtkCPPythonScriptPipeline : Fille de vtkCPPipeline et utilise un script python comme pipeline
  - vtkCPDataDescription : Transmission d'information entre le code et le pipeline
  - vtkCPInputDataDescription : Utilisé par vtkCPDataDescription, transmission des informations sur le maillage et les champs

Dans votre code d'export : Initialisation (utiliser les vtkSmartPointer)

- Initialisation :

```
vtkMPICommunicatorOpaqueComm * opaqueComm =
    new vtkMPICommunicatorOpaqueComm(mpicomm);
vtkCPProcessor * cpp = new vtkCPProcessor();
cpp->Initialize(*(opaqueComm));
```

- Si pipeline en python :

```
vtkCPPythonScriptPipeline * pp =
    new vtkCPPythonScriptPipeline();
pp->Initialize("script.py");
cpp->AddPipeline(pp);
```

- Si pipeline C++ :

```
vtkBaseInsituPipeline * pp = new vtkBaseInsituPipeline();
pp->Initialize();
cpp->AddPipeline(pp);
```

Dans votre code d'export : Traitement

- Traitement des données :

```
vtkMultiBlockDataSet * mdbs = ...

vtkCPDataDescription * dd = new vtkCPDataDescription();
dd->AddInput("input");
dd->SetTimeData(timeValue, timeIndex);

if(cpp->RequestDataDescription(dd) != 0)
{
    dd->GetInputDescriptionByName("input")->SetGrid(mbds);
    cpp->CoProcess(dd);
}
```

- Finalisation :

```
cpp->Finalize();
```

vtkBaseInSituPipeline : Initialisation

- Initialisation :
  - Récupérer l'instance courante du gestionnaire de proxy :

```
spxm = vtkSMProxyManager::GetProxyManager()
    ->GetActiveSessionProxyManager();
```

  - Créer un nouveau proxy sur un PVTrivialProducer et l'enregistrer :

```
vtkSMProxy * px =
    spxm->NewProxy("sources", "PVTrivialProducer");
vtkSMSourceProxy * spx = vtkSMSourceProxy::SafeDownCast(px);
spxm->RegisterProxy("sources", spx);
```

  - Initialisation du lien Catalyst Live :

```
isl = vtkSmartPointer<vtkLiveInsituLink>::New();
isl->SetHostname("hostname");
isl->SetInsituPort(portId);
isl->Initialize(spxm);
```

vtkBaseInSituPipeline : CoProcess(...)

- CoProcess(vtkCPDataDescription * dd) :
  - Mise à jour des données et des paramètres des données :

```
vtkObjectBase * cso = spx->GetClientSideObject();
vtkPVTrivialProducer * rp =
    vtkPVTrivialProducer::SafeDownCast(cso);
rp->SetOutput(
    dd->GetInputDescriptionByName("input")->GetGrid(),
    dd->GetTime() );

double time = dd->GetTime();
vtkIdType timeStep = dd->GetTimeStep();
```

vtkBaseInSituPipeline : CoProcess(...)

- CoProcess(vtkCPDataDescription * dd) :
  - Mise à jour de la visualisation et intéraction :

```
/* Tant que la simulation est en pause */
while(true)
{
    /* M.a.j. état de la simulation, extracts et */
    /* simulationPaused */
    isl->InsituUpdate(time, timeStep);
    spx->UpdatePipeline(time);
    /* envoi des extracts */
    isl->InsituPostProcess(time, timeStep);

    /* En pause ? */
    if(isl->GetSimulationPaused())
    {
        /* Attends une action de l'utilisateur */
        if(isl->WaitForLiveChange())
        { break; }
    }
    else
    { break; }
}
```

Exemples de codes

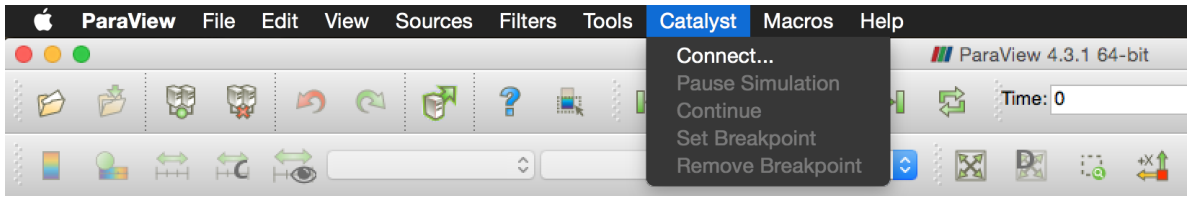
- Python :
  - Simplifié, pas de référence aux vtkSMSessionProxyManager et vtkLiveInsituLink
  - Exemple de code python : http://www.kitware.com/blog/home/post/722
  - API Python : http://www.paraview.org/ParaView/Doc/Nightly/www/py-doc/paraview.coprocessing.html
- Dépots de codes sources Kitware : https://github.com/Kitware/ParaViewCatalystExampleCode
- Implémentation de Catalyst dans Feel++ : https://github.com/feelpp/feelpp/blob/develop/feel/feelfilters/exporterVTK.hpp

