Semi-Lagrangian Advection in Elmer

Applications

Cyrille Mosbeux and P. Raback

In collaboration with

O. Gagliardini & N.Jourdain, A. Gilbert & F.Gillet-Chaulet

8th November 2023 - EGU
ElmerIce User's Meeting

Principle of the semi-Lagrangian Method

- Any quantity can (e.g. damage) be advected in an Eulerian Framework following:

Source term : $\chi(\sigma, D)$

$$\frac{\partial D}{\partial t} + \boldsymbol{u}\,\nabla D = f(\chi)$$

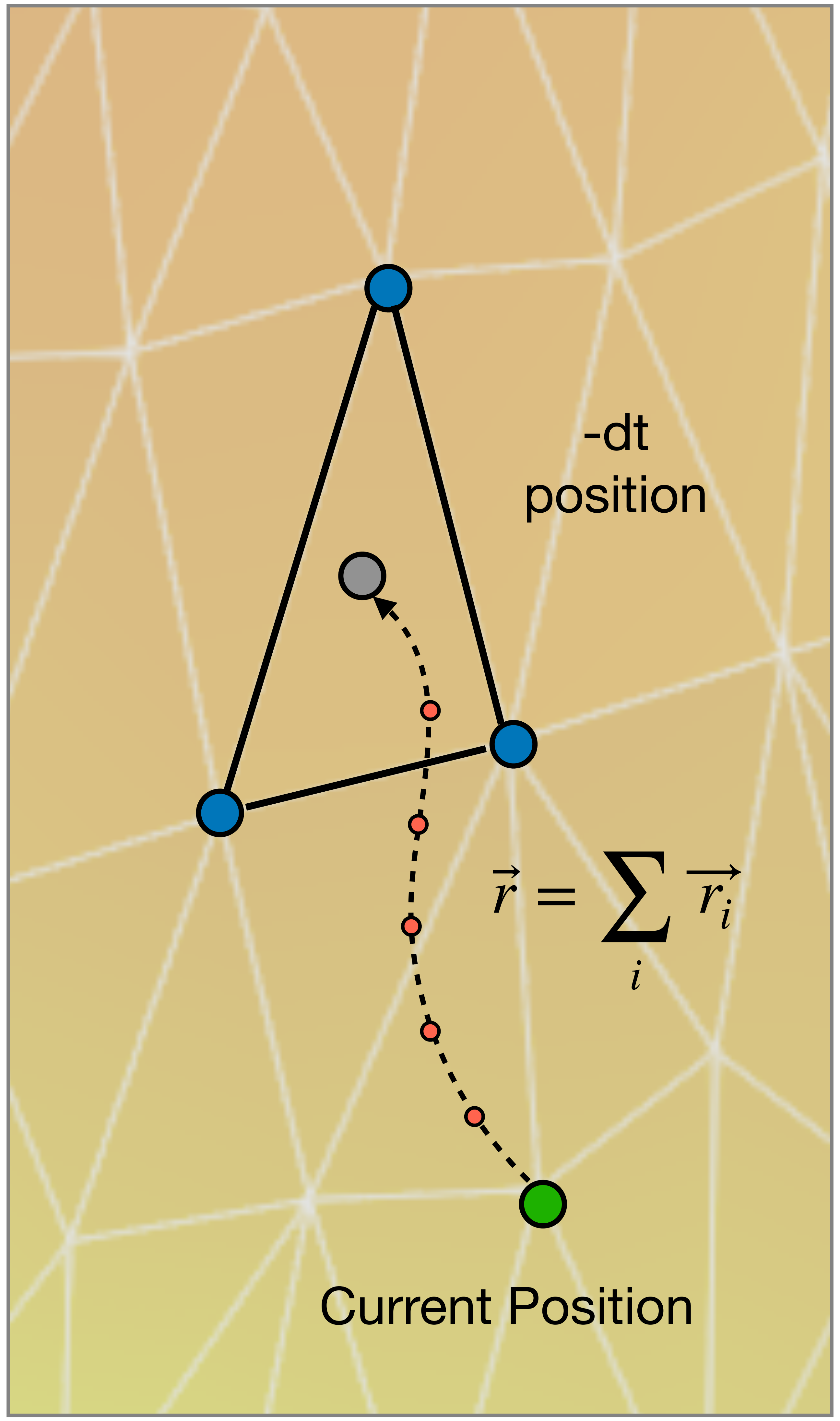
But it comes with substantial numerical diffusion…

- Another solution is to use a Semi-Lagrangian Framework, following a particle (damage defined at a node) trajectory ($\vec{r}$) over time

$$\vec{r} = \vec{r_0} + \int_0^{-\delta t} \boldsymbol{u}\, dt$$

Track the transport of the particle

```
Max Timestep Intervals = Integer 10
```



Principle of the semi-Lagrangian Method

- Any quantity can (e.g. damage)be advected in an Eulerian Framework following:

Source term : $\chi(\sigma, D)$

$$\frac{\partial D}{\partial t} + \boldsymbol{u}\nabla D = f(\chi)$$

But it comes with substantial numerical diffusion…

- Another solution is to use a **Semi-Lagrangian Framework**, following a particle (damage defined at a node) trajectory ($\vec{r}$) over time

$$\vec{r} = \vec{r_0} + \int_0^{-\delta t} \boldsymbol{u}\, dt$$

Track the transport of the particle

Here the source term can be expressed as

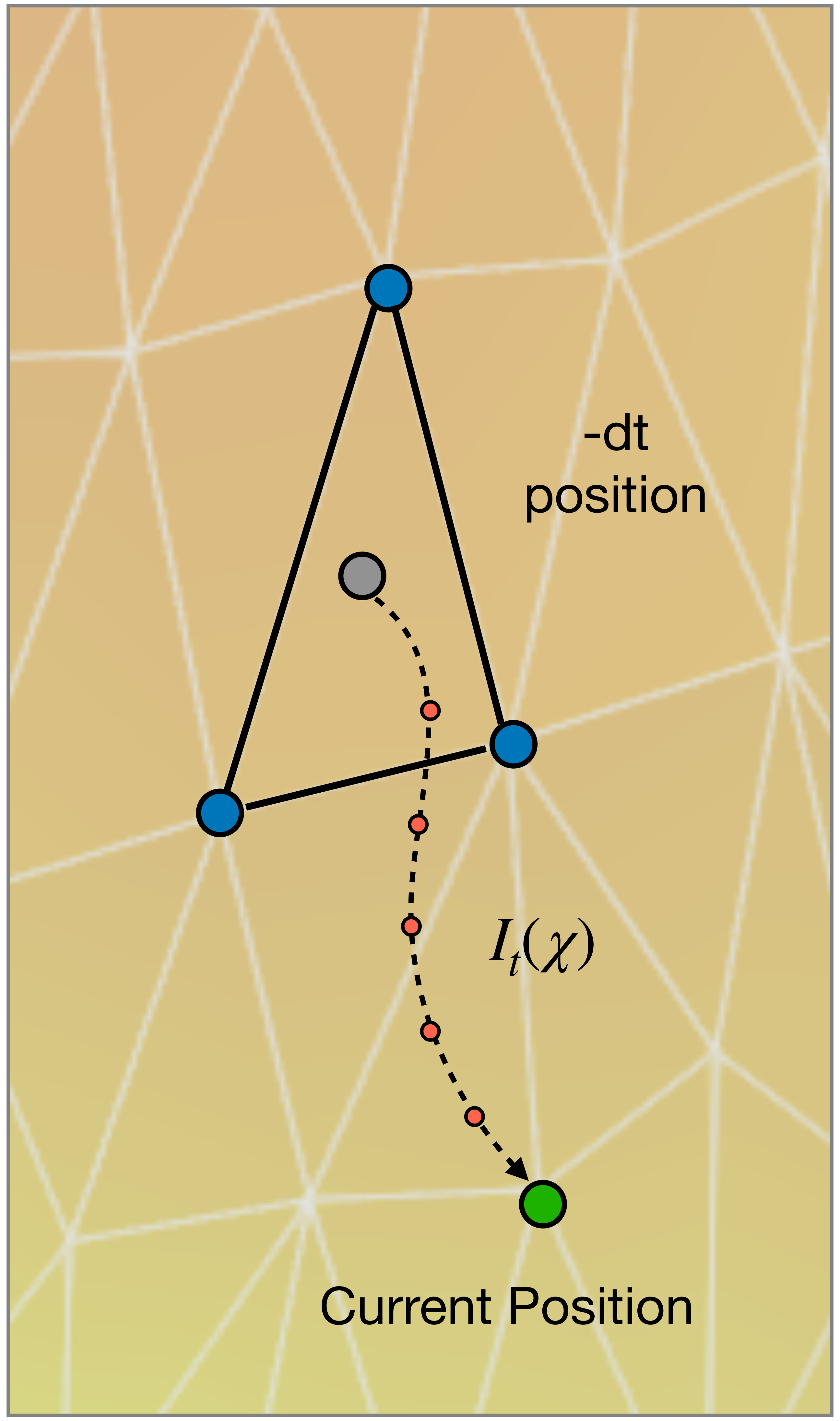
$$I_t(\chi) = \int_0^{-\delta t} f(\chi)\, dt$$

To track the evolution of the particle given $f(\chi)$

```
Particle time reverse = Logical True
```

The evolution of damage over time and space can finally be calculated following

$$D = D(\vec{r_0}, t) + I_t(\chi)$$



Principle of the semi-Lagrangian Method

- Any quantity can (e.g. damage)be advected in an Eulerian Framework following:

Source term : $\chi(\sigma, D)$

$$\frac{\partial D}{\partial t} + \boldsymbol{u} \nabla D = f(\chi)$$

But it comes with substantial numerical diffusion…

- Another solution is to use a Semi-Lagrangian Framework, following a particle (damage defined at a node) trajectory ($\vec{r}$) over time

