Università della Svizzera italiana

Institute of Computational Science ICS

Università della Svizzera italiana

CARDIOCENTROTICINO

Center for Computational Medicine in Cardiology CCMC
Institute of Computational Science ICS
Faculty of Informatics

SCCER SoE
SWISS COMPETENCE CENTER for ENERGY RESEARCH
SUPPLY of ELECTRICITY

FNSNF
FONDS NATIONAL SUISSE
SCHWEIZERISCHER NATIONALFONDS
FONDO NAZIONALE SVIZZERO
SWISS NATIONAL SCIENCE FOUNDATION

Eurohack 2018
Utopia an EDSL for scientific computing

Team: Patrick Zulian, Alena Kopanicakova, Maria Nestola, Nur Fadel, Andreas Fink

Mentors: Carter Edwards (NVIDIA), Christian Trott (SNL)

NVIDIA

Sandia National Laboratories

October 2nd, 2018, Lugano

CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

Introduction

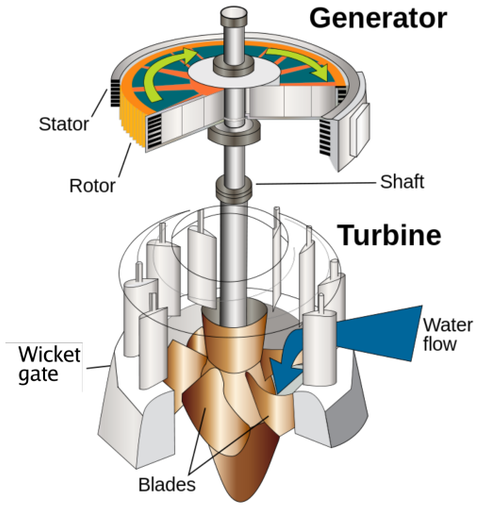
High-performance computing

Usability

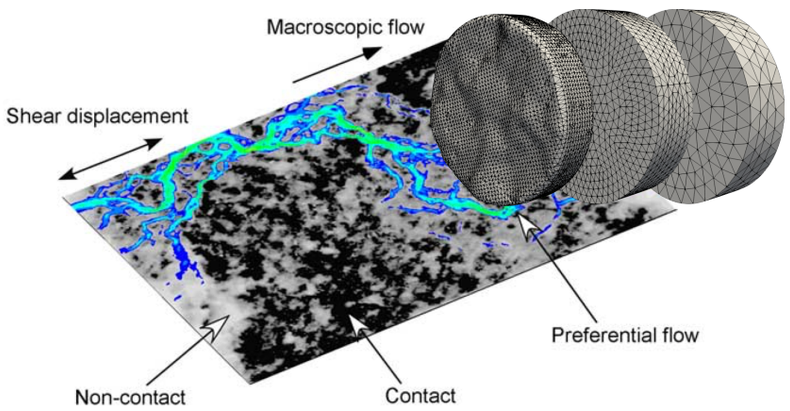
+

Flexibility!