Plan

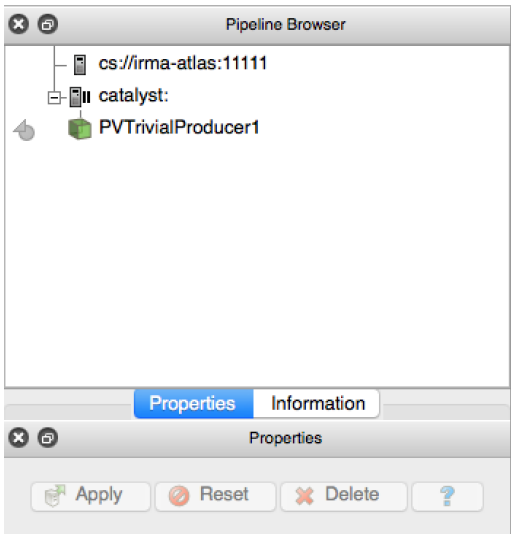
Mise en place

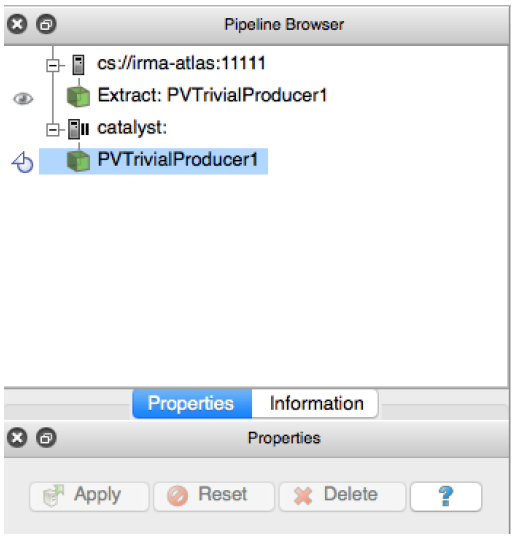
Cas 1 : ParaView et l'application sont lancés localement

- Lancer ParaView
  Catalyst → Connect ..., Catalyst → Pause Simulation



- Lancer l'application
- Interaction dans ParaView





Mise en place

- Cas 2 : Scénario distant (sans firewall)
  - (distant) Lancer pvserver
  - (local) ParaView : Se connecter au pvserver (Client/Server connection: cs://...)
  - (local) ParaView : Catalyst → Connect ..., Catalyst → Pause Simulation
  - (distant) Lancer l'application
  - (local) Interagir dans ParaView
- Scénarios plus avancés (firewall . . .) :
  - Connexion inversée (le pvserver se connecte au client) :

```
--reverse-connection --client-host=clienthname
```

  - Tunneling ssh :