$$\vec{r} = \vec{r_0} + \int_0^{-\delta t} \boldsymbol{u}\, dt$$

Track the transport of the particle

Here the source term can be expressed as

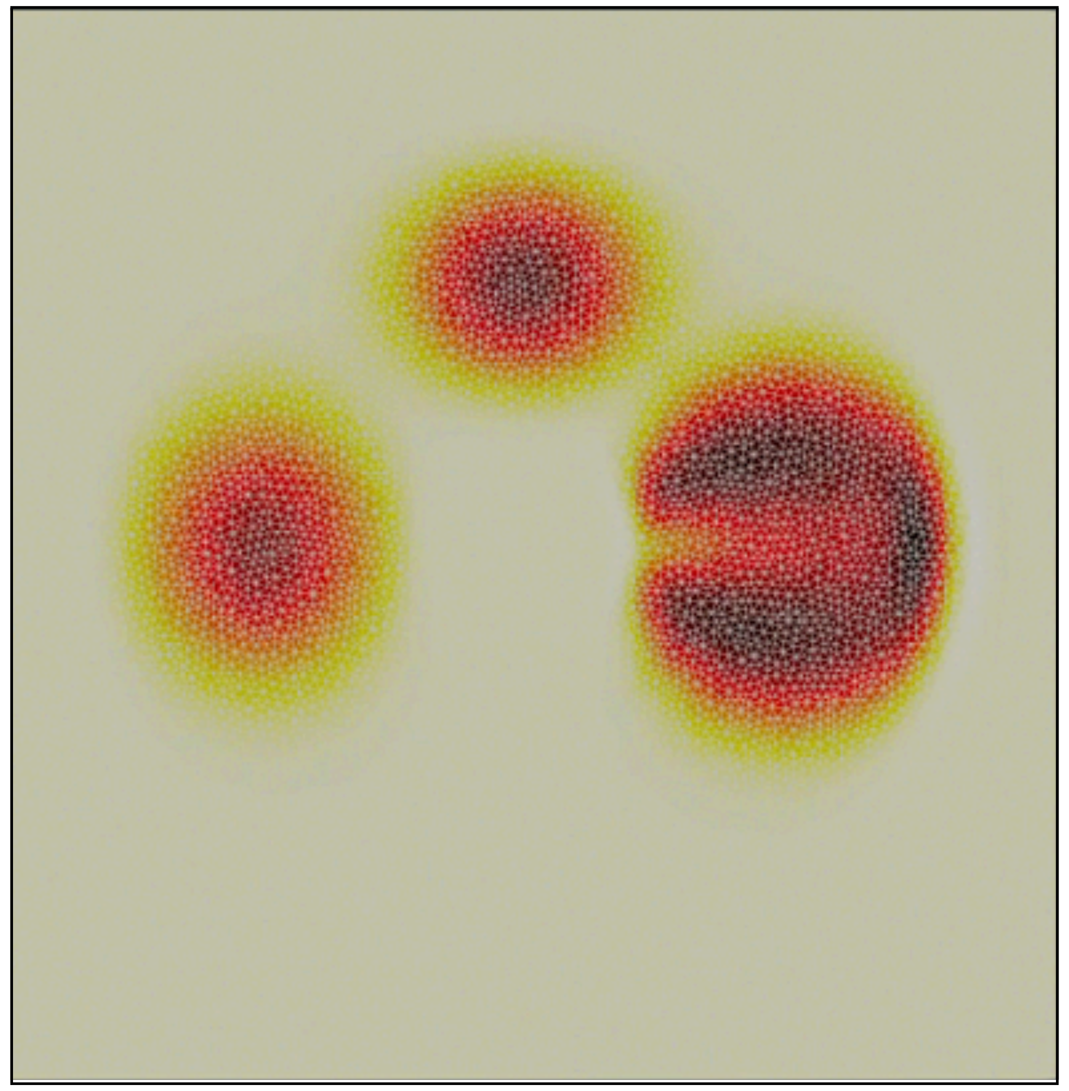
$$I_t(\chi) = \int_0^{-\delta t} f(\chi)\, dt$$

To track the evolution of the particle given $f(\chi)$

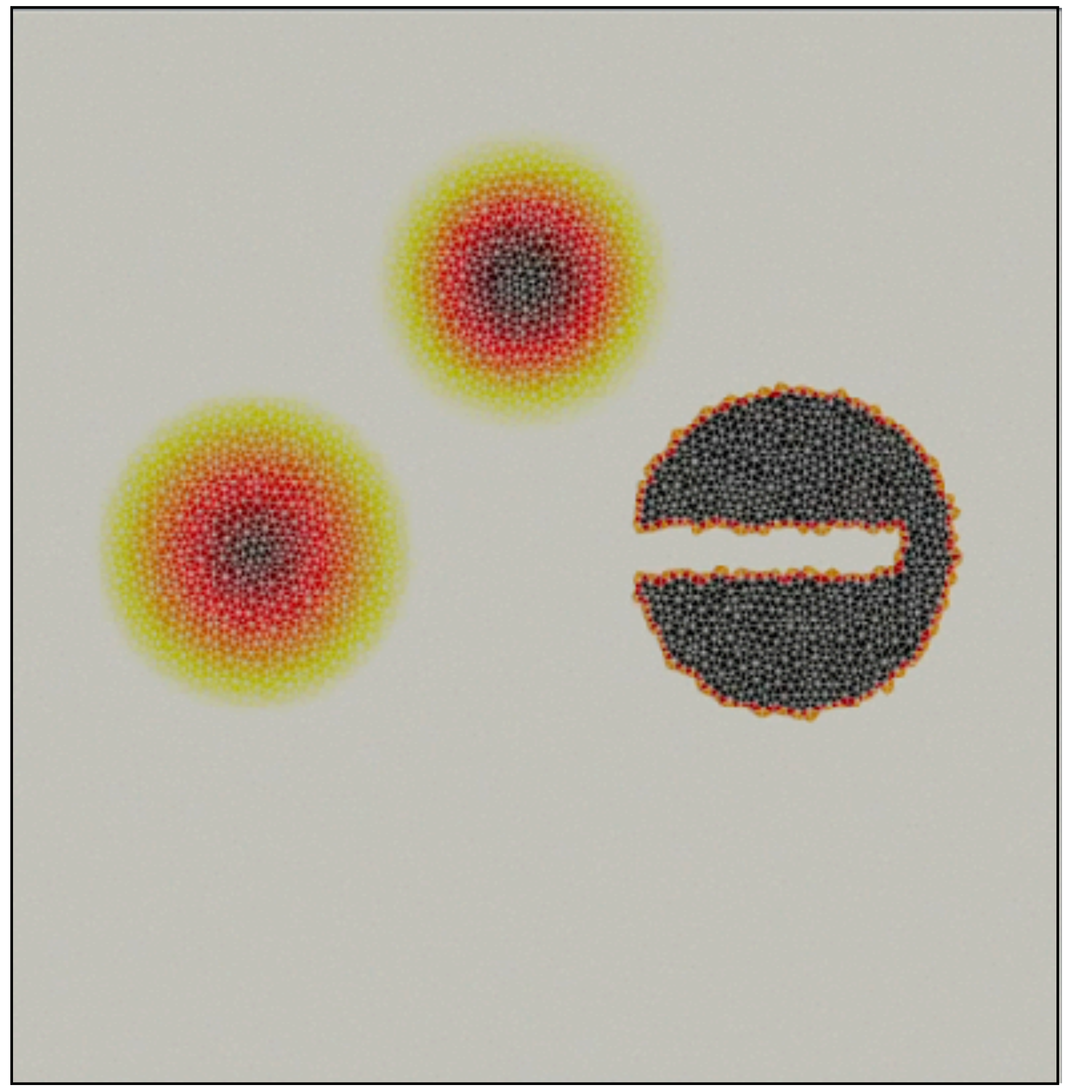
The evolution of damage over time and space can finally be calculated following

$$D = D(\vec{r_0}, t) + I_t(\chi)$$

Advection - Diffusion

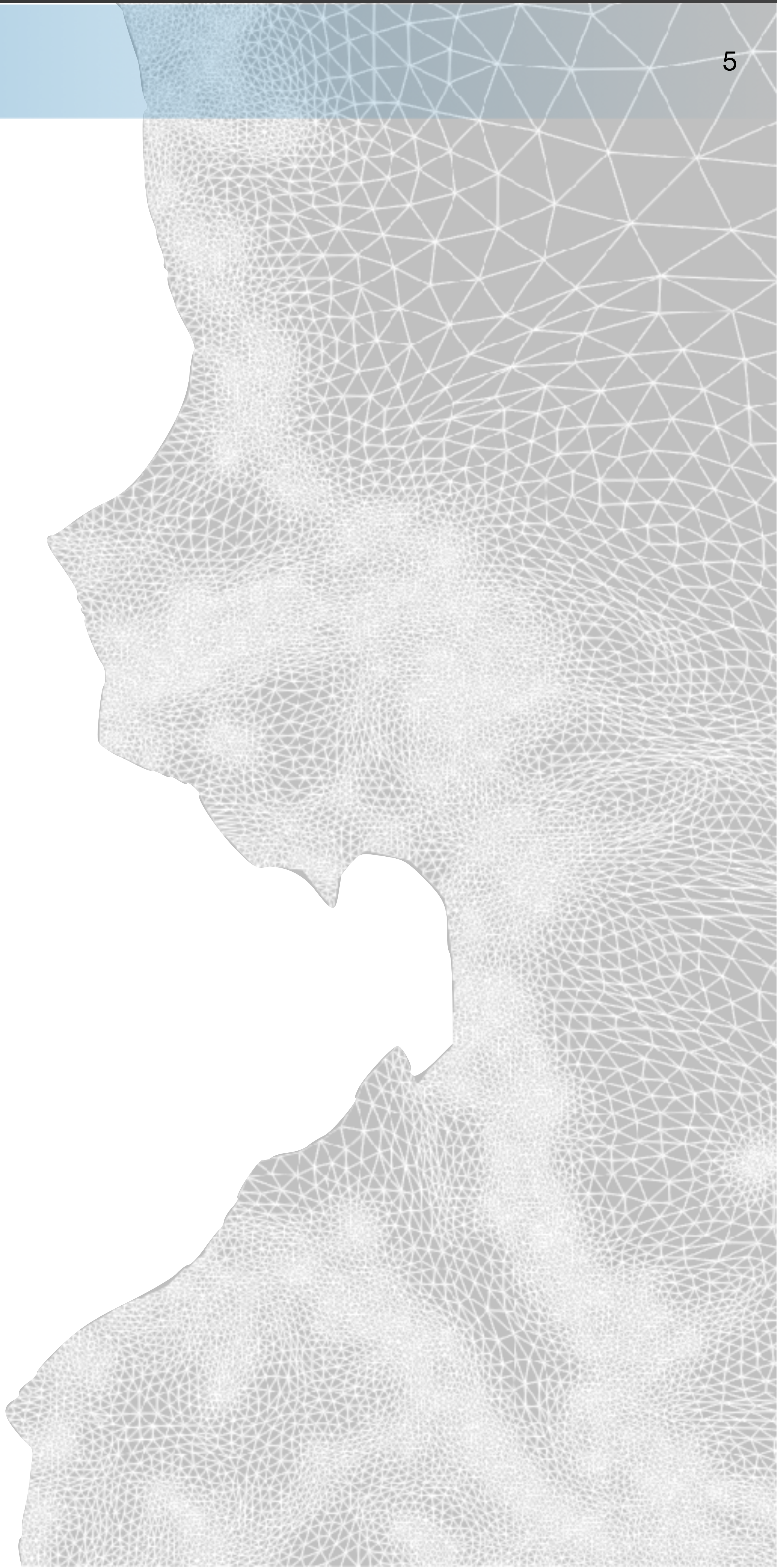


Semi-Lagrangian



The Semi-Lagrangian Solver in Elmer


```
Solver 2
  Equation = ParticleAdvector
  Procedure = "ParticleAdvector" "ParticleAdvector"

!Relative Mesh Level = Integer -1

! Initialize particles at center of elements (as opposed to nodes)
  Advect Elemental = Logical False

  Reinitialize Particles = Logical True
  !Particle Accurate At Face = Logical False

! Timestepping strategy
  Simulation Timestep Sizes = Logical True

  Particle Dt Constant = Logical False
  Max Timestep Intervals = Integer 10 !Accuracy largely decrease for small values

! Time in average 4 steps in each element
  Timestep Unisotropic Courant Number = Real 0.25
  Max Timestep Size = Real 1.0e3

! Give up integration if particles are tool old
  Max Integration Time = Real 1.0e4

! Integration forward in time
  Runge Kutta = Logical False

  Velocity Gradient Correction = Logical True
  Velocity Variable Name = String "Velocity"

! Keywords for sourcing the particle
  Source Time Correction = Logical True
  Particle time reverse = Logical True

! Show some info in the end
  Particle Info = Logical True
  Particle Time = Logical True

! The internal variables for this solver
  Variable 1 = String "Hpart"
  Variable 2 = String "Particle distance"
  Variable 3 = String "Particle time"
  Variable 4 = String "Particle time integral"

! Absolute displacement when going back-and-forth in time.
! For exact this should be zero (computation bugged in parallel)
  Variable 5 = String "Particle disp"

 ! Distance and velocity of the particle can be computed
  Variable 5 = String "Particle distance integral"
  Variable 6 = String "Particle velocity abs"

  Result Variable 1 = String "Hadv"
  Result Variable 4 = String "Damage"

 Particle Integral Dummy Argument = Logical True
End
```



