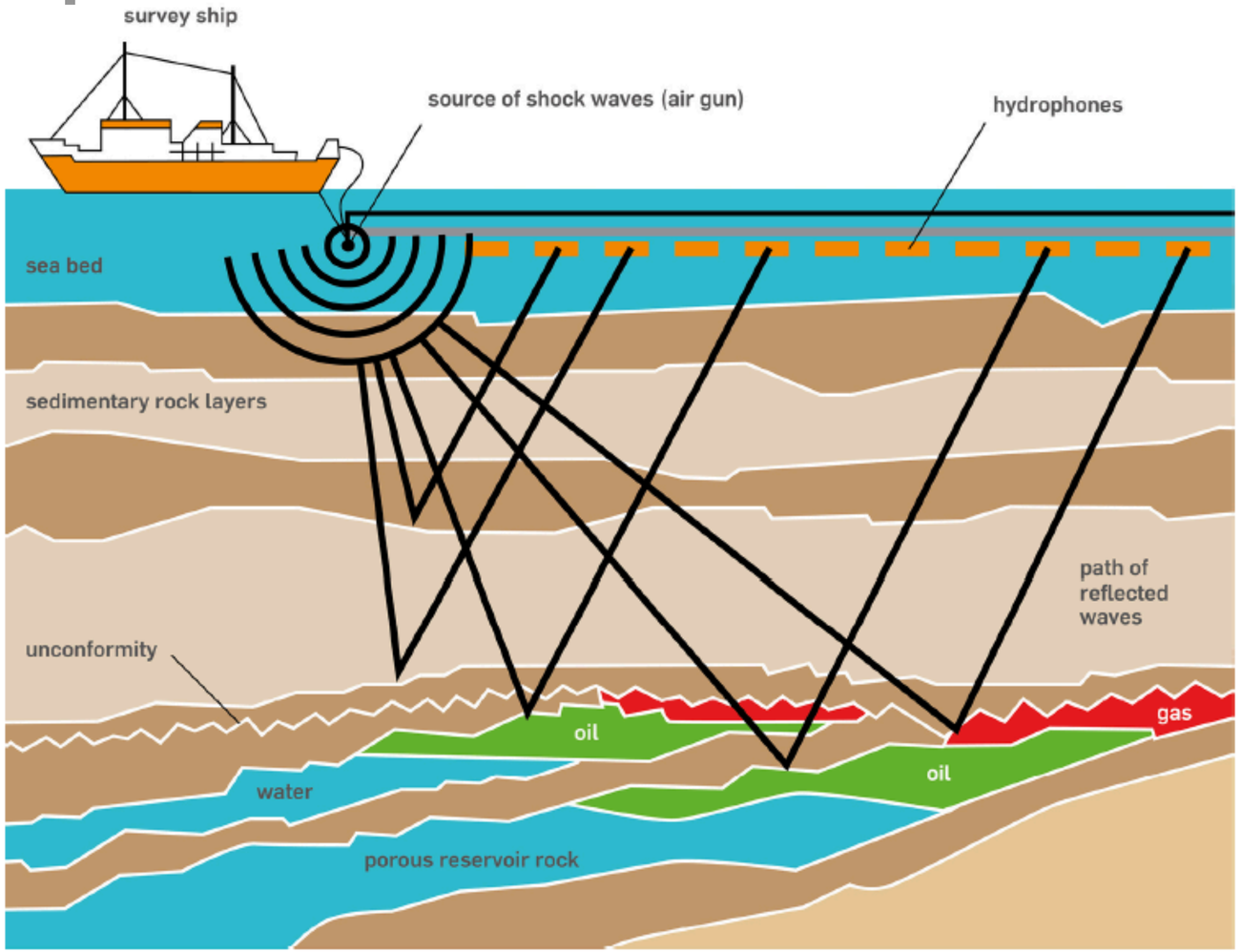
Raising the abstraction to separate concerns: enabling different physics for geophysical exploration

Mathias Louboutin, Michael Lange, Navjot Kukreja, Fabio Luporini,
Felix J. Herrmann and Gerard Gorman

SLIM
University of British Columbia

Wave-equation based geophysical exploration introduction

Physical problem



http://www.open.edu/openlearn/science-maths-technology/science/environmental-science/earths-physical-resources-petroleum/content-section-3.2.1

Mathematical problem

$$\underset{\mathbf{m}}{\text{minimize}} \quad \frac{1}{2}|| \ \mathbf{A}^{-1}(\mathbf{m}) \cdot \mathbf{q} - \mathbf{d}||_2^2$$

(Virieux and Operto, 2009)