Computational energy



Water turbines

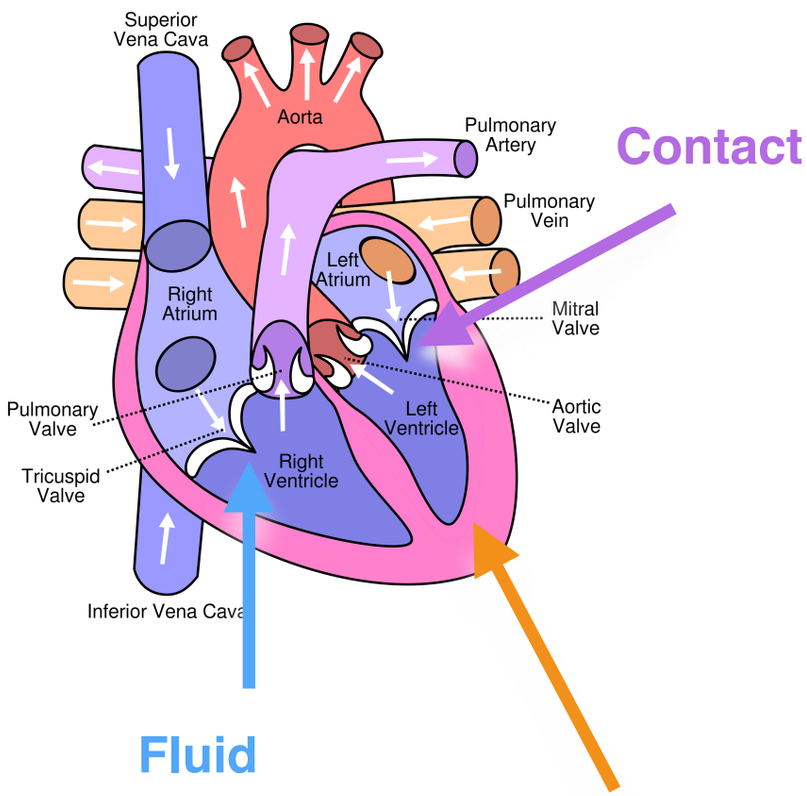


Computational Geophysics

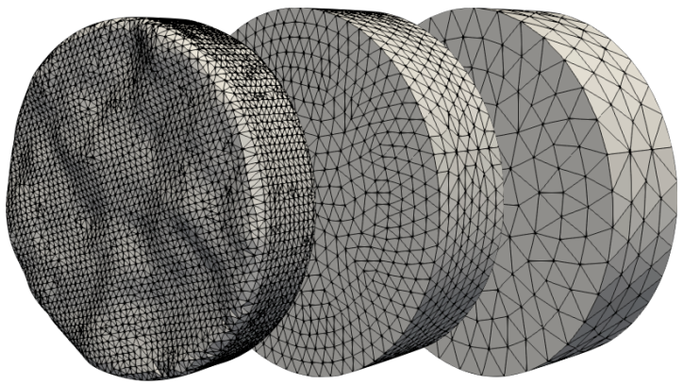
SCCER SoE
SWISS COMPETENCE CENTER for ENERGY RESEARCH
SUPPLY of ELECTRICITY

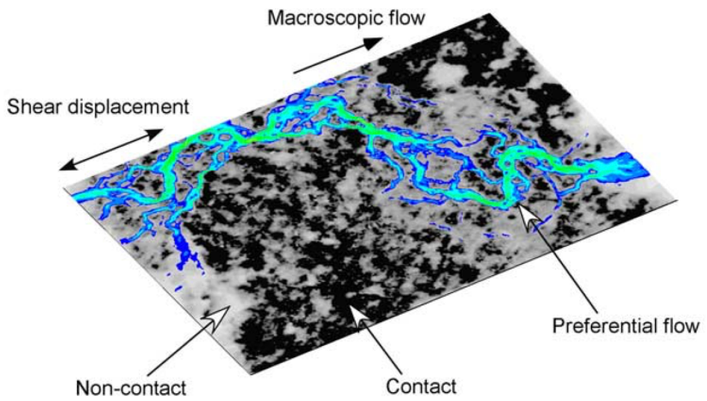
Other applications

Simulation of the heart

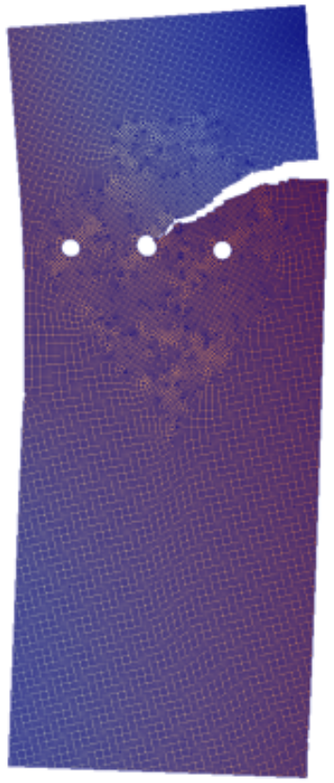


Multilevel solution methods

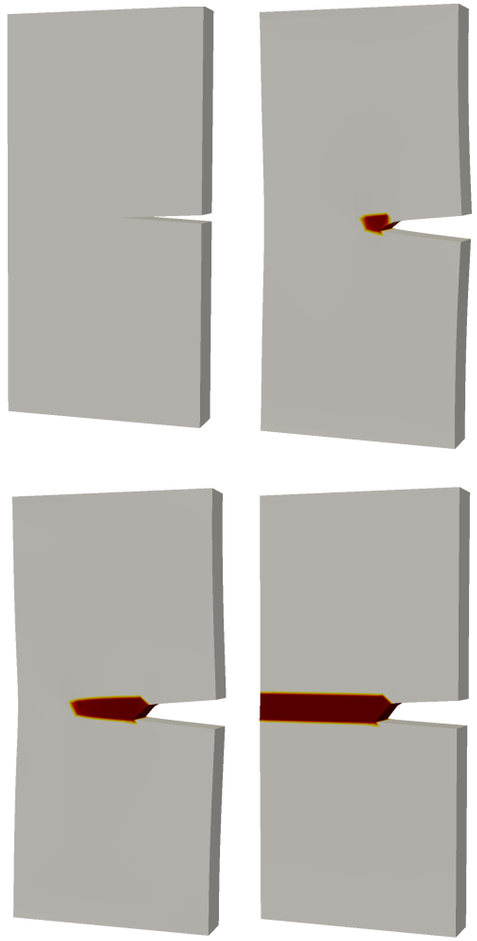




Multigrid solver for **computational geology** (C. Von Planta)



Multigrid solver for **crack simulations** with XFEM[1] (H. Kothari)



Multigrid solver for **Phase-field method**[2] (A. Kopaničáková)

Literature: **1** Z. Goangseup, and T. Belytschko, 2003.
2 C. Miehe, F. Welschinger, and M. Hofacker, 2010.

