# **Software** and **workflows** at SLIM/SINBAD

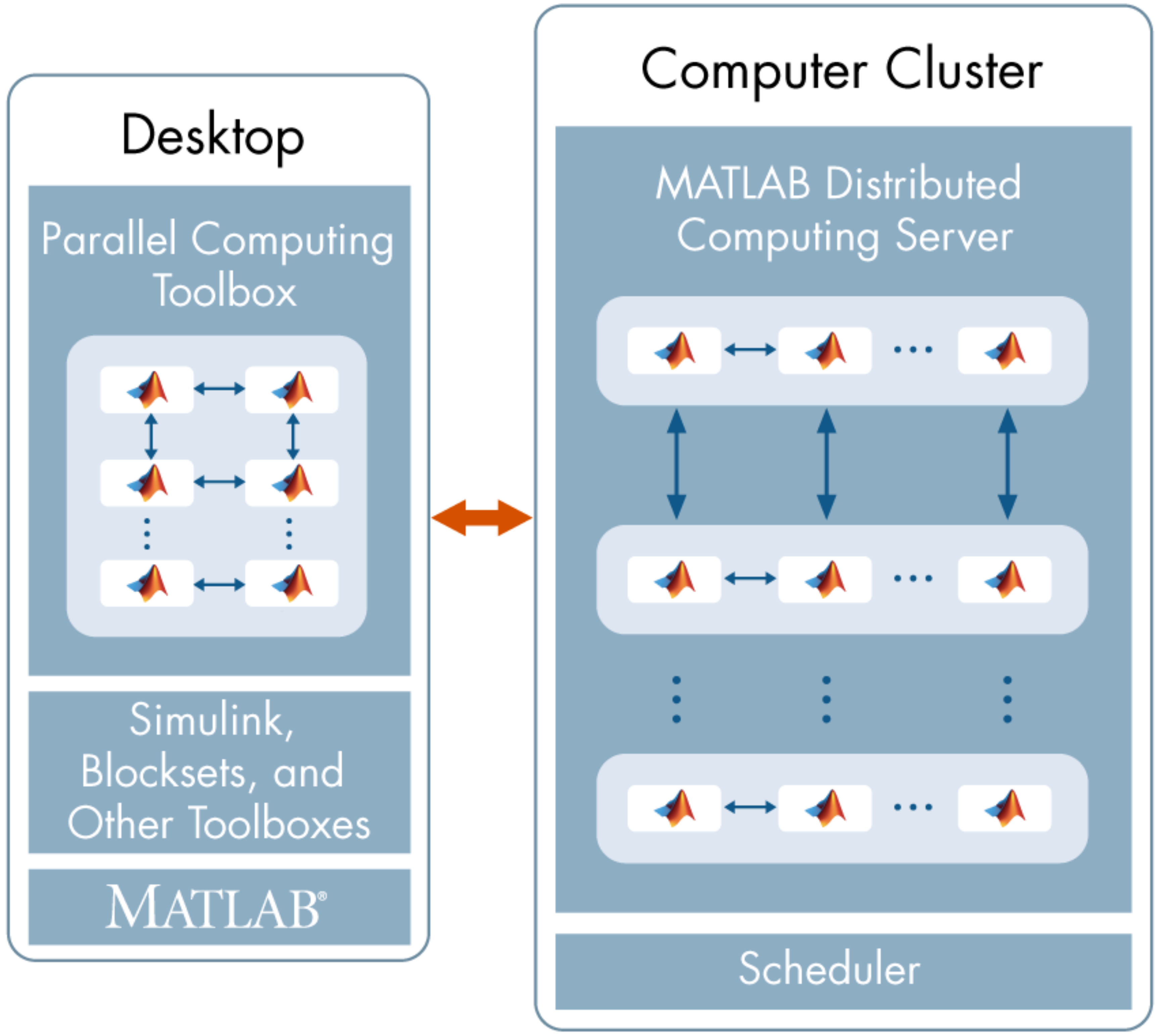
Tim Lin, Brazil IIP FWI Workshop

SLIM
University of British Columbia

*currently, we program in MATLAB, but our abstraction for distributed computation is based on*

**Parallel Matlab (PCT)** **pSPOT**

# “Parallel Matlab”




Officially another “toolbox” on top of Matlab, called “Parallel Computing Toolbox” or PCT

Two components:

- The “toolbox” itself, which provides the parallelization code and can spawn local workers
- The “Distributed Compute Server” (MDCS) which allows spawning workers on external nodes in a cluster
    - can bring own scheduler, i.e., SLIM uses Torque, SENAI uses Slurm

# Matlab PCT operation

Always assumes a “master” supervisor for a pool of workers

Each worker (and master) are independent, complete Matlab processes, and communicate via a MPI-based backend

Workers form a “pool” that can be provisioned and released interactively from the command line

**Local workers free**, individual licensing price for remote workers

## Distributed arrays

Emulates a normal numeric array

- by default distributed evenly across the last dimension
- APIs to change underlying distribution
- can be constructed in many ways… from simple to complex
- easy way to learn about shared-memory/**NUMA** type architecture

**Killer Feature:** overloading of many Matlab functions on local numeric arrays to distributed arrays

# SPOT/pSPOT built on distributed array

A way to encapsulate kernel computations of linear operations into something that "looks like a matrix"

```
F = opDFT(512)
x = randn(512,1);
xf = F * x;
x2 = F' * xf;
```

Inherently express the notion of multilinear transformations on tensors into Kronecker products

```
FK = opKron(opDFT(300), opDFT(512))
x = randn(512,300);
x_fk = FK * x(:);
x2 = FK' * x_fk;
```

# SPOT/pSPOT built on distributed array

Extends to distributed paradigm (implicitly performs transpose)

```
F = opDFT(1024);
F2D = opKron(F,F);
F4D = oppKron2Lo(F2D,F2D);
F5D = oppKron2Lo(F2D,opKron(F,F,F));

x = distributed.randn(1024*1024*1024,1024*1024);
xf = F5D * x(:);
```

# Non-separable example

## Frequency-dependent filtering

```
A = oppDistFun(f,@filter)
```

| | |
|---|---|
| **f** | is (distributed) array of frequencies |
| **@filter(x,f)** | performs filter on x based on frequency f |

## Slice-wise matrix-matrix multiply

```
A = oppDistFun(MAT,@matmult)
```

| | |
|---|---|
| **MAT** | is 3D array distributed over the “slice” dim |
| **@matmult(x,mat)** | performs mat-mult between x and mat |

# SLIM software releases

## Organizational overview

## Software versioned and managed using Git

***Internal*** *in-development* code lives on **private Git-Lab** server

***Public*** *code* (software release, SPOT/pSPOT, etc) lives on **GitHub**

# A new software organization for 3D FWI

Curt Da Silva

SLIM
University of British Columbia

# A new 3D FWI framework

*modularizes* relevant subsystems

- helmholtz discretization, linear solves, computing gradients, hessians, etc.
- software hierarchy manages complexity
- *east to test* individual modules + ensure correctness/efficiency
- *easy to extend*
- *easy to parallelize*
- lower level performance improvements (linear solves, etc.) propagate to entire framework