The Semi-Lagrangian Solver in Elmer


```
Solver 2
  Equation = ParticleAdvector
  Procedure = "ParticleAdvector" "ParticleAdvector"

!Relative Mesh Level = Integer -1

! Initialize particles at center of elements (as opposed to nodes)
  Advect Elemental = Logical False

  Reinit
  !Parti

! Timeste
  Simula

  Partic
  Max Ti

! Time i
  Timeste
  Max Ti

! Give u
  Max In

! Integr
  Runge K

  Velocity Gradient Correction = Logical True
  Velocity Variable Name = String "Velocity"

! Keywords for sourcing the particle
  Source Time Correction = Logical True
  Particle time reverse = Logical True

! Show some info in the end
  Particle Info = Logical True
  Particle Time = Logical True

! The internal variables for this solver
  Variable 1 = String "Hpart"
  Variable 2 = String "Particle distance"
  Variable 3 = String "Particle time"
  Variable 4 = String "Particle time integral"

! Absolute displacement when going back-and-forth in time.
! For exact this should be zero (computation bugged in parallel)
  Variable 5 = String "Particle disp"

 ! Distance and velocity of the particle can be computed
  Variable 5 = String "Particle distance integral"
  Variable 6 = String "Particle velocity abs"

  Result Variable 1 = String "Hadv"
  Result Variable 4 = String "Damage"

 Particle Integral Dummy Argument = Logical True
End
```

```
! Initialize particles at center of elements (as opposed to nodes)
  Advect Elemental = Logical False

  Reinitialize Particles = Logical True
  !Particle Accurate At Face = Logical False

! Timestepping strategy
  Simulation Timestep Sizes = Logical True

  Particle Dt Constant = Logical False
  Max Timestep Intervals = Integer 10 !Accuracy largely decrease for small values
```

Particles can be defined at:

1. Node
2. Element
3. Integration point
4. Discontinuous Galerkin nodes

For transient simulations, particles should be reinitialised every timestep… longer the timestep lesser the interpolation error

Timestep (external) can be fixed to the simulation timestep

More internal timestep intervals give an overall better results but is also more expensive (5-10 has been a good number for test cases)

The Semi-Lagrangian Solver in Elmer


```
Solver 2
  Equation = ParticleAdvector
  Procedure = "ParticleAdvector" "ParticleAdvector"

!Relative Mesh Level = Integer -1

! Initialize particles at center of elements (as opposed to nodes)
  Advect Elemental = Logical False

  Reinitialize Particles = Logical True
  !Particle Accurate At Face = Logical False

! Timestepping strategy
  Simulation Timestep Sizes = Logical True

  Particle Dt Constant = Logical False
  Max Tim

! Time in
  Timestep Unisotropic Courant Number = Real 0.25
  Max Timestep Size = Real 1.0e3

! Give up integration if particles are tool old
  Max Integration Time = Real 1.0e4

! Integration forward in time
  Runge Kutta = Logical False

  Velocity Gradient Correction = Logical True
  Velocity Variable Name = String "Velocity"

! Keywords for sourcing the particle
  Source Time Correction = Logical True
  Particle time reverse = Logical True

! Show some info in the end
  Particle Info = Logical True
  Particle Time = Logical True

! The internal variables for this solver
  Variable 1 = String "Hpart"
  Variable 2 = String "Particle distance"
  Variable 3 = String "Particle time"
  Variable 4 = String "Particle time integral"

! Absolute displacement when going back-and-forth in time.
! For exact this should be zero (computation bugged in parallel)
  Variable 5 = String "Particle disp"

 ! Distance and velocity of the particle can be computed
  Variable 5 = String "Particle distance integral"
  Variable 6 = String "Particle velocity abs"

  Result Variable 1 = String "Hadv"
  Result Variable 4 = String "Damage"

 Particle Integral Dummy Argument = Logical True
End
```

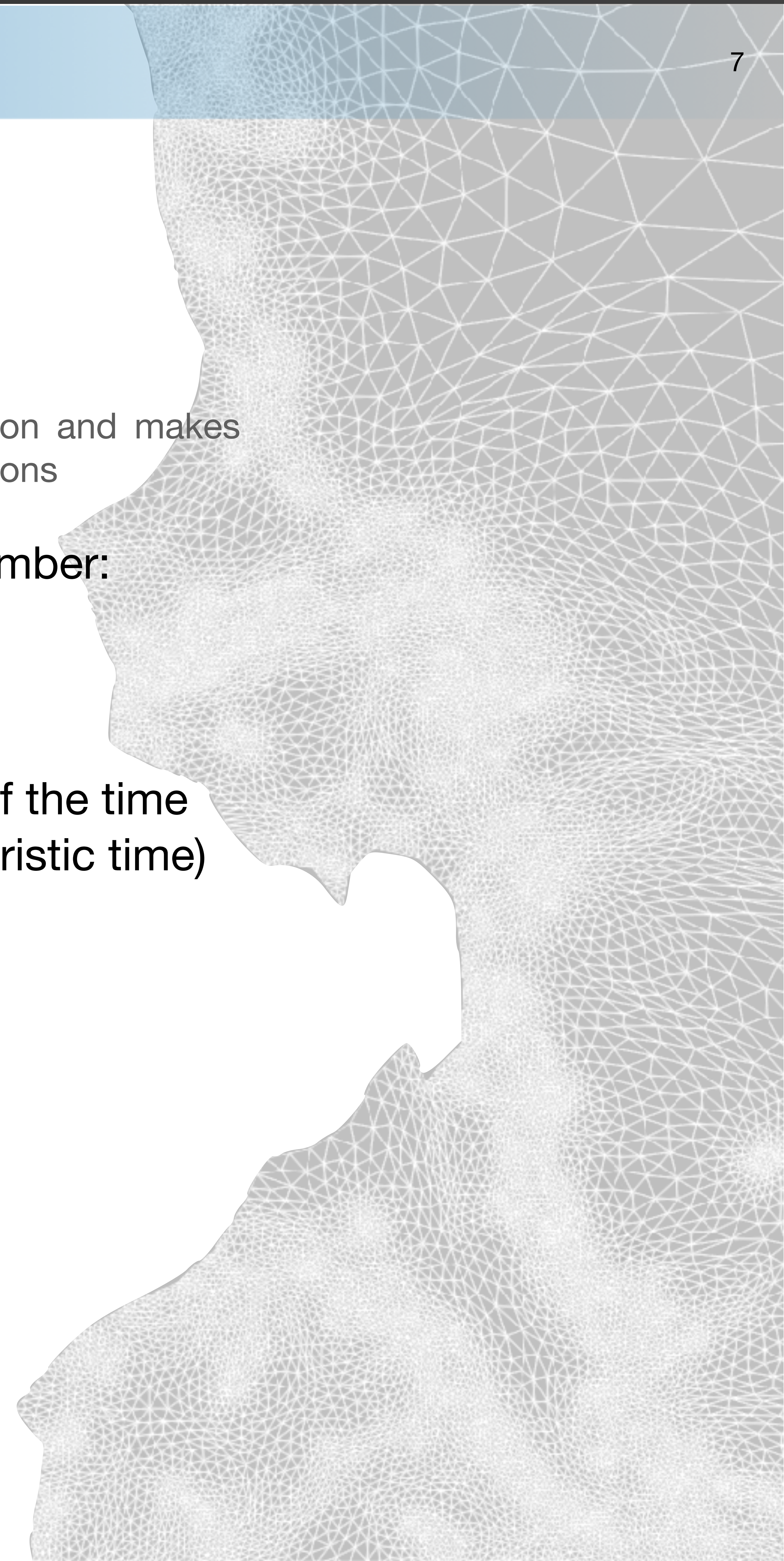
```
! Time in average 4 steps in each element
  Timestep Unisotropic Courant Number = Real 0.25
```

computes it for x,y,z direction and makes different time different directions

Timestep can be fixed to unisotropic current number:

$$c = \frac{u\Delta t}{\Delta x} \Rightarrow \Delta t = c\Delta x/u$$

The internal timestep is defined as a portion c of the time needed to cruise over the element (i.e. characteristic time)



The Semi-Lagrangian Solver in Elmer

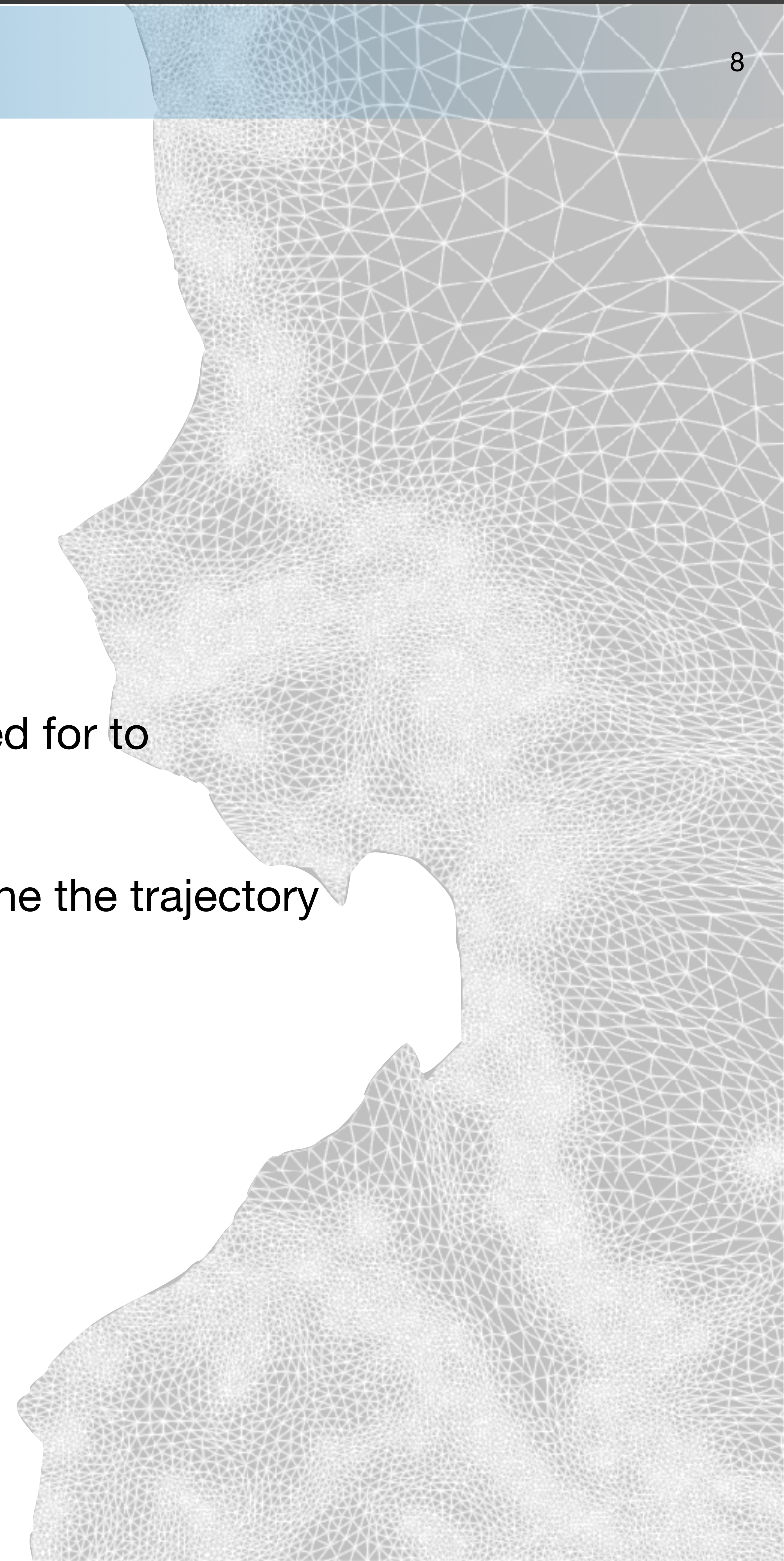

```
Solver 2
  Equation = ParticleAdvector
  Procedure = "ParticleAdvector" "ParticleAdvector"

!Relative Mesh Level = Integer -1

! Initialize particles at center of elements (as opposed to nodes)
  Advect Elemental = Logical False

  Reinitialize Particles = Logical True
  !Particle Accurate At Face = Logical False

! Timestepping strategy
  Simulation Timestep Sizes = Logical True

  Particle Dt Constant = Logical False
  Max Timestep Intervals = Integer 10 !Accuracy largely decrease for small values

! Time in average 4 steps in each element
  Timestep Unisotropic Courant Number = Real 0.25
  Max Timestep Size = Real 1.0e3

! Give up integration if particles are tool old
  Max Integration Time = Real 1.0e4
```

```
! Integration forward in time
  Runge Kutta = Logical False

  Velocity Gradient Correction = Logical True
  Velocity Variable Name = String "Velocity"
```

```
! Show some info in the end
  Particle Info = Logical True
  Particle Time = Logical True

! The internal variables for this solver
  Variable 1 = String "Hpart"
  Variable 2 = String "Particle distance"
  Variable 3 = String "Particle time"
  Variable 4 = String "Particle time integral"

! Absolute displacement when going back-and-forth in time.
! For exact this should be zero (computation bugged in parallel)
  Variable 5 = String "Particle disp"

 ! Distance and velocity of the particle can be computed
  Variable 5 = String "Particle distance integral"
  Variable 6 = String "Particle velocity abs"

  Result Variable 1 = String "Hadv"
  Result Variable 4 = String "Damage"

 Particle Integral Dummy Argument = Logical True
End
```

