SHARC

ARCANE-BASED
POROUS MEDIA FLOW
MINI APPLICATION(S)

RAPHAËL GAYNO

EQUIPE ARCANE-ARCGEOSIM

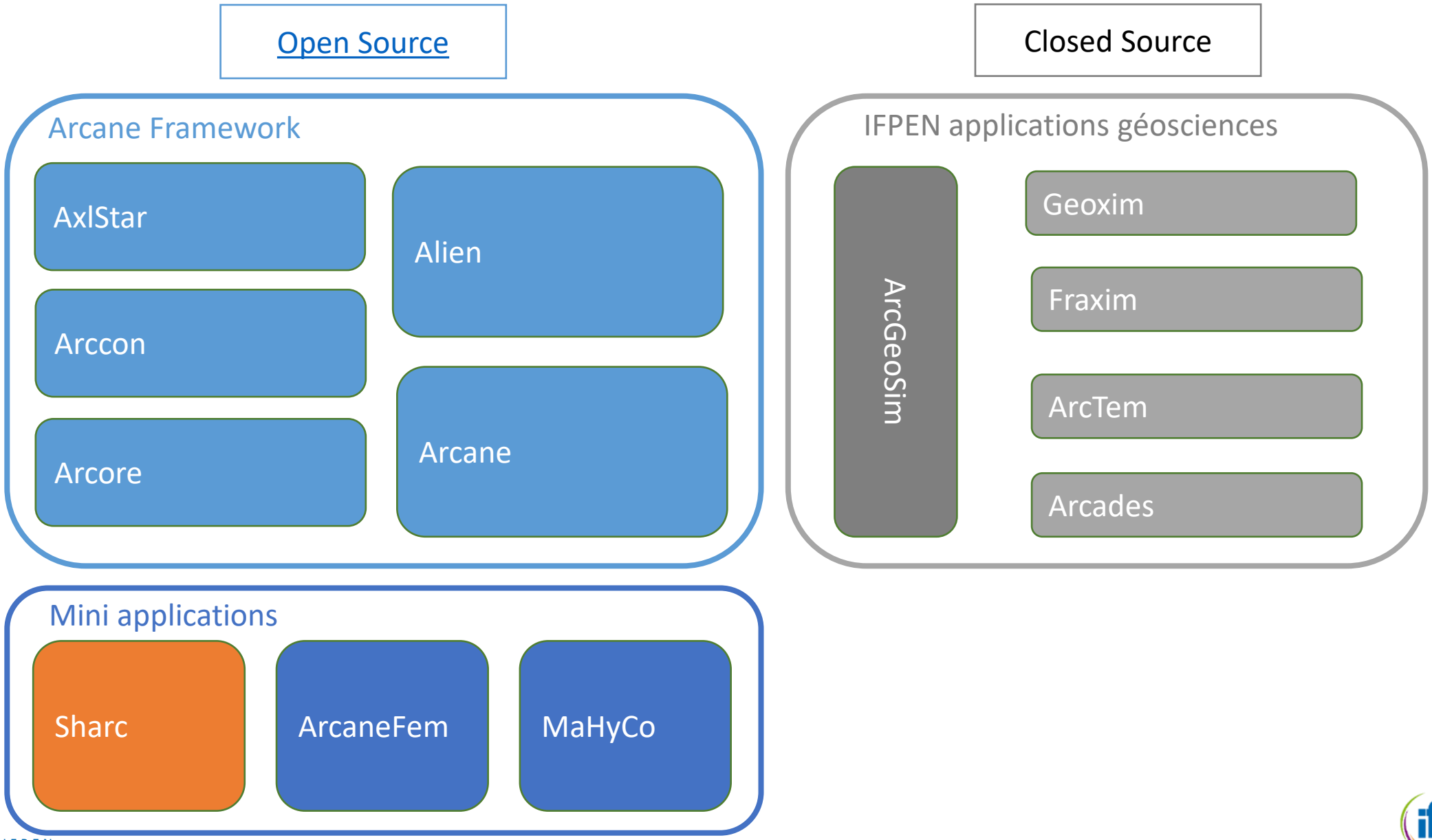
24/03/25

PLAN

1. Contexte
2. Gump (modèle de données)
3. Framework de Lois
4. Audi contributions et assemblage
5. Balade dans les sources