# Software organization

Software hierarchy manages complexity

- human brains have very limited working memory
- if a particular part of a program only has one function, people using/debugging it only have to think about that one function
- if software is easier to reason about -> it's easier to work with, easier to test

# Software organization

Software hierarchy manages complexity

- we don’t have to sacrifice performance
  - lowest level operations implemented in C w/multithreading
- hiding irrelevant details at each level
  - higher level functions don’t have any idea about C/fortran/that gross stuff

# Software organization

Anything that we do that isn't solving PDEs is essentially irrelevant, computation time-wise

- advantageous, for designing our software, because any overhead introduced is negligible compared to solving PDEs

- improvements in solving PDEs quickly propagate to the whole FWI framework

## 3D FWI

Our problem:

$$\min_{m} \sum_{s,\omega} \|P_r u_{s,\omega}(m) - D_{s,\omega}\|_2^2$$

$$\text{such that } H_\omega(m) u_{s,\omega}(m) = q_{s,\omega}$$

- we have *separability* over sources/frequencies
  - objective, gradient, hessian, GN hessian, etc.

- informs our later design decisions

## Software organization

Lowest level: opBandStorage

- SPOT operator, stores necessary information for Helmholtz multiplications, wavefield solves, preconditioners

- Writing

```
U = H \ Q_i;
```

calls the specified solver with the appropriate preconditioner, tolerance, parameters, etc.

# Software organization

opBandStorage

- agnostic to its entries
  - 7 pt, 27 pt stencils, all treated in the same manner
  - acoustic, constant density kernel currently
    - easy to integrate anisotropy, varying density in to FWI framework
- stores minimal amount of information for specific applications
  - e.g., if no adjoint multiplications/divisions are needed, adjoint coefficients are not stored

# Software organization

opBandStorage

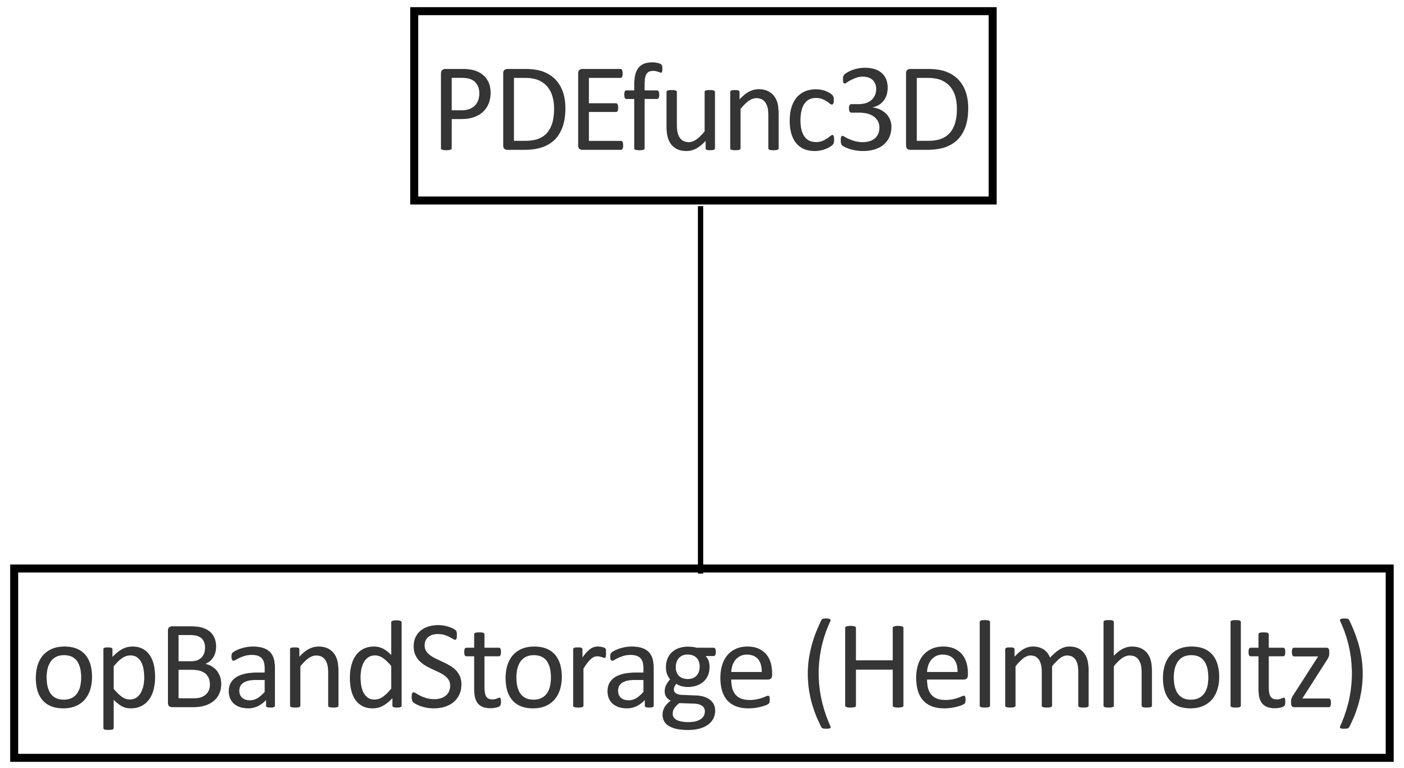
- sweeping Kaczmarz preconditioner (CRMN)
  - (multithreaded) sweeps implemented in C
- matrix-vector products implemented in C (also multithreaded)
- as far as iterative solvers are concerned, the helmholtz operator is just another matrix (SPOT paradigm)

# A new 3D FWI framework

opBandStorage (Helmholtz)

Modeling matrix : multiplication/division

# A new 3D FWI framework




PDE-related quantities
Serial version

Modeling matrix :
multiplication/division

# Software organization

PDEfunc3D

- computes various quantities (objective + gradient, migration/ demigration, gauss-newton hessian, hessian) based on solutions of the helmholtz equation

- serial code that calls (multi-threaded over number of RHS implicitly)

```
U = H \ Q;
```

- function that is tested + satisfies Taylor error estimates, adjoint tests, etc.

