The power of abstraction in Computational Exploration Seismology

Felix J. Herrmann

Smoky Mountains Computational Sciences & Engineering Conference
Gatlinburg, August 29, 2018

SLIM
Georgia Institute of Technology

DownUnder GeoSolutions

The power of abstraction in Computational Exploration Seismology

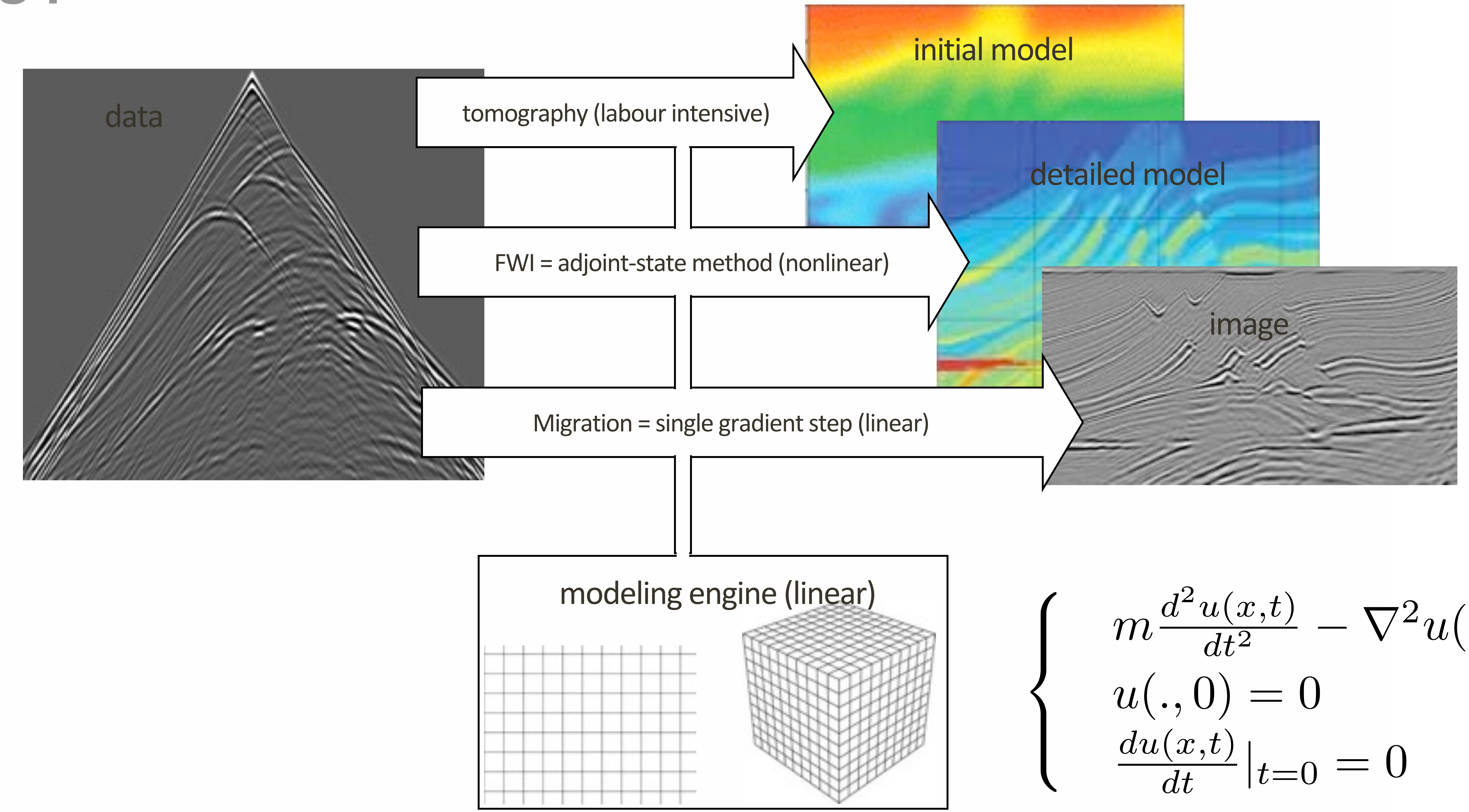
Gerard Gorman, Jan Hückelheim, Keegan Lensink, Paul Kelly, Navjot Kukreja, Henryk Modzelewski, Michael Lange, Mathias Louboutin, Fabio Luporini, Ali SiahKoohi, Phillipp Witte

PETROBRAS woodside ConocoPhillips Schlumberger

SLIM
Georgia Institute of Technology

Imperial College London intel SENAI CIMATEC SISTEMA FIEB Federação das Indústrias do Estado da Bahia

Big picture



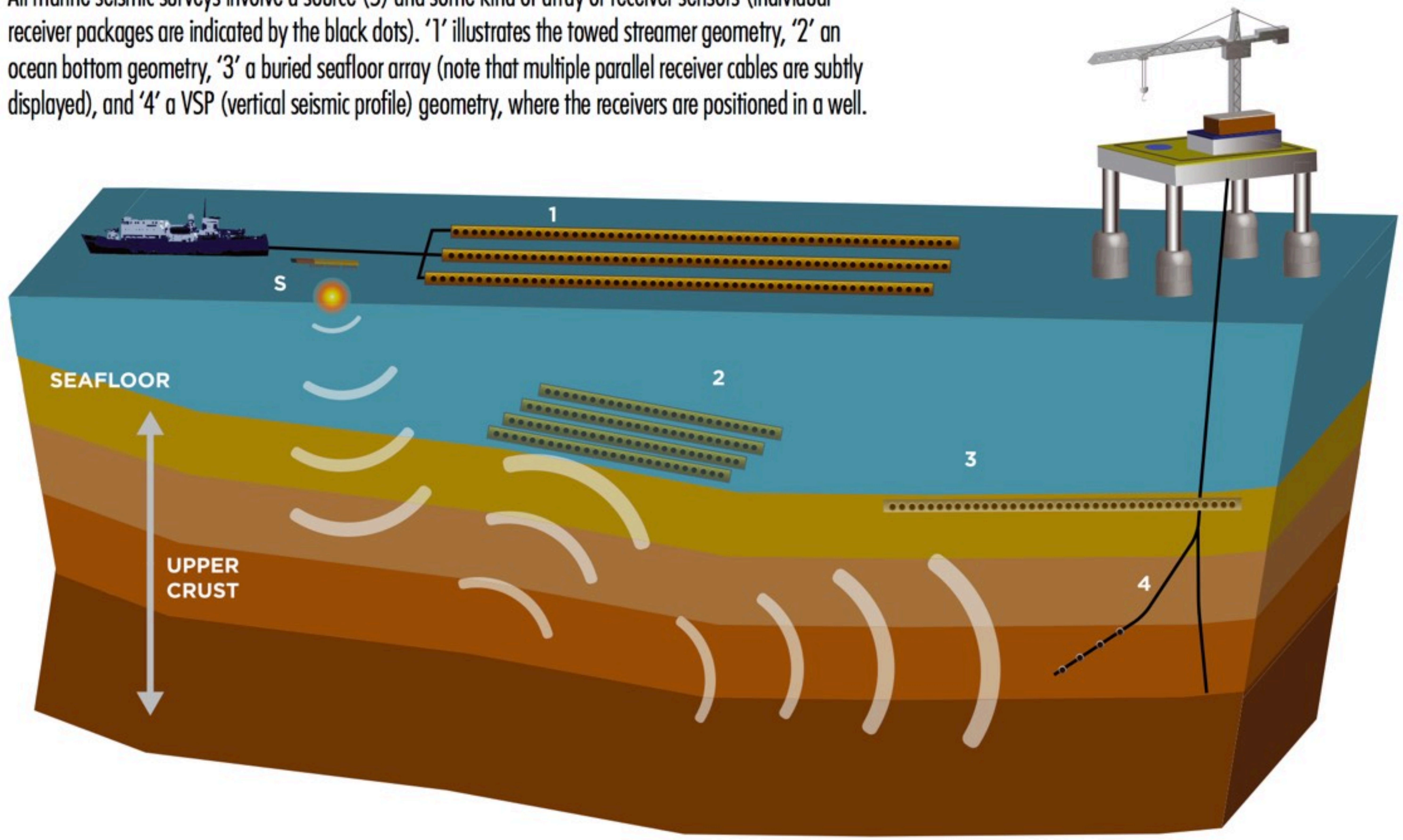
$$\begin{cases} m\frac{d^2u(x,t)}{dt^2} - \nabla^2 u(x,t) = q \\ u(.,0) = 0 \\ \frac{du(x,t)}{dt}\big|_{t=0} = 0 \end{cases}$$

Seismic imaging

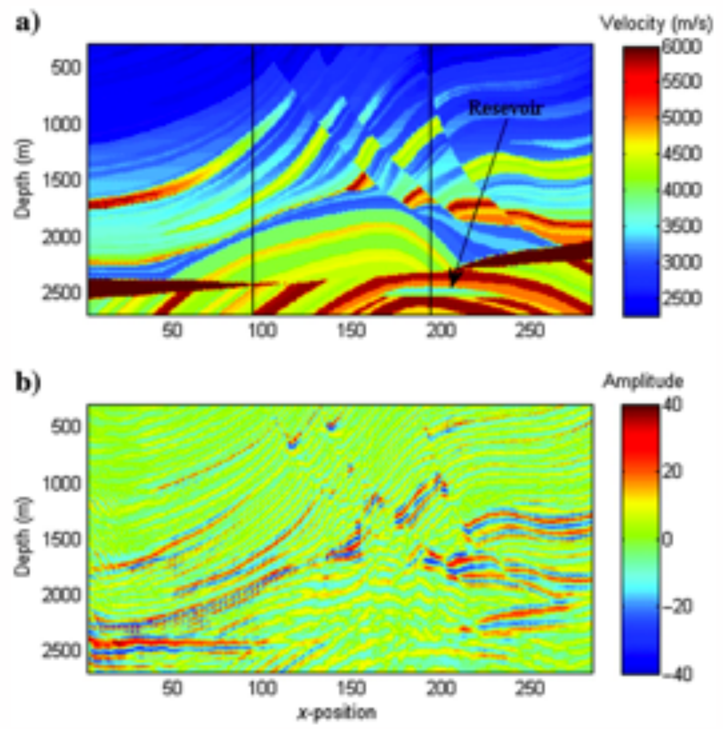
Infer 3D images from massive multi-experiment data:

- $\mathcal{O}(10^9)$ unknowns
- $\mathcal{O}(10^{15})$ datapoints
- propagate $\mathcal{O}(10^2)$ wavelengths