The **gradient of the velocity** can be accounted for to improve the internal particle trajectory

The velocity variable (or flow solution) will define the trajectory of the particles



The Semi-Lagrangian Solver in Elmer

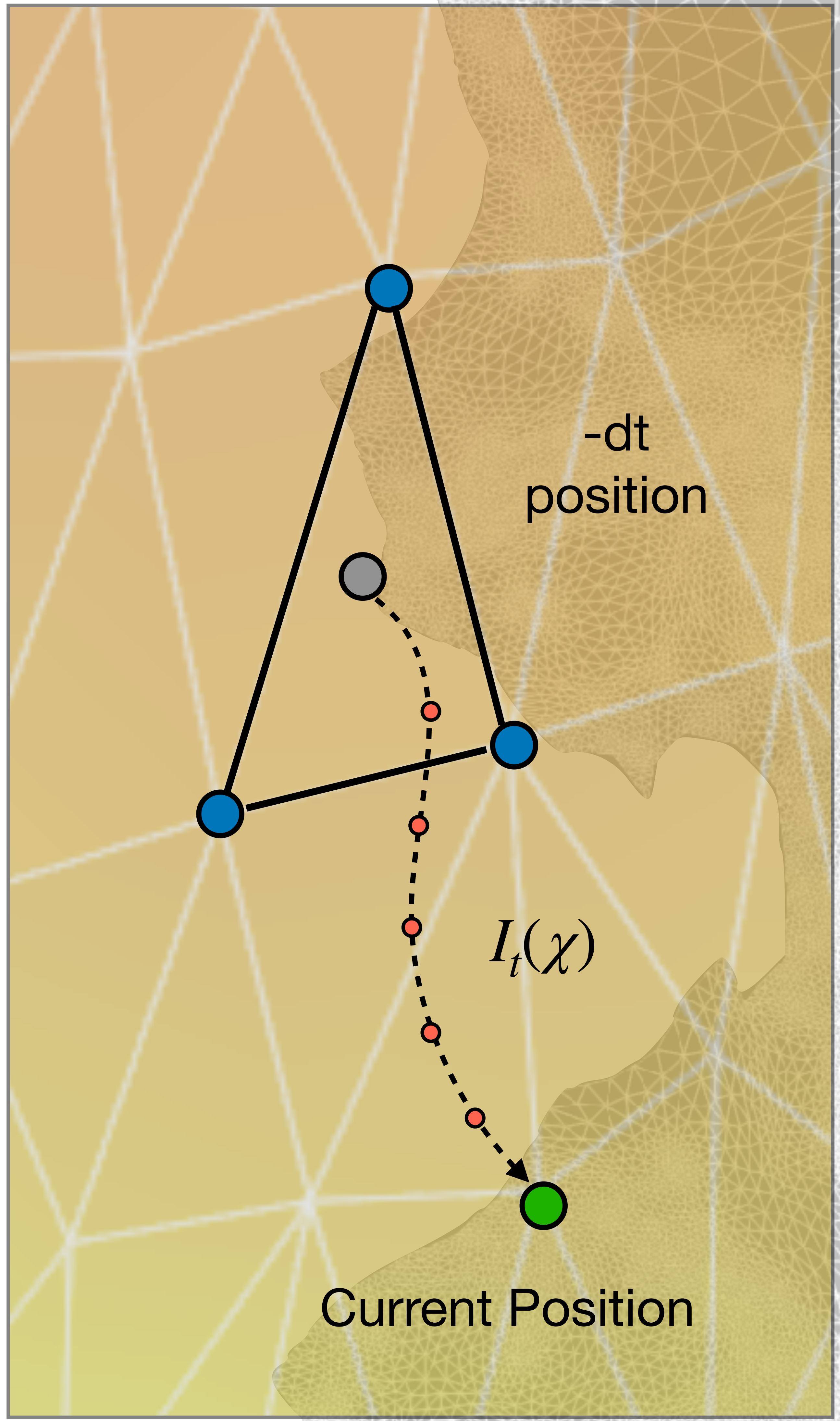

```
Solver 2
  Equation = ParticleAdvector
  Procedure = "ParticleAdvector" "ParticleAdvector"

!Relative Mesh Level = Integer -1

! Initialize particles at center of elements (as opposed to nodes)
  Advect Elemental = Logical False

  Reinitialize Particles = Logical True
  !Particle Accurate At Face = Logical False

! Timestepping strategy
  Simulation Timestep Sizes = Logical True

  Particle Dt Constant = Logical False
  Max Timestep Intervals = Integer 10 !Accuracy largely decrease for small values

! Time in average 4 steps in each element
  Timestep Unisotropic Courant Number = Real 0.25
  Max Timestep Size = Real 1.0e3

! Give up integration if particles are tool old
  Max Integration Time = Real 1.0e4

! Integration forward in time
  Runge Kutta = Logical False

  Velocity Gradient Correction = Logical True
  Velocity Variable Name = String "Velocity"

! Keywords for sourcing the particle
  Source Time Correction = Logical True
  Particle time reverse = Logical True

! Show some info in the end
  Variable 2 = String "Particle distance"
  Variable 3 = String "Particle time"
  Variable 4 = String "Particle time integral"

! Absolute displacement when going back-and-forth in time.
! For exact this should be zero (computation bugged in parallel)
  Variable 5 = String "Particle disp"

 ! Distance and velocity of the particle can be computed
  Variable 5 = String "Particle distance integral"
  Variable 6 = String "Particle velocity abs"

  Result Variable 1 = String "Hadv"
  Result Variable 4 = String "Damage"
```


```
! Keywords for sourcing the particle
  Source Time Correction = Logical True
  Particle time reverse = Logical True
```


```
 Particle Integral Dummy Argument = Logical True
End
```

Particle time reverse is the keyword allowing the solver to go back in time and then reverse time to compute the source forward in time

The **dummy** value is the value of the Particle Time Integrator (evolving) over the integration



The Semi-Lagrangian Solver in Elmer


```
Solver 2
  Equation = ParticleAdvector
  Procedure = "ParticleAdvector" "ParticleAdvector"

!Relative Mesh Level = Integer -1

! Initialize particles at center of elements (as opposed to nodes)
  Advect Elemental = Logical False

  Reinitialize Particles = Logical True
  !Particle Accurate At Face = Logical False

! Timestepping strategy
  Simulation Timestep Sizes = Logical True

  Particle Dt Constant = Logical False
  Max Timestep Intervals = Integer 10 !Accuracy largely decrease for small values

! Time in average 4 steps in each element
  Timestep Unisotropic Courant Number = Real 0.25
  Max Timestep Size = Real 1.0e3

! Give up integration if particles are tool old
  Max Integration Time = Real 1.0e4

! Integration forward in time
  Runge Kutta = Logical False

  Velocity Gradient Correction = Logical True
  Velocity Variable Name = String "Velocity"

! Keywords for sourcing the particle
  Source Time Correction = Logical True
  Particle time reverse = Logical True
```

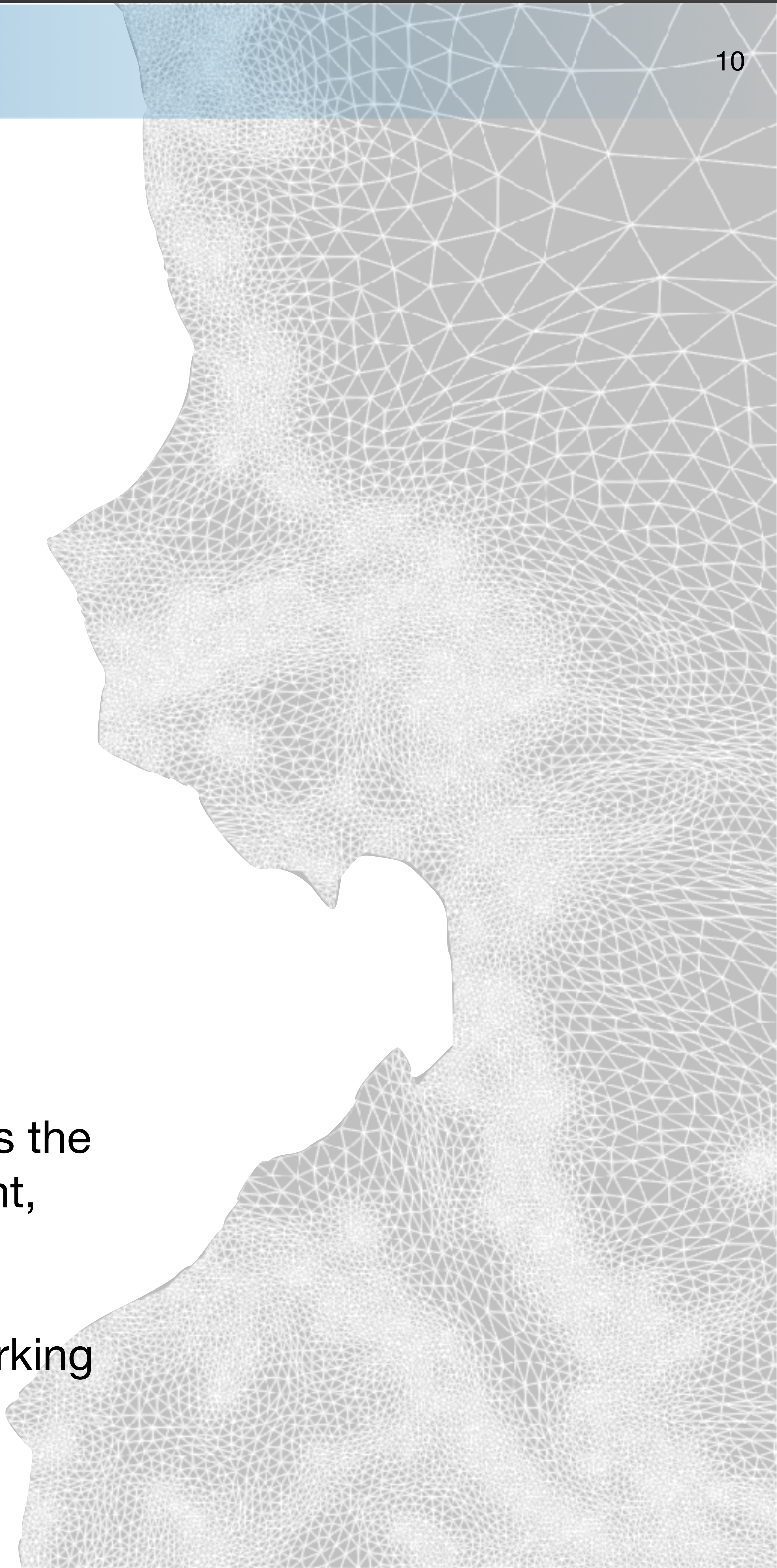
```
! The internal variables for this solver
  Variable 1 = String "Hpart"
  Variable 2 = String "Particle distance"
  Variable 3 = String "Particle time"
  Variable 4 = String "Particle time integral"

! Absolute displacement when going back-and-forth in time.
! For exact this should be zero (computation bugged in parallel)
  Variable 5 = String "Particle disp"

 ! Distance and velocity of the particle can be computed
  Variable 5 = String "Particle distance integral"
  Variable 6 = String "Particle velocity abs"
```

```
 Particle Integral Dummy Argument = Logical True
End
```

External variables can be defined.