## Software organization

PDEfunc3D receives a list of $(\mathrm{src\ x}, \mathrm{src\ y}, \mathrm{freq})$ indices

- for each frequency $f$, gets the $(\mathrm{src\ x}, \mathrm{src\ y})$ indices corresponding to $f$
- each Helmholtz matrix can solve $n_{\mathrm{compsrc}}$ sources efficiently in parallel via multi-threading (user defined)
- the source indices are chunked up in to batches of size $n_{\mathrm{compsrc}}$, each batch is processed sequentially

# Software organization

PDEfunc3D - at this level in the hierarchy

- we care about
  - arranging the ‘pieces’ of wavefields in the right way
  - processing wavefield solves in efficient chunks

- we don’t care about
  - how exactly the linear solve is performed
  - what the helmholtz matrix looks like
  - parallelization

# A new 3D FWI framework

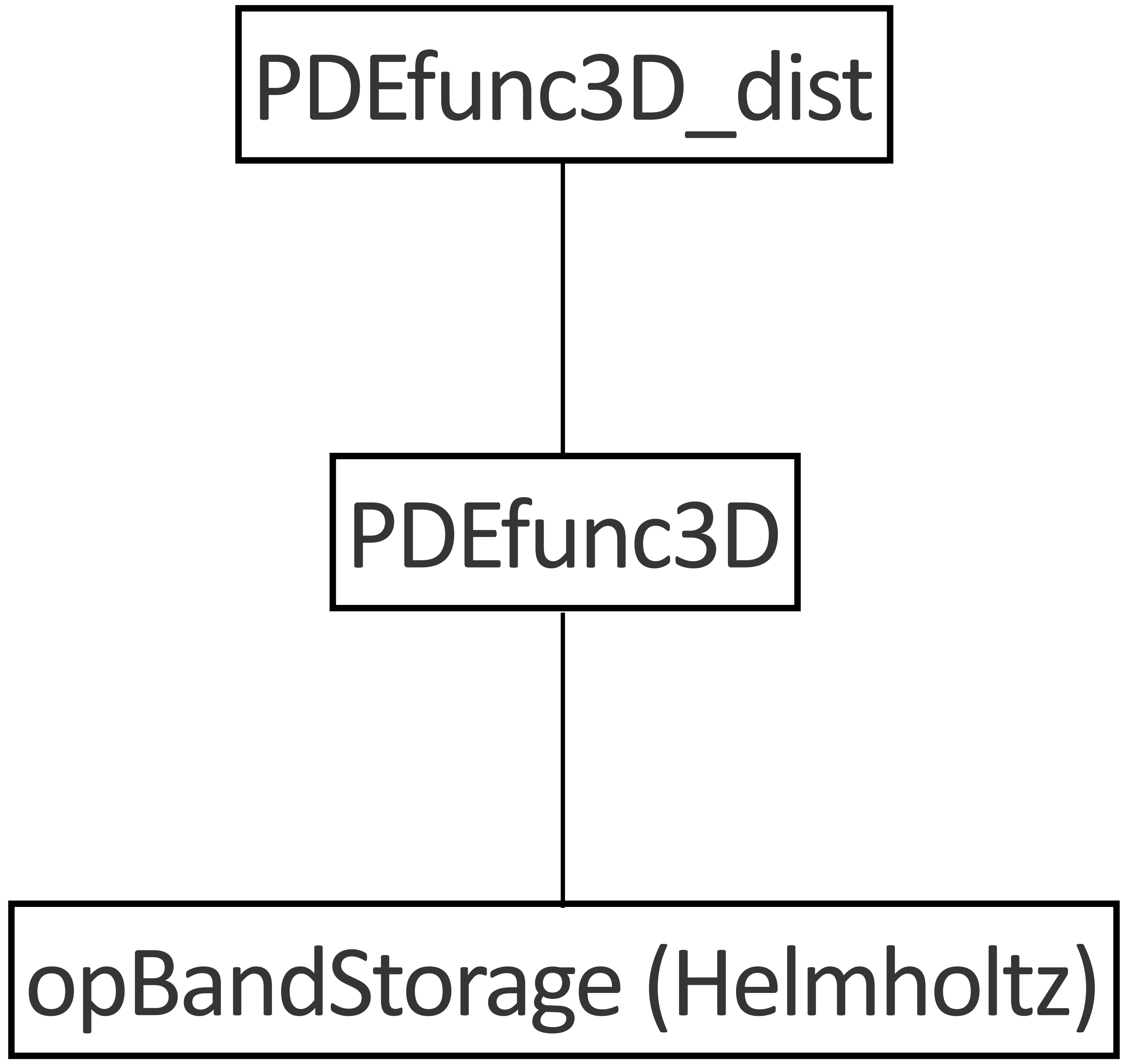
PDEfunc3D

PDE-related quantities
Serial version

opBandStorage (Helmholtz)

Modeling matrix :
multiplication/division

# A new 3D FWI framework

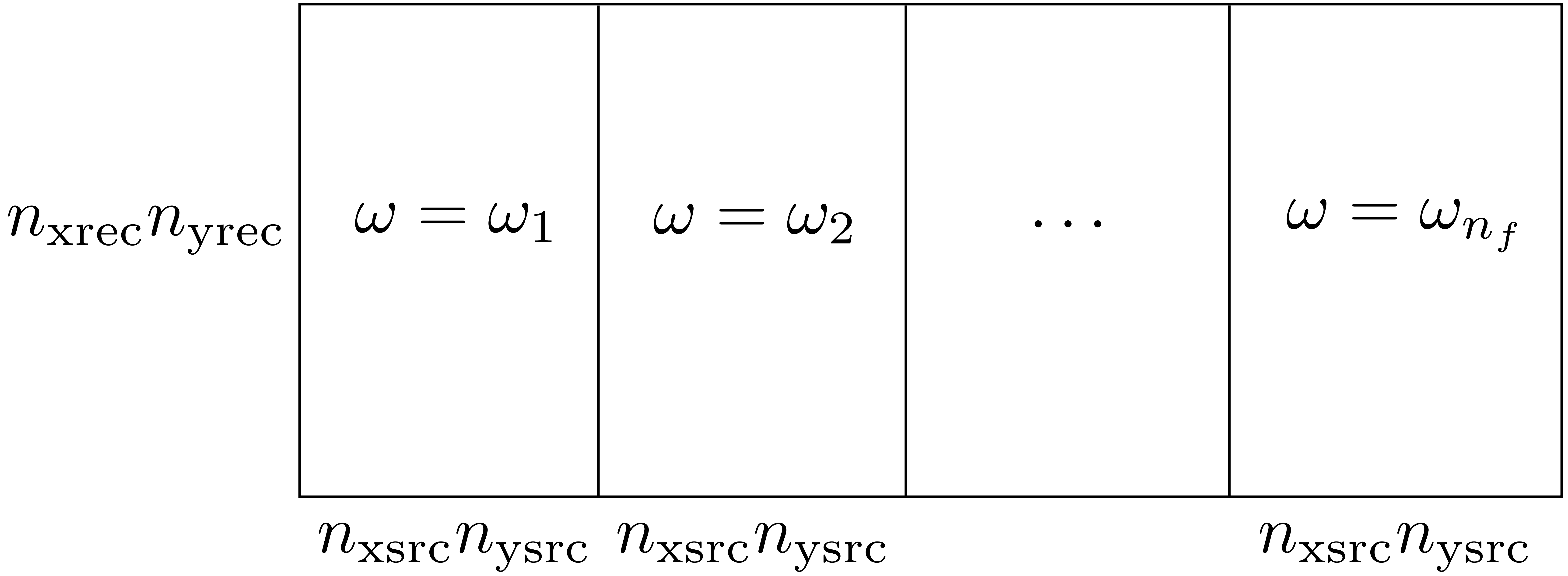



PDE-related quantities
Distributed version

PDE-related quantities
Serial version

Modeling matrix :
multiplication/division

# Data volume



$n_{\text{xrec}} n_{\text{yrec}}$
$\omega = \omega_1$
$\omega = \omega_2$
$\dots$
$\omega = \omega_{n_f}$
$n_{\text{xsrc}} n_{\text{ysrc}}$
$n_{\text{xsrc}} n_{\text{ysrc}}$
$n_{\text{xsrc}} n_{\text{ysrc}}$