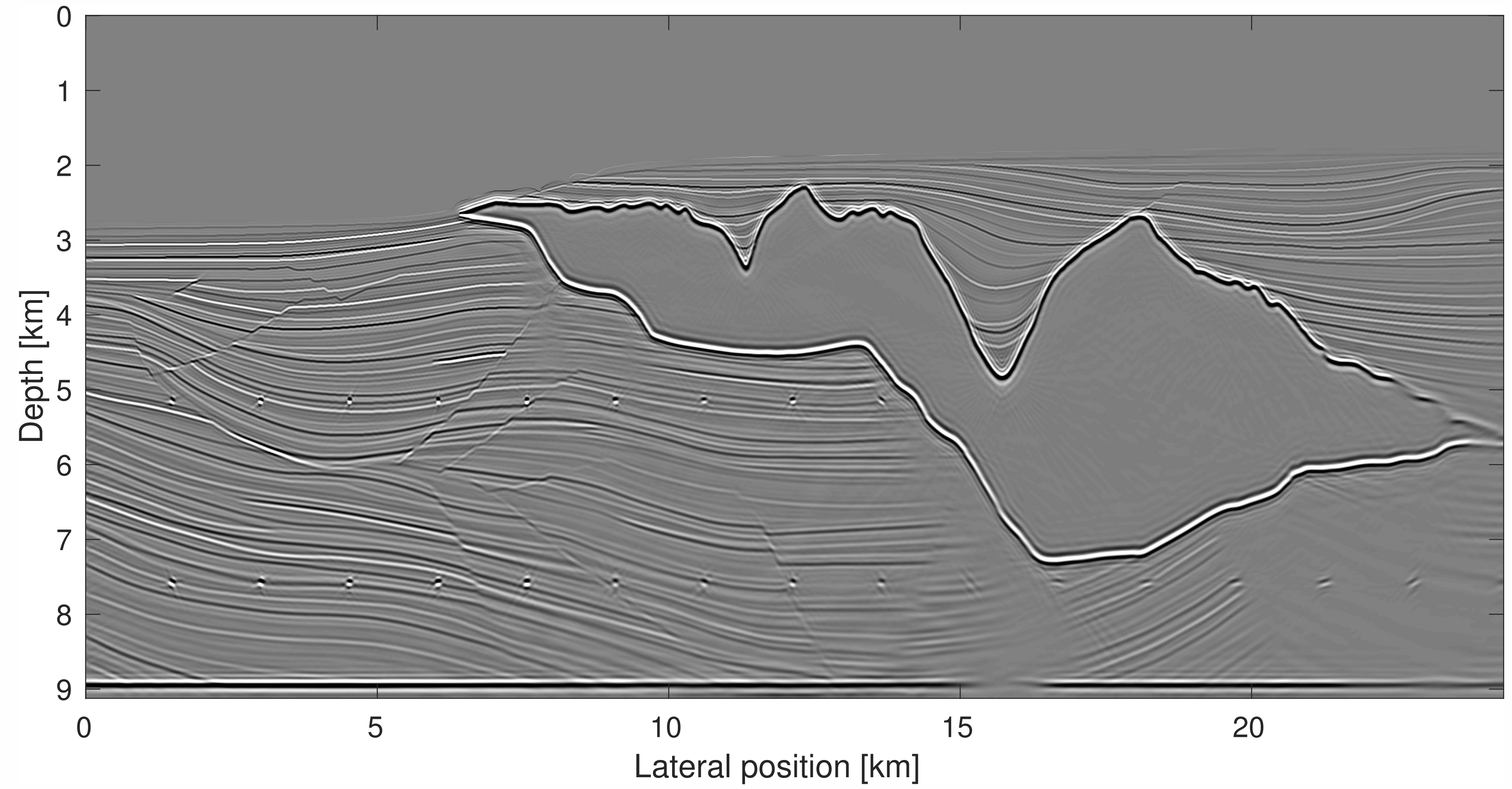
All marine seismic surveys involve a source (S) and some kind of array or receiver sensors (individual receiver packages are indicated by the black dots). '1' illustrates the towed streamer geometry, '2' an ocean bottom geometry, '3' a buried seafloor array (note that multiple parallel receiver cables are subtly displayed), and '4' a VSP (vertical seismic profile) geometry, where the receivers are positioned in a well.



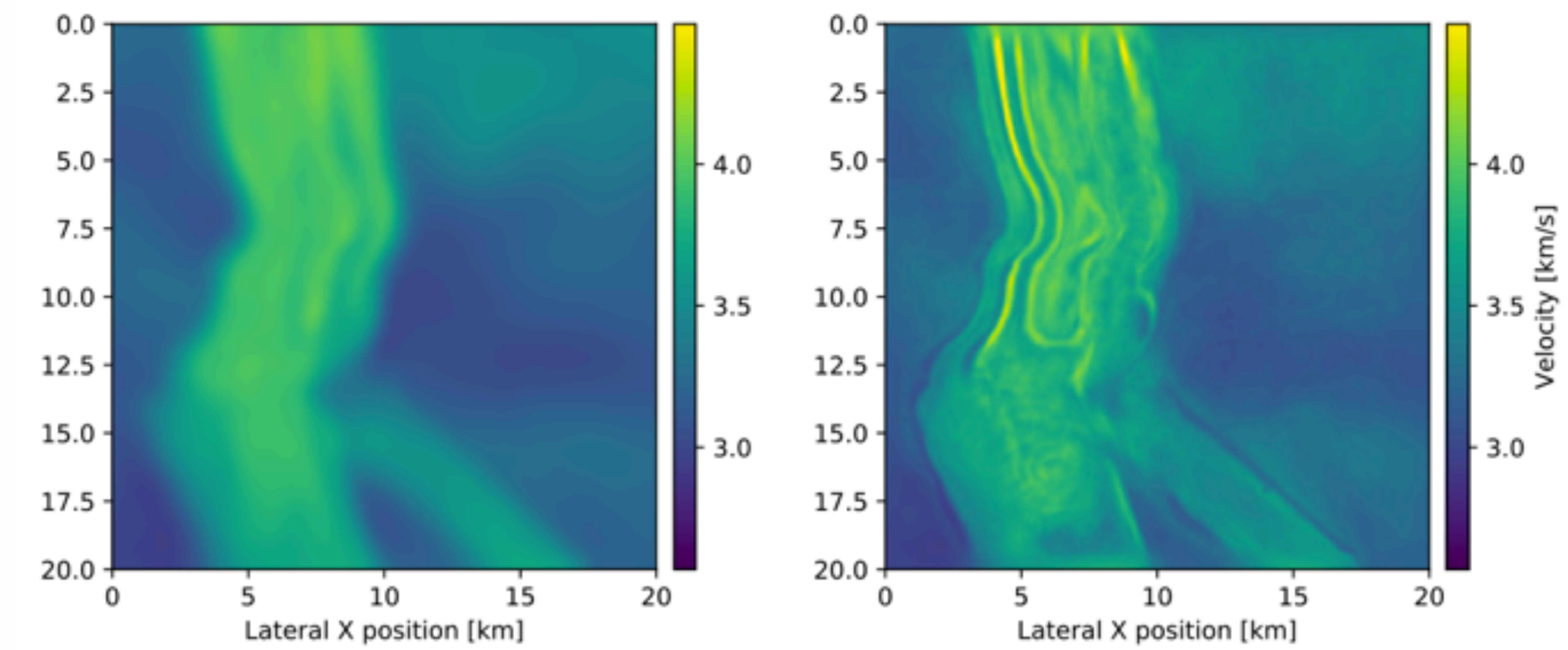
- **>10k time steps**
- **acoustic only**
- **elastic much larger**
- **extreme compute & IO**



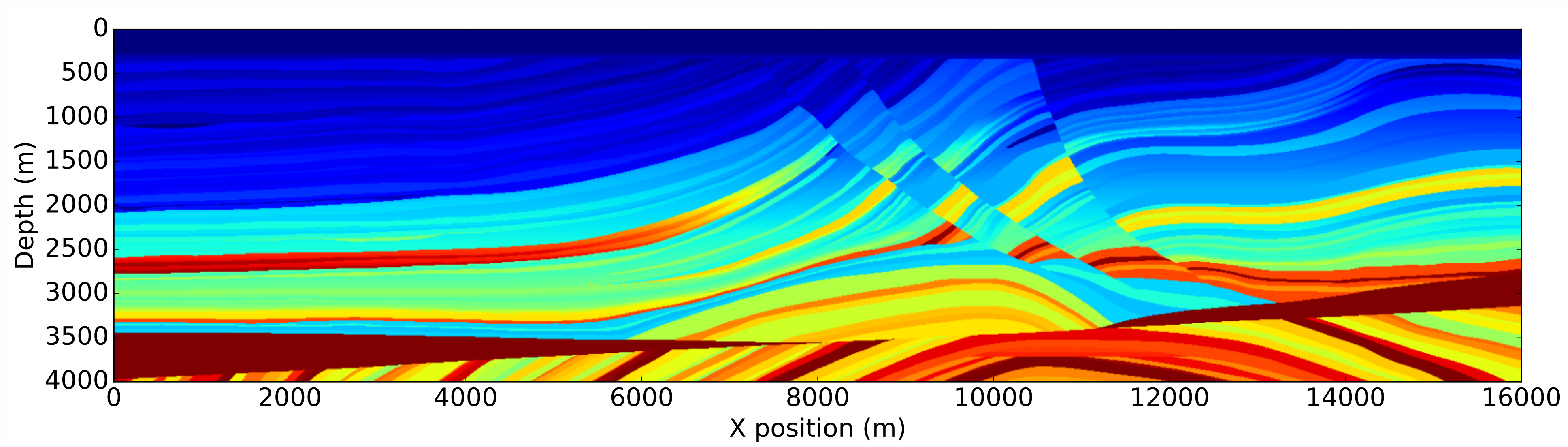
Tiny marmousi
62k gridpoints
1.5MFlop/time-step



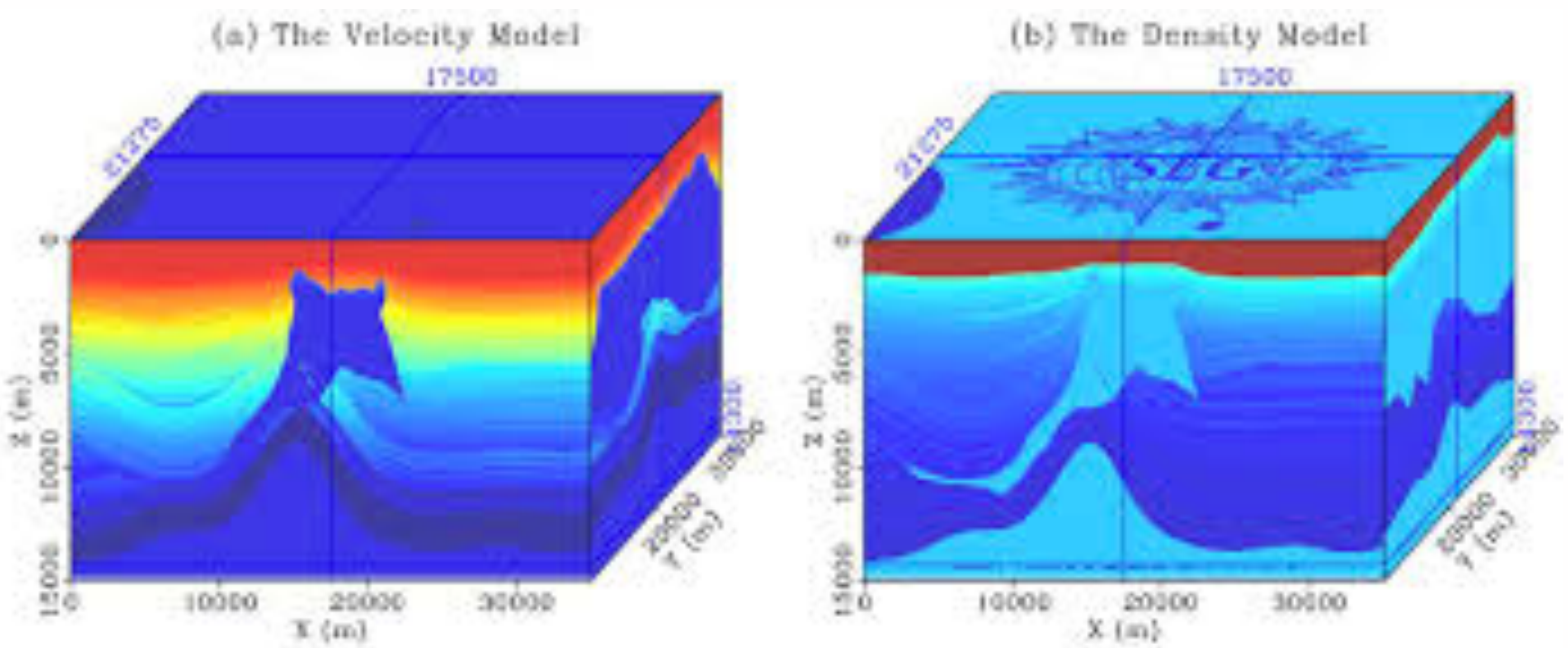
Sigsbee
2.2M gridpoints
56MFlop/time-step



3D overthrust
222M gridpoints
6GFlop/time-step



Full marmousi
640k gridpoints
15MFlop/time-step



SEAM
2.2G gridpoints
56GFlop/time-step

Wave-equation based inversion & learning

Large-scale learning problem:

training data

generator

10k "CNNs"

$$\underset{\mathbf{m}}{\text{minimize}}\,\Phi(\mathbf{m}) = \frac{1}{M}\sum_{i=1}^{M}\phi_i(\mathbf{m}) = \frac{1}{2}\|F_i(\mathbf{m})\mathbf{q}_i - \mathbf{d}_i\|^2, \quad F_i(\mathbf{m})\mathbf{q}_i = P_i A^{-1}(\mathbf{m})\mathbf{q}_i$$

parameters

input

"network"

- forward model = generative convolutional network
- adjoint state = back propagation
- simultaneous sourcing = sketching + stochastic gradients
- like learning 5k+ video where we care about the parameters

Research questions

"How can we exploit abstractions & connections w/ machine learning?"