Some **internal variables** can be called, such as the particle time/path integral, particle displacement, particle velocity, particle status,…

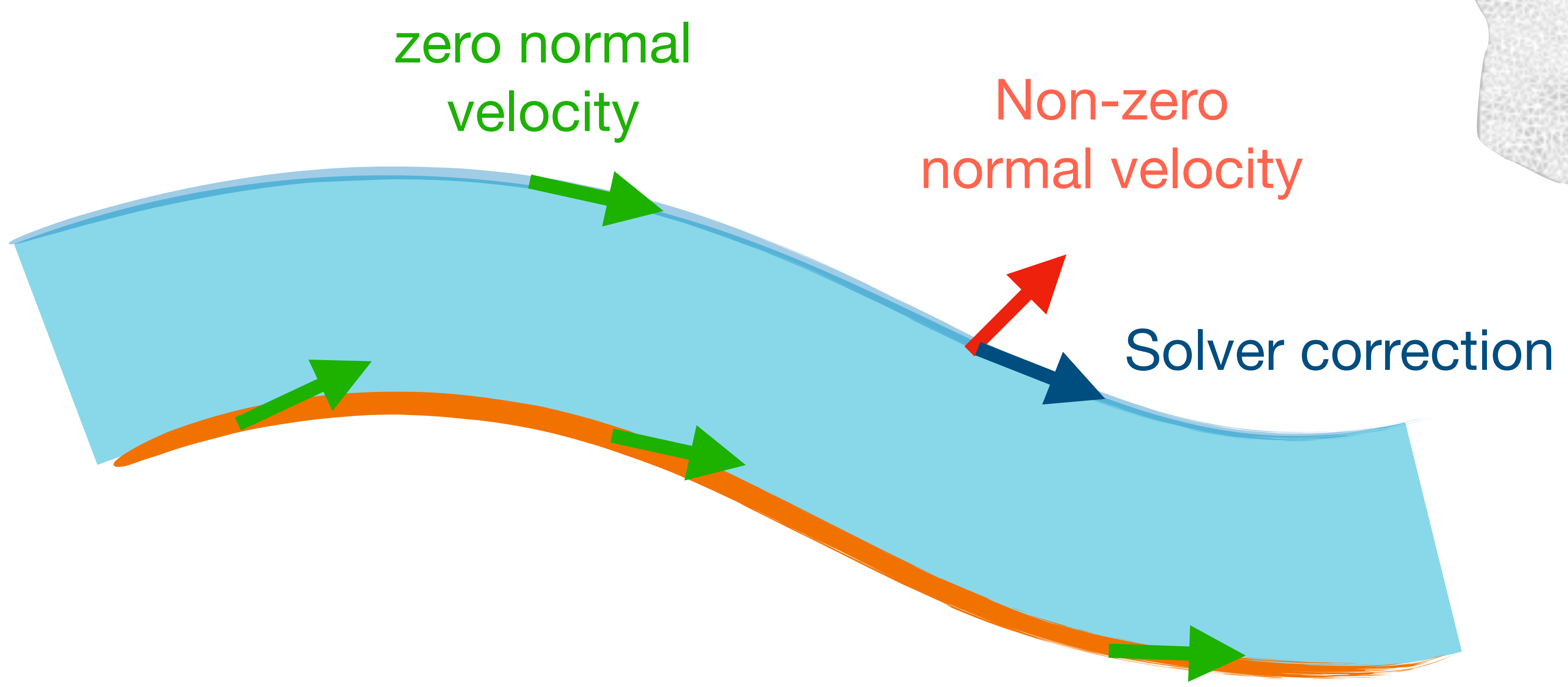
Some are useful to check that everything is working fine!



The Semi-Lagrangian Solver in Elmer

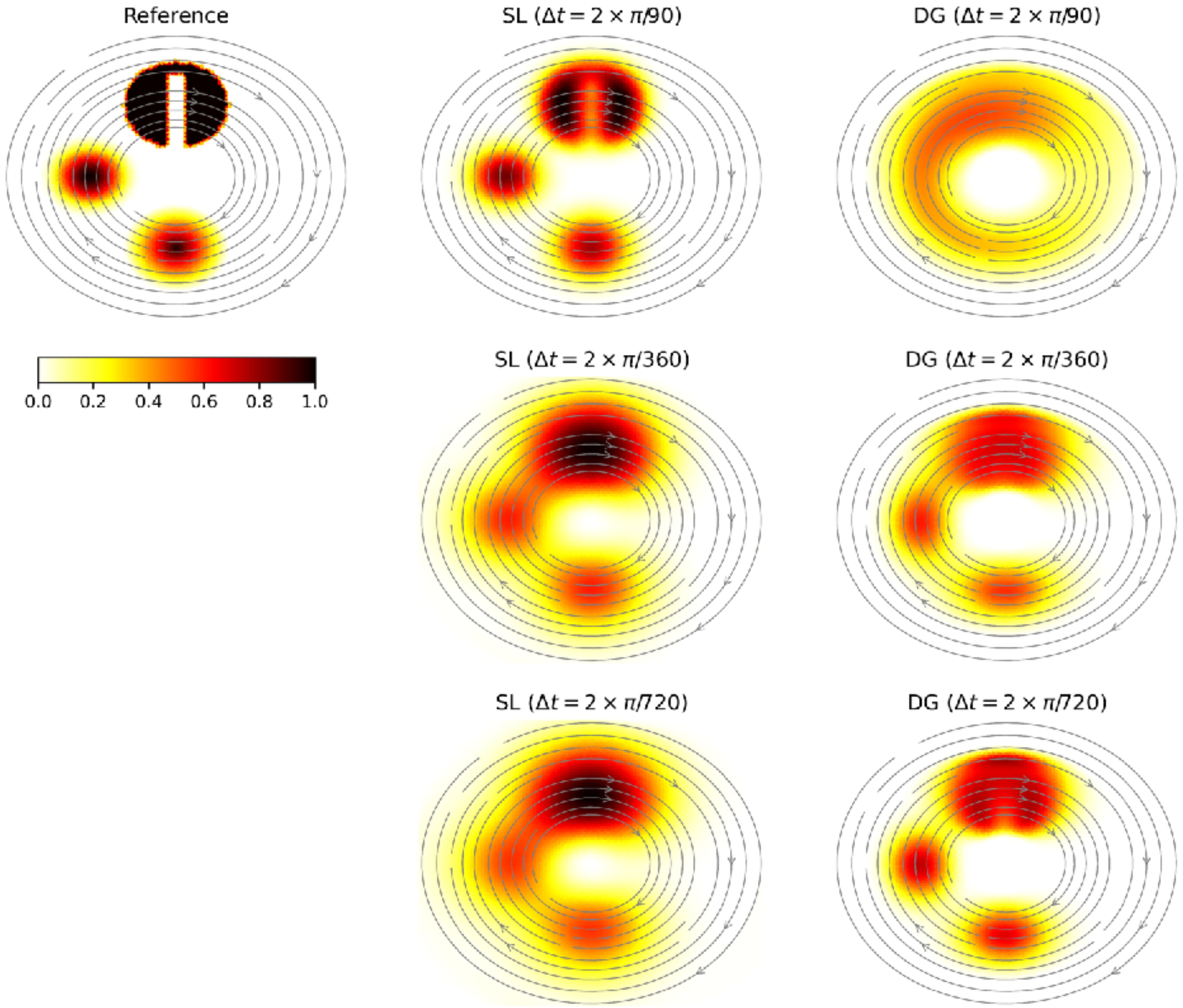

```
Solver 2
  Equation = ParticleAdvector
  Procedure = "ParticleAdvector" "ParticleAdvector"

!Relative Mesh Level = Integer -1

! Initialize particles at center of elements (as opposed to nodes)
  Advect Elemental = Logical False

  Reinitialize Particles = Logical True
  !Particle Accurate At Face = Logical False

! Timestepping strategy
  Simulation Timestep Sizes = Logical True

  Particle Dt Constant = Logical False
  Max Timestep Intervals = Integer 10 !Accuracy largely decrease for small values

! Time in average 4 steps in each element
  Timestep Unisotropic Courant Number = Real 0.25
  Max Timestep Size = Real 1.0e3

! Give up integration if particles are tool old
  Max Integration Time = Real 1.0e4

! Integration forward in time
  Runge Kutta = Logical False

  Velocity Gradient Correction = Logical True
  Velocity Variable Name = String "Velocity"

! Keywords for sourcing the particle
  Source Time Correction = Logical True
  Particle time reverse = Logical True

! Show some info in the end
  Particle Info = Logical True
  Particle Time = Logical True

! The internal variables for this solver
  Variable 1 = String "Hpart"
  Variable 2 = String "Particle distance"
  Variable 3 = String "Particle time"
  Variable 4 = String "Particle time integral"

! Absolute displacement when going back-and-forth in time.
! For exact this should be zero (computation bugged in parallel)
  Variable 5 = String "Particle disp"

 ! Distance and velocity of the particle can be computed
  Variable 5 = String "Particle distance integral"
  Variable 6 = String "Particle velocity abs"

  Result Variable 1 = String "Hadv"
  Result Variable 4 = String "Damage"

 Particle Integral Dummy Argument = Logical True
End
```

```
Boundary Condition 5
  Particle Wall = Logical True
  Particle Tangent = Logical True
```

