# Implementation of a new heat transfer model in OpenFOAM for lagrangian particle tracking solvers for use in porous media

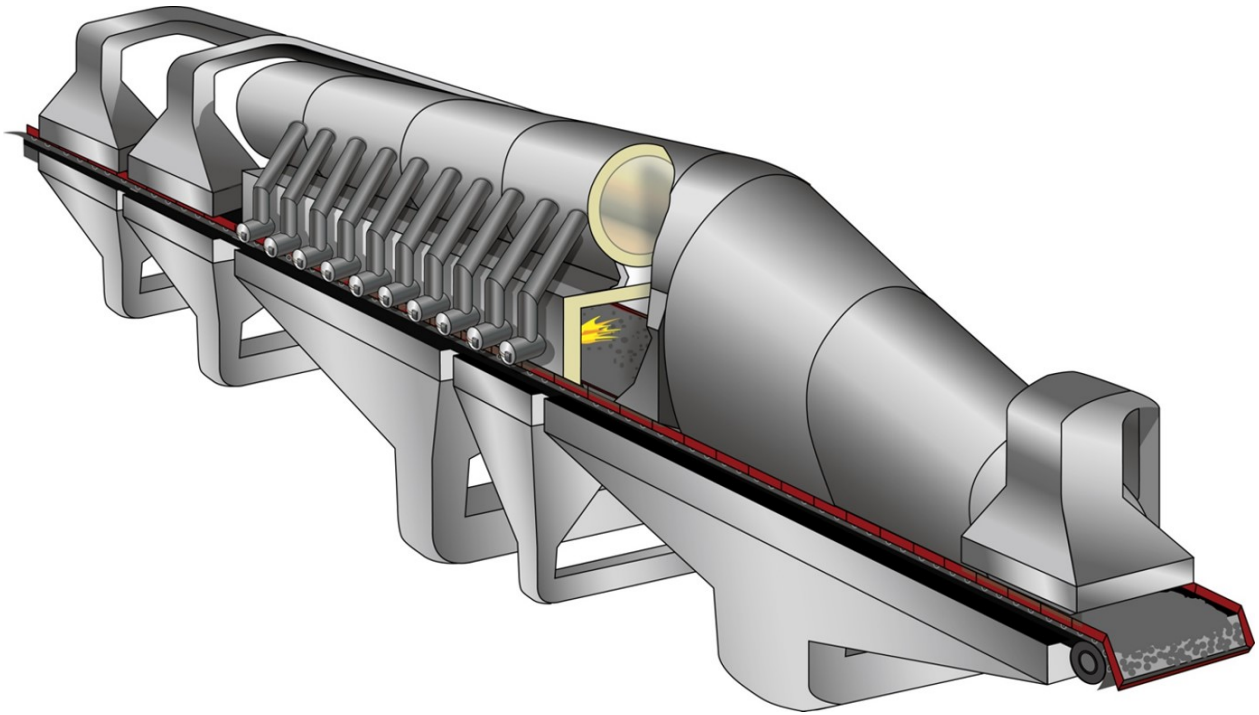
Örjan Fjällborg

Process Metallurgy
Luleå University of Technology
Luleå, Sweden

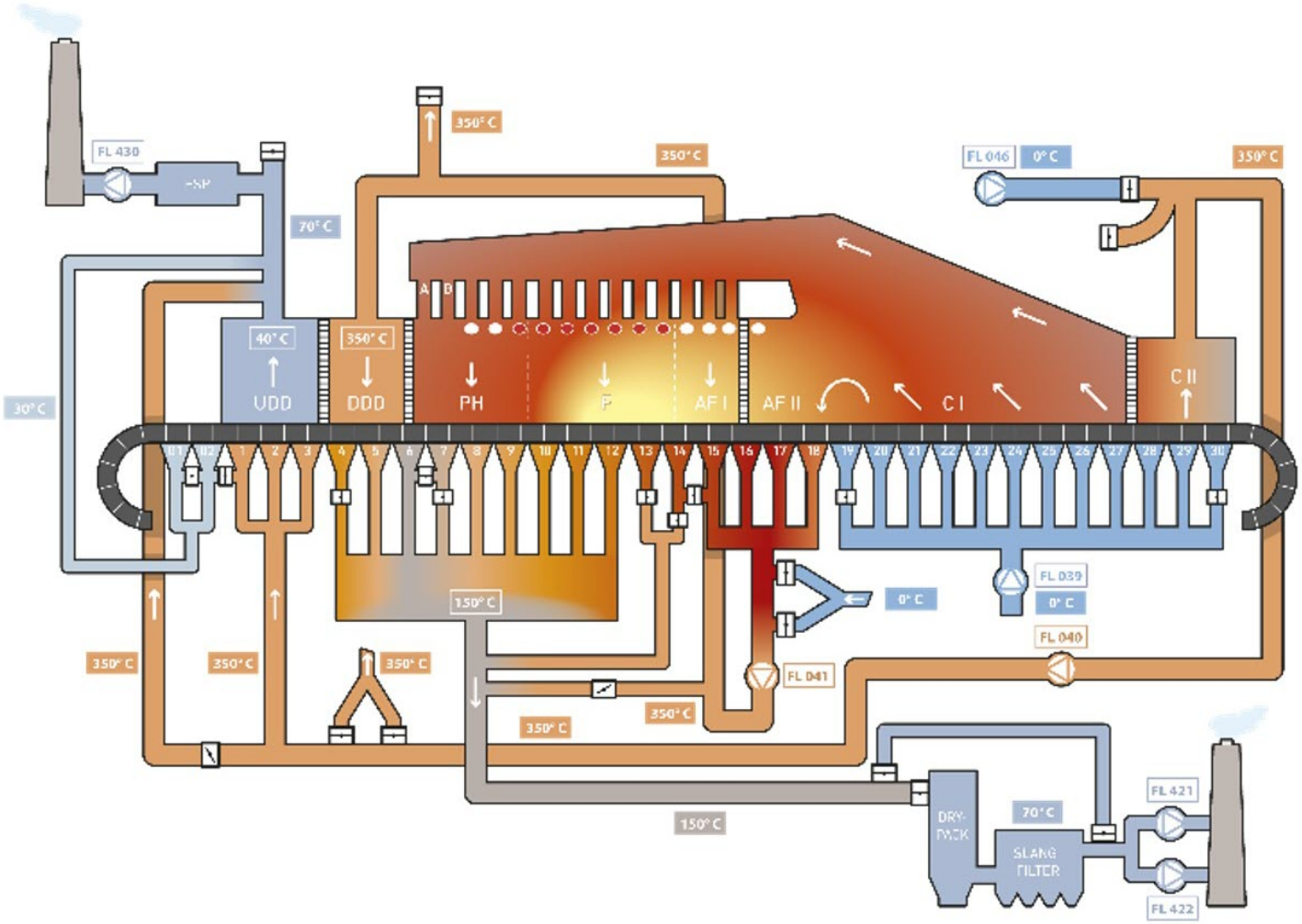
January 15, 2023

# Table of Contents

In the pelletizing machine green pellets are dried, oxidized and sintered to the finished product to the right. Big fans are blowing gas through the pellets bed which is circulated in the process to save energy.



This is a schematic figure showing the process with fans and gas flows coloured by temperature. This machine is about 90 meters long and 3.5 meters wide.



The Experimental Pilot Pot Furnace contains a porous bed of iron ore green pellets with a gas mixture going through it for simulating a real pelletizing plant or to do other investigations like validation of simulation models.

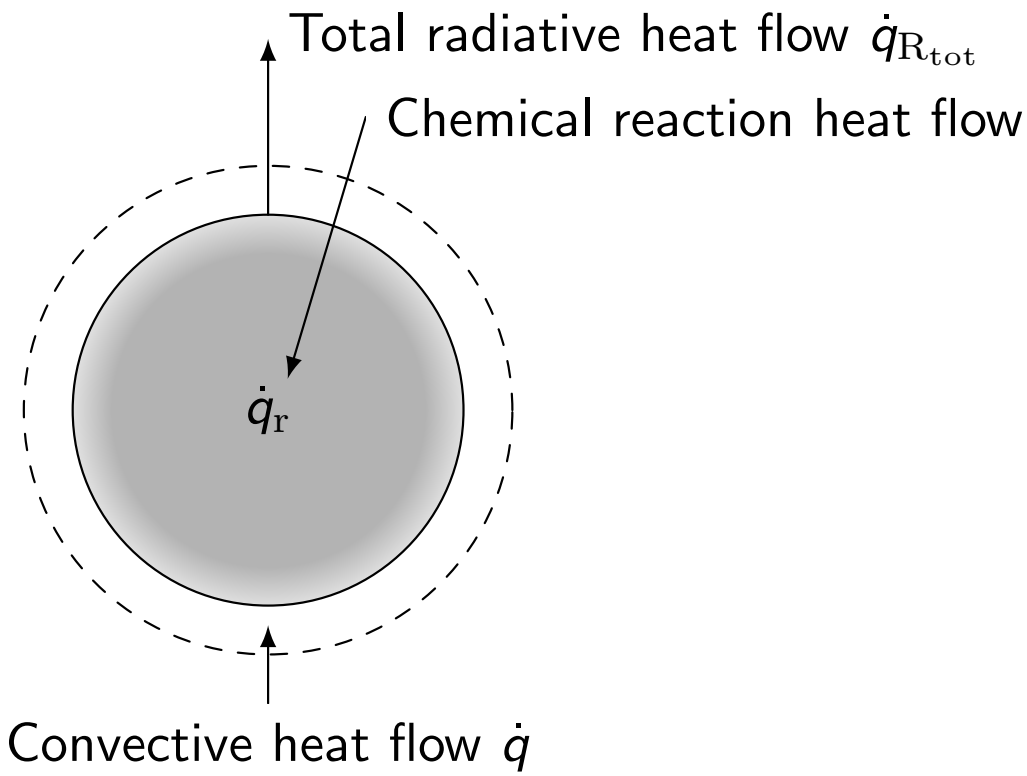



Figure: Heat transfer mechanisms in a gas particle system

# Heat transfer in a gas - solid particles system

The convective heat flow between a particle and a moving gas stream is given by

$$\dot{q} = hA_{\text{srfc}}(T_{\text{g}} - T_{\text{srfc}})$$

ThermoParcel.C

```
template<class ParcelType>
template<class TrackCloudType>
Foam::scalar Foam::ThermoParcel<
    ParcelType>::calcHeatTransfer
...
    const scalar d = this->d();
    const scalar rho = this->rho();
    const scalar As = this->areaS(d);
    const scalar V = this->volume(d);
    const scalar m = rho*V;

    // Calc heat transfer coefficient
    scalar htc = cloud.heatTransfer().
     htc(d, Re, Pr, kappa, NCpW);

    // Calculate the integration
     coefficients
    const scalar bcp = htc*As/(m*Cp_);
    const scalar acp = bcp*td.Tc();
...
```