Example utopia code (Neo-hookean material Hessian and gradient)

```
auto mu     = params_.var_mu();
auto lambda = params_.var_lambda();

auto u  = trial(V_);
auto v  = test(V_);
auto uk = interpolate(x, u);

auto F       = identity() + grad(uk);
auto F_t     = transpose(F);
auto F_inv   = inv(F);
auto F_inv_t = transpose(F_inv);
auto J       = det(F);

auto P = mu * (F - F_inv_t) + (lambda * logn(J)) * F_inv_t;

auto stress_lin = mu * grad(u)
-(lambda * logn(J) - mu) * F_inv_t * transpose(grad(u)) * F_inv_t
+ inner(lambda * F_inv_t, grad(u)) * F_inv_t;

auto l_form = inner(P, grad(v)) * dX;
auto b_form = inner(stress_lin, grad(v)) * dX;
```

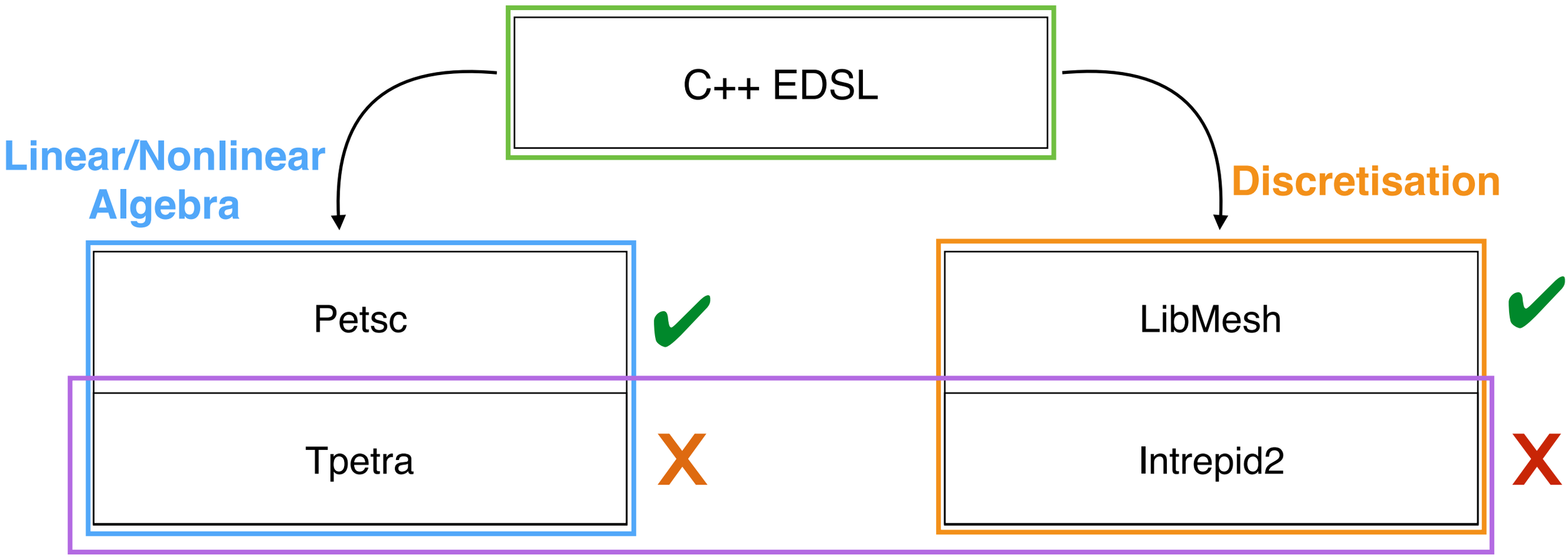
Front-end

```
assemble(b_form == l_form, hessian, gradient);
```