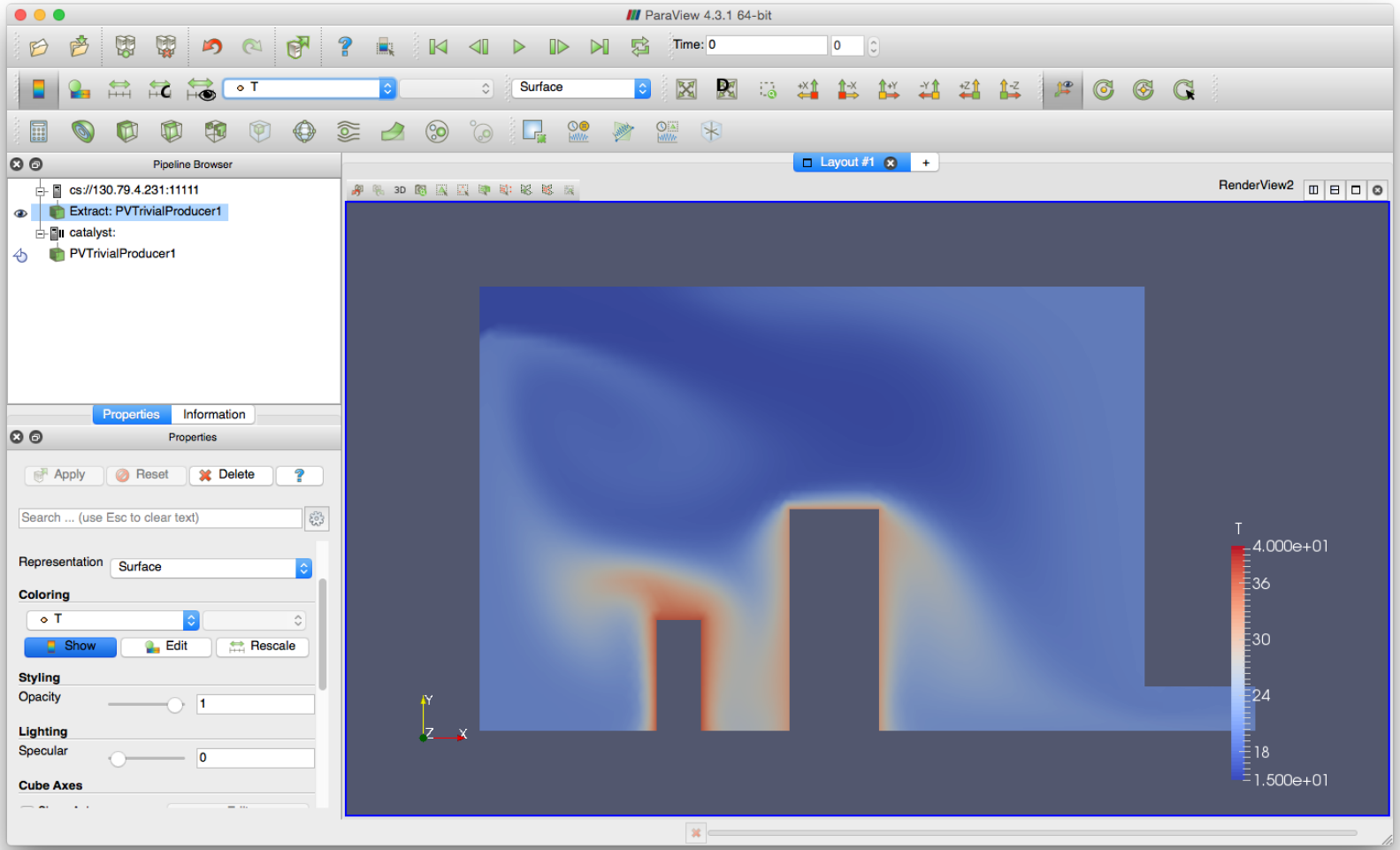
```
ssh -L port-local:frontalnode:port-distant frontalnode
```

  - Tunneling ssh avec connexion inversée :

```
ssh -R port-distant:localhost:port-local frontalnode
```

- Informations complémentaires : http://www.paraview.org/Wiki/Reverse_connection_and_port_forwarding

Démonstration (Crédits : Jean-Baptiste Wahl)



Plan

Conclusion et perspectives

- Catalyst Live permet d'ajouter une composante In-situ facilement à vos codes
- Cela permet de(d') :
  - Raccourcir le pipeline de visualisation en intégrant le post-processing à la simulation
  - Eviter les écritures de fichiers, qui peuvent se révéler coûteuses
- De plus, on garde le contrôle de ses données :
  - En amont, avec la possibilité de les pré-traiter (élagage, . . .)
  - En aval, avec la possibilité d'utiliser des filtres dans ParaView
- Concernant les implémentations disponibles :
  - L'approche python est plus simple à intégrer et ne requiert pas de recompiler son code
  - L'approche C++ est plus intégrée à votre code et permet de mieux comprendre les mécanismes mis en jeu
- Perspectives
  - Certaines applications que nous développons utilisent des Plugins ParaView pour contrôler la simulation
    Possibilité de les intégrer dans le pipeline Catalyst ?

- Merci de votre attention !
- Documentation & liens utiles :
  - Page Insitu de ParaView : http://www.paraview.org/in-situ
  - Documentation : http://www.paraview.org/files/catalyst/docs/ParaViewCatalystUsersGuide_v2.pdf
  - Documentation de ParaView : http://www.paraview.org/ParaQ/Doc/Nightly/html/index.html
  - Dépôts de codes sources Kitware : https://github.com/Kitware/ParaViewCatalystExampleCode
  - Mailing list ParaView : http://www.paraview.org/mailing-lists/
- Contact : ancel@math.unistra.fr