# Heat transfer in a gas - solid particles system

The total radiative heat flow between a particle and a moving gas stream is given by

$$\dot{q}_{\mathrm{R_{tot}}} = A_{\mathrm{srfc}} \epsilon \left( \frac{G}{4} - \sigma T_{\mathrm{p}}^4 \right)$$

ThermoParcel.C

```
if (cloud.radiation())
{
    const tetIndices tetIs = this->
     currentTetIndices();
    const scalar Gc = td.GInterp().
     interpolate(this->coordinates(),
     tetIs);
    const scalar sigma = physicoChemical
     ::sigma.value();
    const scalar epsilon = cloud.
     constProps().epsilon0();

    ancp += As*epsilon*(Gc/4.0 - sigma*
     pow4(T_));
}
ancp /= m*Cp_;
```

# Heat transfer in a gas - solid particles system

Internally generated heat in a particle from chemical reactions is sent to `ThermoParcel::calcHeatTransfer()` as the input/output parameter `dhsTrans`

$$\dot{q}_{\mathrm{r}} = \text{result from chemical reaction model}$$

ThermoParcel.C

```
template<class ParcelType>
template<class TrackCloudType>
Foam::scalar Foam::ThermoParcel<
    ParcelType>::calcHeatTransfer
(
    TrackCloudType& cloud,
    trackingData& td,
    const scalar dt,
    const scalar Re,
    const scalar Pr,
    const scalar kappa,
    const scalar NCpW,
    const scalar Sh,
    scalar& dhsTrans,
    scalar& Sph
)
```

# Heat transfer in a gas - solid particles system

Putting together convective-, radiative- and chemical heat transfer in an energy balance, and assuming a low Biot number (neglecting internal heat conduction) gives

$$mC_p\frac{\mathrm{d}T}{\mathrm{d}t} = \underbrace{hA_{\mathrm{srfc}}(T_{\mathrm{g}} - T_{\mathrm{p}})}_{\dot{q}} + \underbrace{A_{\mathrm{srfc}}\epsilon\left(\frac{G}{4} - \sigma T_{\mathrm{p}}^4\right)}_{\dot{q}_{\mathrm{R}_{\mathrm{tot}}}} + \dot{q}_{\mathrm{r}}.$$

# Heat transfer in a gas - solid particles system

This first order differential equation can be written on the general form as

$$\frac{\mathrm{d}\phi}{\mathrm{d}t} = A - B\phi,$$

where in this case

$$B = \frac{hA_{\mathrm{srfc}}}{mC_p},$$

$$A = B \cdot T_{\mathrm{g}} + \frac{A_{\mathrm{srfc}}\epsilon\left(\frac{G}{4} - \sigma T_{\mathrm{p}}^4\right) + \dot{q}_{\mathrm{r}}}{mC_p},$$

$$\phi = T_{\mathrm{p}}$$

giving the analytical solution

$$\Delta T_{\mathrm{p}} = (A - B \cdot T_{\mathrm{p}}^n) \underbrace{\frac{1 - Ce^{-B\Delta t}}{B}}_{\Delta t_{\mathrm{e}}}$$

and the Euler solution

$$\Delta T_{\mathrm{p}} = (A - B \cdot T_{\mathrm{p}}^n) \underbrace{\frac{\Delta t}{1 + B\Delta t}}_{\Delta t_{\mathrm{e}}}.$$

# Heat transfer in a gas - solid particles system

The new particle (=surface) temperature $T_{\mathrm{p}}^{n+1}$ is calculated with

$$\begin{aligned} \Delta t_{\mathrm{e}} &= f(\Delta t, B), \\ \Delta T_{\mathrm{p}} &= (A - B \cdot T_{\mathrm{p}}^{n}) \Delta t_{\mathrm{e}} \\ \textit{and} & \\ T_{\mathrm{p}}^{n+1} &= T_{\mathrm{p}}^{n} + \Delta T_{\mathrm{p}}. \end{aligned}$$

ThermoParcel.C

```
...
    // Integrate to find the new parcel
     temperature
    const scalar deltaT = cloud.
     TIntegrator().delta(T_, dt, acp +
     ancp, bcp);
    const scalar deltaTncp = ancp*dt;
    const scalar deltaTcp = deltaT -
     deltaTncp;

    // Calculate the new temperature and
      the enthalpy transfer terms
    scalar Tnew = T_ + deltaT;
    Tnew = min(max(Tnew, cloud.
     constProps().TMin()), cloud.
     constProps().TMax());

    dhsTrans -= m*Cp_*deltaTcp;
...
```

# Heat transfer in a gas - solid particles system

The effective time step $\Delta t_e$ is calculated in the `delta()` method

integrationSchemeTemplates.C

```
template<class Type>
inline Type Foam::integrationScheme::
    delta
(
    const Type& phi,
    const scalar dt,
    const Type& Alpha,
    const scalar Beta
) const
{
    return explicitDelta(phi, dtEff(dt,
    Beta), Alpha, Beta);
}
```

# Heat transfer in a gas - solid particles system

Which then calls the `dtEff()` method implemented in the two subclasses:

analytical.C

```
Foam::scalar Foam::integrationSchemes::
    analytical::dtEff
(
    const scalar dt,
    const scalar Beta
) const
{
    return
        mag(Beta*dt) > SMALL
      ? (1 - exp(- Beta*dt))/Beta
      : dt;
}
```

Euler.C

```
Foam::scalar Foam::integrationSchemes::
    Euler::dtEff
(
    const scalar dt,
    const scalar Beta
) const
{
    return dt/(1 + Beta*dt);
}
```

$$\Delta t_e = \frac{1 - Ce^{-B\Delta t}}{B}$$

$$\Delta t_e = \frac{\Delta t}{1 + B\Delta t}$$

# Heat transfer in a gas - solid particles system

With the effective time step calculated, the `explicitDelta()` method calculates

$$\Delta T_{\mathrm{p}} = (A - B \cdot T_{\mathrm{p}}^{n})\Delta t_{\mathrm{e}}$$

integrationSchemeTemplates.C

```
template<class Type>
inline Type Foam::integrationScheme::
    explicitDelta
(
    const Type& phi,
    const scalar dtEff,
    const Type& Alpha,
    const scalar Beta
)
{
    return (Alpha - Beta*phi)*dtEff;
}
```

# Heat transfer in a gas - solid particles system

The integration coefficients is implemented

$$B = \frac{hA_{\mathrm{srfc}}}{mC_p},$$

$$A = B \cdot T_{\mathrm{g}} + \frac{A_{\mathrm{srfc}}\epsilon\left(\frac{G}{4} - \sigma T_{\mathrm{p}}^4\right) + \dot{q}_{\mathrm{r}}}{mC_p}$$

but $\dot{q}_{\mathrm{r}}$ is not included in the integration coefficient A, which means that heat from reaction is not affecting the particle temperature directly!

ThermoParcel.C

```
// Calculate the integration coefficients
const scalar bcp = htc*As/(m*Cp_);
const scalar acp = bcp*td.Tc();
scalar ancp = Sh;
if (cloud.radiation())
{
    const tetIndices tetIs = this->currentTetIndices
     ();
    const scalar Gc = td.GInterp().interpolate(this->
     coordinates(), tetIs);
    const scalar sigma = physicoChemical::sigma.value
     ();
    const scalar epsilon = cloud.constProps().
     epsilon0();

    ancp += As*epsilon*(Gc/4.0 - sigma*pow4(T_));
}
ancp /= m*Cp_;
```

# Heat transfer in a gas - solid particles system

But dhsTrans already contains heat from reaction which will directly heat or cool the gas. The temperature change for the particle (without radiation) will draw heat from the gas (last line)

ThermoParcel.C

```
...
// Integrate to find the new parcel
    temperature
const scalar deltaT = cloud.TIntegrator().
    delta(T_, dt, acp + ancp, bcp);
const scalar deltaTncp = ancp*dt;
const scalar deltaTcp = deltaT - deltaTncp;

// Calculate the new temperature and the
    enthalpy transfer terms
scalar Tnew = T_ + deltaT;
Tnew = min(max(Tnew, cloud.constProps().TMin()
    ), cloud.constProps().TMax());

dhsTrans -= m*Cp_*deltaTcp;
...
```