Back-end

Overview

Architecture



Goal for the hackathon

GPU porting approach and status

Approach

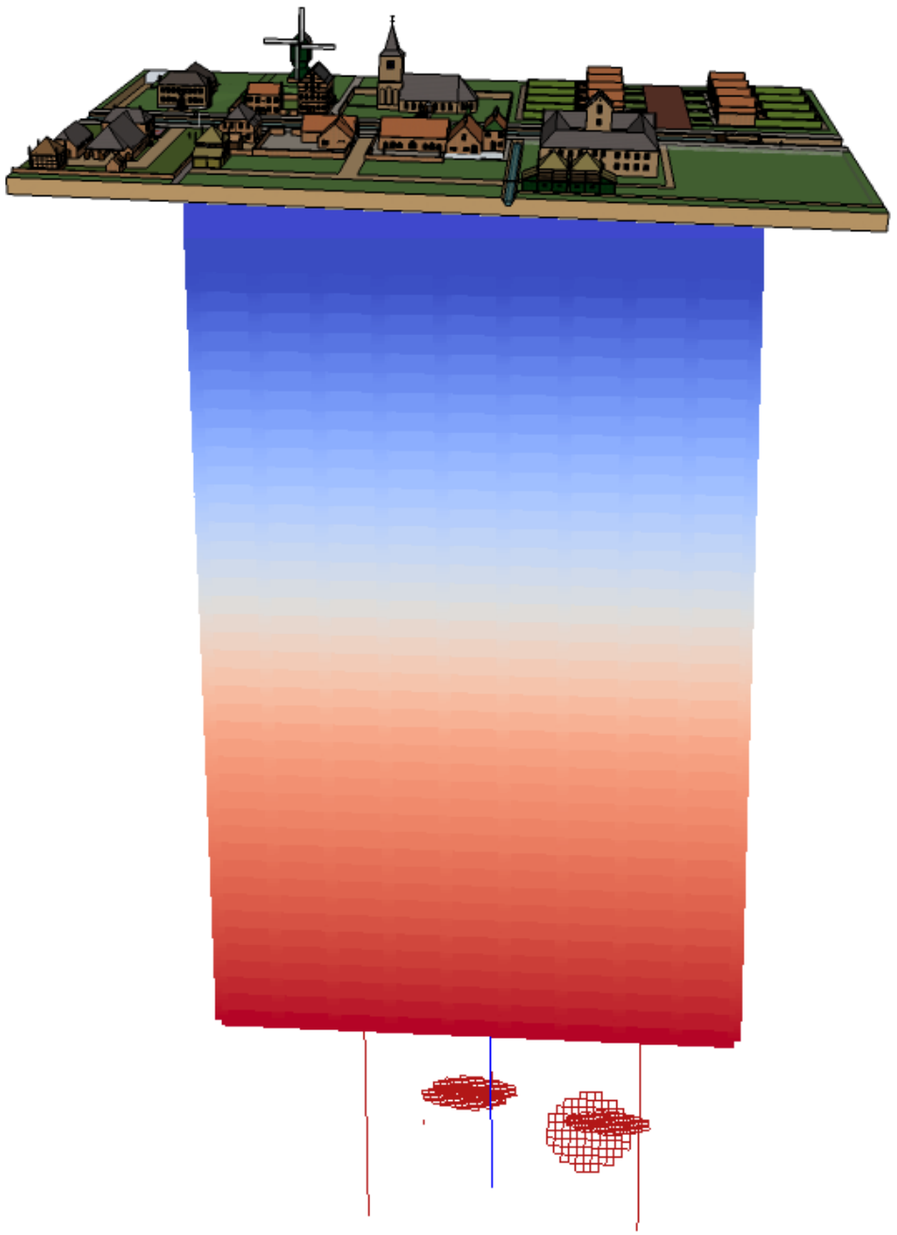
- Exploit existing functionalities from Trilinos (**Tpetra** and **Intrepid2**)
- Implementing missing functionalities with **Kokkos**