$\mathbf{m}$: squared slowness

$\mathbf{d}$: field recorded data

$\mathbf{A}(\mathbf{m})$: discretized wave-equation

$\mathbf{q}$: source term

Challenges

- Multiple representations of the physics
- Problem sizes are huge:
  - seismic surveys consist of tens of thousands of individual experiments
  - model wave propagation over thousands of time steps in large domains
  - typical size of modeling matrix ➡ $\mathbf{A}(\mathbf{m}) \in \mathbb{R}^{n \times n}, n = 1e16$
- Least square problem requires adjoints of the system matrix
- Discrete wave-equation can not be formed as an explicit matrix
  - stencil-based implementation
- Needs scalable, flexible, performant and portable discretization

Scientific motivations

Different physics

- Isotropic acoustic
- Isotropic acoustic with density
- Anisotropic acoustic
- Isotropic elastic
- Anisotropic elastic
- ...

Simulation for inversion

- Adjoint PDE
- Gradients
- ...

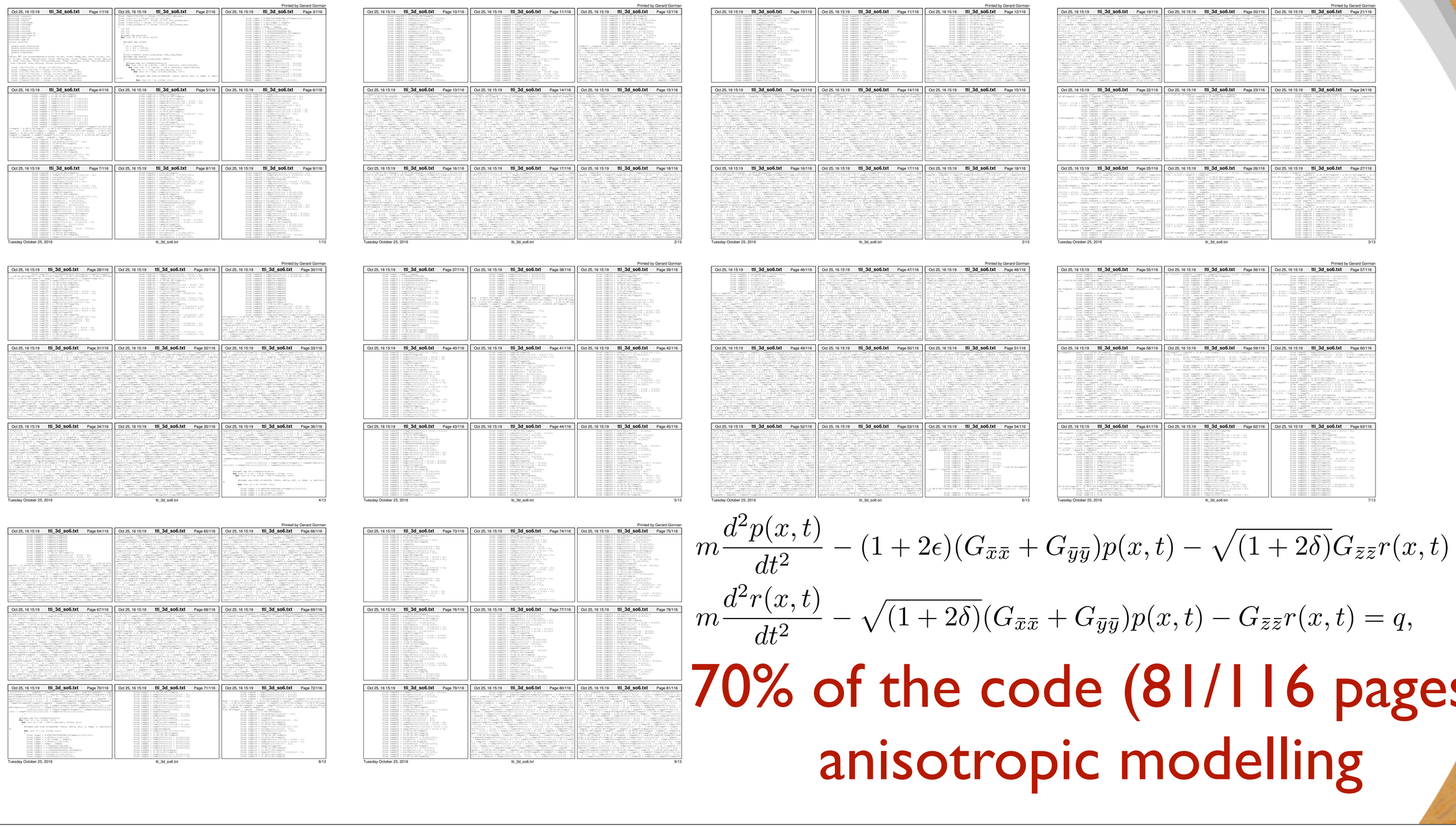
Design motivation

- Writing stencil codes is time consuming and hard for complicated equations. It is even harder because of the need for highly optimized implementations on a range of different computer architectures.