# Software organization

Each column of this matrix is *independent*

- split up + compute these in parallel

Manifested as a sum structure, e.g., for the objective function

$$f(m) = \sum_{s,\omega} f_{s,\omega}(m)$$

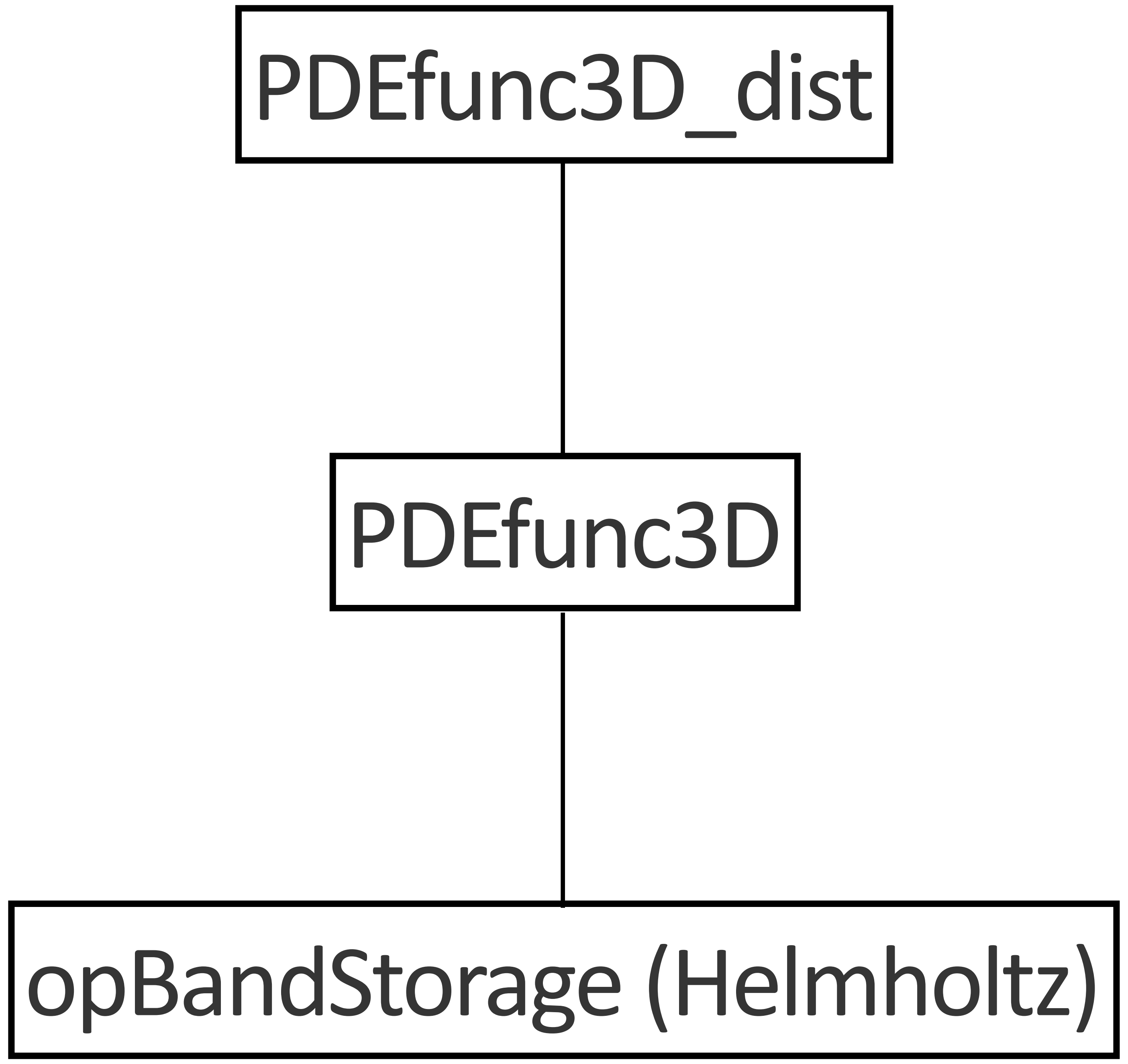
which is also computed in parallel over source, freq

## Software organization

PDEfunc3D_dist

- manages the computation of PDEfunc3D in parallel
- responsible for no actual computation, merely distributing + calling PDEfunc3D in parallel with the correct source/frequency indices
- separating parallelization from computation