- manage complexities of often monolythic code bases
- be more agile, reduce development time & costs
- avoid interference of meta data w/ linear algebra & optimization
- use ML to remove growing computational costs

Proposed solution

DEVITO – Domain specific language for stencil-based finite difference code generation for PDEs w/ explicit time stepping in Python using SymPy.

https://www.devitoproject.org

Design motivation

- Stencil codes:
  - Time consuming
  - Complex
  - Architecture dependent

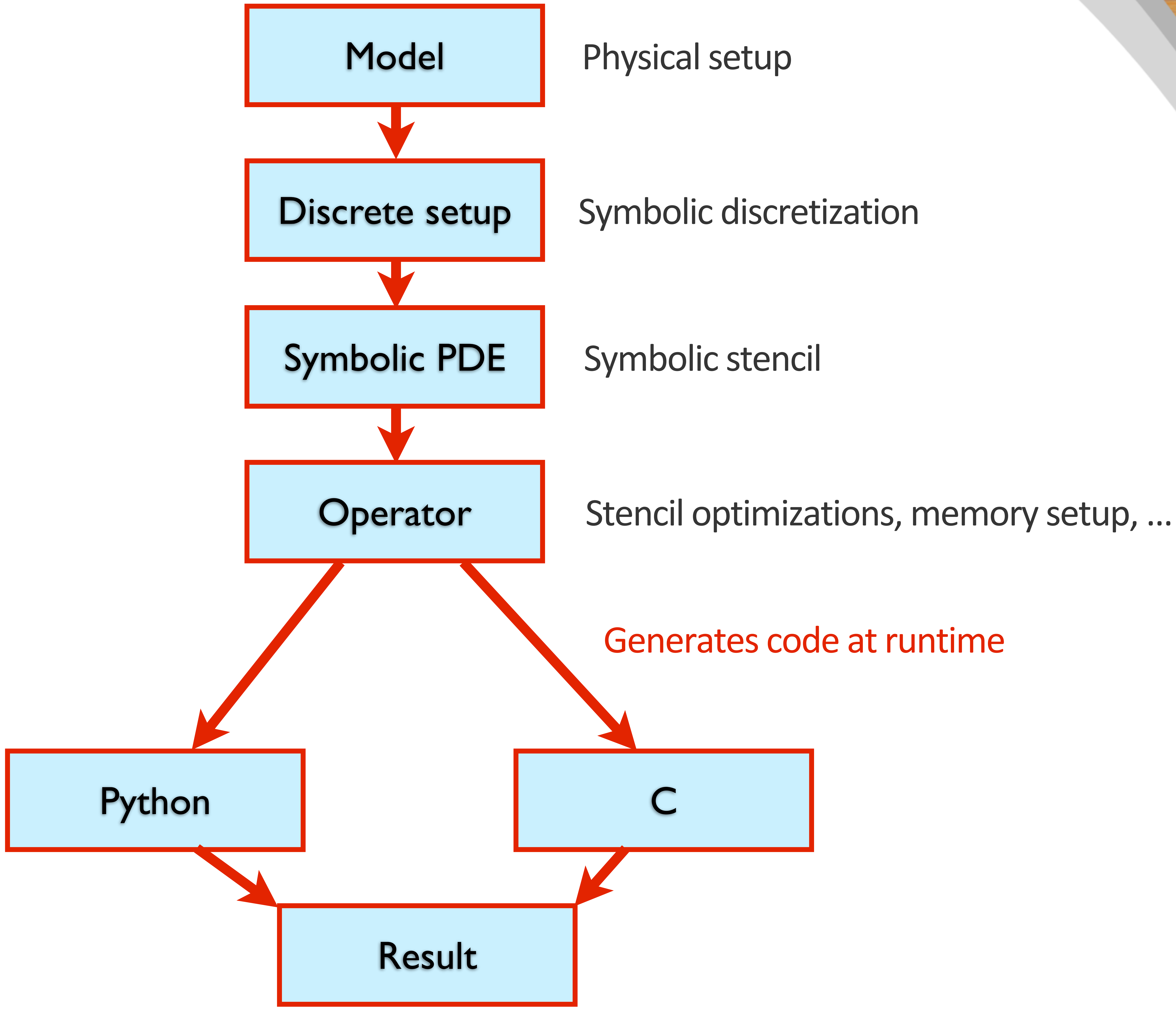
Finite-difference DSL

- **Separation of Concerns:**
  - Geophysicists focus on physics
  - Computer scientists focus on software
  - Mathematicians focus on numerical analysis

Devito

Michael Lange, Navjot Kukreja, Mathias Louboutin, Fabio Luporini, Felippe Vieira Zacarias, Vincenzo Pandolfo, Paulius Velesko, Paulius Kazakas, and Gerard Gorman,

“Devito: Towards a generic finite difference DSL using symbolic python”, in 6th Workshop on Python for High-Performance and Scientific Computing, 2016, p. 67-75.



Raising the level of abstraction

$$m\frac{\partial^2 u}{\partial t^2} + \eta\frac{\partial u}{\partial t} - \Delta u = 0$$

```
void kernel(…) {
  …
  <impenetrable code with crazy
   performance optimizations>
  …
}
```

Raising the level of abstraction

$$m\frac{\partial^2 u}{\partial t^2} + \eta\frac{\partial u}{\partial t} - \Delta u = 0$$

```
void kernel(…) {
  …
   <impenetrable code with crazy
   performance optimizations>
  …
}
```

Raising the level of abstraction

$$m\frac{\partial^2 u}{\partial t^2} + \eta\frac{\partial u}{\partial t} - \Delta u = 0$$

Raising the level of abstraction

$$m\frac{\partial^2 u}{\partial t^2} + \eta\frac{\partial u}{\partial t} - \Delta u = 0$$

```
eqn = m * u.dt2 + eta * u.dt - u.laplace
solve(eqn, u.forward)
```

Raising the level of abstraction

$$m\frac{\partial^2 u}{\partial t^2} + \eta\frac{\partial u}{\partial t} - \Delta u = 0$$

```
eqn = m * u.dt2 + eta * u.dt - u.laplace
solve(eqn, u.forward)
```

```
void kernel(…) { … }
```

Raising the level of abstraction

$$m\frac{\partial^2 u}{\partial t^2} + \eta\frac{\partial u}{\partial t} - \Delta u = 0$$

Devito

```
eqn = m * u.dt2 + eta * u.dt - u.laplace
solve(eqn, u.forward)
```

```
void kernel(…) { … }
```

Flexibility in space/time discretization

```
u = TimeFunction(…, space_order=so)
eqn = m * u.dt2 + eta * u.dt - u.laplace
solve(eqn, u.forward)
```

so=4

```
  for (int time = time_m, t0 = (time)%(3), t1 = (time + 1)%(3), t2 = (time +
2)%(3); time <= time_M; time += 1, t0 = (time)%(3), t1 = (time + 1)%(3), t2 =
(time + 2)%(3)) {
    for (int x = x_m; x <= x_M; x += 1) {
      for (int y = y_m; y <= y_M; y += 1) {
        for (int z = z_m; z <= z_M; z += 1) {
          u[t1][x + 4][y + 4][z + 4] = 2*pow(dt,
3)*(-2.08333333333333e-4F*u[t0][x + 2][y + 4][z + 4] +
3.33333333333333e-3F*u[t0][x + 3][y + 4][z + 4] - 2.08333333333333e-4F*u[t0]
[x + 4][y + 2][z + 4] + 3.33333333333333e-3F*u[t0][x + 4][y + 3][z + 4] -
2.08333333333333e-4F*u[t0][x + 4][y + 4][z + 2] + 3.33333333333333e-3F*u[t0]
[x + 4][y + 4][z + 3] - 1.875e-2F*u[t0][x + 4][y + 4][z + 4] +
3.33333333333333e-3F*u[t0][x + 4][y + 4][z + 5] - 2.08333333333333e-4F*u[t0]
[x + 4][y + 4][z + 6] + 3.33333333333333e-3F*u[t0][x + 4][y + 5][z + 4] -
2.08333333333333e-4F*u[t0][x + 4][y + 6][z + 4] + 3.33333333333333e-3F*u[t0]
[x + 5][y + 4][z + 4] - 2.08333333333333e-4F*u[t0][x + 6][y + 4][z + 4])/
(pow(dt, 2)*damp[x + 1][y + 1][z + 1] + 2*dt*m[x + 4][y + 4][z + 4]) +
pow(dt, 2)*damp[x + 1][y + 1][z + 1]*u[t2][x + 4][y + 4][z + 4]/(pow(dt,
2)*damp[x + 1][y + 1][z + 1] + 2*dt*m[x + 4][y + 4][z + 4]) + 4*dt*m[x + 4][y
+ 4][z + 4]*u[t0][x + 4][y + 4][z + 4]/(pow(dt, 2)*damp[x + 1][y + 1][z + 1]
+ 2*dt*m[x + 4][y + 4][z + 4]) - 2*dt*m[x + 4][y + 4][z + 4]*u[t2][x + 4][y +
4][z + 4]/(pow(dt, 2)*damp[x + 1][y + 1][z + 1] + 2*dt*m[x + 4][y + 4][z +
4]);
        }
      }
    }
  }
```