- **Separation of Concerns** with a finite-difference DSL

- Geophysicists need to be able to focus on the physics
- Computer scientists need to be able to focus on the software
- Mathematicians need to focus on numerical analysis

Design motivations



$$m\frac{d^2p(x,t)}{dt^2} - (1+2\epsilon)(G_{\bar{x}\bar{x}} + G_{\bar{y}\bar{y}})p(x,t) - \sqrt{(1+2\delta)}G_{\bar{z}\bar{z}}r(x,t) = q,$$

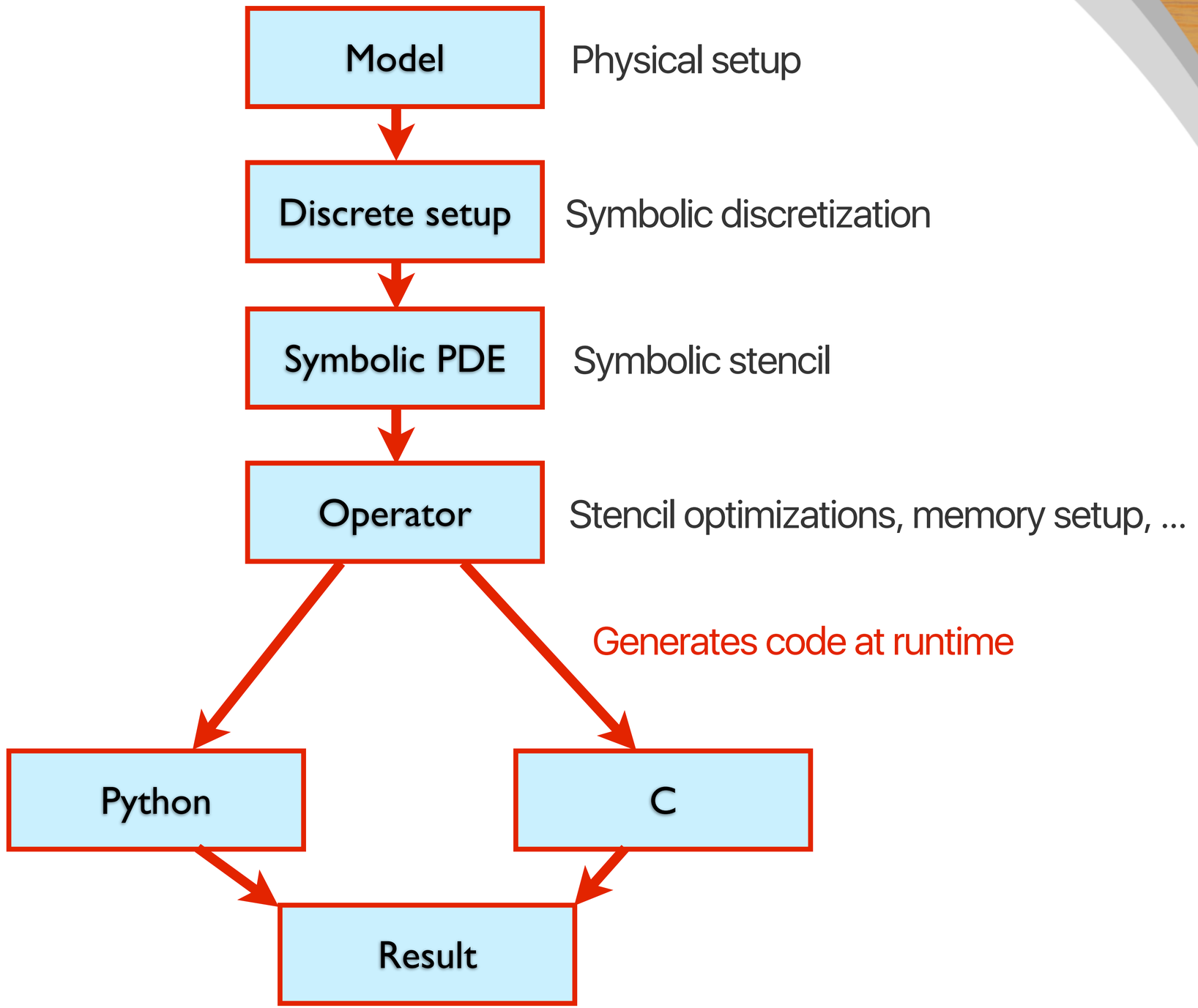
$$m\frac{d^2r(x,t)}{dt^2} - \sqrt{(1+2\delta)}(G_{\bar{x}\bar{x}} + G_{\bar{y}\bar{y}})p(x,t) - G_{\bar{z}\bar{z}}r(x,t) = q,$$