# MyThermoParcel

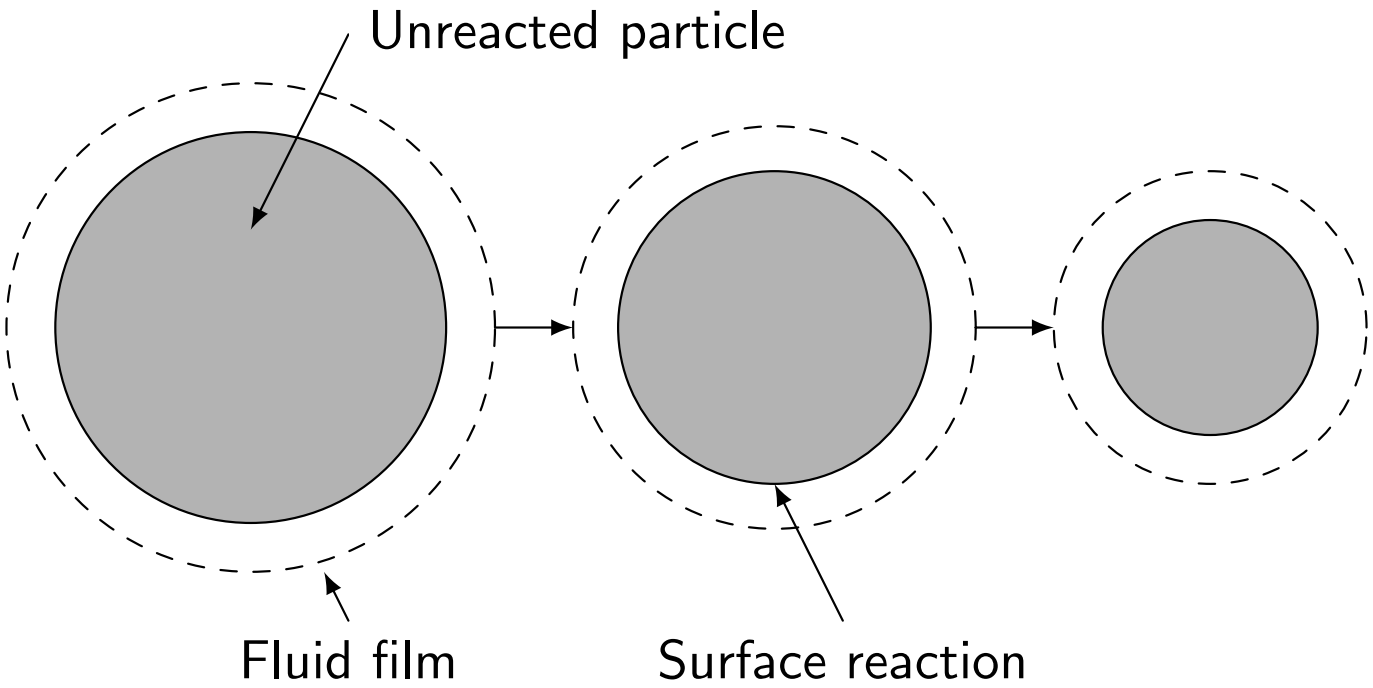



Figure: Gasification of a solid particle under conditions where the external dimensions of the particle shrink. Most heat of reaction is transfered externally from the particle.

# MyThermoParcel

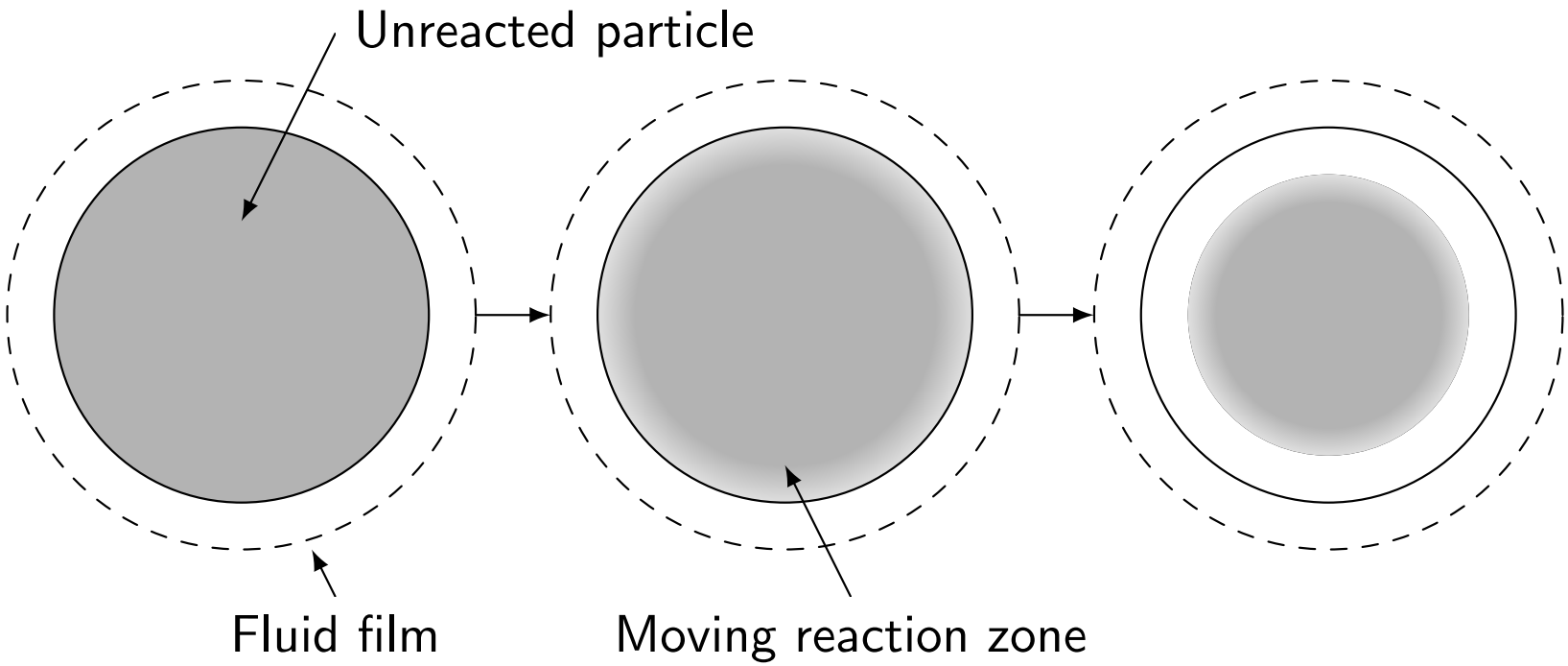



Figure: The progress of a reaction of a porous particle reactant with a gas to form a porous solid product. Most heat of reaction is transfered internally in the particle.

# MyThermoParcel

To make the term $\dot{q}_r$ affect the particle temperature the boolean flag `internalHeatOfReaction` is added to the code

MyThermoParcel.C

```
...
if (cloud.constProps().
    internalHeatOfReaction())
{
    ancp += dhsTrans / dt;
}
ancp /= m*Cp_;

// Integrate to find the new parcel
    temperature
const scalar deltaT = cloud.TIntegrator
    ().delta(T_, dt, acp + ancp, bcp);
const scalar deltaTncp = ancp*dt;
const scalar deltaTcp = deltaT -
    deltaTncp;
```

MyThermoParcel.C

```
// Calculate the new temperature and the
     enthalpy transfer terms
scalar Tnew = T_ + deltaT;
Tnew = min(max(Tnew, cloud.constProps().
    TMin()), cloud.constProps().TMax())
    ;

if (cloud.constProps().
    internalHeatOfReaction())
{
    dhsTrans = -m*Cp_*deltaTcp;
}
else
{
    dhsTrans -= m*Cp_*deltaTcp;
}
...
```

# MyThermoParcel

To replace `ThermoParcel` with `MyThermoParcel`, a new cloud type definition was needed and used in a new version of the solver.

## New cloud type

```
Foam::ReactingHeterogeneousCloud
<
    Foam::ReactingCloud
    <
        Foam::ThermoCloud
        <
            Foam::KinematicCloud
            <
                Foam::Cloud
                <
                    See right side
                >
            >
        >
    >
>
```

## New parcel type

```
Foam::ReactingHeterogeneousParcel
<
    Foam::ReactingParcel
    <
        Foam::MyThermoParcel
        <
            Foam::KinematicParcel
            <
                Foam::particle
            >
        >
    >
>
```

# MyThermoParcel

Files that need to be added or modified are shown in the file structure below:

File structure

```
.
├── Make
│   ├── files
│   └── options
├── clouds
│   └── derived
│       └── basicMyHeterogeneousReactingCloud
│           └── basicMyHeterogeneousReactingCloud.H
└── parcels
    └── Templates
        └── MyThermoParcel
            ├── MyThermoParcel.C
            ├── MyThermoParcel.H
            ├── MyThermoParcelI.H
            ├── MyThermoParcelIO.C
            └── MyThermoParcelTrackingDataI.H
```

# MyThermoParcel

## Continued

### File structure

```
.
└── parcels
    ├── ReactingHeterogeneousParcel
    │   ├── ReactingHeterogeneousParcel.C
    │   ├── ReactingHeterogeneousParcel.H
    │   ├── ReactingHeterogeneousParcelI.H
    │   └── ReactingHeterogeneousParcelIO.C
    └── derived
        ├── basicMyHeterogeneousReactingParcel
        │   ├── basicMyHeterogeneousReactingParcel.H
        │   ├── defineBasicMyHeterogeneousReactingParcel.C
        │   └── makeBasicMyHeterogeneousReactingParcelSubmodels.C
        ├── basicMyReactingParcel
        │   ├── basicMyReactingParcel.H
        │   └── defineBasicMyReactingParcel.C
        └── basicMyThermoParcel
            ├── basicMyThermoParcel.H
            └── defineBasicMyThermoParcel.C
```

# Ranz-Marshall and Rowe heat transfer model

The convective heat transfer coefficient $h$ is related to the Nusselt number as $\mathrm{Nu} = hL/k_\mathrm{g}$, with the Nusselt number given for each model by

Ranz-Marshall:

$$\mathrm{Nu} = f(\mathrm{Re}, \mathrm{Pr})$$
$$\mathrm{Nu} = 2.0 + 0.6\mathrm{Re}^{1/2}\mathrm{Pr}^{1/3}$$