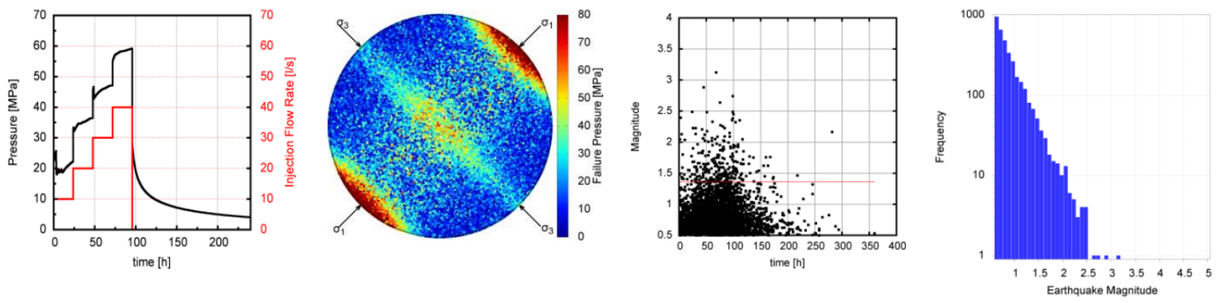
Status

- Basic **Tpetra** backend implemented with still few bugs
- Integration of **Intrepid2** has to be done from scratch

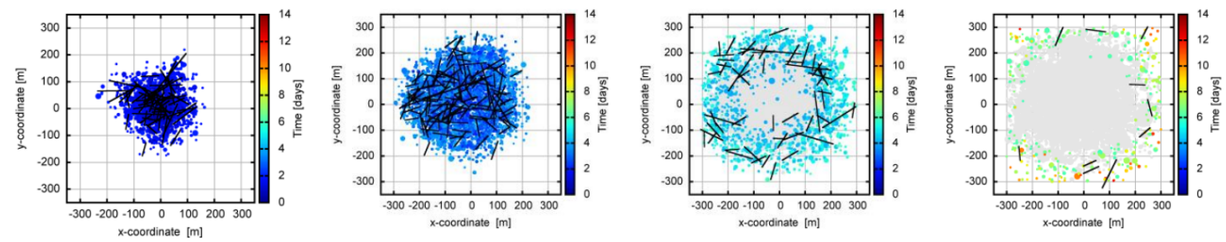
Plan (FASTER)

- Profile of FASTER with utopia and optimise it for the least overhead possible (**Andreas**)

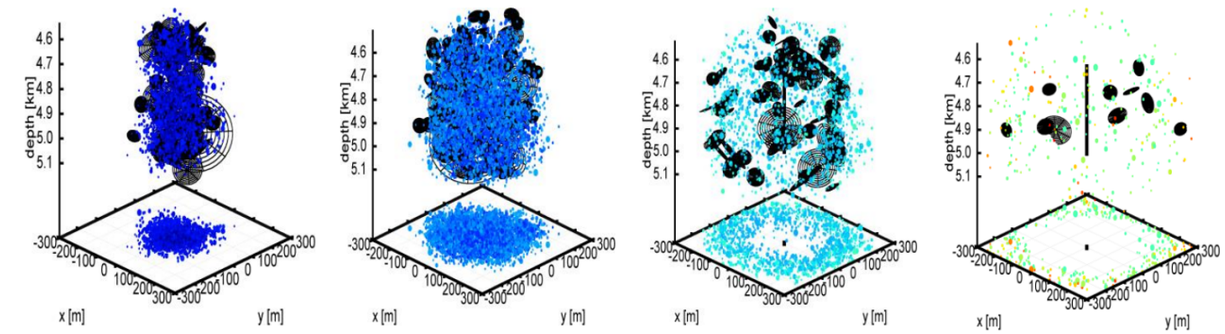




(a) (b) (c) (d)