Contexte

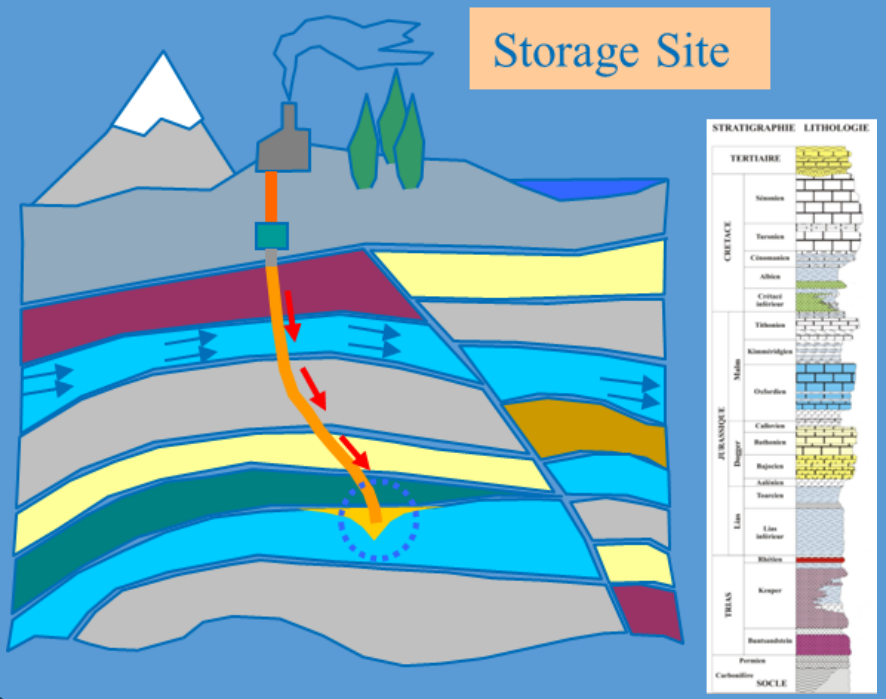
ARCANE ET SES APPLICATIONS



SHARC PROXY APP DE GEOXIM

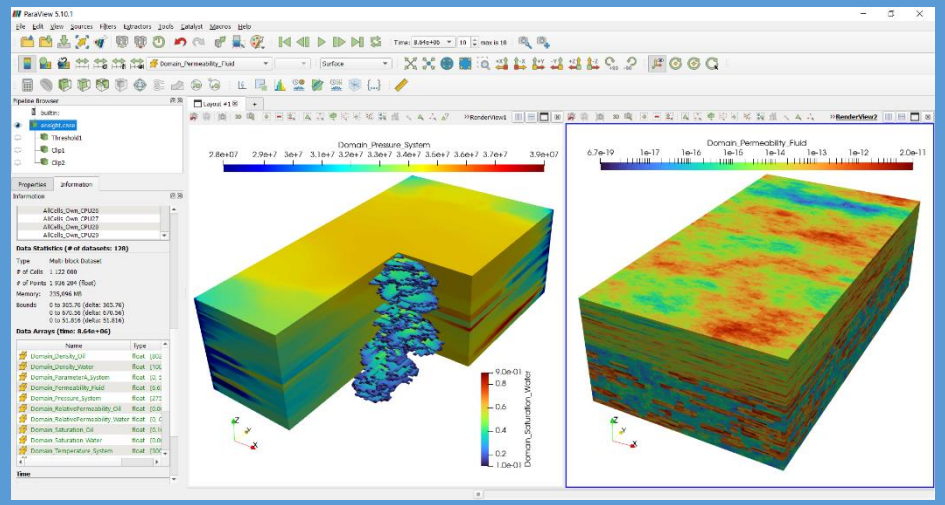
Geoxim

multiphase flow
advanced multipoints schemes
reactive transport
geomechanic



Sharc

two-phase flow (W-G)
only two-points scheme



Sharc : Modèle de données Gump

GUMP

Entités hiérarchisées

```
<gump>
  <model namespace="ArcRes">
    <entity name="System">
      <contains>
        <entity name="FluidSubSystem" unique="true" />
      </contains>
    </entity>
    <entity name="FluidSubSystem">
      <contains>
        <entity name="FluidPhase" />
      </contains>
    </entity>
    <entity name="FluidPhase">
      <contains>
        <entity name="Species" />
      </contains>
    </entity>
    <entity name="Species">
      <contains>
        <entity name="Component" />
      </contains>
    </entity>
  </model>
</gump>
```