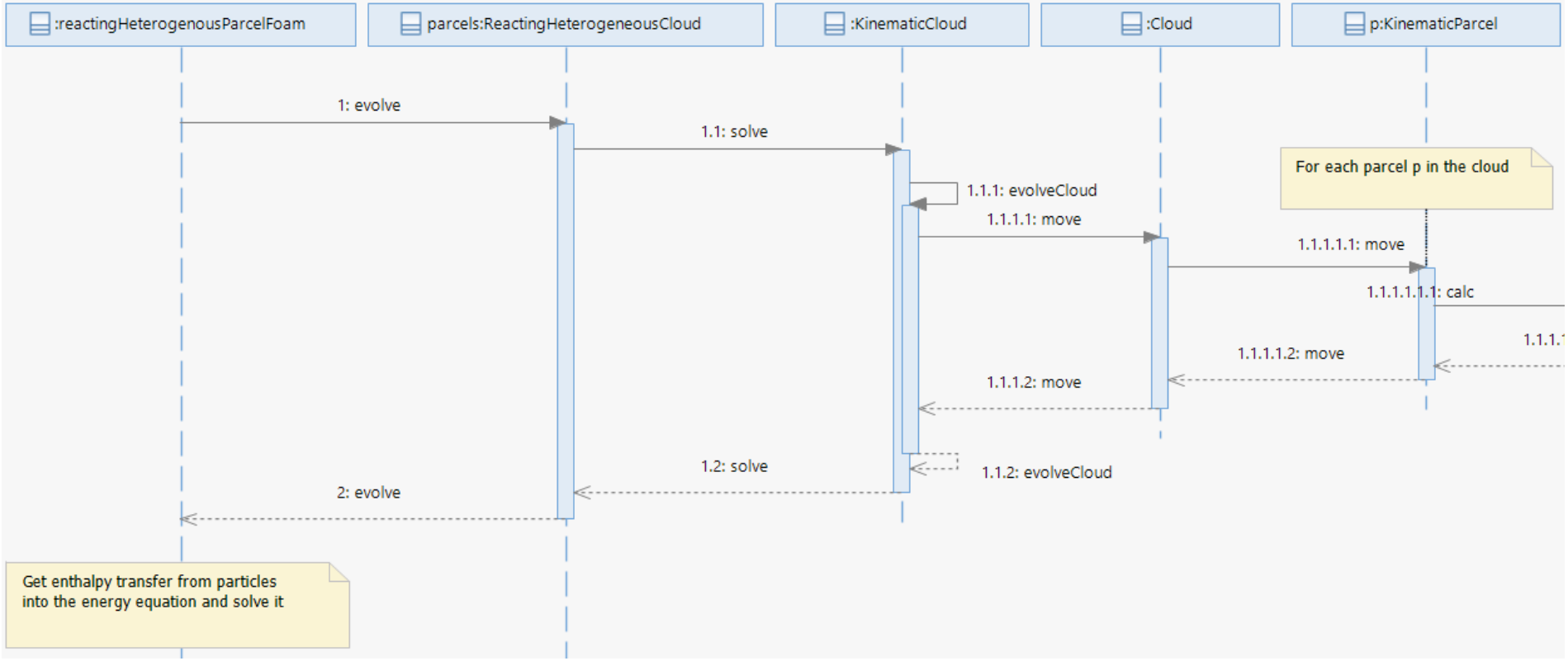
Rowe:

$$\mathrm{Nu} = f(\mathrm{Re}, \mathrm{Pr}, \epsilon_\mathrm{v})$$
$$\mathrm{Nu} = A + B\mathrm{Re}^{n}\mathrm{Pr}^{2/3}$$

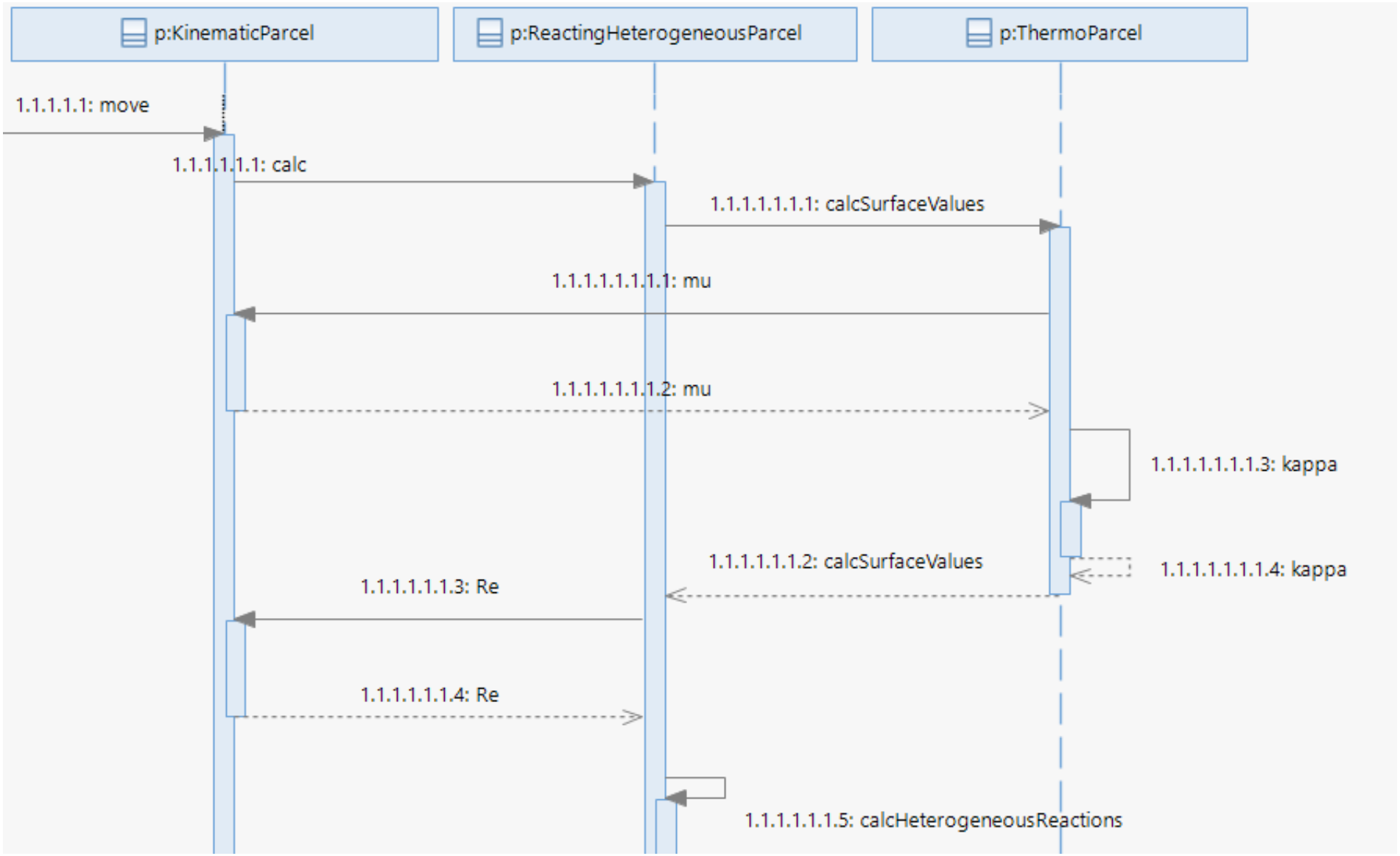
where

$$A = \frac{2}{1-(1-\epsilon_\mathrm{v})^{1/3}},$$
$$B = \frac{2}{3\epsilon_\mathrm{v}},$$
$$\frac{2-3n}{3n-1} = 4.65\mathrm{Re}^{-0.28}$$

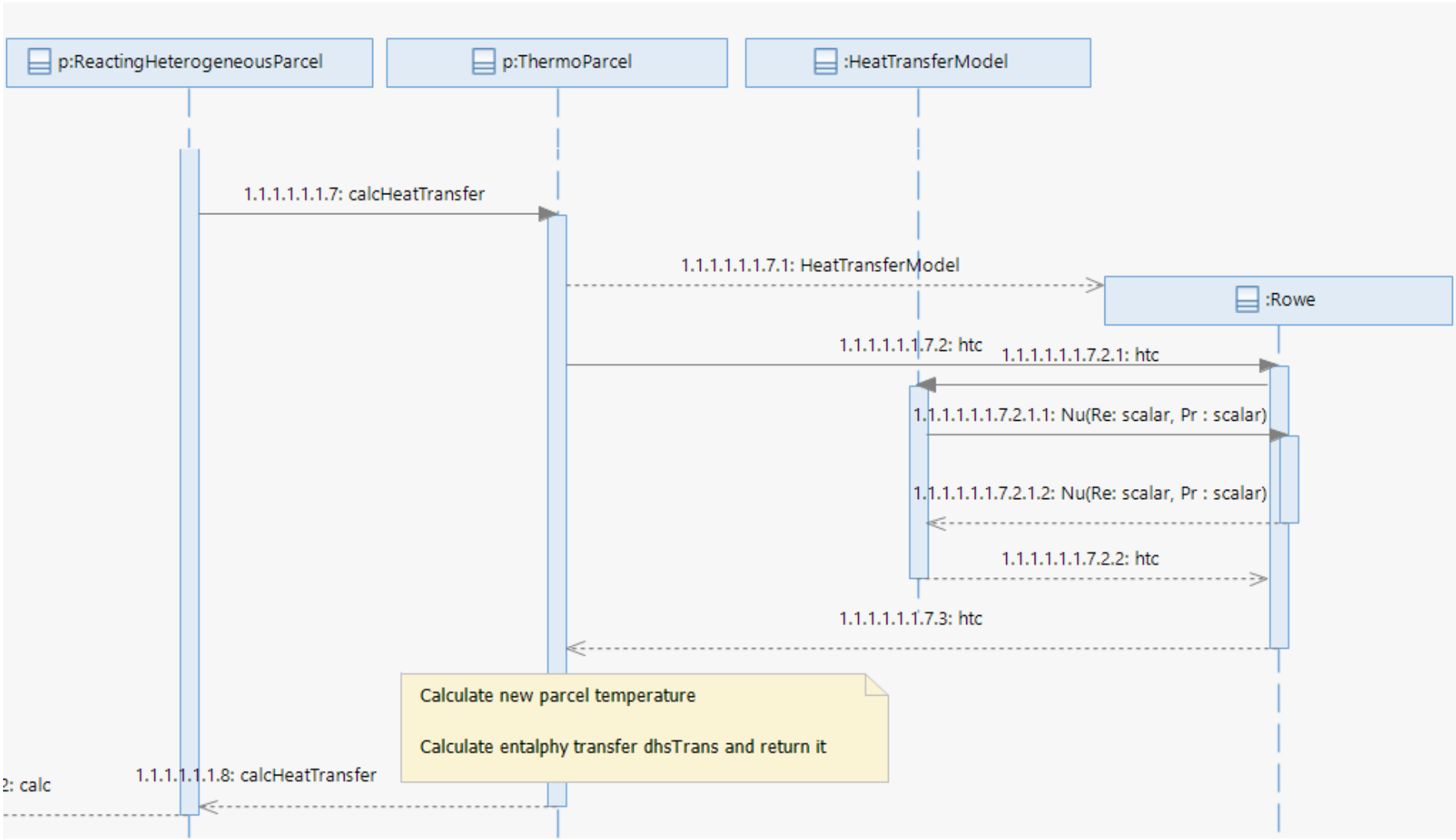
The call `parcels.evolve()` in the solver code file `reactingParcelFoam.C` starts a long chain of calls, first through some cloud classes:



and then through some parcel classes to calculate particle surface values and chemical heat of reaction:



before it is time to create the heat transfer model from a dictionary entry via the runtime selection table. Then the method `htc()` is called:



# Rowe heat transfer model

The Prandtl number is given by

$$\mathrm{Pr} = C_\mathrm{p}\mu/k_\mathrm{g},$$

and the particle surface Prandtl number is implemented as shown to the right.

ThermoParcel.C

```
void Foam::MyThermoParcel<ParcelType>::
     calcSurfaceValues
    ...
    // Surface temperature using two thirds rule
    Ts = (2.0*T + td.Tc())/3.0;
    ...
    // Assuming thermo props vary linearly with T for
      small d(T)
    const scalar TRatio = td.Tc()/Ts;

    rhos = td.rhoc()*TRatio;

    tetIndices tetIs = this->currentTetIndices();
    mus = td.muInterp().interpolate(this->coordinates
     (), tetIs)/TRatio;
    kappas = td.kappaInterp().interpolate(this->
     coordinates(), tetIs)/TRatio;

    Pr = td.Cpc()*mus/kappas;
```

# Rowe heat transfer model

The Reynolds number is given by

$$\mathrm{Re} = UL/\nu,$$

and it's particle surface value is implemented as:

ReactingHeterogeneousParcel.C

```
// Calc surface values
scalar Ts, rhos, mus, Prs, kappas;
this->calcSurfaceValues(cloud, td, T0,
    Ts, rhos, mus, Prs, kappas);
scalar Res = this->Re(rhos, U0, td.Uc(),
     d0, mus);
```

KinematicParcelI.H

```
template<class ParcelType>
inline Foam::scalar Foam::
    KinematicParcel<ParcelType>::Re
(
    const scalar rhoc,
    const vector& U,
    const vector& Uc,
    const scalar d,
    const scalar muc
)
{
    return rhoc*mag(U - Uc)*d/max(muc,
     ROOTVSMALL);
}
```

# Rowe heat transfer model

The convective heat transfer coefficient $h$ (in code `htc`) is related to the Nusselt number by

$$\mathrm{Nu} = hL/k_\mathrm{g}$$
$$\mathrm{Nu} = f(\mathrm{Re}, \mathrm{Pr})$$
$$\mathrm{Nu} = f(\mathrm{Re}, \mathrm{Pr}, \epsilon_\mathrm{v})$$

ThermoParcel.C

```
...
// Calc heat transfer coefficient
scalar htc = cloud.heatTransfer().htc(d,
      Re, Pr, kappa, NCpW);
...
```

HeatTransferModel.C

```
template<class CloudType>
Foam::scalar Foam::HeatTransferModel<
     CloudType>::htc
    (
    const scalar dp,
    const scalar Re,
    const scalar Pr,
    const scalar kappa,
    const scalar NCpW
    ) const
    {
        const scalar Nu = this->Nu(Re,
     Pr);

        scalar htc = Nu*kappa/dp;
        ...
```

# Rowe heat transfer model

The Rowe model calculates the Nusselt number from the Reynolds and the Prandtl number and bed voidage $\epsilon_v$ as

$$\mathrm{Nu} = A + B\mathrm{Re}^n\mathrm{Pr}^{2/3}$$

where

$$A = \frac{2}{1-(1-\epsilon_v)^{1/3}},$$

$$B = \frac{2}{3\epsilon_v},$$

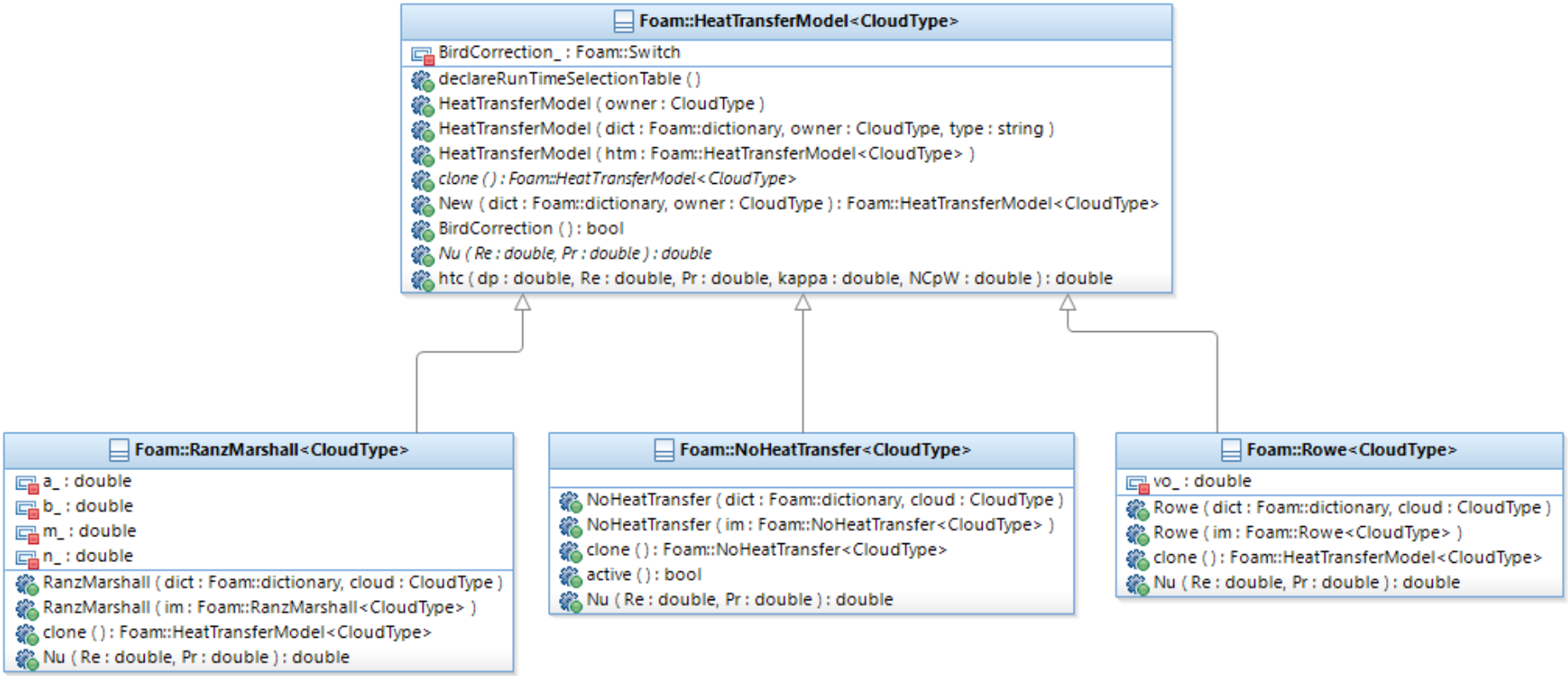
$$\frac{2-3n}{3n-1} = 4.65\mathrm{Re}^{-0.28}.$$

Rowe.C

```
template<class CloudType>
Foam::scalar Foam::Rowe<CloudType>::Nu
(
    const scalar Re,
    const scalar Pr
) const
{
    const scalar a = 2/(1 - cbrt(1 - vo_));
    const scalar b = 2/(3*vo_);
    const scalar m = 2.0/3.0;
    scalar n = 1.0/3.0;
    if (Re != 0) {
        const scalar R_hat = 4.65*pow(Re, -0.28);
        n = (2 + R_hat)/(3*R_hat + 3);
    }
    const scalar Nu = a + b*pow(Re, n)*pow(Pr, m);
```

# Rowe heat transfer model

The class diagram shows the base class `HeatTransferModel` and subclasses with methods, where the method `Nu` is abstract in the base class.