70% of the code (81/116 pages) anisotropic modelling

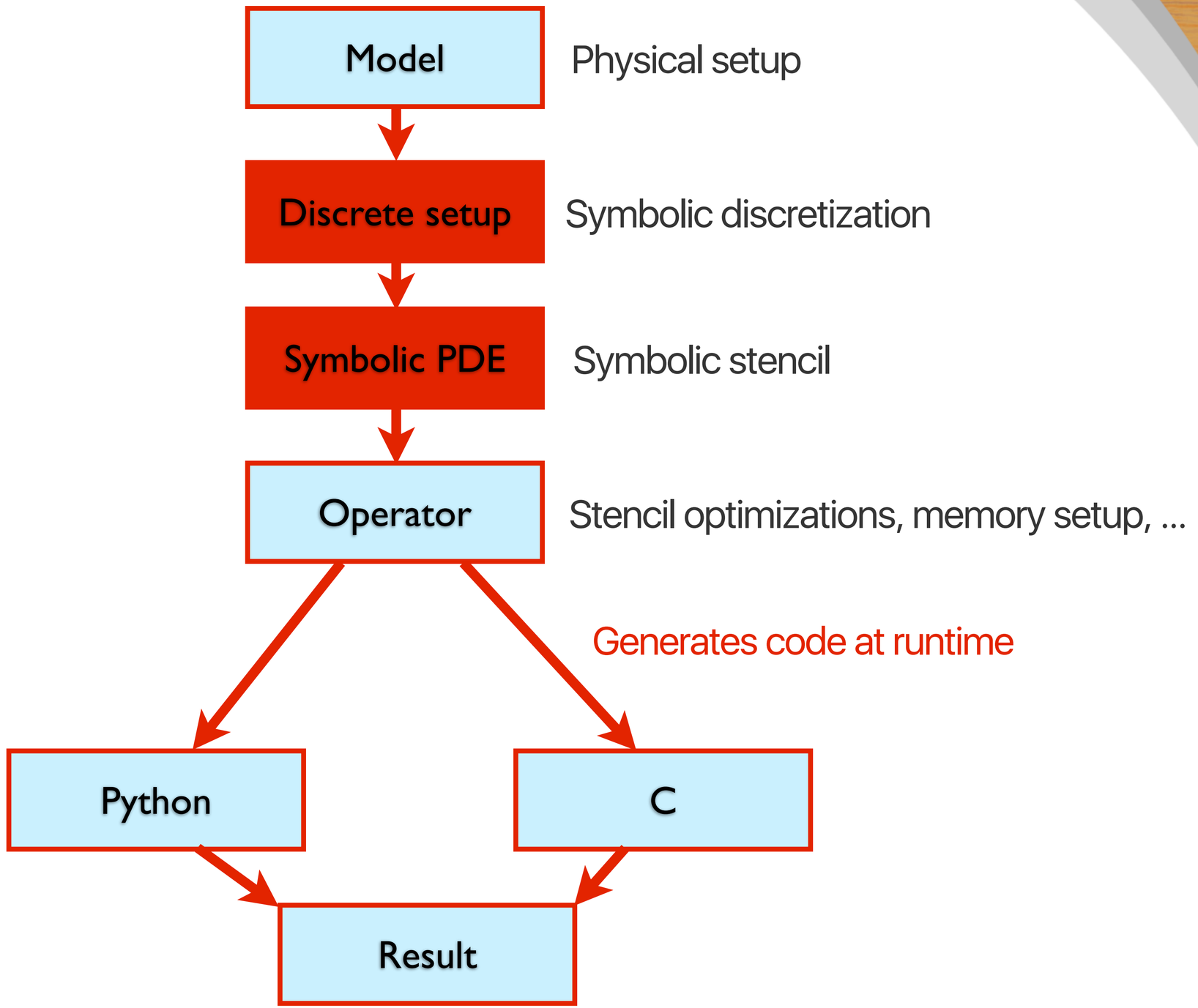
Objectives

- Flexible finite-differences discretization
- One framework for all equations
- Symbolic interface
  - easy to use
  - simple introduction of new PDEs

Summary


Model
Physical setup
Discrete setup
Symbolic discretization
Symbolic PDE
Symbolic stencil
Operator
Stencil optimizations, memory setup, ...
Generates code at runtime
Python
C
Result