# A new 3D FWI framework

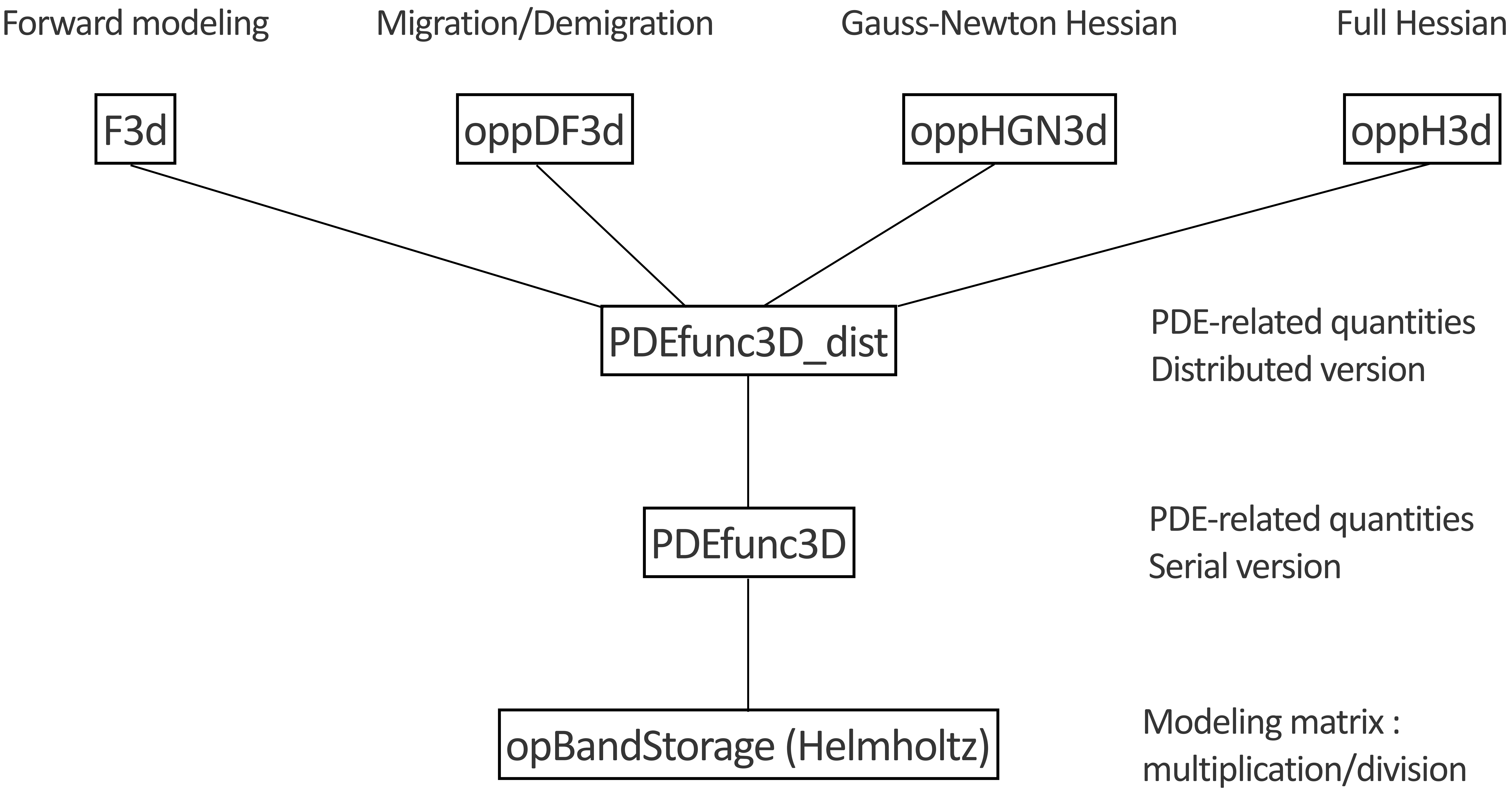



PDE-related quantities
Distributed version

PDE-related quantities
Serial version

Modeling matrix :
multiplication/division

# A new 3D FWI framework



# Software organization

F3d, oppDF3d, oppHGN3d, oppH3d

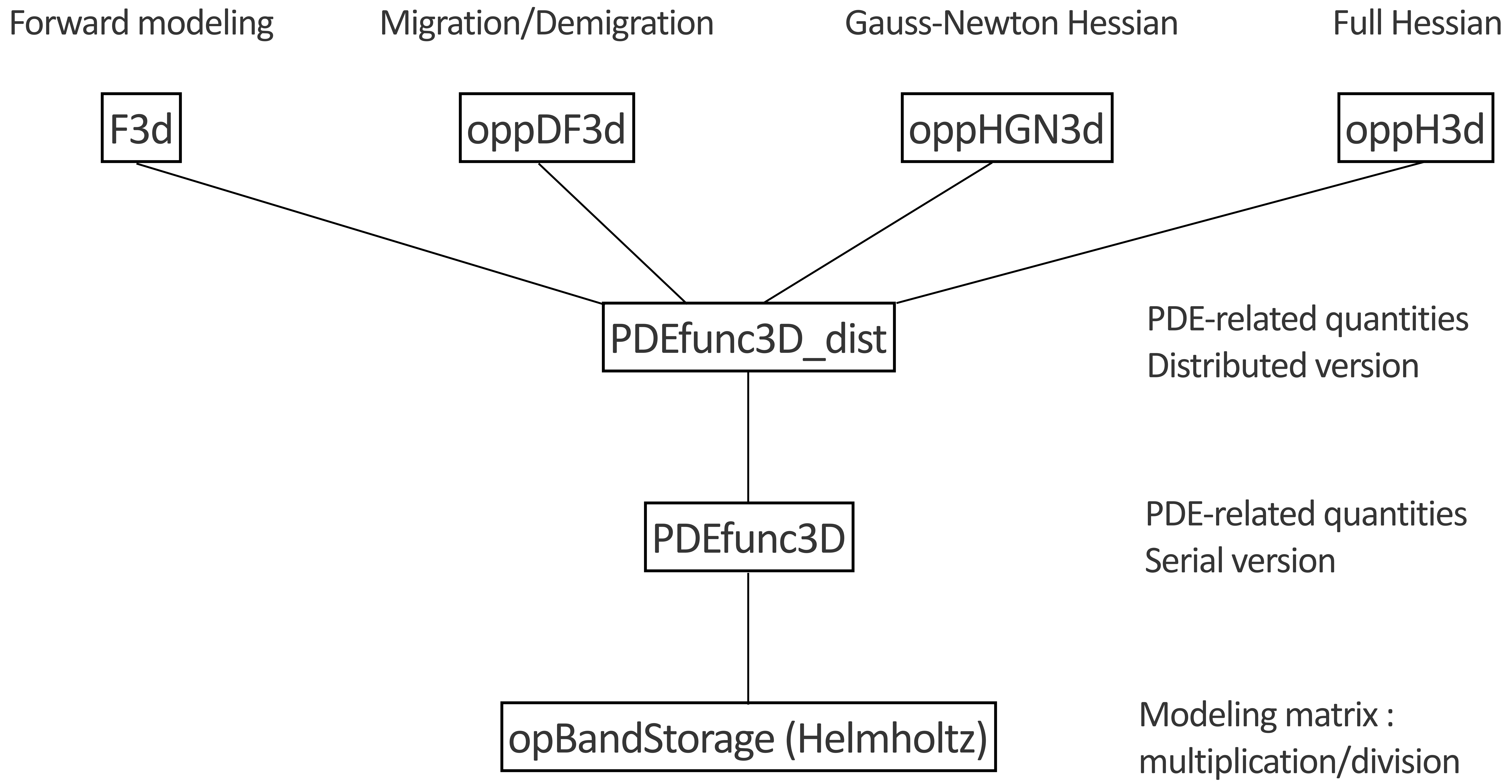
- essentially shallow wrappers around PDEfunc3D_dist
- building blocks for setting up an FWI optimization scheme
- good for operations using all of the data at once