Propriétés par entités

```
<entity name="FluidPhase">
  <supports>
    <property name="Viscosity" dim="scalar" type="real" />
    <property name="Density" dim="scalar" type="real" />
    <property name="CapillaryPressure" dim="scalar" type="real" />
  </supports>
</entity>
```

A la compilation, une version c++ de ce modèle statique Sharc.gump est générée dans le build

Spécialisation du modèle

```
<physical-model>
  <system name="UserSystem">
    <name>System</name>
    <fluid-system>
        <name>Fluid</name>
        <fluid-phase>
            <name>Water</name>
            <species>H2O</species>
        </fluid-phase>
        <fluid-phase>
          <name>Gas</name>
          <species>CO2</species>
        </fluid-phase>
    </fluid-system>
  </system>
</physical-model>
```

A l'exécution, ce modèle très générique est ensuite spécialisé grâce au .arc (fichier d'entrée de la simulation)

GUMP

- Ce mécanisme permet une uniformisation du code:
  - Dans le code:
    - Boucle sur les entités
    - Accès aux propriétés par nom et entité

```
ENUMERATE_PHASE(iphase, m_system) {
    sharc::Saturation(iphase)=;
    ENUMERATE_SPECIES(ispecies, iphase) {
        sharc::MolarFraction(ispecies))=;
    }
}
```

  - Dans le .arc fichier d'entrée de la simulation
    - Accès aux propriétés

    par nom et entité convention XPath

```
<initial-condition name="Constant">
    <property>[SubSystem]Fluid::VolumeFraction</property>
    <condition>0.4</condition>
</initial-condition>
```

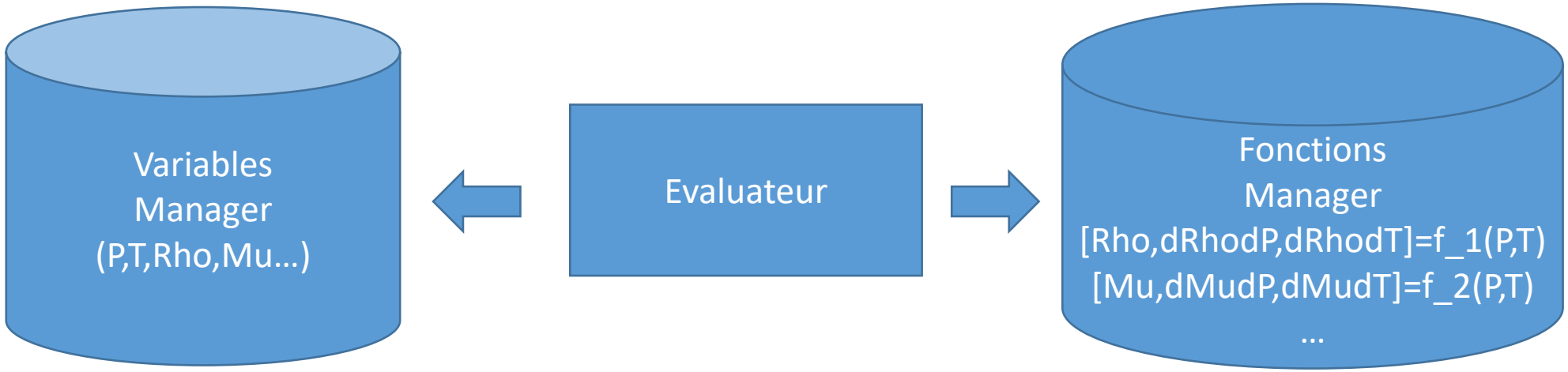
Sharc : Law Framework

LAW FRAMEWORK

Conteneurs:
- Scalaire
- Tableaux…

Un support:
- Un tableau
- Tout le maillage
- Une zone…

Fonction val + deriv:
- En un point
- En une maille…



CREATION ET ENREGITREMENT D'UNE LOI

signature

Fichier: RhoType.law

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<law name="RhoType">
 <input name="pressure" type="real" dim="scalar" />
 <input name="temperature" type="real" dim="scalar" />
 <output name="rho" type="real" dim="scalar" />
</law>
```

Intégré au système de compilation: génère un fichier RhoType_law.h

fonction

```
class RhoLaw {
void  eval(Real pressure, Real temperature, Real& rho,
            Real& drho_dp, Real drho_dt)  {
    rho = …
    drho_dp = …
    drho_dt = …
  }
}
```

enregistrement

```
#include "ArcGeoSim/Physics/Law2/FunctionManager.h"
#include "RhoType_law.h"
#include "RhoLaw.h "