so=12

```
  for (int time = time_m, t0 = (time)%(3), t1 = (time + 1)%(3), t2 = (time +
2)%(3); time <= time_M; time += 1, t0 = (time)%(3), t1 = (time + 1)%(3), t2 =
(time + 2)%(3)) {
    for (int x = x_m; x <= x_M; x += 1) {
      for (int y = y_m; y <= y_M; y += 1) {
        for (int z = z_m; z <= z_M; z += 1) {
          u[t1][x + 12][y + 12][z + 12] = 2*pow(dt,
3)*(-1.5031265031265e-7F*u[t0][x + 6][y + 12][z + 12] +
2.5974025974026e-6F*u[t0][x + 7][y + 12][z + 12] - 2.23214285714286e-5F*u[t0][x
+ 8][y + 12][z + 12] + 1.32275132275132e-4F*u[t0][x + 9][y + 12][z + 12] -
6.69642857142857e-4F*u[t0][x + 10][y + 12][z + 12] + 4.28571428571429e-3F*u[t0]
[x + 11][y + 12][z + 12] - 1.5031265031265e-7F*u[t0][x + 12][y + 6][z + 12] +
2.5974025974026e-6F*u[t0][x + 12][y + 7][z + 12] - 2.23214285714286e-5F*u[t0][x
+ 12][y + 8][z + 12] + 1.32275132275132e-4F*u[t0][x + 12][y + 9][z + 12] -
6.69642857142857e-4F*u[t0][x + 12][y + 10][z + 12] + 4.28571428571429e-3F*u[t0]
[x + 12][y + 11][z + 12] - 1.5031265031265e-7F*u[t0][x + 12][y + 12][z + 6] +
2.5974025974026e-6F*u[t0][x + 12][y + 12][z + 7] - 2.23214285714286e-5F*u[t0][x
+ 12][y + 12][z + 8] + 1.32275132275132e-4F*u[t0][x + 12][y + 12][z + 9] -
6.69642857142857e-4F*u[t0][x + 12][y + 12][z + 10] + 4.28571428571429e-3F*u[t0]
[x + 12][y + 12][z + 11] - 2.23708333333333e-2F*u[t0][x + 12][y + 12][z + 12] +
4.28571428571429e-3F*u[t0][x + 12][y + 12][z + 13] - 6.69642857142857e-4F*u[t0]
[x + 12][y + 12][z + 14] + 1.32275132275132e-4F*u[t0][x + 12][y + 12][z + 15] -
2.23214285714286e-5F*u[t0][x + 12][y + 12][z + 16] + 2.5974025974026e-6F*u[t0]
[x + 12][y + 12][z + 17] - 1.5031265031265e-7F*u[t0][x + 12][y + 12][z + 18] +
4.28571428571429e-3F*u[t0][x + 12][y + 13][z + 12] - 6.69642857142857e-4F*u[t0]
[x + 12][y + 14][z + 12] + 1.32275132275132e-4F*u[t0][x + 12][y + 15][z + 12] -
2.23214285714286e-5F*u[t0][x + 12][y + 16][z + 12] + 2.5974025974026e-6F*u[t0]
[x + 12][y + 17][z + 12] - 1.5031265031265e-7F*u[t0][x + 12][y + 18][z + 12] +
4.28571428571429e-3F*u[t0][x + 13][y + 12][z + 12] - 6.69642857142857e-4F*u[t0]
[x + 14][y + 12][z + 12] + 1.32275132275132e-4F*u[t0][x + 15][y + 12][z + 12] -
2.23214285714286e-5F*u[t0][x + 16][y + 12][z + 12] + 2.5974025974026e-6F*u[t0]
[x + 17][y + 12][z + 12] - 1.5031265031265e-7F*u[t0][x + 18][y + 12][z + 12])/
(pow(dt, 2)*damp[x + 1][y + 1][z + 1] + 2*dt*m[x + 12][y + 12][z + 12]) +
```

Devito for inversion

Adjoint PDE

Gradients/Sensitivities

Adjoint state gradient

FWI objective

$$\Phi(\mathbf{m}) = \frac{1}{2}||\,\mathbf{P}_r\mathbf{A}^{-1}(\mathbf{m})\mathbf{P}_s^T\mathbf{q} - \mathbf{d}||_2^2$$

with gradient with respect to $\mathbf{m}$

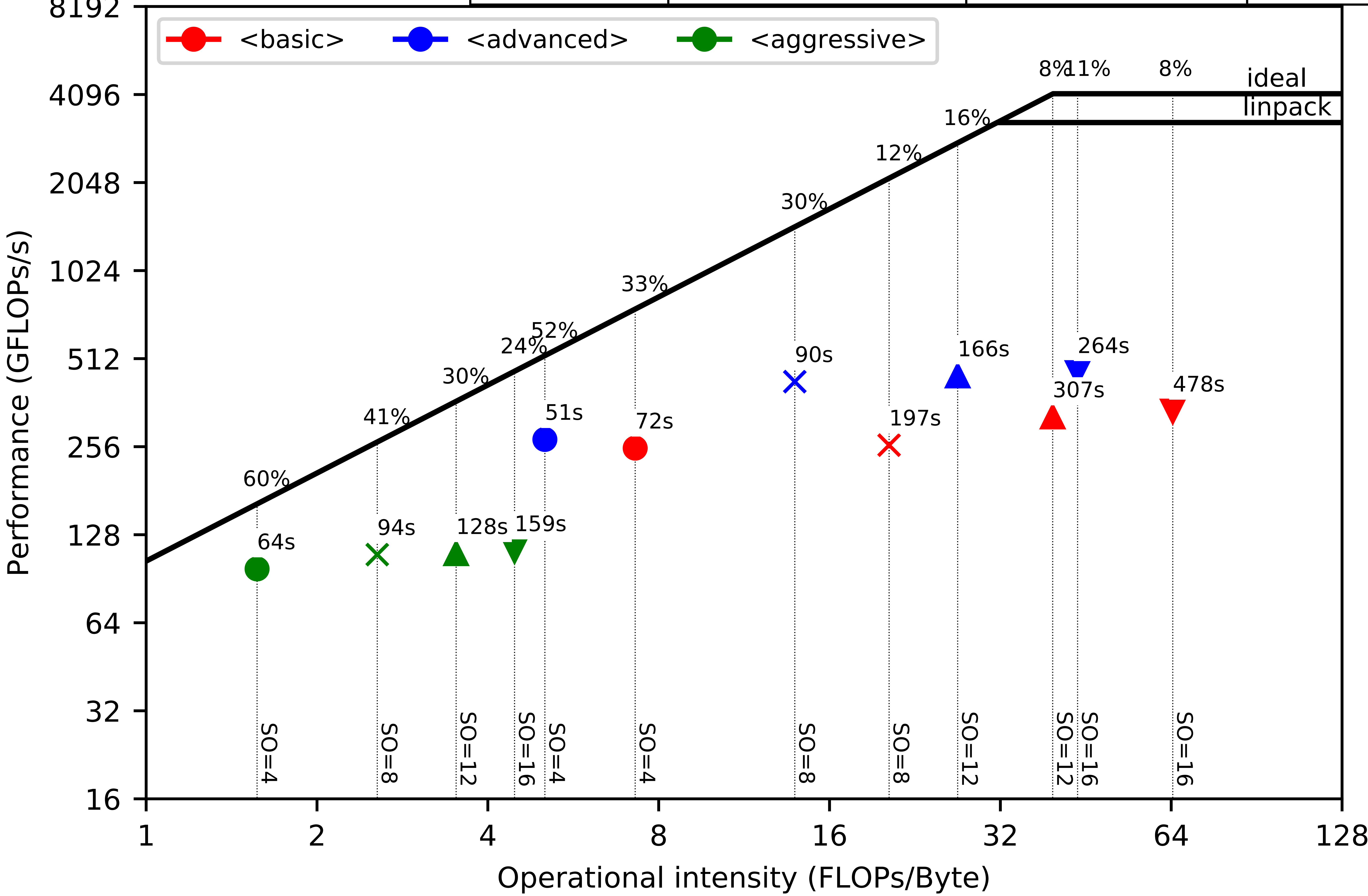
$$\nabla\Phi(\mathbf{m}) = -(\frac{d^2\mathbf{u}}{d\mathbf{t}^2})^T\mathbf{A}(\mathbf{m})^{-T}\mathbf{P}_r^T(\mathbf{P}_r\mathbf{A}(\mathbf{m})^{-1}\mathbf{P}_s\mathbf{q} - \mathbf{d})$$

requires adjoint wave-equation

3D TTI performance w/roofline models:

- 512x512x512 grid points
- 1000ms propagation (416 time steps)

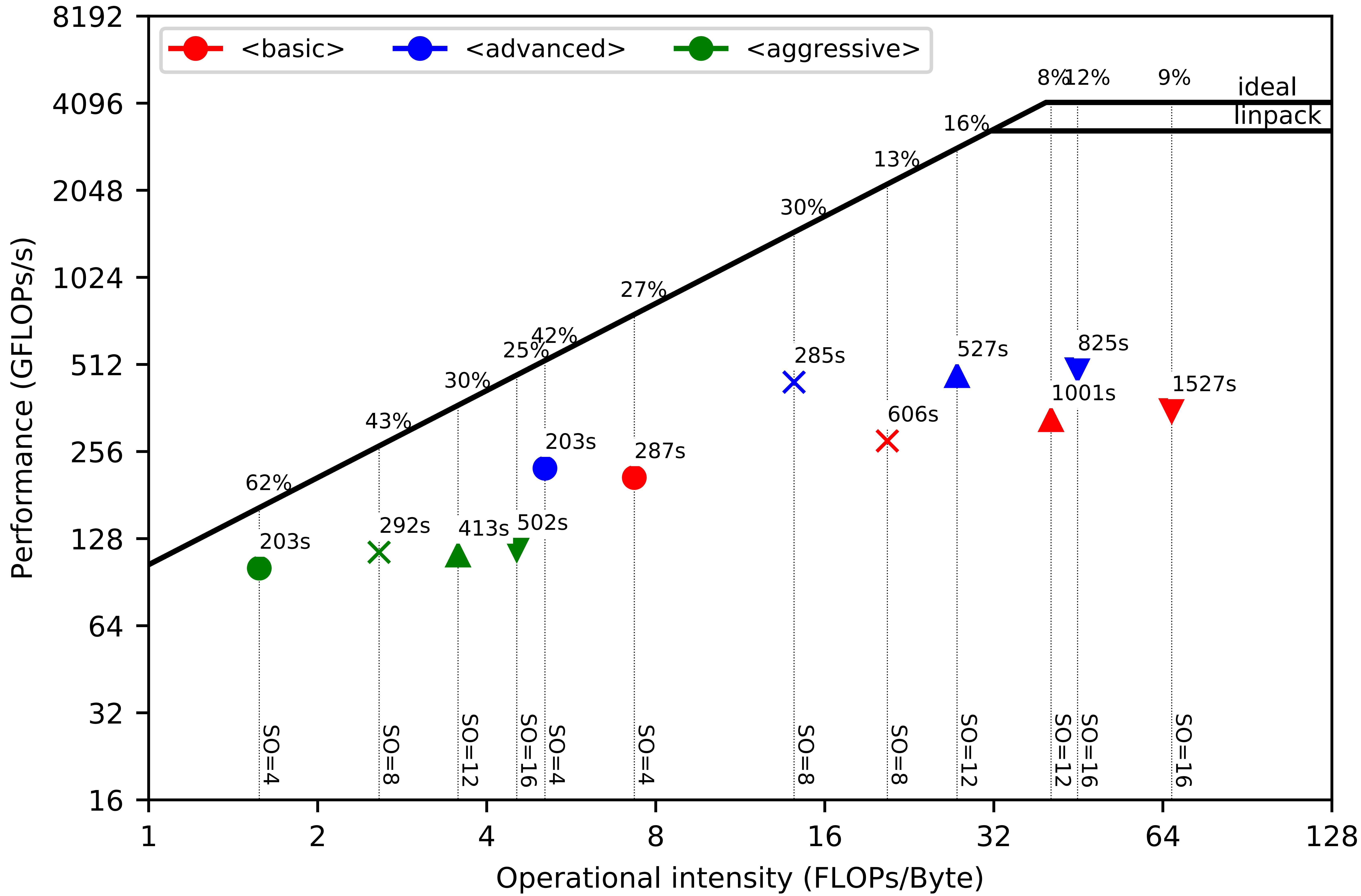
SO	Flops basic	Flops advanced	Flops aggressive
2	299	260	102
4	857	707	168
8	1703	1370	234
16	2837	2249	300



3D TTI performance:

- 768x768x768 grid points
- 1000ms propagation (416 time steps)

We scale linearly!



Observations

Highly abstracted JIT compiler w/ pathways to

- C, MPI+OpenMP+C, CUDA, MPI+CUDA…
- backend w/ YASK implemented (3 X speed up on Xeon Phi)
- backend w/ OPS library for CPU-GPU(+MPI)

Take home message: getting the abstraction right is key!

- highly productive environment w/ flexibility w.r.t. discretization (stencil)
- connection w/ linear algebra
- parallel IO & access to meta data
- intuitive data parallelism to work w/ multiple instances

https://github.com/intel/yask ***https://github.com/OP-DSL/OPS***

Philipp A. Witte, Mathias Louboutin, Navjot Kukreja, Fabio Luporini, Michael Lange, Gerard J. Gorman, and Felix J. Herrmann, "**A large-scale framework for symbolic implementations of seismic inversion algorithms in Julia**". 2018.