# Rowe heat transfer model

Files that need to be added or modified are shown in the file structure below:

File structure

```
└── src
    └── lagrangian
        └── intermediate
            ├── Make
            │   ├── files
            │   └── options
            ├── parcels
            │   ├── derived
            │   │   └── basicMyHeterogeneousReactingParcel
            │   │       └──
makeBasicMyHeterogeneousReactingParcelSubmodels.C
            │   └── include
            │       └── makeParcelHeatTransferModels.H
            └── submodels
                └── Thermodynamic
                    └── HeatTransferModel
                        └── Rowe
                            ├── Rowe.C
                            └── Rowe.H
```

# Rowe heat transfer model

To compile in the user directory, `files` and `options` are created and the most important parts are shown below:

`files`

```
12 $(REACTINGHETERMPPARCEL)/makeBasicMyHeterogeneousReactingParcelSubmodels.C
13
14 LIB = $(FOAM_USER_LIBBIN)/libmylagrangianIntermediate
```

Paths to include files and libraries to link to is put in the `options` file. Note that include files from the user directory needs to be in the path as shown at line 2 below:

`options`

```
1 EXE_INC = \
2     -IlnInclude \
3     -I$(LIB_SRC)/finiteVolume/lnInclude \
```

# Rowe heat transfer model

Template macros are created as shown below:

makeParcelHeatTransferModels.H

```
30 #include "Rowe.H"
31
32 // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
33
34 #define makeParcelHeatTransferModels(CloudType)                                  \
35                                                                                   \
36     makeHeatTransferModel(CloudType);                                             \
37                                                                                   \
38     makeHeatTransferModelType(Rowe, CloudType);
```

# Rowe heat transfer model

A Template for the Rowe heat transfer model is instantiated as shown below.

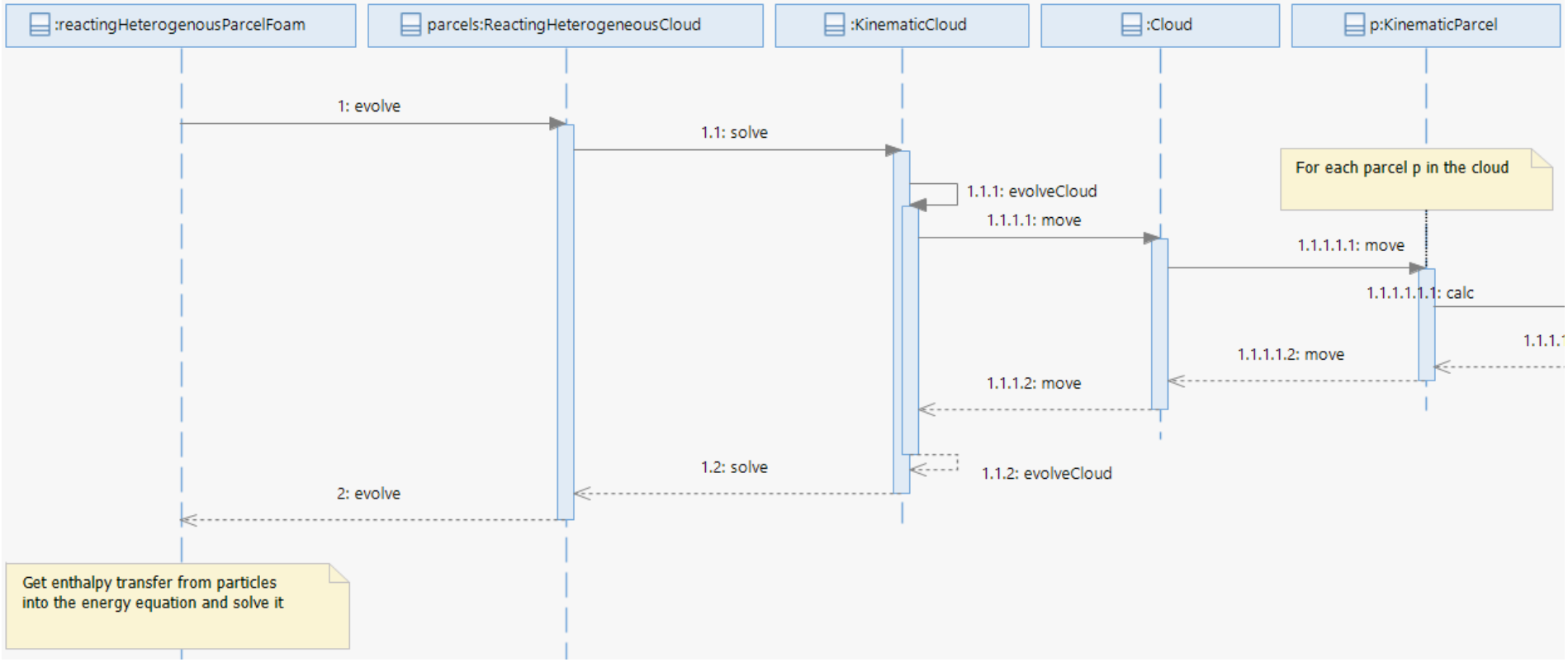
`makeBasicMyHeterogeneousReactingParcelSubmodels.C`

```
30 #include "makeParcelHeatTransferModels.H"
31
32 // Thermo sub-models
33 makeParcelHeatTransferModels(basicHeterogeneousReactingCloud);
```

This file is compiled into the custom library `libmylagrangianIntermediate.so` which can be loaded from `controlDict` for the simulation.

# Energy equation

After the call to `parcels.evolve()`, the enthalpy/internal energy `volScalarField` can be fetched from the cloud variable `parcels`.



# Energy equation

and be used in the energy equation (enthalpy equation shown).

$$\frac{\partial \rho h}{\partial t} + \nabla \cdot (\rho \mathbf{U} h) + \frac{\partial \rho K}{\partial t} + \nabla \cdot (\rho \mathbf{U} K) - \frac{\partial p}{\partial t} = -\nabla \cdot \mathbf{q} + \nabla \cdot (\boldsymbol{\tau} \cdot \mathbf{U}) + \rho r + \rho \mathbf{g} \cdot \mathbf{U}$$

EEqn.H

```
fvm::ddt(rho, he) + mvConvection->fvmDiv(phi, he)
  + fvc::ddt(rho, K) + fvc::div(phi, K)
  + (
        he.name() == "e"
      ? fvc::div
        (
            fvc::absolute(phi/fvc::interpolate(rho), U
    ),
            p,
            "div(phiv,p)"
        )
      : -dpdt
    )
  - fvm::laplacian(turbulence->alphaEff(), he)
```

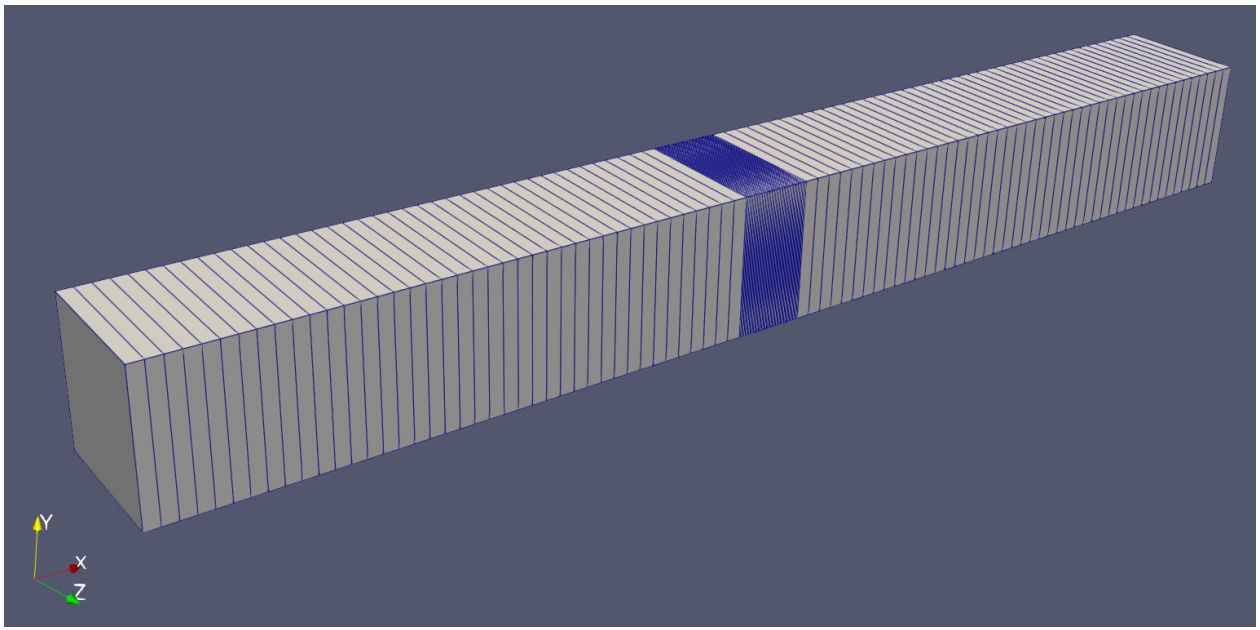
EEqn.H

```
 ==
    rho*(U&g)
  + parcels.Sh(he)
  + surfaceFilm.Sh()
  + radiation->Sh(thermo,
     he)
  + Qdot
  + fvOptions(rho, he)
-
```

# Case setup

The `rectangularDuct` tutorial is copied and modified.