Summary


Model
Physical setup
Discrete setup
Symbolic discretization
Symbolic PDE
Symbolic stencil
Operator
Stencil optimizations, memory setup, ...
Generates code at runtime
Python
C
Result

Enabling different physics

Wave-equations

$$\frac{1}{c^2}\frac{d^2p(x,t)}{dt^2} - \Delta p(x,t) = 0$$

Acoustic isotropic

$$\frac{1}{\rho c^2}\frac{d^2p(x,t)}{dt^2} - \nabla.(\frac{1}{\rho}\nabla p(x,t)) = 0$$

Acoustic isotropic with density

$$m\frac{d^2p(x,t)}{dt^2} - (1+2\epsilon)(G_{\bar{x}\bar{x}} + G_{\bar{y}\bar{y}})p(x,t) - \sqrt{(1+2\delta)}G_{\bar{z}\bar{z}}r(x,t) = q,$$
$$m\frac{d^2r(x,t)}{dt^2} - \sqrt{(1+2\delta)}(G_{\bar{x}\bar{x}} + G_{\bar{y}\bar{y}})p(x,t) - G_{\bar{z}\bar{z}}r(x,t) = q,$$

Acoustic anisotropic

$$m\frac{d^2p(x,t)}{dt^2} - (1+2\epsilon)(D_{xx} + D_{yy})p(x,t) - \sqrt{(1+2\delta)}D_{zz}r(x,t) = q,$$
$$m\frac{d^2r(x,t)}{dt^2} - \sqrt{(1+2\delta)}(D_{xx} + D_{yy})p(x,t) - D_{zz}r(x,t) = q,$$

Acoustic tilted anisotropic

Zhang, Y., Zhang, H., and Zhang, G., 2011, A stable TTI reverse time migration and its implementation: Geophysics, 76

Symbolic discretization

Symbolic object with finite-difference discretization as a property

```
u =TimeData(name="u", shape=(nx, ny, nz),
            time_order=self.t_order,
            space_order=self.s_order)
```

is a symbolic object with derivatives properties

```
u.dx, u.dy, u.dz, u.dx2, ..., u.laplace,
```

In[69]: u.dx
Out[69]: -u(t - s, x - 3*h, y, z)/(60*h) + 3*u(t - s, x - 2*h, y, z)/(20*h) - 3*u(t - s, x - h, y, z)/(4*h) + 3*u(t - s, x + h, y, z)/(4*h) - 3*u(t - s, x + 2*h, y, z)/(20*h) + u(t - s, x + 3*h, y, z)/(60*h)

Symbolic wave-equations

Acoustic

$$0 = m\frac{d^2u(x,t)}{dt^2} - \Delta u(x,t) + \text{damp}\frac{du(x,t)}{dt}$$

```
eqn = m * u.dt2 - u.laplace + damp * u.dt
```

Acoustic 4th order in time

$$0 = m\frac{d^2u(x,t)}{dt^2} - \Delta u(x,t) - \frac{dt^2}{12}\Delta\left(\frac{1}{m}\Delta u(x,t)\right) + \text{damp}\frac{du(x,t)}{dt}$$

```
eqn = m * u.dt2 - u.laplace - (s**2)/12 * u.laplace2(1/m)+ damp * u.dt
```

Worked example

Acoustic modelling

Wave-equation setup

$$\mathbf{m}(\mathbf{x})\frac{\mathbf{u}(\mathbf{x},\mathbf{t}+dt)-2\mathbf{u}(\mathbf{x},t)+\mathbf{x},\mathbf{t}-dt)}{dt^2}-\Delta\mathbf{u}(\mathbf{x},\mathbf{t})=0$$

```
equation = m * u.dt2 - u.laplace + damp * u.dt
```

Absorbing boundary condition

$\mathbf{u}$: discretized wavefield

$\mathbf{m}$: discretized square slowness

Δ : discretized Laplacian

```
u = TimeData(name="u", shape=model.get_shape_comp(),
             time_dim=nt, time_order=time_order,
             space_order=spc_order,
             save=save, dtype=damp.dtype)

m = DenseData(name="m", shape=model.get_shape_comp(),
              dtype=damp.dtype)


Lap = u.laplace
```

Stencil

$$\mathbf{u}(\mathbf{x}, \mathbf{t} + dt) = 2\mathbf{u}(\mathbf{x}, t) - \mathbf{u}(\mathbf{x}, \mathbf{t} - dt) + \frac{dt^2}{\mathbf{m}(\mathbf{x})}\Delta\mathbf{u}(\mathbf{x}, \mathbf{t})$$

u.forward = solve(equation, u.forward)

```
stencil = Eq(u.forward, solve(equation, u.forward))
```