# Software organization

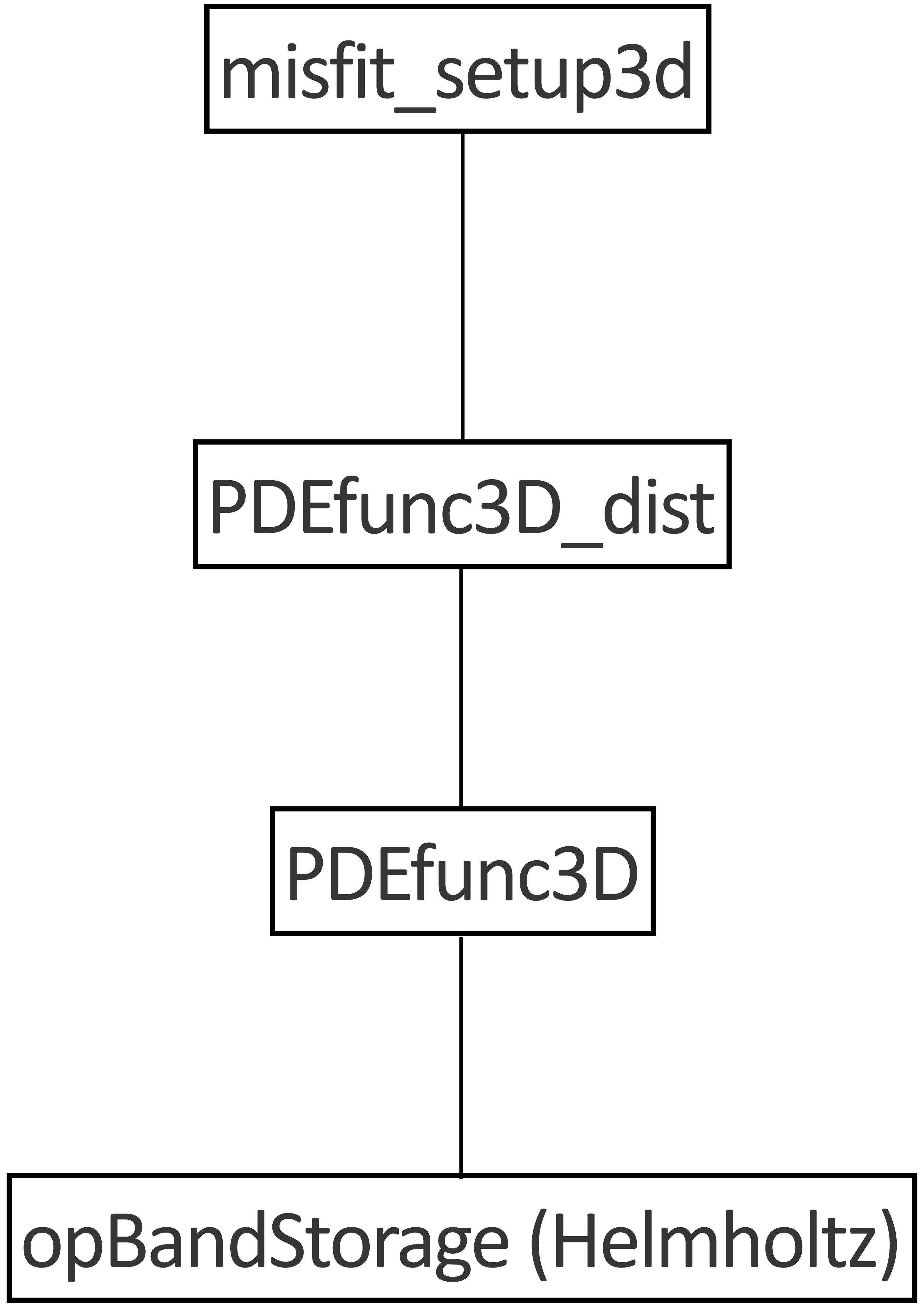
F3d, oppDF3d, oppHGN3d, oppH3d

- using these functions, not a straightforward way to set up simultaneous sources, frequency batching, frequency-based model subsampling, etc.
- don’t use these functions for production-level problems
- instead…

# A new 3D FWI framework



# A new 3D FWI framework




FWI objective setup

PDE-related quantities
Distributed version

PDE-related quantities
Serial version

Modeling matrix :
multiplication/division

# Software organization

misfit_setup3d

- outputs an FWI objective function handle according to the specified options
  - sim. sources, subsampled sources/freqs, model decimation, GN or full Hessian, etc.

- function handle can be minimized with a black-box optimization routine, no knowledge of the underlying problem required
  - parallelization, data movement, etc. handled automatically due to this software design

# Very simple example

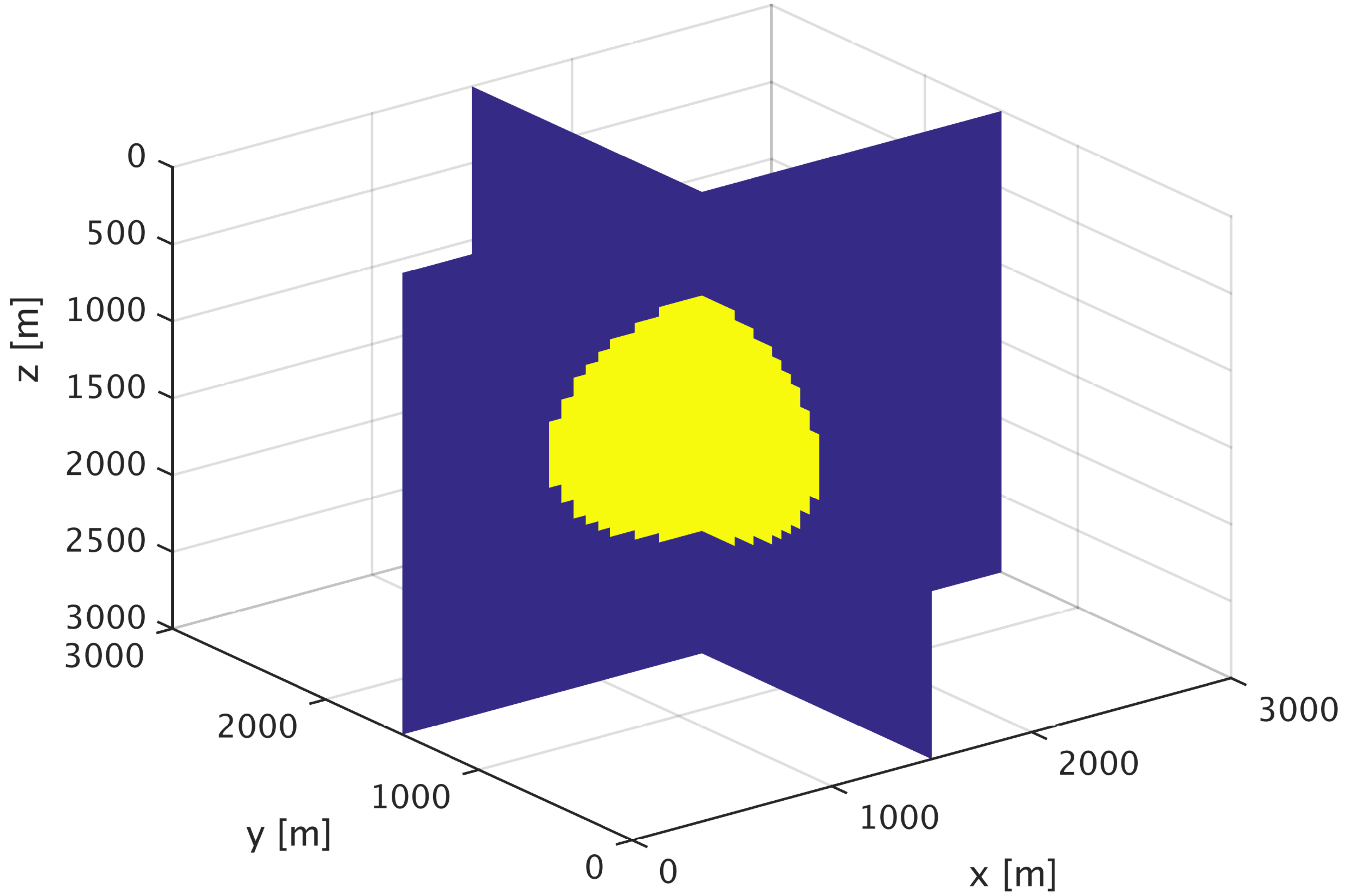
Transmission experiment - edam model

- 2 sources in x-y plane at the top of the model
- Dense receiver sampling at bottom of the model
- 3 frequencies, 2Hz - 4Hz
- 2000m/s background velocity + 2200m/s perturbation
- 10 LBFGS iterations per frequency