- From 90×10×10 m to 9×1×1 m (L×B×H)
- 40000 cells in 3D → 135 cells in 1D
- Fixed bed with particles in finer mesh at X between 4 - 4.5m
- Transient (as before)
- No turbulence
- 100% hematite first, later with 100% magnetite



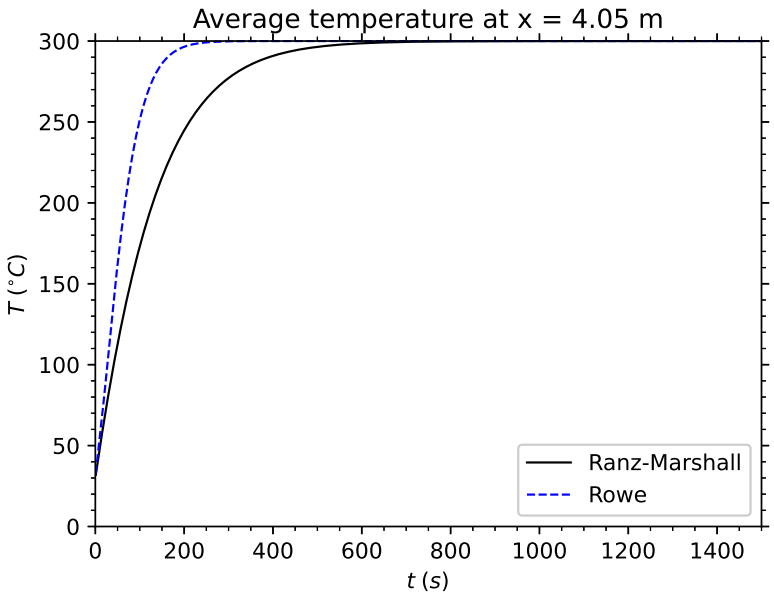
# Case setup

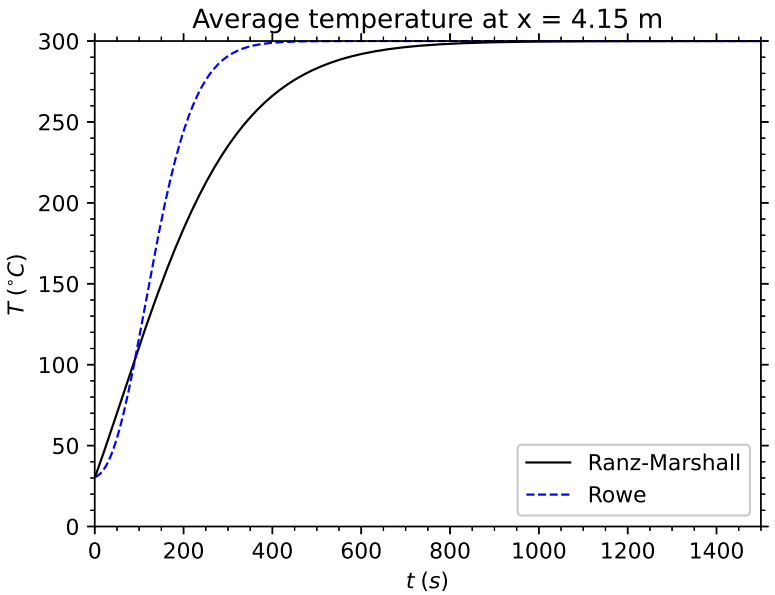
Table: Particle injection properties

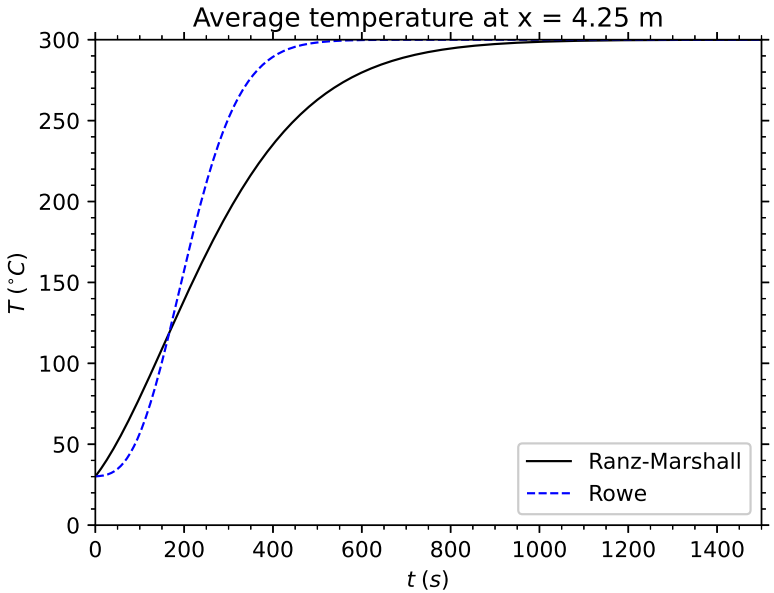
| Property | Value | Description |
|---|---|---|
| `type` | `manualInjection` | manual injection |
| `parcelBasisType` | `mass` | mass based |
| `massTotal` | 1080 | total mass $V_{\mathrm{bed}}(1-\epsilon_{\mathrm{bed}})\rho_{\mathrm{p}} = 0.5 * (1-0.4) * 3600 = 1.08\mathrm{e}3$ |
| `positionsFile` | `reactingCloud1Positions` | parcel positions |

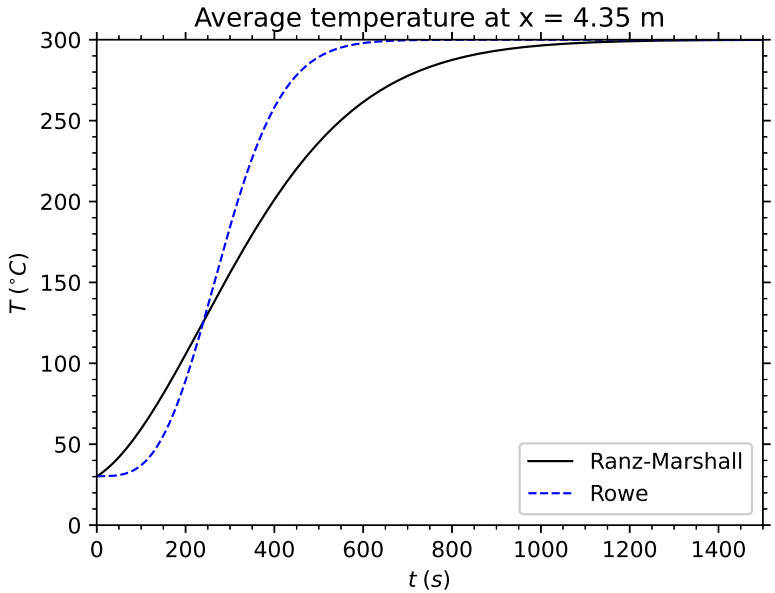
# Conclusions

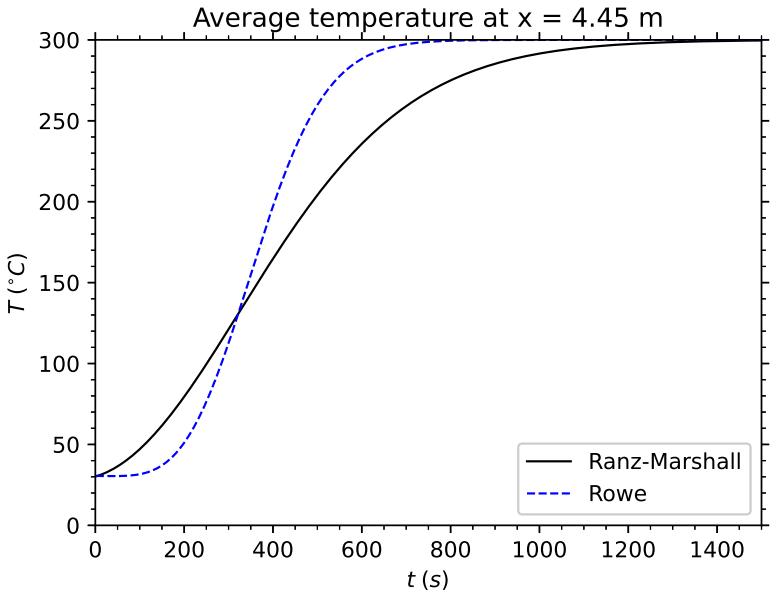
The Rowe heat transfer model gives a much faster heat transfer to a packet porous bed than the Ranz-Marshall model and corresponds better to experimental data.