SLIM

Proposed solution

JUDI – Domain specific language for linear algebra abstractions, data parallelism & meta data in Julia

https://github.com/slimgroup/JUDI.jl

JUDI – true vertical integration

Julia

linear opeartors, data containers, IO

↓ parallel modeling function

parallelization: distribute sources, data

↓ serial modeling function

interface to Devito (Python)

↓

Python

Devito: symbolic definition of PDE

↓ automatic code generation and JIT compilation

C

solve PDE w/ OpenMP parallelism

students

math/optimizers/cs/ seismic practitioners

students

CS/math/physics people

polyhydral compiler people

SLIM

Linear operators & data containers

Challenges for time-domain modeling/inversion:

- seismic data are multidimensional with lots of meta data
- simply vectorizing the input data not an option
- data typically too big to fit in memory

$$\mathbf{d} = \mathcal{P}_r \, \mathbf{F} \, \mathcal{P}_s^\top \mathbf{q}$$

Linear operators & data containers

Challenges of this approach for time-domain modeling/inversion:

- seismic data is multidimensional volume with meta data
- simply vectorizing the input data not an option
- data typically too big to fit in memory

$$\mathbf{d} = \mathcal{P}_r \, \mathbf{F} \, \mathcal{P}_s^\top \mathbf{q}$$

- cannot be formed explicitly
- need physical information (model, source/receiver locations)

Linear operators & data containers

Challenges of this approach for time-domain modeling/inversion:

- seismic data is multidimensional volume with meta data
- simply vectorizing the input data not an option
- data typically too big to fit in memory

$$\mathbf{d} = \mathcal{P}_r \, \mathbf{F} \, \mathcal{P}_s^\top \mathbf{q}$$

- cannot be kept in memory
- not a vector
- contains header information

- cannot be formed explicitly
- need physical information (model, source/receiver locations)

Linear operators & data containers

Abstract in-core & out-of-core data vectors:

- inspired by iWave, RVL and others (Symes, Padula, Trilinos)
- can be formed directly from single/multiple SEG-Y files
- parallel read/write chunks of data via compressed lookup table

```
julia> container = segy_scan(pwd(), "overthrust_shots", ["GroupX","GroupY"]);
Scanning ... /home/slim/pwitte/overthrust_shots_41_60.segy
Scanning ... /home/slim/pwitte/overthrust_shots_21_40.segy
Scanning ... /home/slim/pwitte/overthrust_shots_61_80.segy
Scanning ... /home/slim/pwitte/overthrust_shots_1_20.segy
Scanning ... /home/slim/pwitte/overthrust_shots_81_97.segy

julia> d = joData(container)
(opesciSLIM.TimeModeling.joData{Float32}, "Julia seismic data container", 15029763, 1)

julia> size(d)
(15029763, 1)

julia> norm(d)
7371.35f0

julia> dot(d,d)
5.432854f7

julia> typeof(d.data[1])
SeisIO.SeisCon
```

(Instructional video at: https://www.youtube.com/watch?v=tx530QOPeZo)

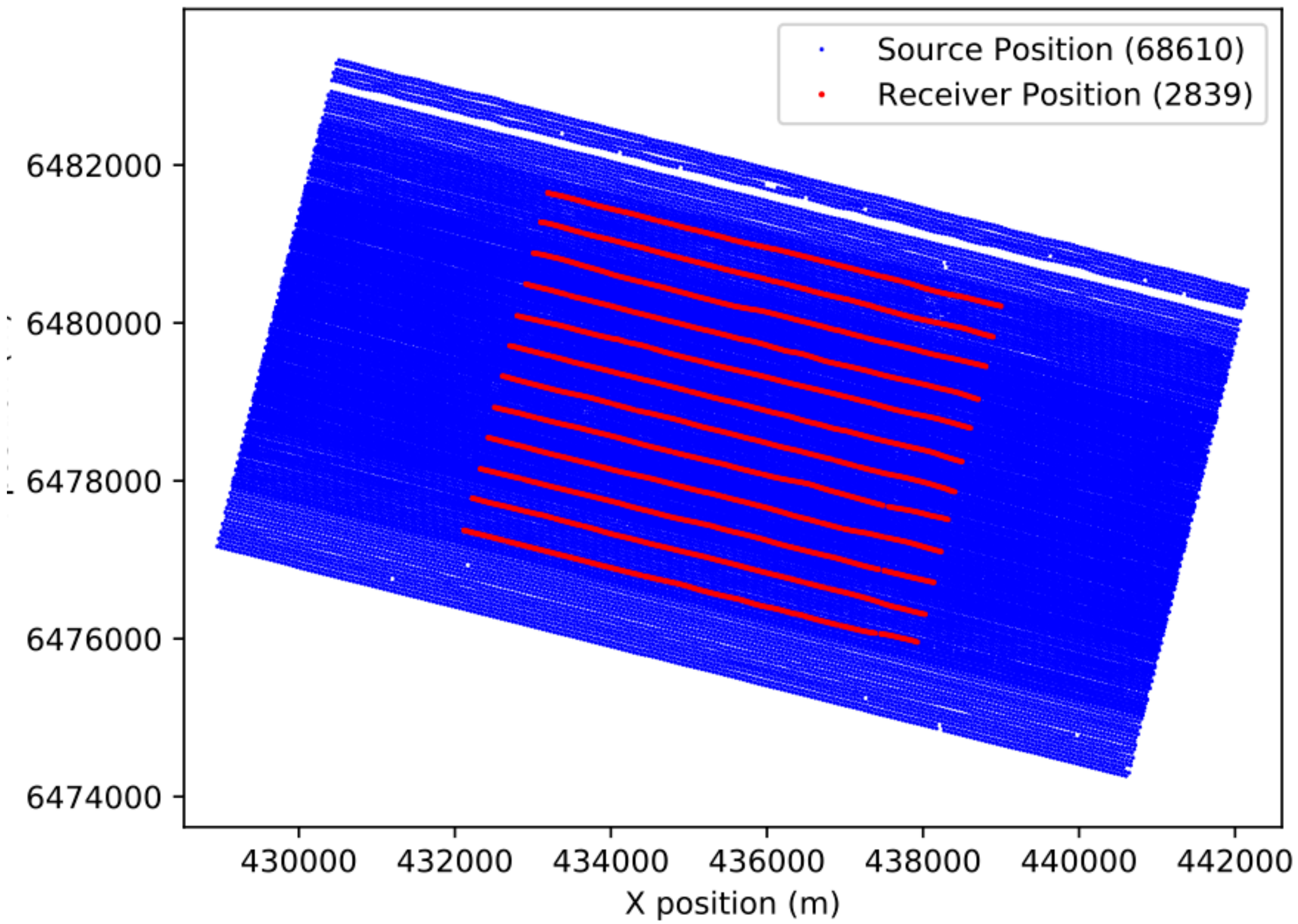
Linear operators & data containers

Massive metadata

Total data size : 110Gb

Metadata: 1Gb

Small field data geometry



Linear operators & data containers

Matrix-free linear operators

- read necessary meta information from data objects
- use like explicit matrices

```
julia> F = joModeling(info,model0)
(opesciSLIM.TimeModeling.joModeling{Float32,Float32}, "forward wave equation", 27566740206, 27566740206)

julia> Pr = joProjection(info,d.geometry)
(opesciSLIM.TimeModeling.joProjection{Float32,Float32}, "restriction operator", 15029763, 27566740206)

julia> Ps = joProjection(info,q.geometry)
(opesciSLIM.TimeModeling.joProjection{Float32,Float32}, "restriction operator", 72847, 27566740206)

julia> d_pred = Pr*F*Ps'*q
```

Example: LS-RTM w/ serial & parallel SGD

Algorithm 1 Preconditioned LS-RTM with SGD

for $j = 1$ to n **do**

$\mathbf{r}_j = \mathbf{M}_l^{-1}\mathbf{J}_{r(j)}\mathbf{M}_r^{-1}\mathbf{x}_j - \mathbf{M}_l^{-1}\delta\mathbf{d}_{r(j)}$

$\mathbf{g}_j = \mathbf{M}_r^{-\top}\mathbf{J}_{r(j)}^{\top}\mathbf{M}_l^{-\top}\mathbf{r}_j$

$t_j = \frac{\|\mathbf{r}_j\|^2}{\|\mathbf{g}_j\|^2}$

$\mathbf{x}_{j+1} = \mathbf{x}_j - t_j\mathbf{g}_j$

end for

```
# Stochastic gradient descent
batchsize = 10
niter = 32

for j=1:niter
   # Select batch
   idx = randperm(dD.nsrc)[1:batchsize]
   Jsub = subsample(J,idx)
   dsub = subsample(dD,idx)

   # Compute residual and gradient
   r = Ml*Jsub*Mr*x - Ml*dsub
   g = Mr'*Jsub'*Ml'*r

   # Step size and update variable
   t = norm(r)^2/norm(g)^2
   x -= t*g
end
```

Example: LS-RTM w/ serial & parallel SGD

Algorithm 2 Preconditioned LS-RTM with elastic average SGD

for $j = 1$ to n **do**
  for $k = 1$ to p **do**
    $\mathbf{r}_j = \mathbf{M}_l^{-1}\mathbf{J}_{jk}\mathbf{M}_r^{-1}\mathbf{x}_j^k - \mathbf{M}_l^{-1}\delta\mathbf{d}_{jk}$
    $\mathbf{g}_j = \mathbf{M}_r^{-\top}\mathbf{J}_{jk}^{\top}\mathbf{M}_l^{-\top}\mathbf{r}_j$ and
    $\mathbf{x}_{j+1}^k = \mathbf{x}_j^k - \eta\mathbf{g}_j^k(\mathbf{x}_j^k) - \alpha(\mathbf{x}_j^k - \tilde{\mathbf{x}}_j)$
  end for
  $\tilde{\mathbf{x}}_{j+1} = (1-\beta)\tilde{\mathbf{x}}_j + \beta(\frac{1}{p}\sum_{i=1}^{p}\mathbf{x}_j^i)$
end for

```
# Gradient function
@everywhere function update_x(Ml,J,Mr,x,d,eta,alpha,xav)
  r = Ml*J*Mr*x - Ml*d
  g = Mr'*J'*Ml'*r
  return x - eta*g - alpha*(x - xav)
end
update_x_par = remote(update_x) # Parallel function wrapper

for j=1:niter
  @sync begin
    for k=1:p

      # Select batch
      idx = randperm(dD.nsrc)[1:batchsize]
      Jsub = subsample(J,idx)
      dsub = subsample(dD,idx)

      # Calculate x update in parallel
      xnew[:,k] = update_x_par(Ml,Jsub,Mr,x[:,k],
                               dsub,eta,alpha,xav)
    end
  end

  # Update average variable
  xav = (1 - beta)*xav + beta*(1/p *sum(x,2))
  x = copy(xnew)
end
```

TTI RTM: BP 2007 dataset

16th order TTI:

- 336 Flops/gridpoint
- 1200 x 4500 grid points
- 12k time steps

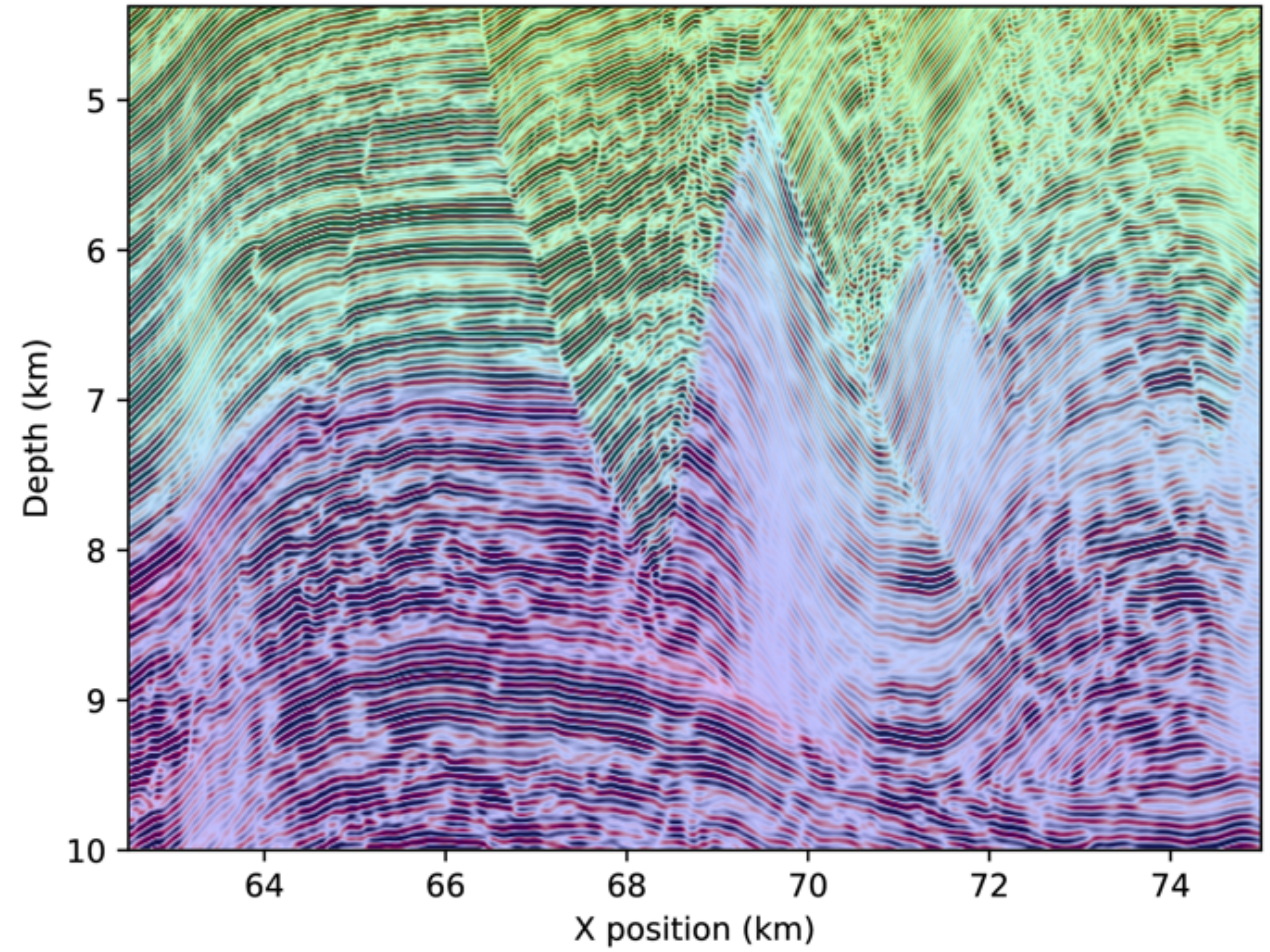
= 22 TFlops

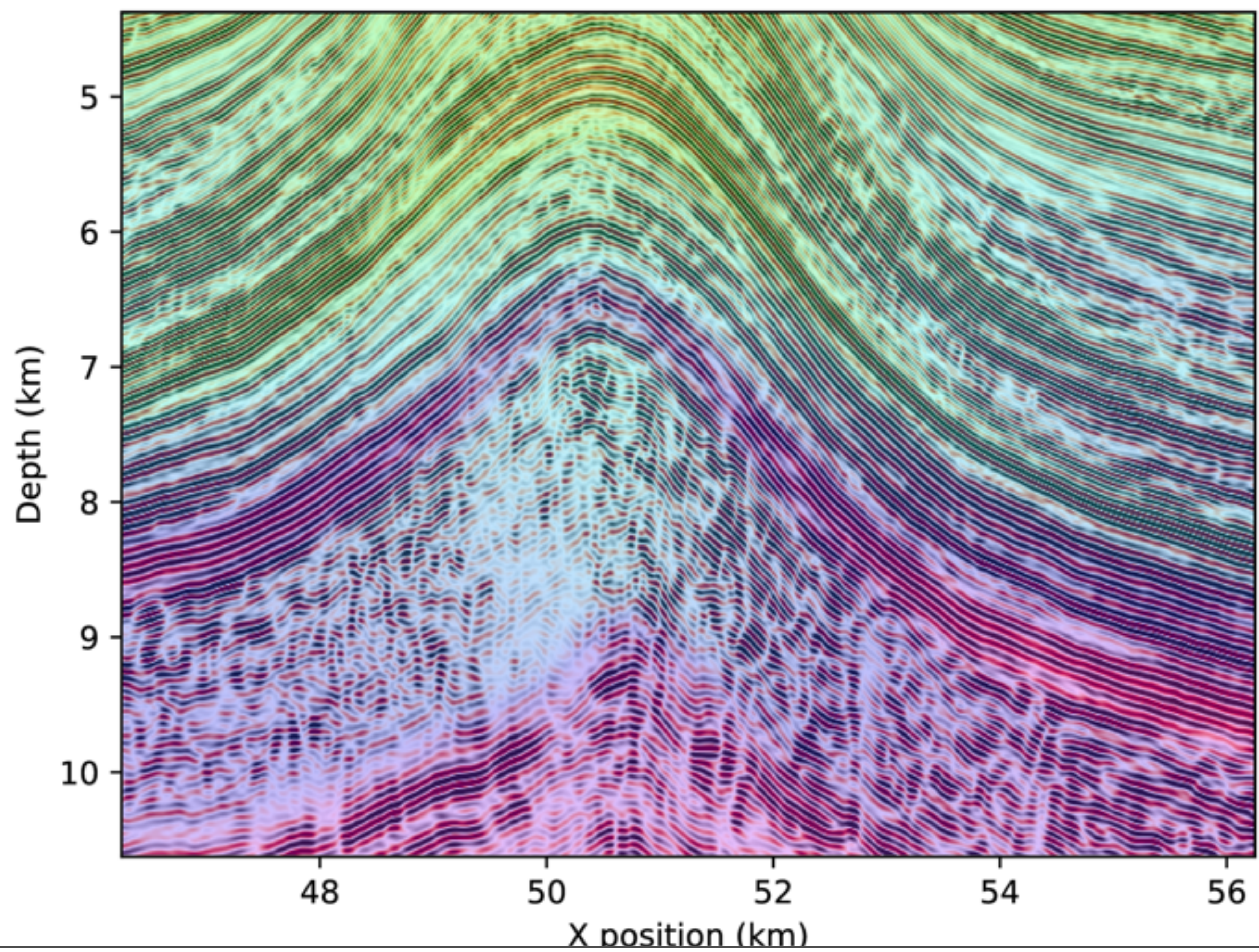
- 1600 Source

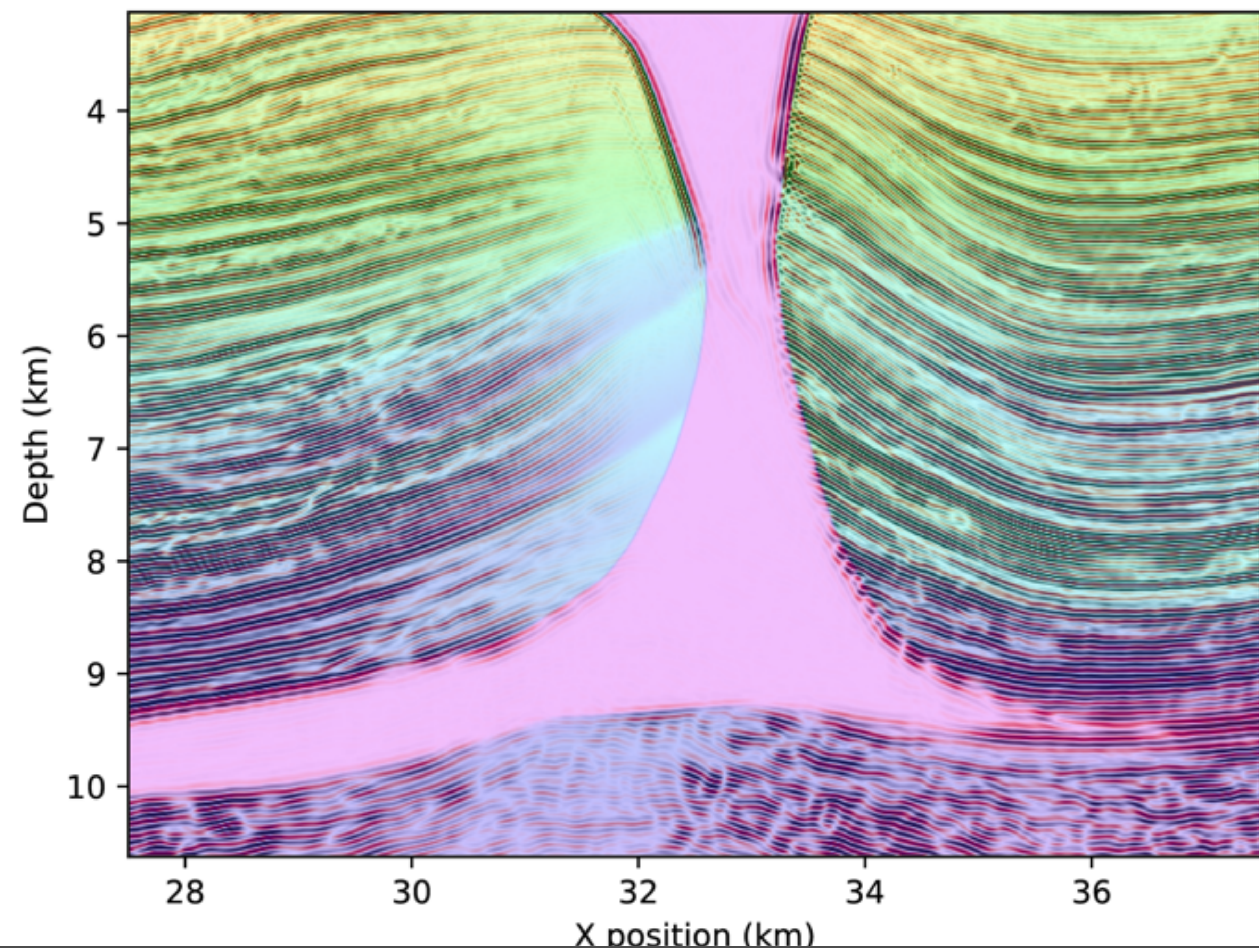
= 36 PFlops total

Summit LINPACK benchmark at 122 PFlop/s

This is only acoustic & 2D!!!