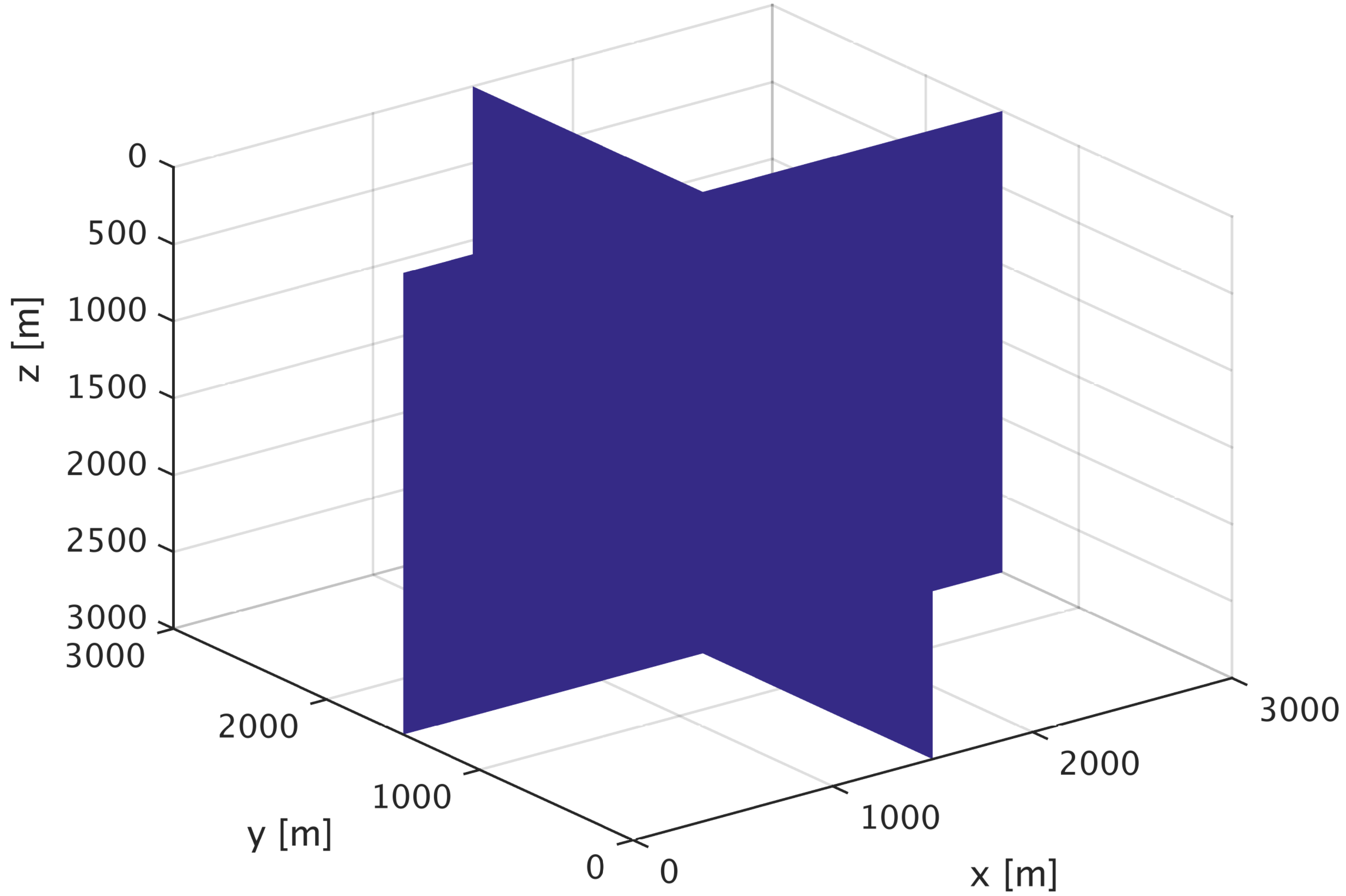
# Example 3D FWI script - easy frequency continuation

```
% Work on a single frequency at a time
freq_batch = num2cell(1:nfreq,[1,nfreq]);

vest = v0;
opts.subsample_model = true;

for j=1:length(freq_batch)
    % Select only sources at this frequency batch
    srcfreqmask = false(nsrc,nfreq);
    srcfreqmask(:,freq_batch{j}) = true;
    opts.srcfreqmask = srcfreqmask;
    [obj,vest_sub,comp_grid] = misfit_setup3d(vest,Q,Dobs,model,opts);

    % Optimization on coarser grid
    vest_sub = minConf_TMP(obj,vest_sub,vlow,vhi,minfunc_opts);
    vest = comp_grid.to_fine*vest_sub;
end
```