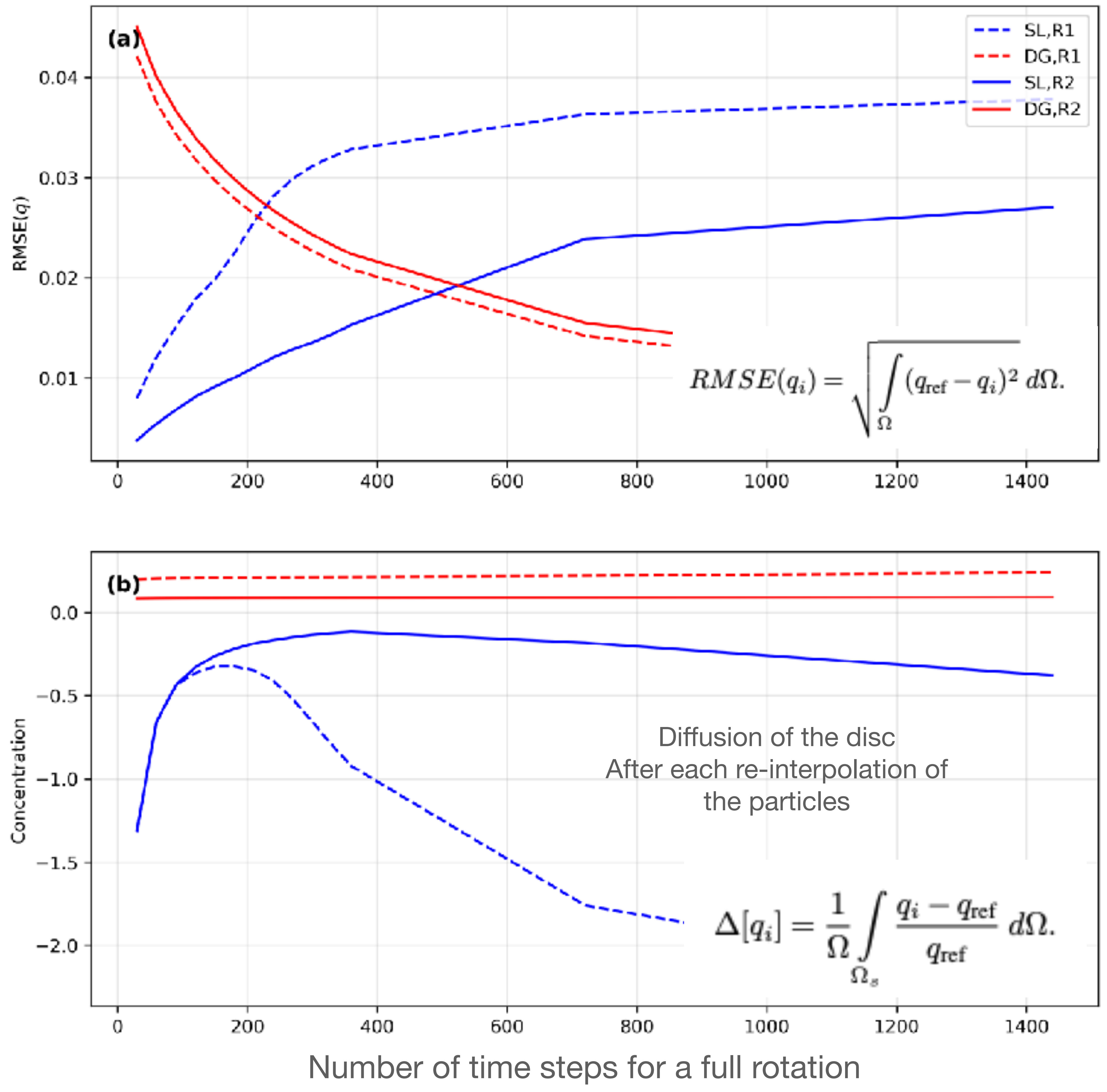


Check that you have this on, especially if you work with unrelaxed surfaces

Principle of the semi-Lagrangian Method

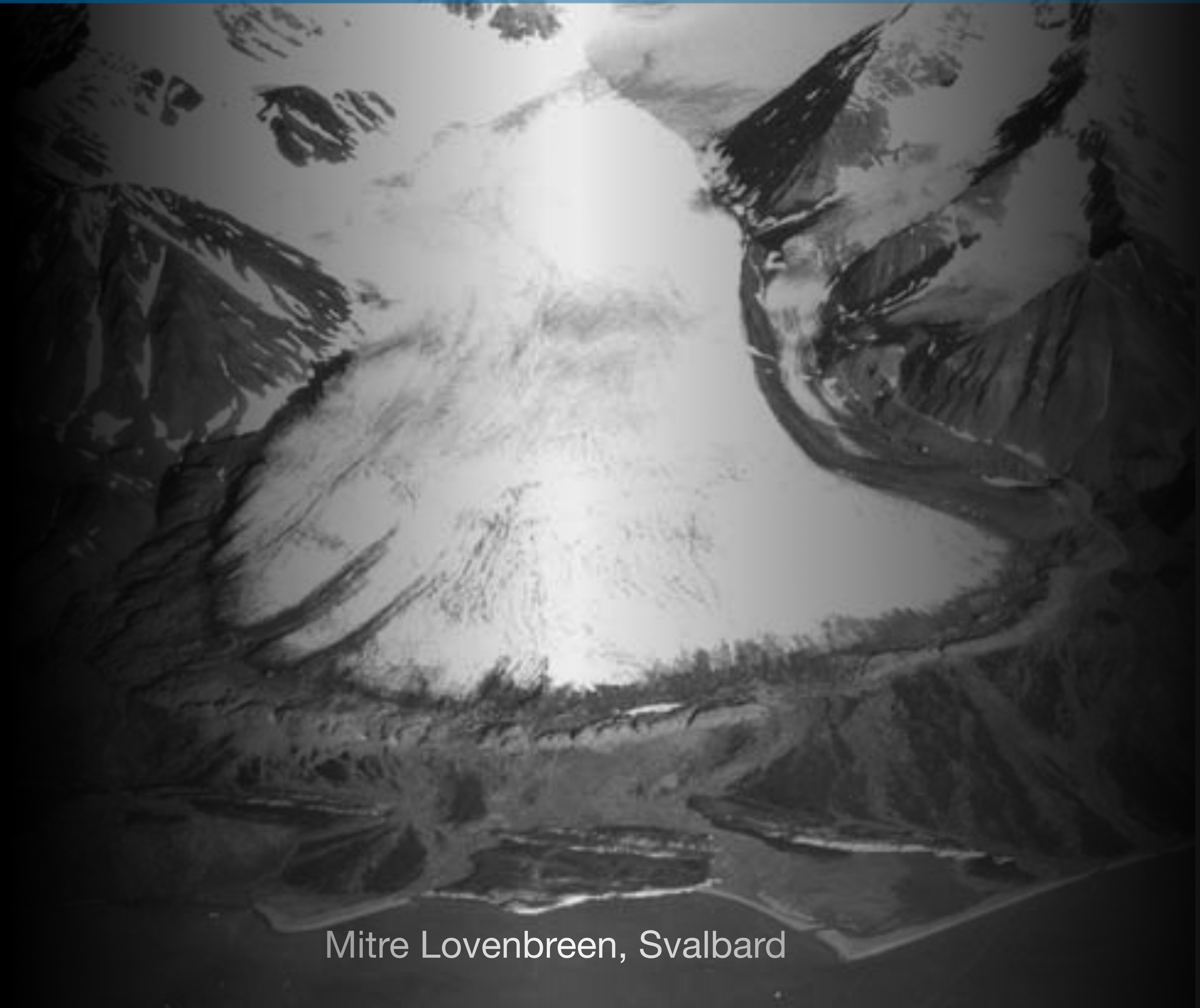


More reinitialisation > More reinterpolation > Less accuracy



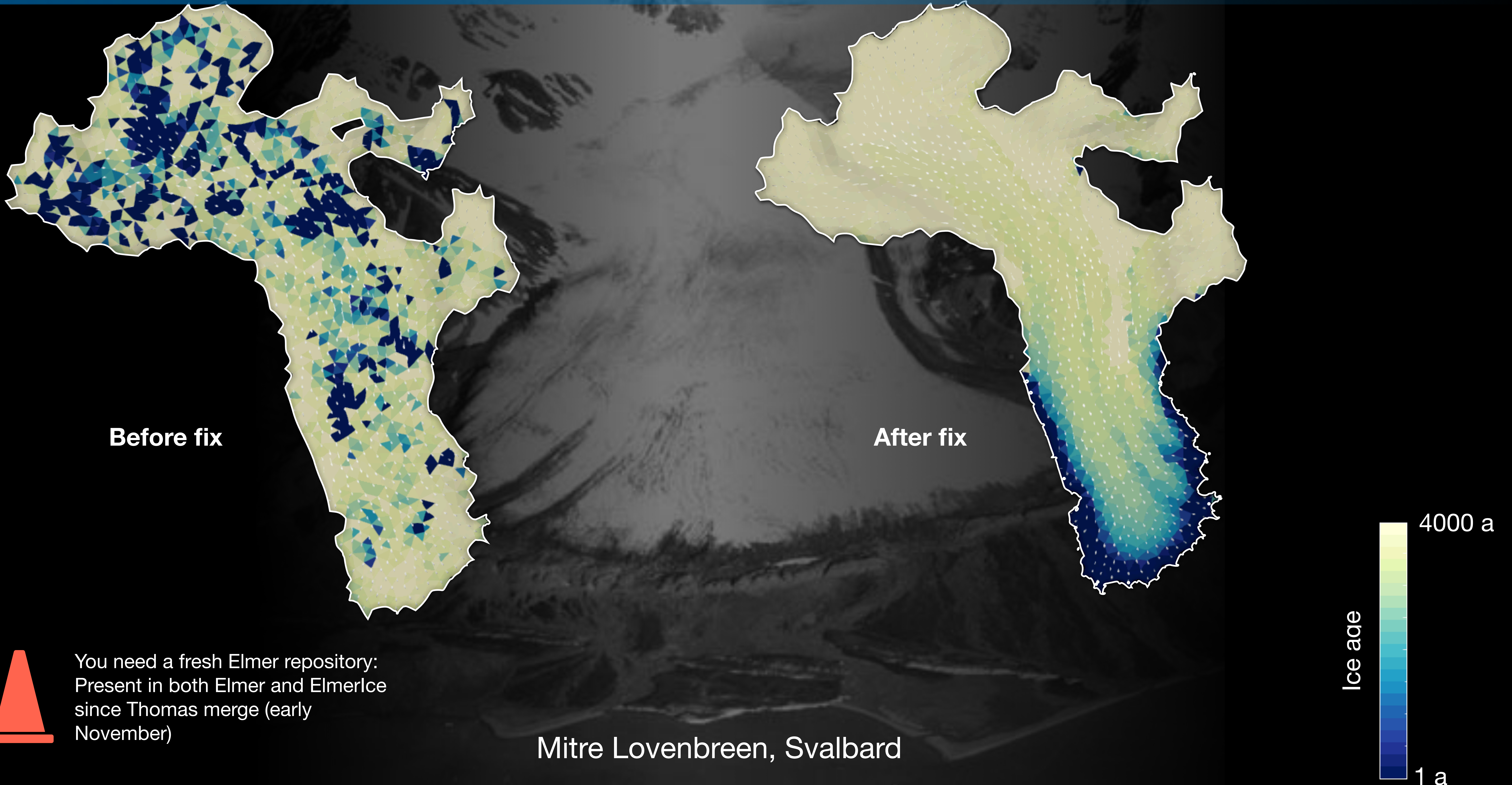
Simulations with Particle Reinitialization at each timestep

Application to ice age



Mitre Lovenbreen, Svalbard

Application to ice age



You need a fresh Elmer repository:
Present in both Elmer and ElmerIce since Thomas merge (early November)