RhoType::Signature s;
s.pressure = prop_pressure;
s.temperature = prop_temperature;
s.rho = prop_rho;

auto f = std::make_shared<RhoType::Function>(s, m_law,  &RhoLaw::eval);
Law::FunctionManager  funcs;
funcs << f;
```

Présentation Audi - Contributions

PRINCIPES GENERAUX CONTRIBUTIONS

- Idée gump + lois + audi (différentiation automatiques) pour assemblages (accumulation, flux…).
- Rassembler les propriétés de lois sur un support commun (même groupe de mailles…)
  - Folder
- Unknown Manager (sélection des dérivées assemblées)
- Contribution

```
auto press = Geoxim::contribution<Geoxim::Pressure>(domain(),um,system());
auto rho = Geoxim::contribution<Geoxim::Density>(domain(),um,fluid.phases());
ENUMERATE_CELL(icell,domain())
{
   Law::Contribution current _pressure = press[icell];
   ENUMERATE_PHASE(iphase,fluid.phases)
   {
      Law::Contribution current _density = density[icell][iphase];
      Law::Contribution current_op_with_audi = current _pressure * current _density;  // operator (+ - * /)
   }
}
```

ASSEMBLAGE DES FLUX INITIALISATION

```
void TwoPhaseFlowSimulationModule::
_buildFluxInternal(ArcNum::Vector& residual, ArcNum::Matrix& jacobian)
{
    // Get Geoxim Entities
    ArcRes::FluidSubSystem fluid = system().fluidSubSystem();

    // Get Discrete Operator Transmissibility
    const Arcane::VariableFaceReal& T = m_transmissivities["T"][ArcNum::Classical];

    const auto& um = unknownsManager();

    // Get Geoxim Domain Implicit Variables
    auto P = Law::contribution<ArcRes::Pressure>(domain(),functionMng(),um,um,system());
    auto Pc = Law::contribution<ArcRes::CapillaryPressure>(domain(),functionMng(),um,um,fluid.phases());
    auto kr = Law::contribution<ArcRes::RelativePermeability>(domain(),functionMng(),um,um,fluid.phases());
    auto mu = Law::contribution<ArcRes::Viscosity>(domain(),functionMng(),um,um,fluid.phases());
    auto rho = Law::contribution<ArcRes::Density>(domain(),functionMng(),um,um,fluid.phases());

    //boucle d'assemblage slide suivante
```

ASSEMBLAGE DES FLUX BOUCLE

```
Arcane::FaceGroup inner_faces = mesh()->allCells().innerFaceGroup();
ENUMERATE_FACE(iface,inner_faces) {

    ArcNum::TwoPointsStencil stencil(iface);
    const Law::Cell& cell_k = stencil.back();
    const Law::Cell& cell_l = stencil.front();
```

Gestion du stencil

```
    ENUMERATE_PHASE(iphase, fluid.phases()) {
```

Calcul du flux (audi: différentiation automatique)

```
        const auto grad_kl = T[iface] * (P[cell_k] + Pc[iphase][cell_k]  - P[cell_l] - Pc[iphase][cell_l] );

        const auto mobility_k = rho[iphase][cell_k] * kr[iphase][cell_k] / mu[iphase][cell_k];
        const auto mobility_l = rho[iphase][cell_l] * kr[iphase][cell_l] / mu[iphase][cell_l];

        const auto flux_kl = (audi::value(grad_kl)>=0) ? mobility_k*grad_kl : mobility_l*grad_kl ;
```

```
        const Arcane::Integer iequation = iphase.index();
        if (cell_k.isOwn()) {
            residual[iequation][cell_k]          += flux_kl;
            jacobian[iequation][cell_k][stencil] += flux_kl;
        }
        if (cell_l.isOwn()) {
            residual[iequation][cell_l]          -= flux_kl;
            jacobian[iequation][cell_l][stencil] -= flux_kl;
        }
    }
}
```

Ajout du flux au résidu et jacobienne

CONCLUSION

- « Pas encore » open source
  - Composition des lois
  - Validations automatiques des dérivées
  - Différentiation automatique creuse
  - Stencil multipoints
  - Lois pour l'IA (inférence onnx)

- Dans la pile
  - Lois IA apprentissage en ligne
  - Lois sur GPU
  - Gump et système d'unité
  - Use case diphasique compositionnel
  - Audi ajout fonctions mathématiques basiques (sin,cos,exp…)