(e) 0-2 days (f) 2-4 days (g) 4-6 days (h) 6-14 days



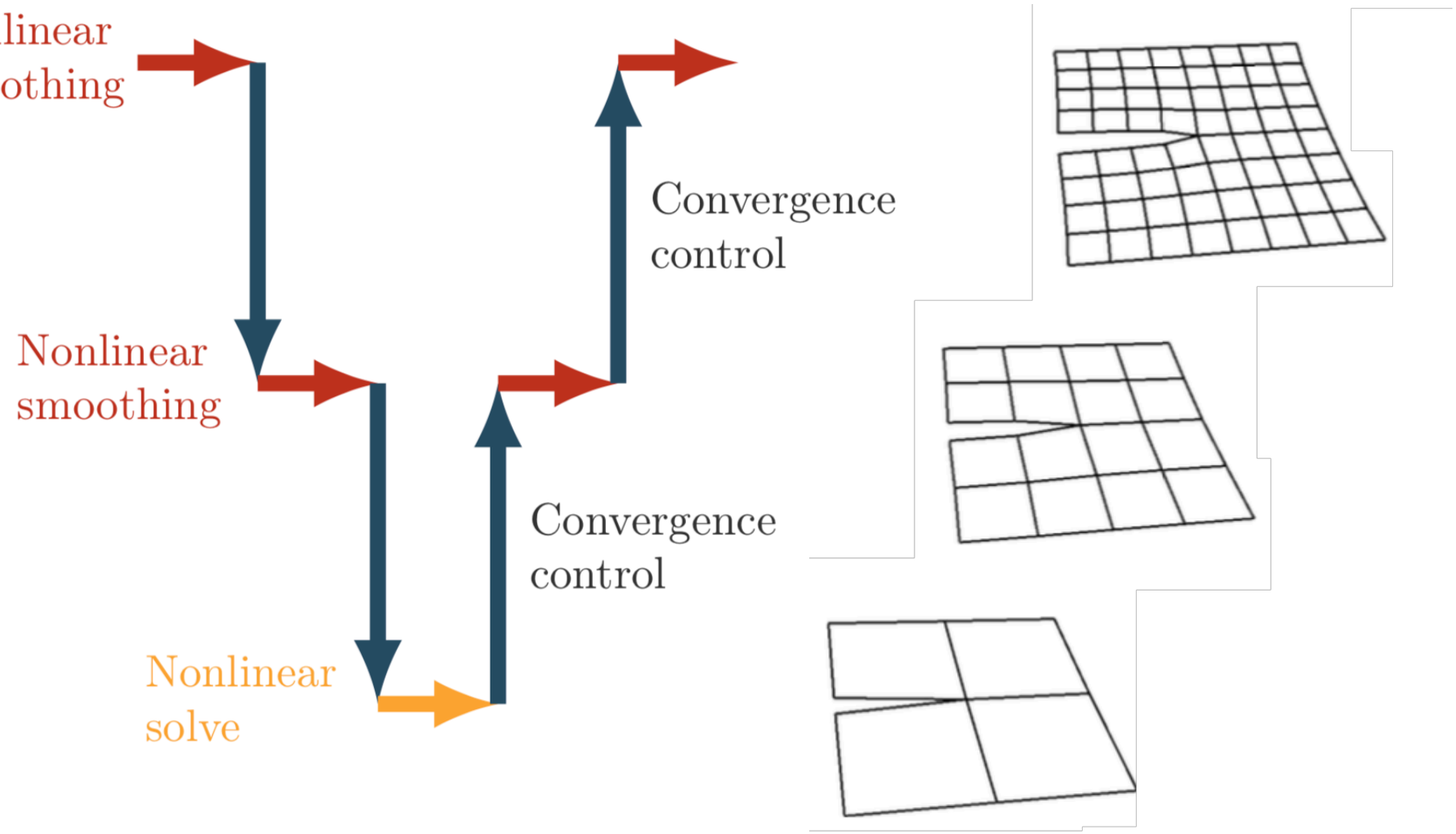
(i) 0-2 days (j) 2-4 days (k) 4-6 days (l) 6-14 days

Plan (Utopia algebra)

- Implement all *TODOs* for the tpetra-backend (**Maria**, **Patrick**, **Nur**)
  - Fix transpose operations for rectangular matrices
  - Fix ghost value handling for fe
  - Implement operations that are missing in tpetra using kokkos
  - Find performance bugs and fix them

Plan (RMTR)

- Recursive Multilevel Trust Region aka RMTR (Alena)
  - Make implementation robust
  - Profile, find bottlenecks, improve, profile,…
  - Compare performance against petsc version



Plan (Utopia FE)

- Proof of concept for utopia_fe for new backend with Intrepid2 (Patrick/Nur)
  - Re-factor code for allowing new backend usage
  - Laplace equation

```
auto u   = trial(V);
auto v   = test(V);
auto b_form = inner(grad(u), grad(v)) * dX;
utopia::assemble(b_form, H);
utopia::assemble(inner(coeff(1.), v) * dX, rhs);
```