Application to damage

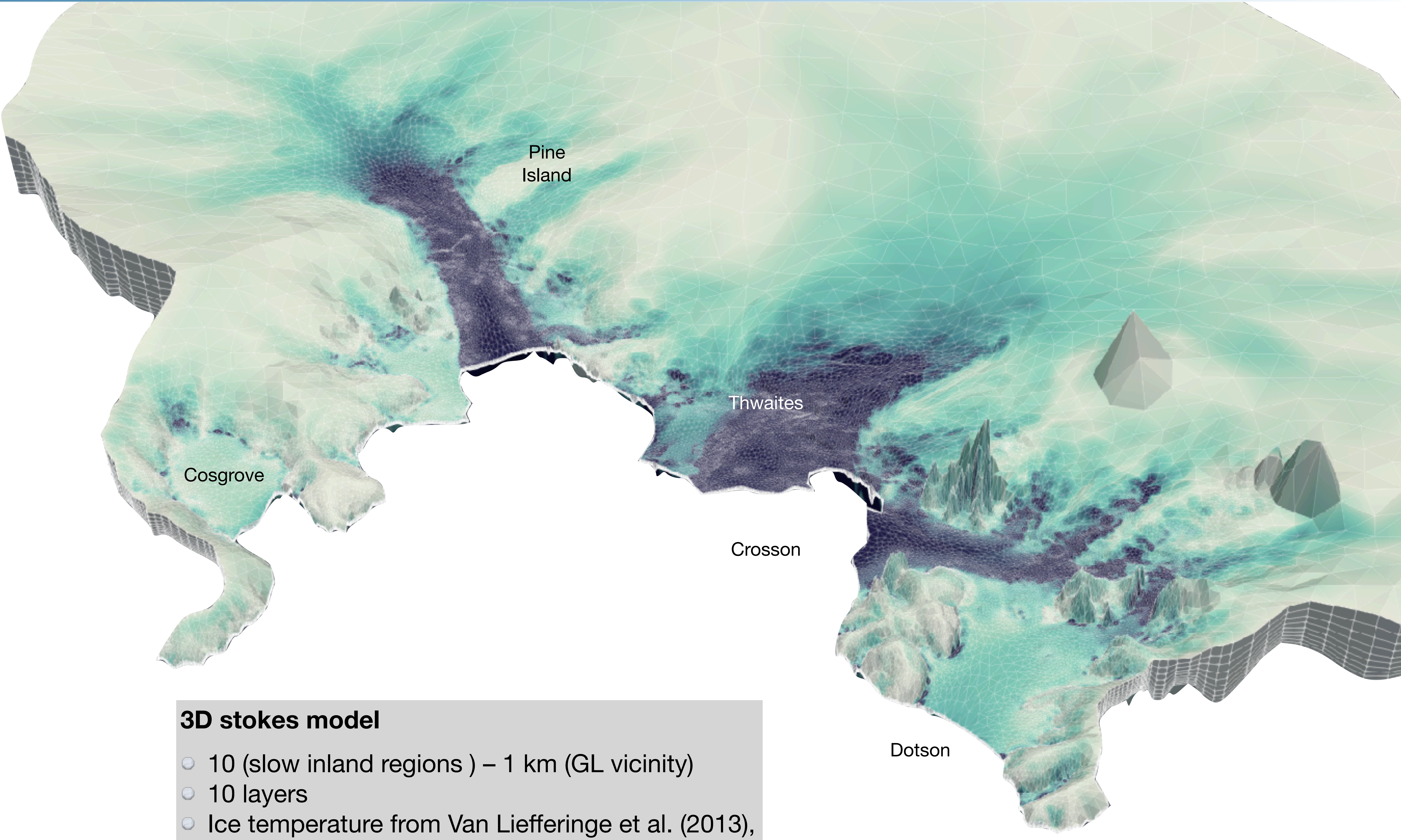
Reconstruct the past ice sheet

i.e. build a 1996 ice sheet state
(Original idea from de Rydt et al., 2020)

- Bedmachine correction with observed dhdt (IceSat1-2 gives us the trend over 2003-2019)
- Surface velocity observations are very good for the Amundsen sea back in 1996
- We can compare the reconstructed grounding line to observations.

Simulate the ice sheet with damage

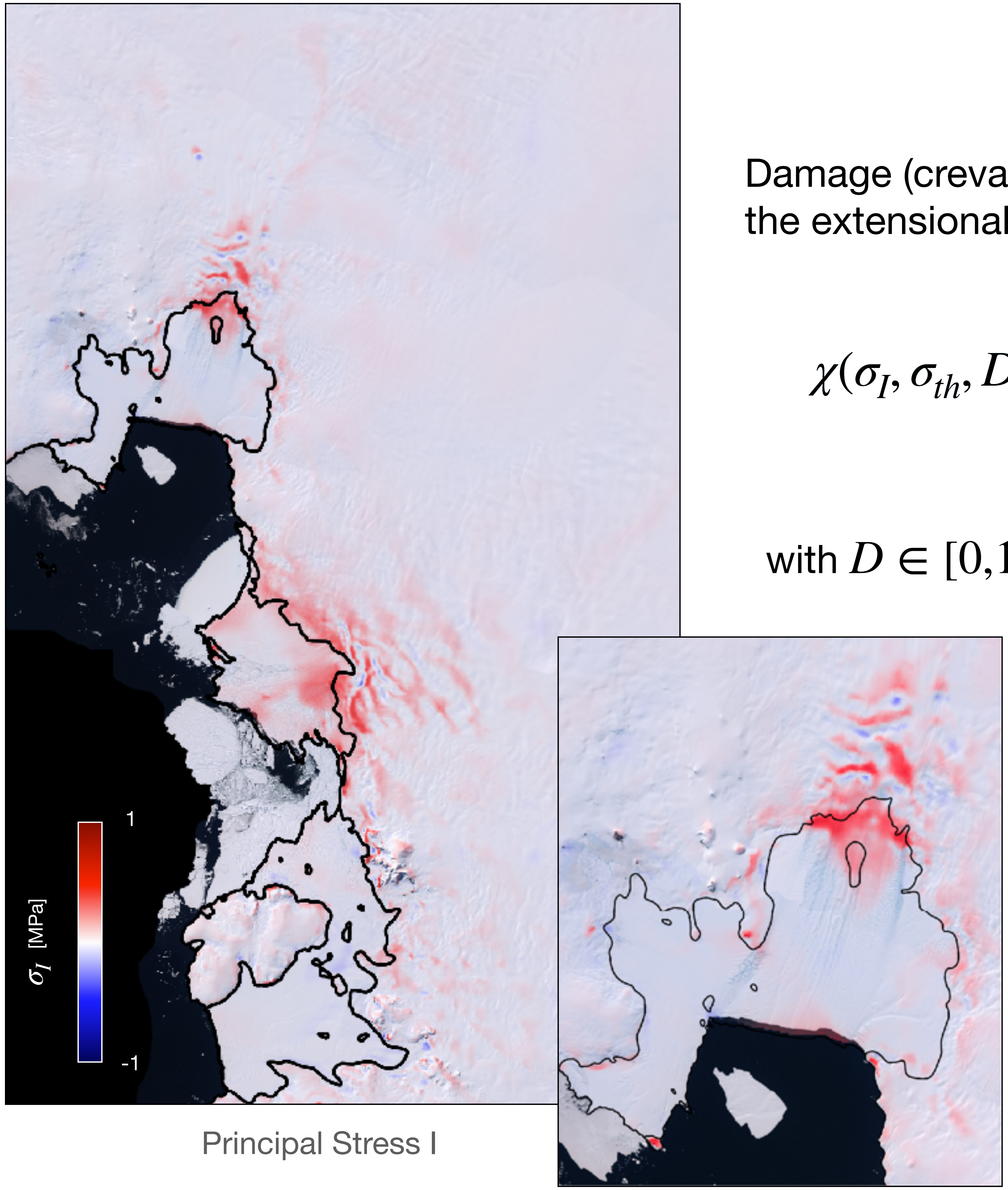
- Trend of dhdt, surface velocity and grounding line migration
- Comparison with :
  - damage observation
  - Inverted damage (data assimilation)



3D stokes model

- 10 (slow inland regions) – 1 km (GL vicinity)
- 10 layers
- Ice temperature from Van Liefferinge et al. (2013), updated in 2019
- Surface mass balance from MAR
- Ocean melt parametrized to have a steady and undamaged ice sheet in 1996

Application to damage

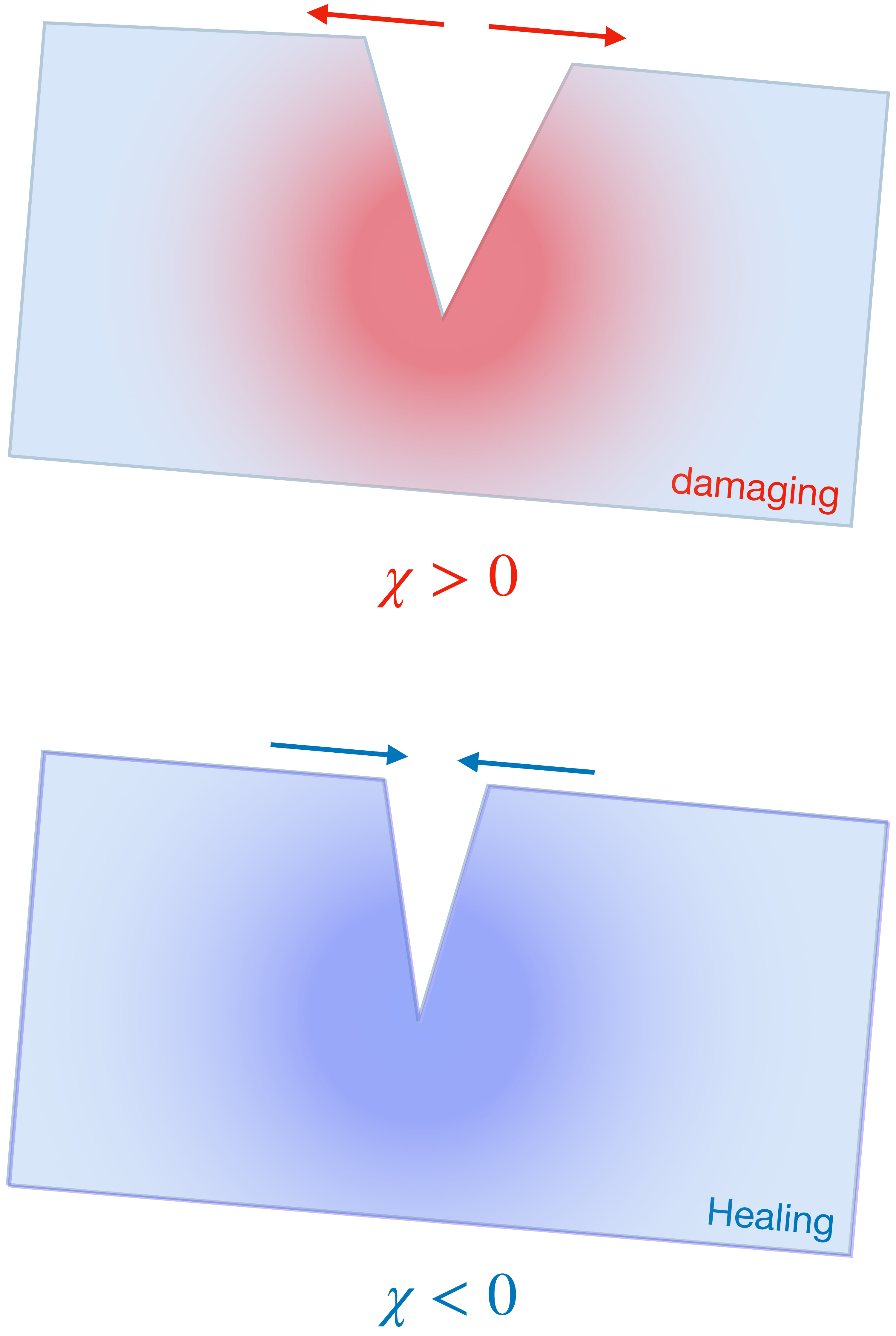


Principal Stress I

Damage (crevasses, rifts,…) creates where the extensional stress is large enough:

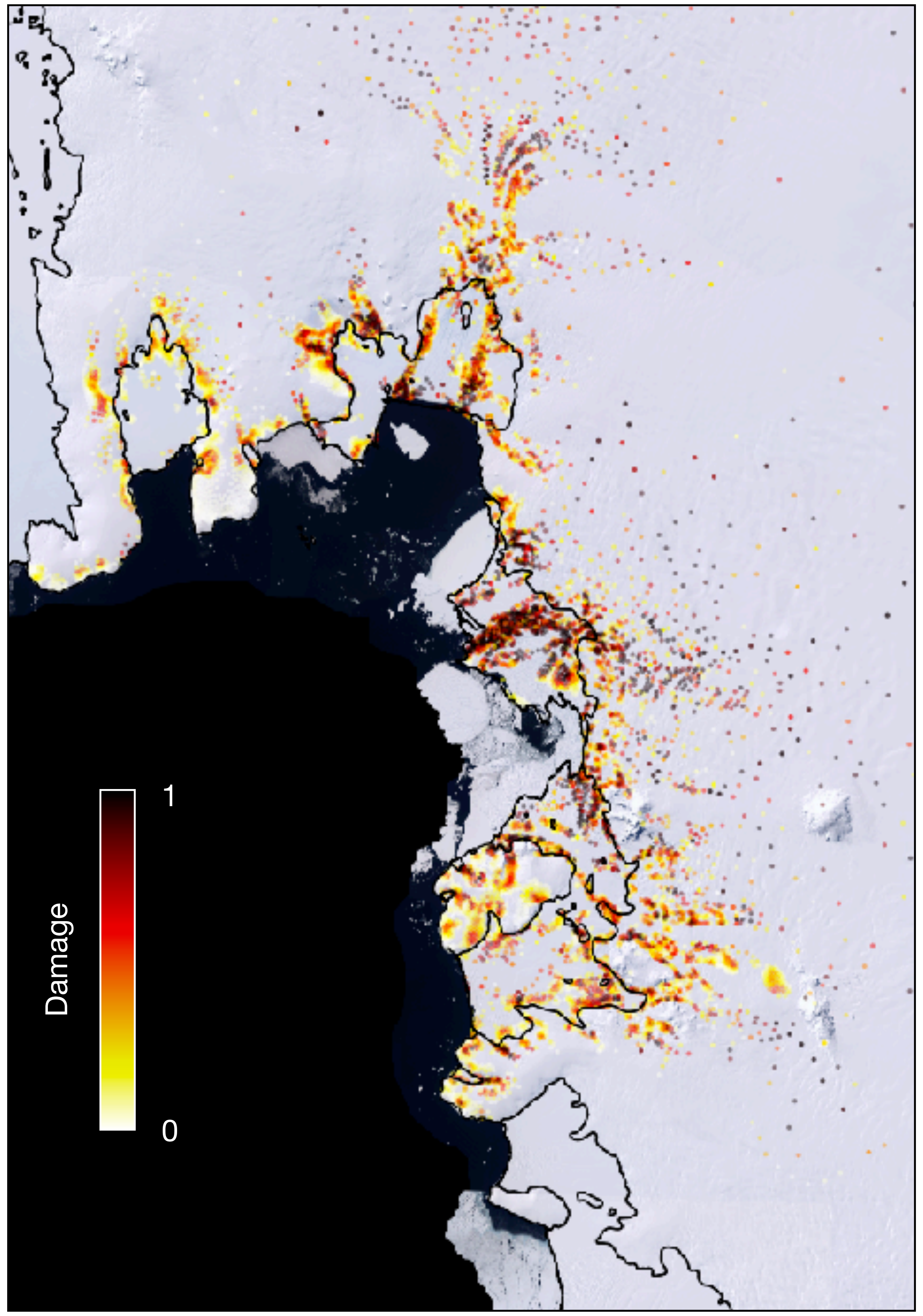
$$\chi(\sigma_I, \sigma_{th}, D) = \frac{\sigma_I}{1-D} - \sigma_{th} + p_w$$

with $D \in [0,1[$ and σ_{th} a minimal threshold.

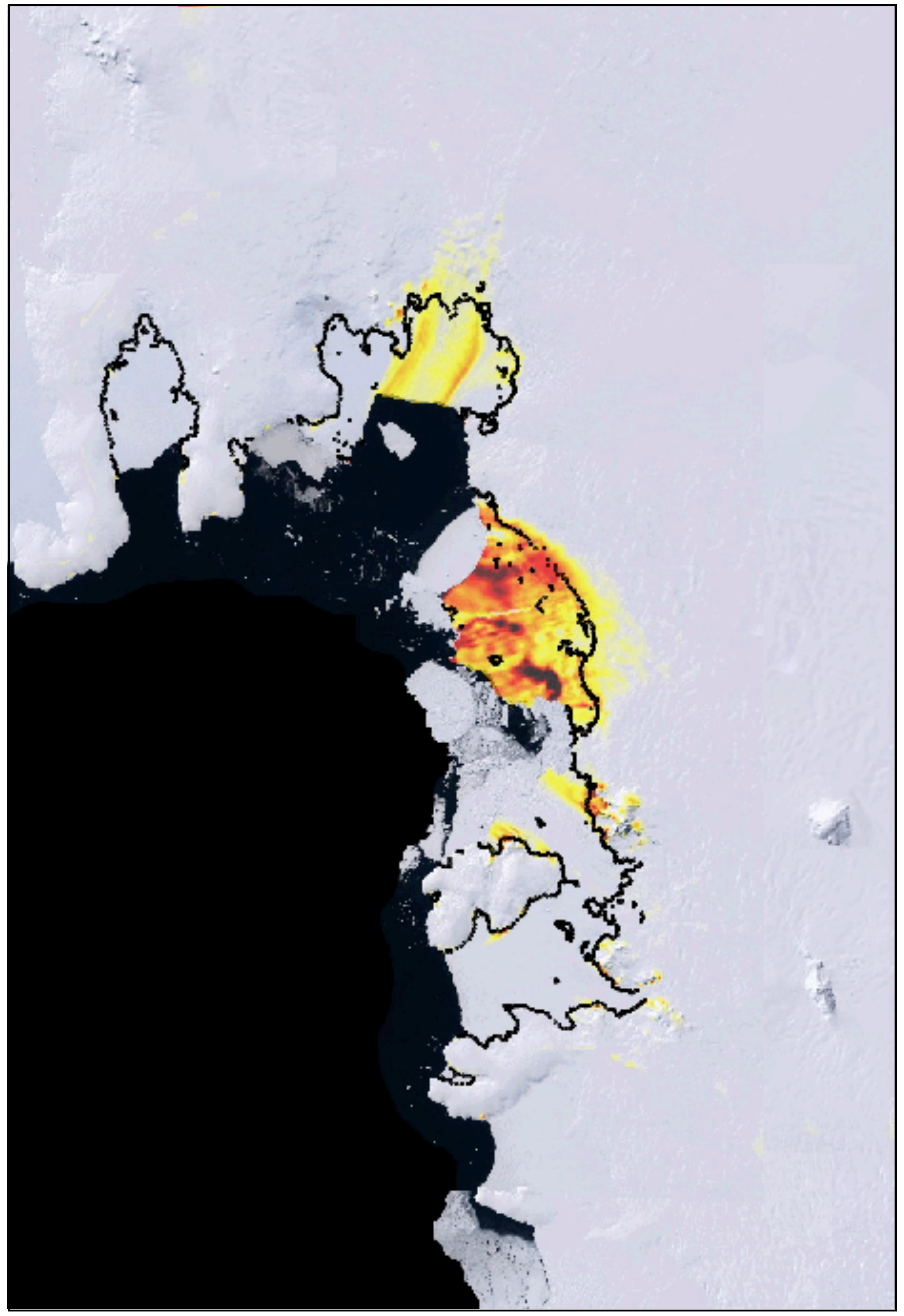


(e.g. Krug et al., 2014, Sun et al. 2017)

Application to damage

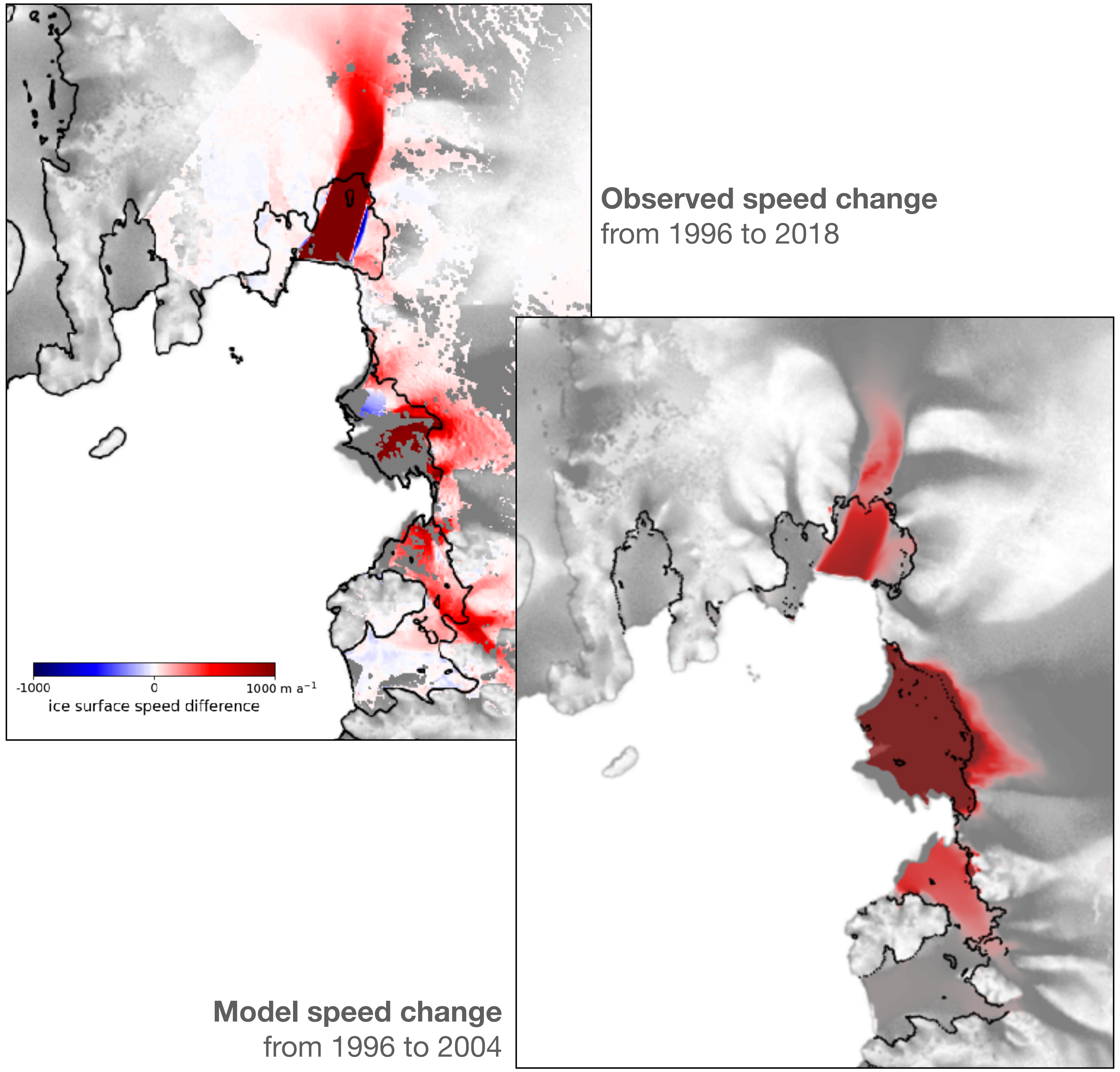


Damage reconstruction by
data assimilation for the year 2018



Damage evolution over 1996-2004
assuming no damage

(Set of parameters : $\sigma_{th} = 0.20, \quad B = 0.5$ and no healing)



Observed speed change
from 1996 to 2018

Model speed change
from 1996 to 2004

C. Mosbeux, N. Jourdain, G. Gagliardini, P. Raback & F. Gillet-Chaulet

Conclusions

- The solver works as expected in 2D serial simulations
- ~~Some bugs remain to fix for 3D parallel simulations~~
- Please reach us if you still have issues
- The "precision" of the semi-Lagrangian method increases with the size of the timestep (opposite to classical advection method)…
  - See how we can combine this with recent work on "increasing the larger possible timestep"