# Output of previous code

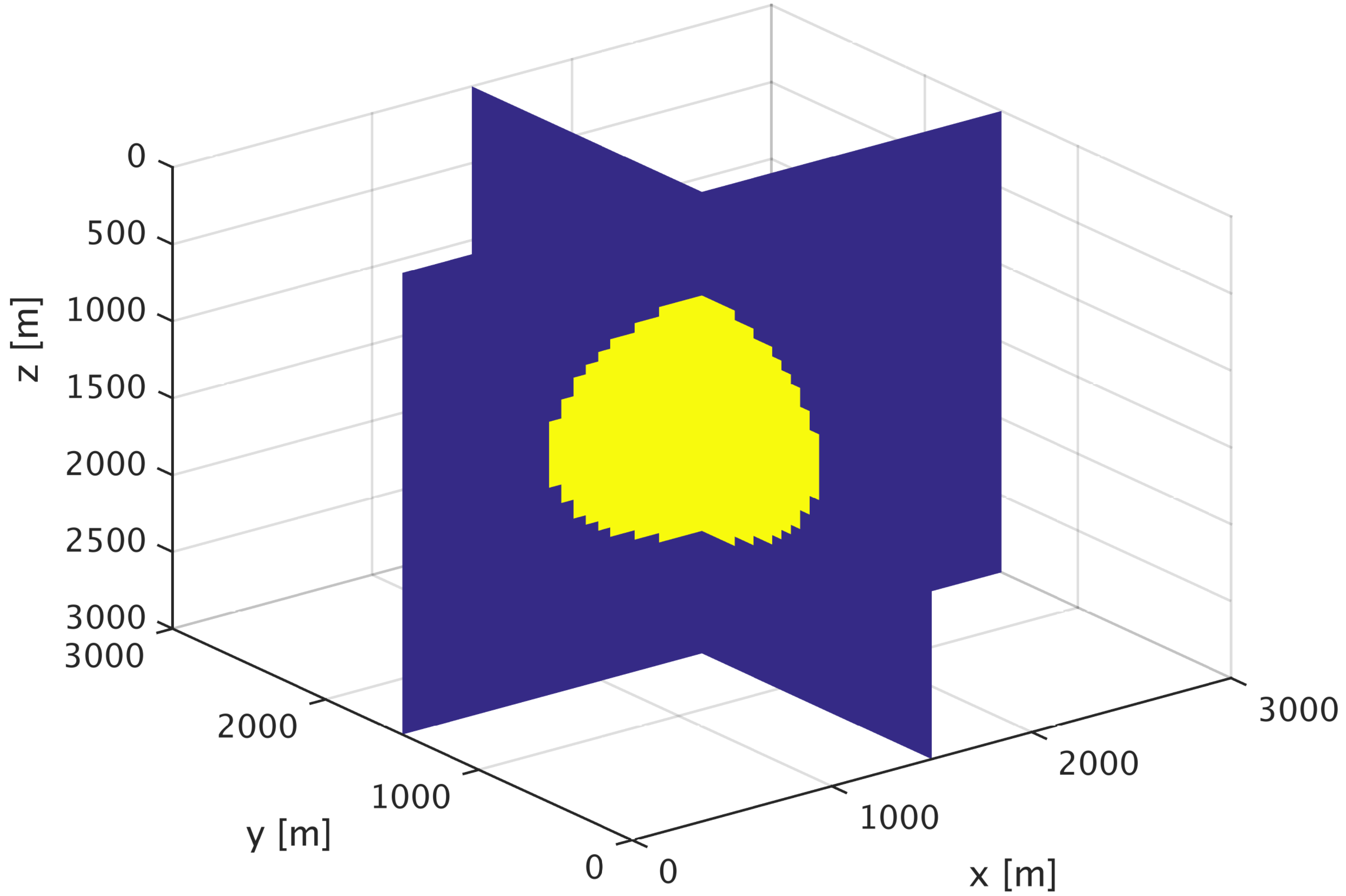



True model

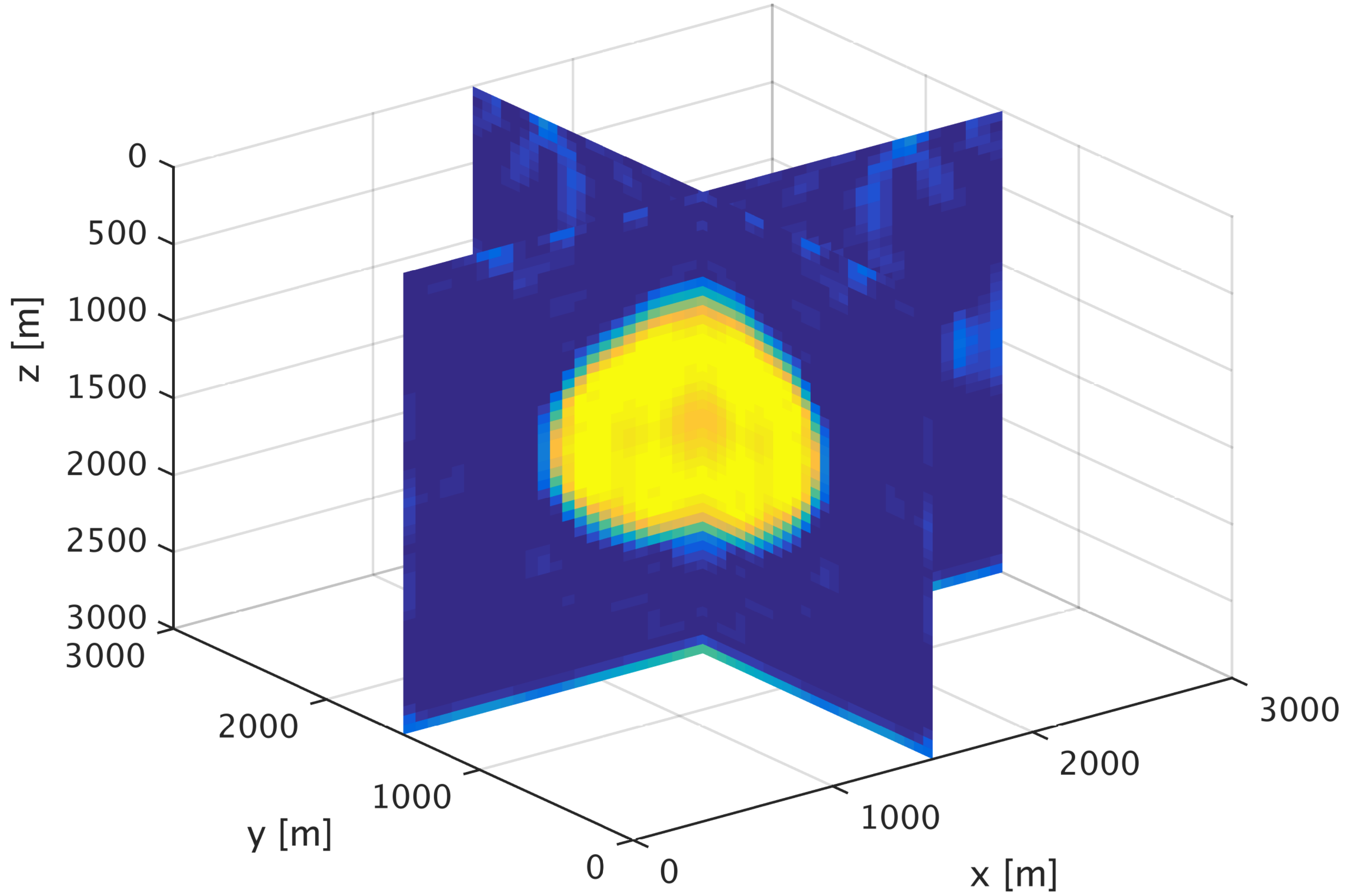



Initial model

# Output of previous code




True model




Inverted model

# Challenges

Managing parameters at different levels in the hierarchy

- ton of options to specify for the whole FWI process
- some are only used at certain parts of the hierarchy
- consistent naming, referencing, etc.

# Software organization

We now have an FWI framework that

- manages complexity/is easy to reason about
- is fully tested
- scales efficiently
- is extendable
- easy to do frequency continuation, randomized source/freq subsampling

# Software organization

## Future extensions

- anisotropic/varying density Helmholtz
- 3DWRI

## Future demonstrations

- realistic examples
- randomized source-frequency subsampling
- demonstration of curvelet-based FWI/least squares migration for 3D