Forward operator

```
# Create a forward operator
super(ForwardOperator, self).__init__(nt, m.shape,
                                      stencils=stencil,
                                      subs=subs,
                                      spc_border=spc_order/2,
                                      time_order=time_order,
                                      forward=True,
                                      dtype=m.dtype,
                                      **kwargs)

# Insert source and receiver terms post-hoc
self.propagator.time_loop_stencils_a = source.add(m, u) + rec.read(u)
```

Generate code at runtime

Application developer

```
(rec, u) = Acoustic.Forward()
```

Generated code

```
#include <cassert>
#include <cstdlib>
#include <cmath>
#include <iostream>
#include <fstream>
#include <vector>
#include <cstdio>
#include <string>
#include <inttypes.h>
#include <sys/time.h>
#include <math.h>
struct profiler
{
  double loop_stencils_a;
  double loop_body;
  double kernel;
} ;
struct flops
{
  long long loop_stencils_a;
  long long loop_body;
  long long kernel;
} ;
extern "C" int ForwardOperator(double *m_vec, double *u_vec, double *damp_vec, double *src_vec, float
*src_coords_vec, double *rec_vec, float *rec_coords_vec, long i1block, struct profiler *timings, struct flops *flops)
{
  double (*m)[280] = (double (*)[280]) m_vec;
  double (*u)[280][280] = (double (*)[280][280]) u_vec;
  double (*damp)[280] = (double (*)[280]) damp_vec;
  double (*src)[2] = (double (*)[2]) src_vec;
  float (*src_coords)[2] = (float (*)[2]) src_coords_vec;
  double (*rec)[101] = (double (*)[101]) rec_vec;
  float (*rec_coords)[2] = (float (*)[2]) rec_coords_vec;
  {
    struct timeval start_kernel, end_kernel;
    gettimeofday(&start_kernel, NULL);
    int t0;
    int t1;
    int t2;
    ;
    for (int i3 = 0; i3<500; i3+=1)
    {
      flops->kernel += 2.000000;
      {
        {
          t0 = (i3)%(3);
          t1 = (t0 + 1)%(3);
          t2 = (t1 + 1)%(3);
        }
        struct timeval start_loop_body, end_loop_body;
        gettimeofday(&start_loop_body, NULL);
        {
          for (int i1b = 1; i1b<279 - (278 % i1block); i1b+=i1block)
            for (int i1 = i1b; i1<i1b+i1block; i1++)
              {
```

Scientific motivations

Multi-physics

- Isotropic acoustic
- Isotropic acoustic with density
- Anisotropic acoustic
- Isotropic elastic
- Anisotropic elastic
- ...

Simulation for inversion

- Adjoint PDE
- Gradients
- ...

Adjoint-state

$$\underset{\mathbf{m}}{\text{minimize}} \quad \frac{1}{2}|| \ \mathbf{A}^{-1}(\mathbf{m}) \cdot \mathbf{q} - \mathbf{d}||_2^2$$

(Virieux and Operto, 2009)

Gradient with respect to $\mathbf{m}$ requires adjoint

$$\mathbf{g} = -\left(\frac{\partial \mathbf{u}}{\partial m}\right)^\top \mathbf{A}^{-\top}(\mathbf{m})(\mathbf{A}^{-1}(\mathbf{m}).\mathbf{q} - \mathbf{d})$$

Discretization for inversion

Extend symbolic discretization to adjoints

```
first_derivative(u,  dim=x, side=centered, order=spc_order, matvec=transpose)
```

CRITICAL for odd order derivatives (anti-symmetric stencil)

Not required for acoustic (self-adjoint equation)

Acoustic adjoint operator

Self-adjoint => Same stencil

Backward in time

Data as source

```
# Create a forward operator
super(AdjointOperator, self).__init__(nt, m.shape,
                                      stencils=stencil,
                                      subs=subs,
                                      spc_border=spc_order/2,
                                      time_order=time_order,
                                      forward=False,
                                      dtype=m.dtype,
                                      **kwargs)

# Insert source and receiver terms post-hoc
self.propagator.time_loop_stencils_a = source.read(m, u) + rec.add(u)
```