Observations

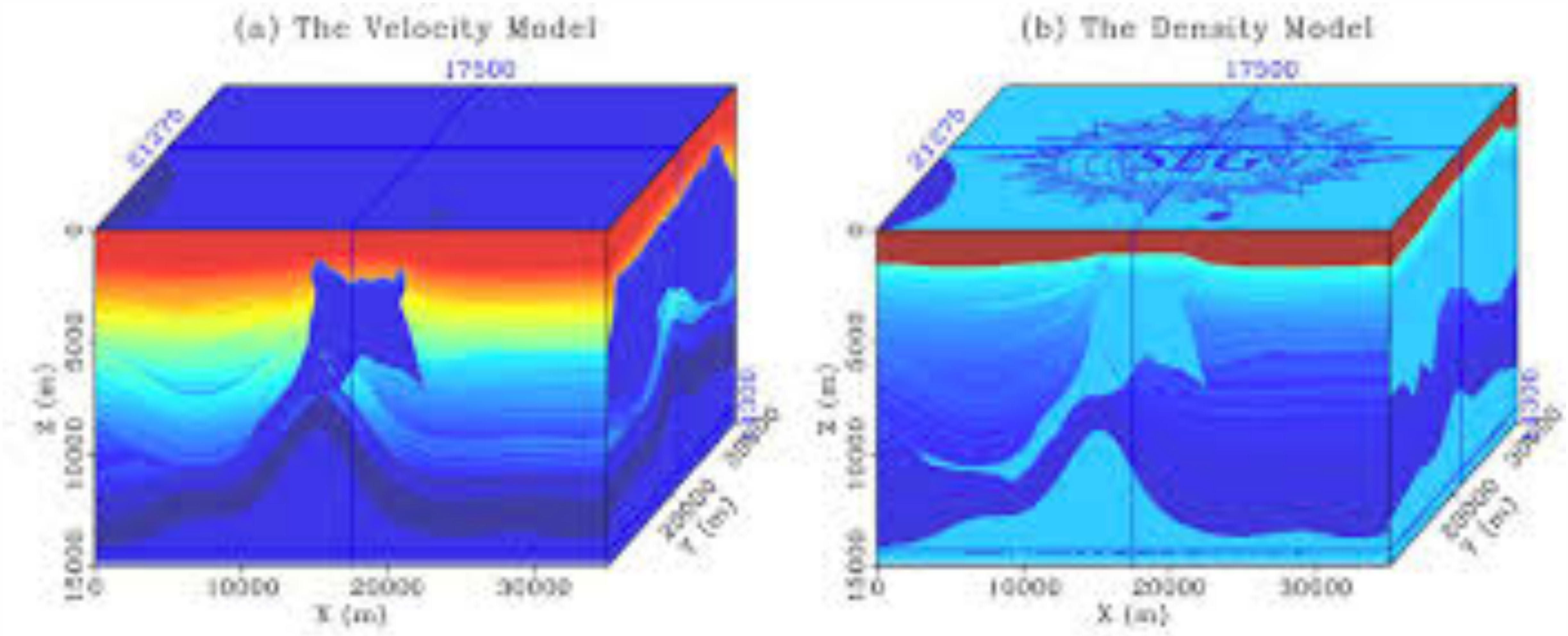
Demonstrated power of abstractions towards data & compute intensive tasks

- flexibility w.r.t hardware (GPU, different CPUs etc.)
- exploits data-space parallelism
- exploits model space w/ multithreading & domain decompositions
- ready to scale technology in Cloud & dedicated HPC solutions (Summit?)

Not ready for

- elastic
- UQ

RTM on SEAM – naive calculation



35km x 45km x 15km

5m grid at 60Hz (156 G points)

2.5m grid at 120Hz (1251 G points)

Single RTM for SEAM feasible @60 Hz requires 8100TB memory per source & 3000 Pflops (32k time steps for 8sec recording) @34h runtime on 6230 nodes

RTM @120 Hz requires 130PB memory per source and 48000 PFlops (64k time steps for 8 sec recording) @21 days but does not fit

Challenges

How to overcome

- stringent discretization required for high-fidelity numerics
- reliance on accurate physics of inverse problems

by merging ideas from ML where deep convolutional neural nets

- allow us to map low- to high-fidelity solutions
- act as proxies for complex impossible to model physics

Will require abstractions synergizing ML & CSE...

Similarities ML & FWI

They both are

- highly non-convex & solutions depend on heuristics
- reliant on back propagation & amenable to stochastic optimization
- in need of prior information (constraints), different objectives, & fast codes
- atomic runtimes too long when operating @scale

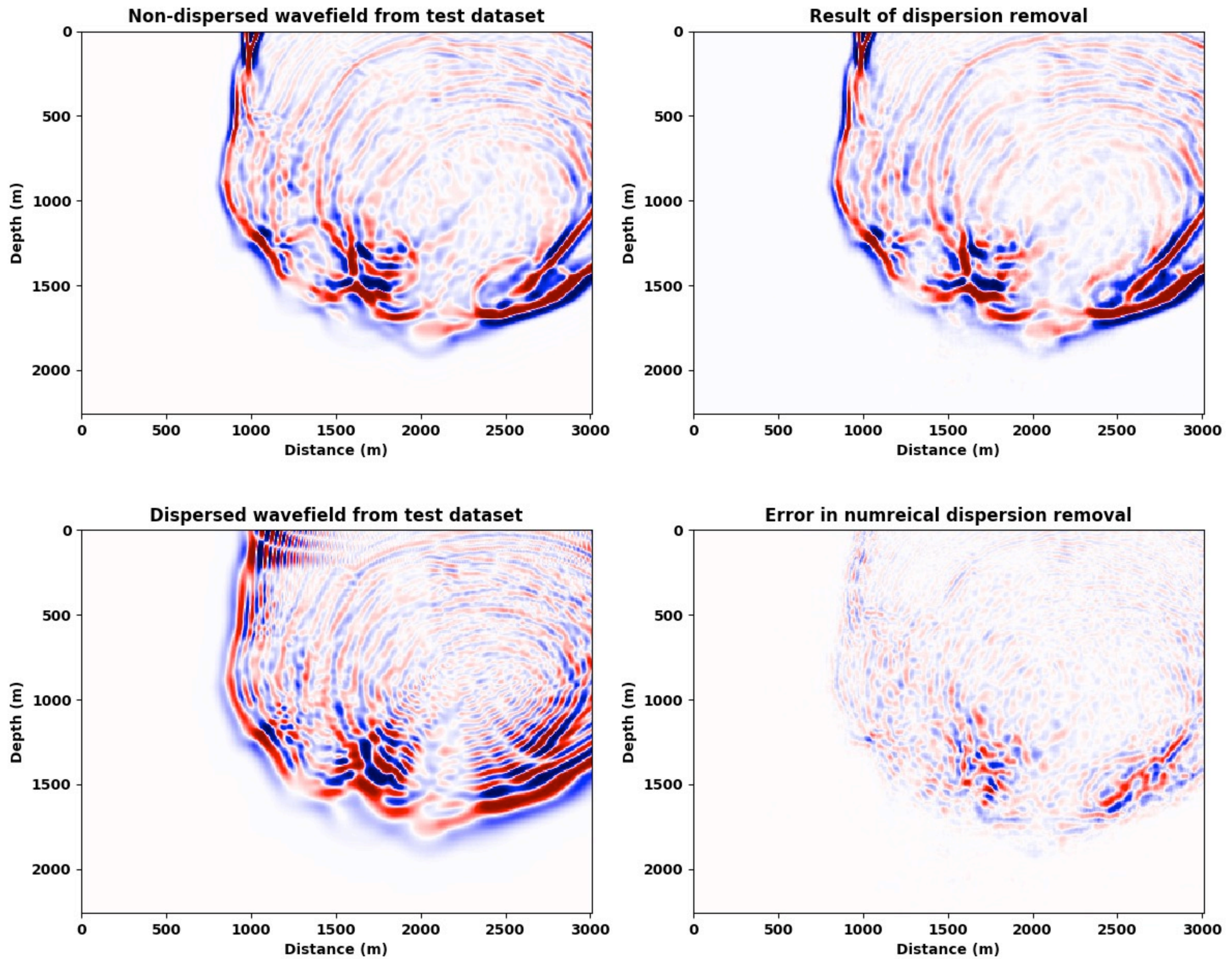
ML will benefit from:

- Devito like abstractions for fast & scalable computations
- interpretability of hidden variables

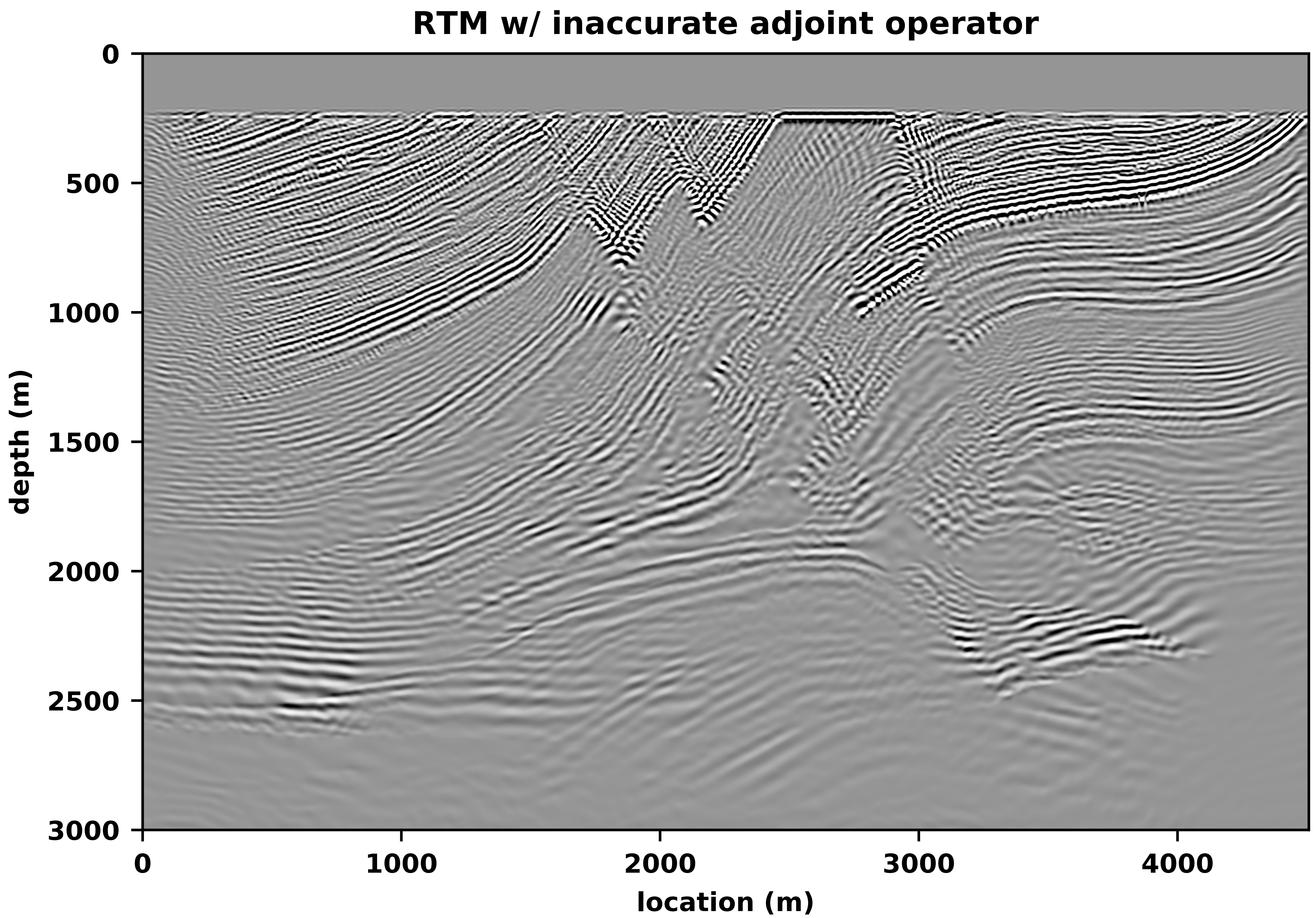
Numerical dispersion removal w/ transfer learning

Ali Siahkoohi, Mathias Louboutin, Rajiv Kumar, and Felix J. Herrmann, "Deep Convolutional Neural Networks in prestack seismic---two exploratory examples", 2018.