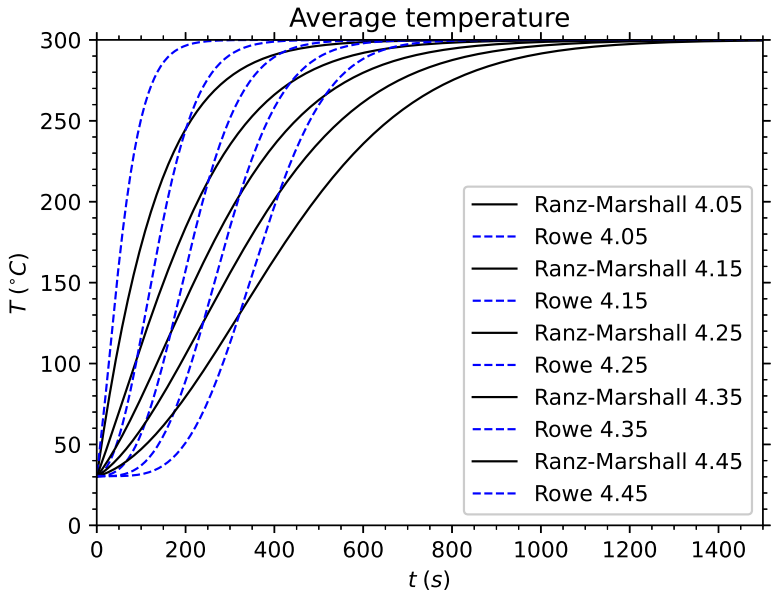



Average temperature at x = 4.25 m
T (°C)
t (s)
Ranz-Marshall
Rowe

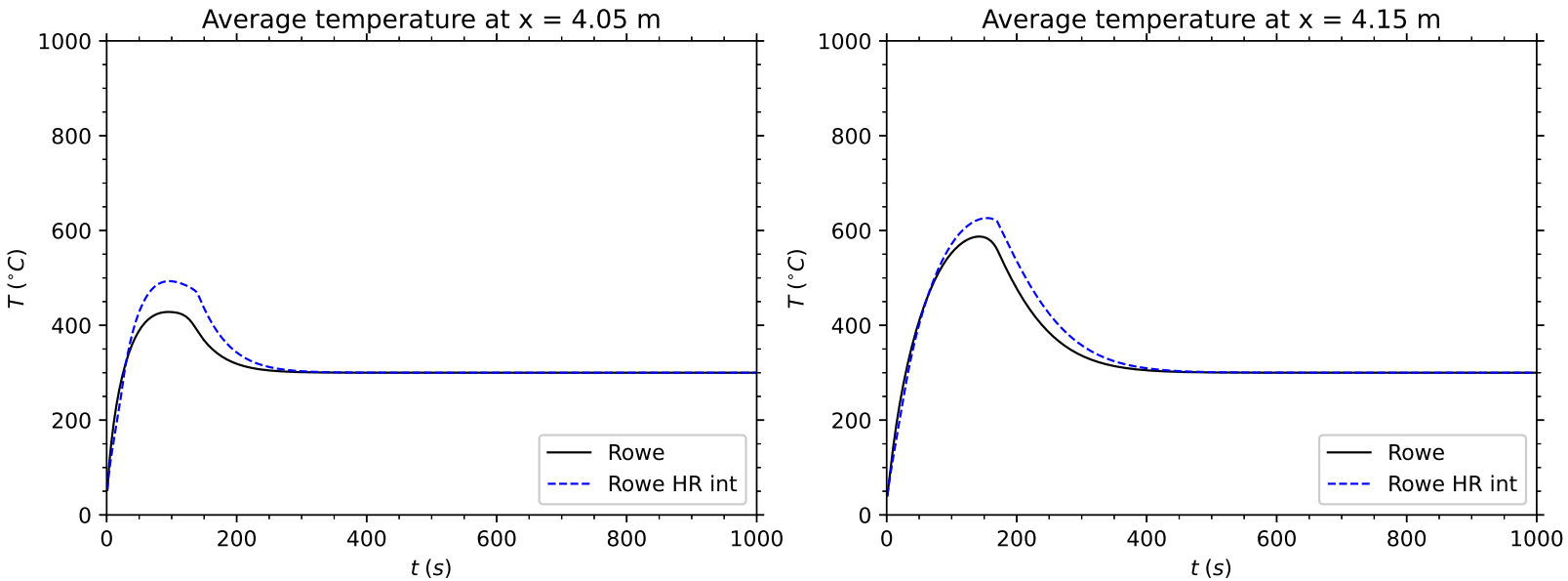

Average temperature at x = 4.35 m
T (°C)
t (s)
Ranz-Marshall
Rowe

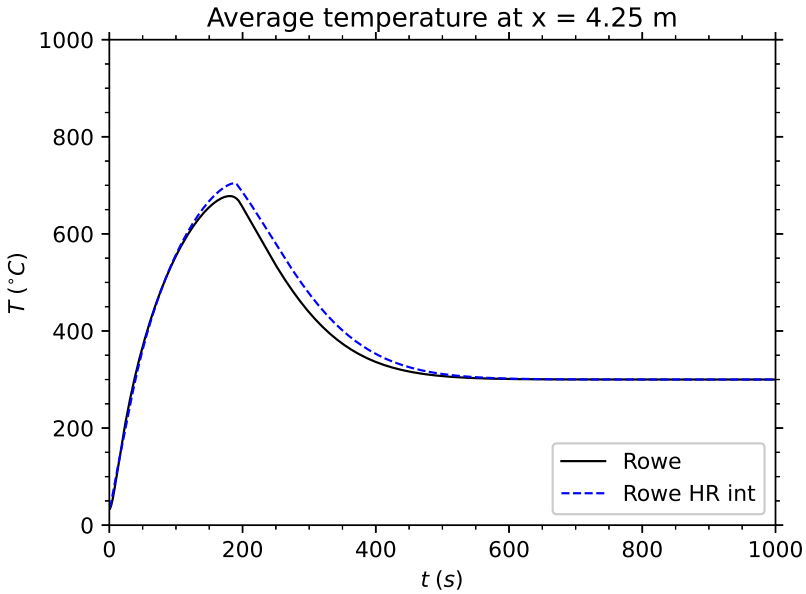

Average temperature at x = 4.45 m
T (°C)
t (s)
Ranz-Marshall
Rowe

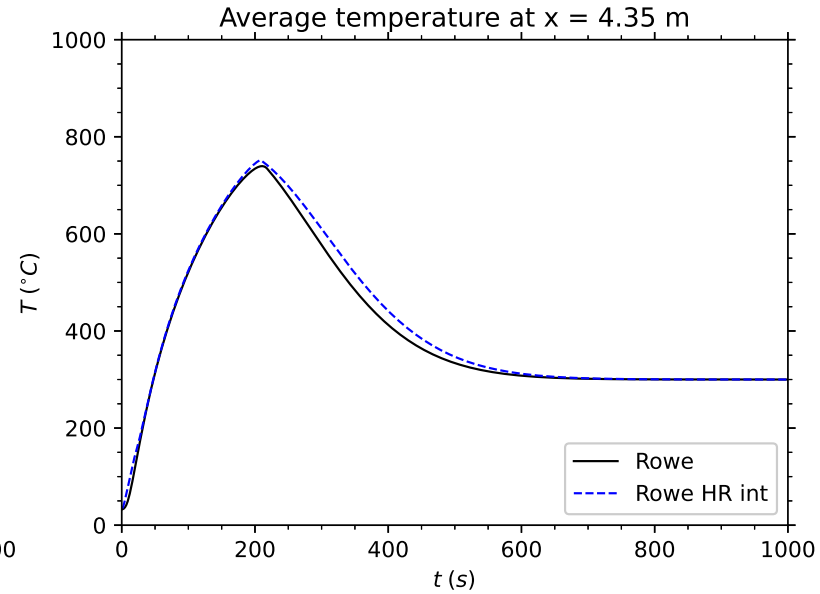

Average temperature
T (°C)
t (s)
Ranz-Marshall 4.05
Rowe 4.05
Ranz-Marshall 4.15
Rowe 4.15
Ranz-Marshall 4.25
Rowe 4.25
Ranz-Marshall 4.35
Rowe 4.35
Ranz-Marshall 4.45
Rowe 4.45

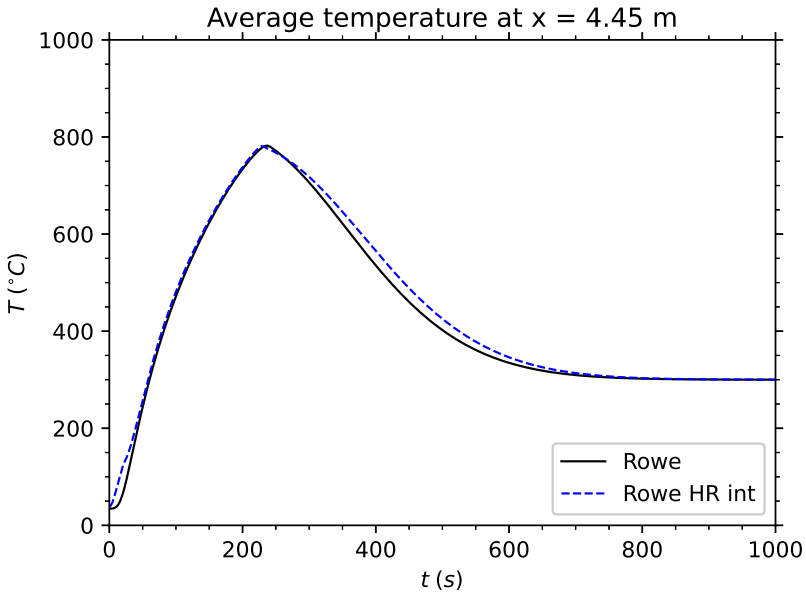
# Conclusions

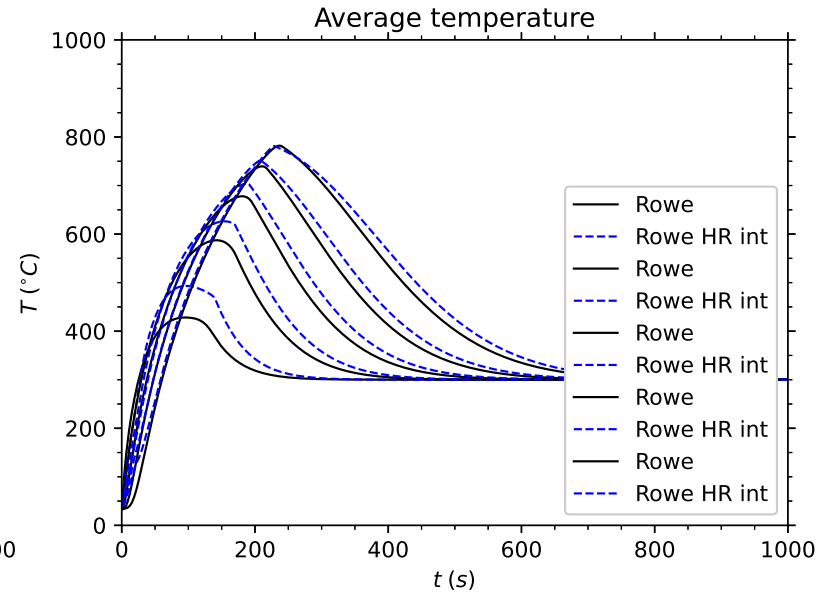
The effect of intrinsic heat in the particles from heat of reaction, compared to directly heating up the gas (external heat) is shown below:




Average temperature at x = 4.25 m
T (°C)
t (s)
Rowe
Rowe HR int


Average temperature at x = 4.35 m
T (°C)
t (s)
Rowe
Rowe HR int


Average temperature at x = 4.45 m
T (°C)
t (s)
Rowe
Rowe HR int


Average temperature
T (°C)
t (s)
Rowe
Rowe HR int

# That's it

Thank you for your attention!