Generate code at runtime

Application developer

```
srca = Acoustic.Adjoint(dSyn - dObs)
```

Generated code

```
#include <cassert>
#include <cstdlib>
#include <cmath>
#include <iostream>
#include <fstream>
#include <vector>
#include <cstdio>
#include <string>
#include <inttypes.h>
#include <sys/time.h>
#include <math.h>
struct profiler
{
  double loop_stencils_a;
  double loop_body;
  double kernel;
} ;
struct flops
{
  long long loop_stencils_a;
  long long loop_body;
  long long kernel;
} ;
extern "C" int AdjointOperator(double *m_vec, double *u_vec, double *damp_vec, double *src_vec, float
*src_coords_vec, double *rec_vec, float *rec_coords_vec, long i1block, struct profiler *timings, struct flops *flops)
{
  double (*m)[280] = (double (*)[280]) m_vec;
  double (*u)[280][280] = (double (*)[280][280]) u_vec;
  double (*damp)[280] = (double (*)[280]) damp_vec;
  double (*src)[2] = (double (*)[2]) src_vec;
  float (*src_coords)[2] = (float (*)[2]) src_coords_vec;
  double (*rec)[101] = (double (*)[101]) rec_vec;
  float (*rec_coords)[2] = (float (*)[2]) rec_coords_vec;
  {
    struct timeval start_kernel, end_kernel;
    gettimeofday(&start_kernel, NULL);
    int t0;
    int t1;
    int t2;
    ;
    for (int i3 = 500; i3>0; i3-=1)
    {
      flops->kernel += 2.000000;
      {
        {
          t0 = (i3)%(3);
          t1 = (t0 + 1)%(3);
          t2 = (t1 + 1)%(3);
        }
        struct timeval start_loop_body, end_loop_body;
        gettimeofday(&start_loop_body, NULL);
        {
          for (int i1b = 1; i1b<279 - (278 % i1block); i1b+=i1block)
            for (int i1 = i1b; i1<i1b+i1block; i1++)
              {
```

Standardized verification for optimization

Rigorousness tests used as unit tests

Verify implementation

Allows stable continuous software integration with automated testing (TRAVIS)

Testing framework

Forward-adjoint test:

for any random $\mathbf{x} \in \text{span}(\mathbf{P}_s\mathbf{A}^T\mathbf{P}_r^T), \mathbf{y} \in \text{span}(\mathbf{P}_r\mathbf{A}\mathbf{P}_s^T)$

$$< \mathbf{P}_r\mathbf{A}\mathbf{P}_s^T\mathbf{x}, \mathbf{y} > - < \mathbf{x}, \mathbf{P}_s\mathbf{A}^T\mathbf{P}_r^T\mathbf{y} >= 0$$

$$\frac{< \mathbf{P}_r\mathbf{A}\mathbf{P}_s^T\mathbf{x}, \mathbf{y} >}{< \mathbf{x}, \mathbf{P}_s\mathbf{A}^T\mathbf{P}_r^T\mathbf{y} >} = 1$$

Gradient test:

for any small model perturbation $\mathbf{dm}$

$$\Phi_s(\mathbf{m} + h\mathbf{dm}) = \Phi_s(\mathbf{m}) + \mathcal{O}(h)$$

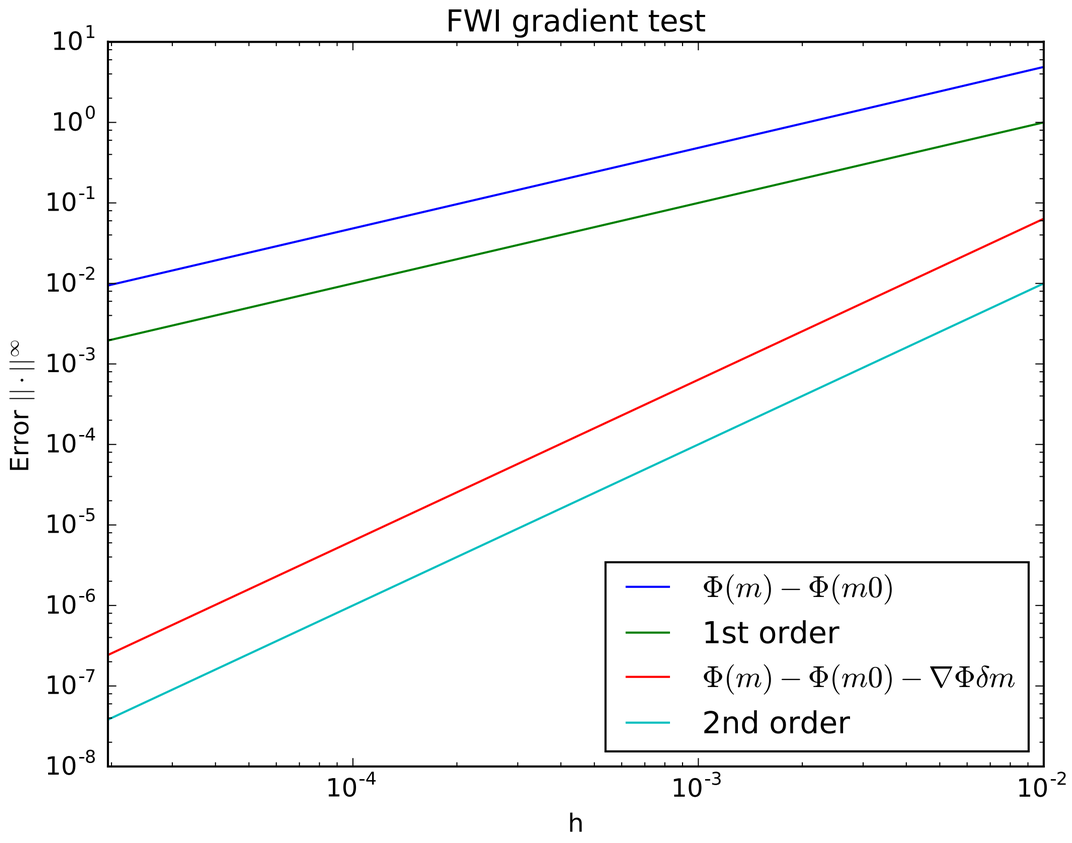
$$\Phi_s(\mathbf{m} + h\mathbf{dm}) = \Phi_s(\mathbf{m}) + h(\mathbf{J}[\mathbf{m}]^T\delta\mathbf{d})\mathbf{dm} + \mathcal{O}(h^2)$$