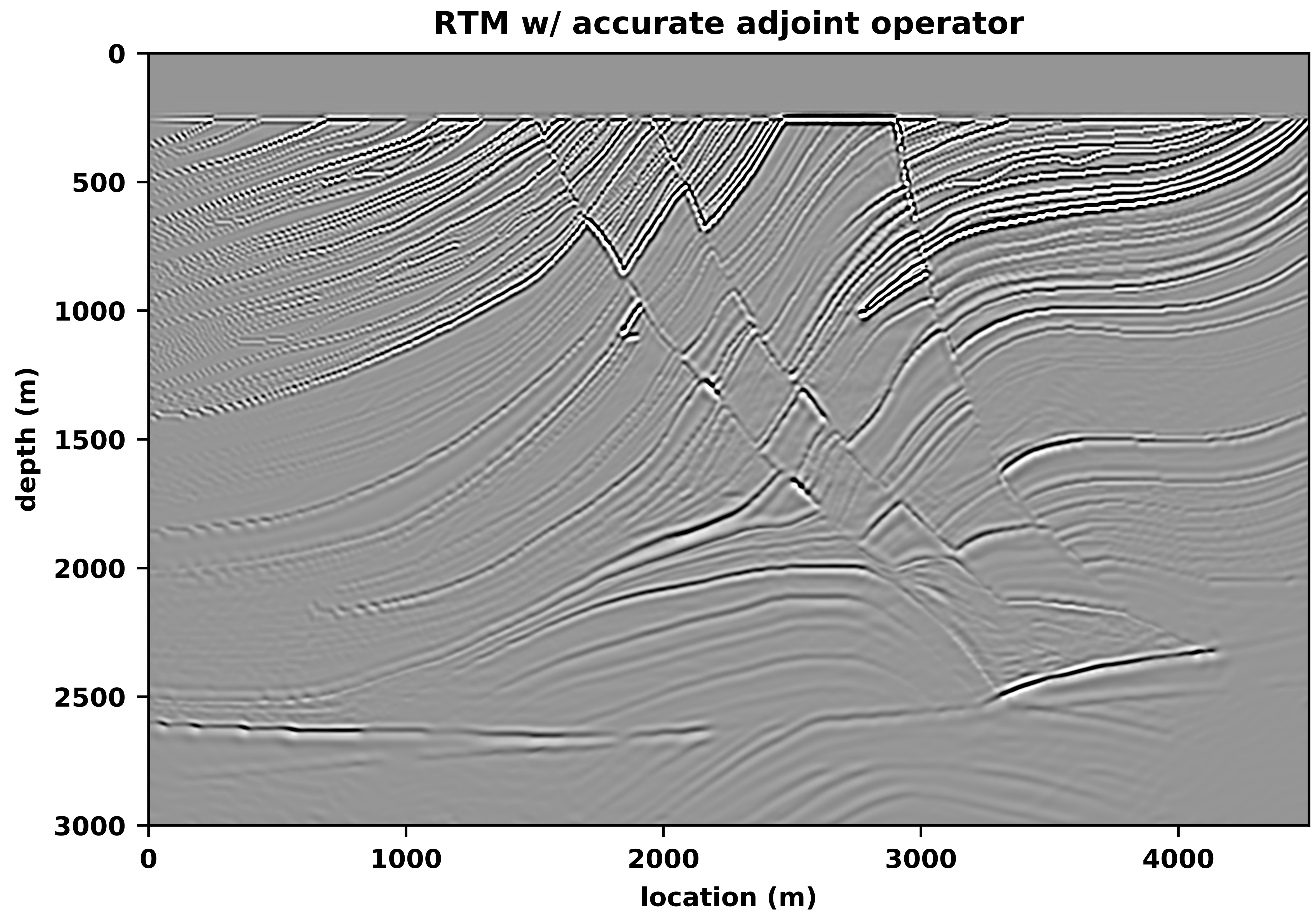
Top-left: w/ 20-point stencil - testing velocity
Bottom-left: w/ 2-point stencil - testing velocity
Top-right: Removing numerical dispersion in w/ transfer learning
Bottom-right: Difference between result and non-dispersed wavefield



RTM image w/ numerical dispersion artifacts

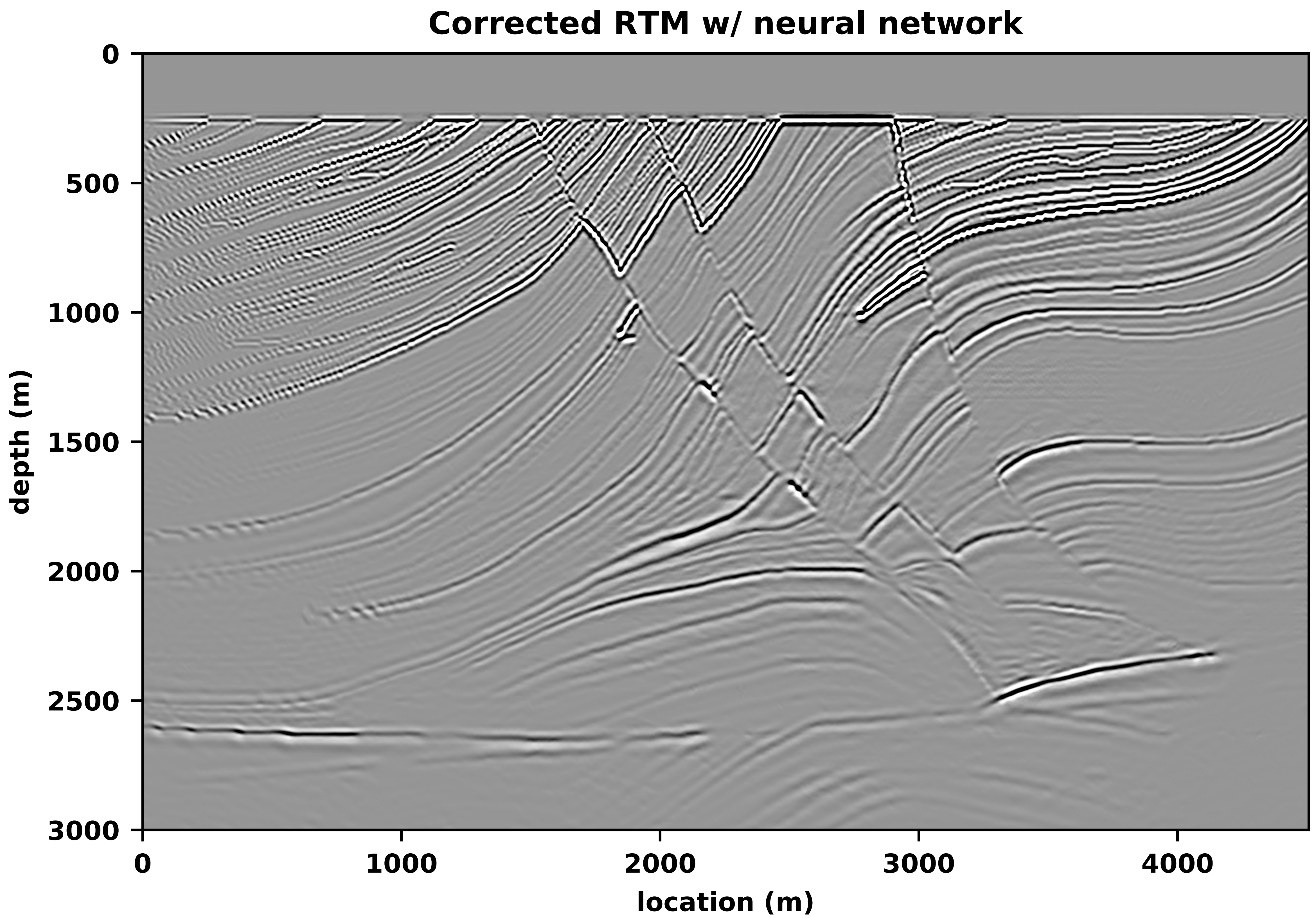


RTM image obtained from inaccurate FD simulations (2-point stencil)

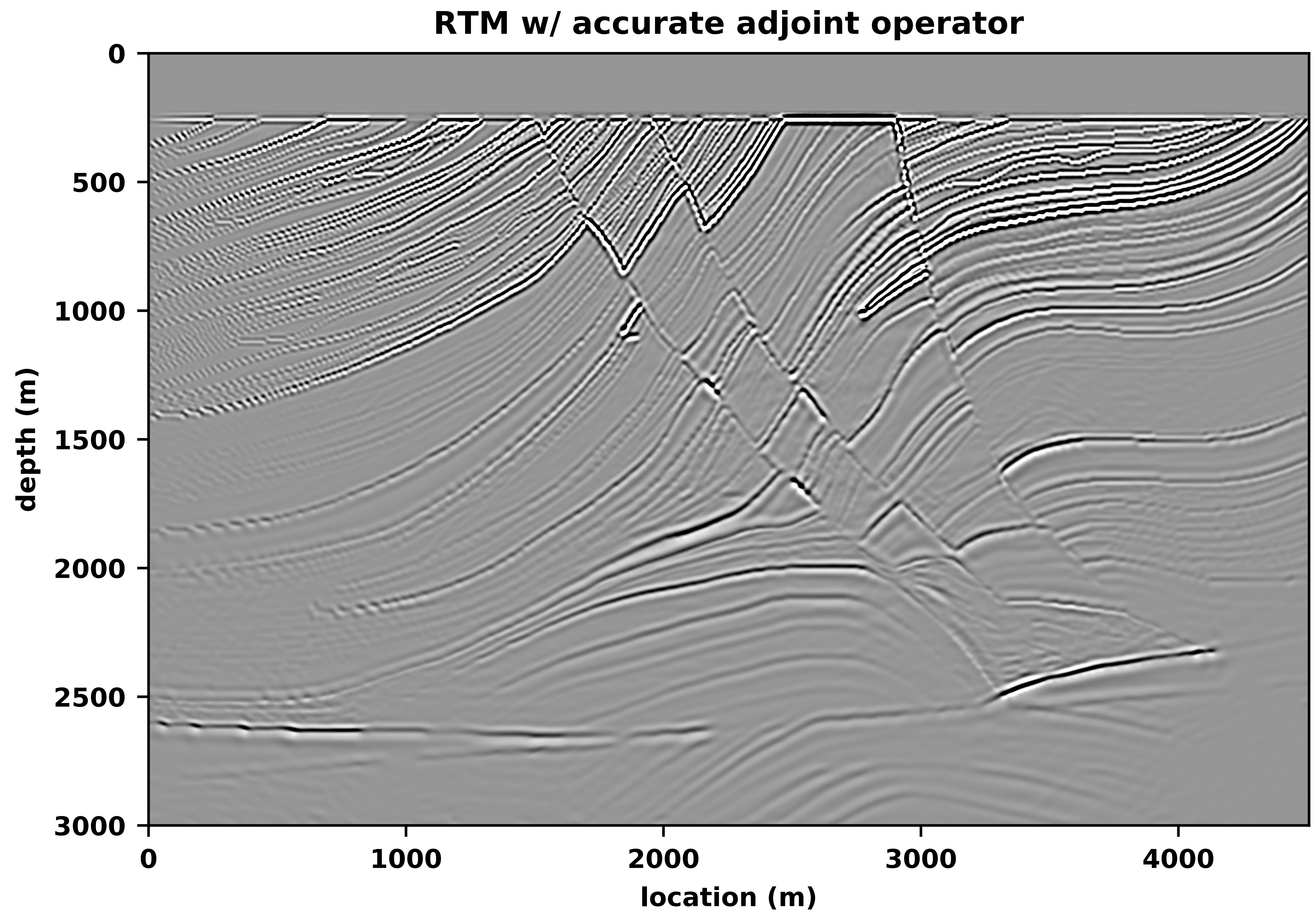


RTM image obtained from accurate FD simulations (20-point stencil)

Removing artifacts w/ CNN



RTM image obtained from correcting the inaccurate RTM image w/ CNN



RTM image obtained from accurate FD simulations (20-point stencil)

Conclusions

The right abstractions hold key to

- manage complexity of data & compute intensive imaging problems
- merging ideas from CSE & ML

Example: Low-fidelity numerics corrected w/ CNNs

To be successful we will need

- access to major machines such as Summit to do high-fidelity simulations
- scale up CNNs to carry out low-to-high fidelity maps...