Adjoint test

Order	Dimension	$< \mathbf{F}\vec{x}, \vec{y} >$	$< \vec{x}, \mathbf{F}^t\vec{y} >$	Difference	ratio
2nd order	2D:	373323.7976042	373323.7975435	6.07169350e-05	1.0
4th order	2D:	340158.1486528	340158.1485253	0.00012756	1.0
6th order	2D:	341557.3948828	341557.3947399	0.00014287	1.0
8th order	2D:	358240.8513606	358240.8511932	0.00016741	1.0
10th order	2D:	393488.5561654	393488.5559270	0.00023841	1.0
12th order	2D:	439561.4005613	439561.4002034	0.00035794	1.0
2nd order	3D:	2.17496552	2.17496553	-1.23030883e-08	0.99999999
4th order	3D:	3.64447937	3.64447939	-2.13132316e-08	0.99999999
6th order	3D:	3.78730372	3.78730375	-2.22477072e-08	0.99999999
8th order	3D:	3.80286229	3.80286231	-2.23545817e-08	0.99999999
10th order	3D:	3.80557957	3.80557959	-2.23736993e-08	0.99999999
12th order	3D:	3.80318675	3.80318677	-2.23587757e-08	0.99999999

Gradient test

Check correct gradient implementation of FWI objective: $\Phi(\mathbf{m}) = \frac{1}{2}||\mathbf{d}_{obs} - \mathbf{P}\mathbf{A}(\mathbf{m})^{-1}\mathbf{q}||_2^2$



$\Leftarrow \quad \Phi(\mathbf{m} + h\mathbf{dm}) - \Phi(\mathbf{m})$

$\Leftarrow \quad \Phi(\mathbf{m} + h\mathbf{dm}) - \Phi(\mathbf{m}) - h(\mathbf{J}[\mathbf{m}]^T \delta\mathbf{d})\mathbf{dm}$

Future tests

Discretization tests

- test the finite-difference scheme generated against polynomials

Accuracy test

- test against analytical solution
- method of manufactured solutions (MMS)

...

Computational performance and optimization

MS84
Domain-Specific Abstractions for Full-Waveform Inversion
Symbolic Math for Automated Fast Finite Difference Computations

Navjot Kukreja, Imperial College London, United Kingdom

MS44
Efficiency of High-Order Methods on the 2nd Generation Intel Xeon Phi Processor

Vectorization and Locality Optimizations for Seismic Imagining Methods Through Automated Code Generation

Fabio Luporini, Imperial College, United Kingdom; Gerard J Gorman, Paul Kelly, and Michael Lange, Imperial College London, United Kingdom

Slides available upon request

Conclusions

Flexible physics with a simple finite-difference interface

- weeks, months, ... of development time saved
- write your own problem

Minimal coding required for geophysicists/mathematicians

- domain specialists only focus on their own problem
- improves collaborations with a high-level common ground

Simulation for inversion with adjoint-aware discretization

- not only restricted to modelling
- adjoints are inherently hard, specially for complicated physics

And all advantages of code generation with Devito (performance, architecture portability, ...)

References and link

OPESCI : http://www.opesci.org/

- documentation
- list of publications
- examples
- link to DEVITO source code

SLIM : https://www.slim.eos.ubc.ca/

- examples
- documentation
- list of publications

Acknowledgements

This research was carried out as part of the SINBAD project with the support of the member organizations of the SINBAD Consortium in collaboration with the Imperial College London Intel Parallel Computing Centre