Expressing next-generation seismic wavefield processing from an HPC perspective

Tim Lin

SLIM
University of British Columbia

Key trends

- Rapid increase in data size (*curse* of *dimensionality*)
- Multi-dimension operations (both *separable* and *non*-separable)
- Involves *loop*-driven optimization and/or *iterative* inversion
- Increased *sophistication* of *algorithm*

Rapid size increase

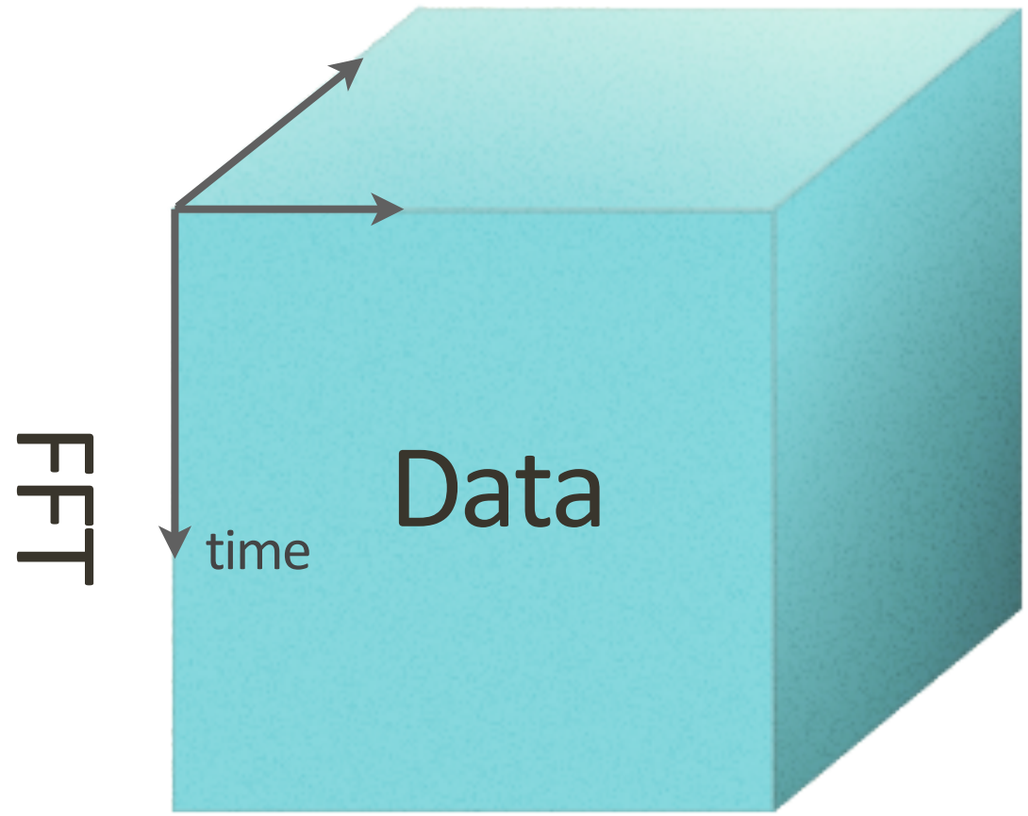
- Seismic data $P(t, x_r, y_r; x_s, y_s)$ discretized into 5-dimensional array

$$\mathbf{P} = \mathrm{nt} \times \mathrm{nx_r} \times \mathrm{ny_r} \times \mathrm{nx_s} \times \mathrm{ny_s}$$

- 750 GBs for
  $\mathrm{nt} = 512,\ \mathrm{nx_r} = \mathrm{ny_r} = \mathrm{nx_s} = \mathrm{ny_s} = 120$
- 465 TBs for
  $\mathrm{nt} = 1024,\ \mathrm{nx_r} = \mathrm{ny_r} = \mathrm{nx_s} = \mathrm{ny_s} = 500$

Separable transforms

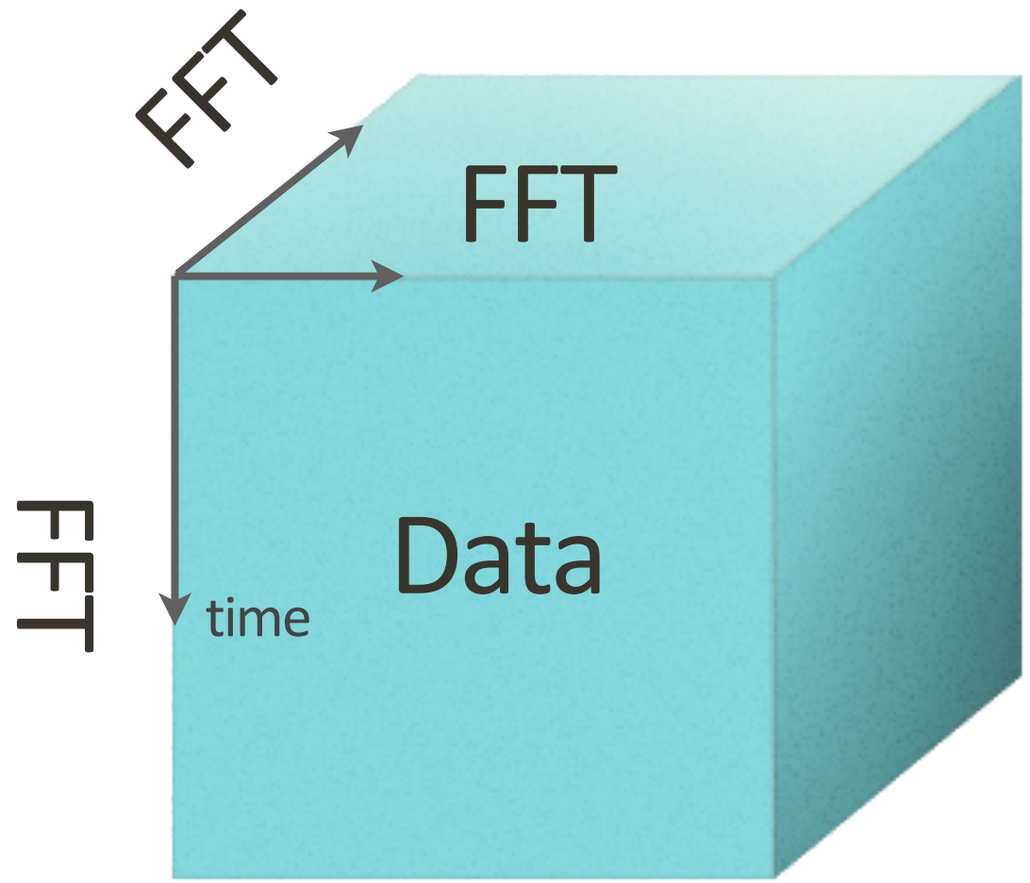
frequency slices



One Transpose

Separable transforms

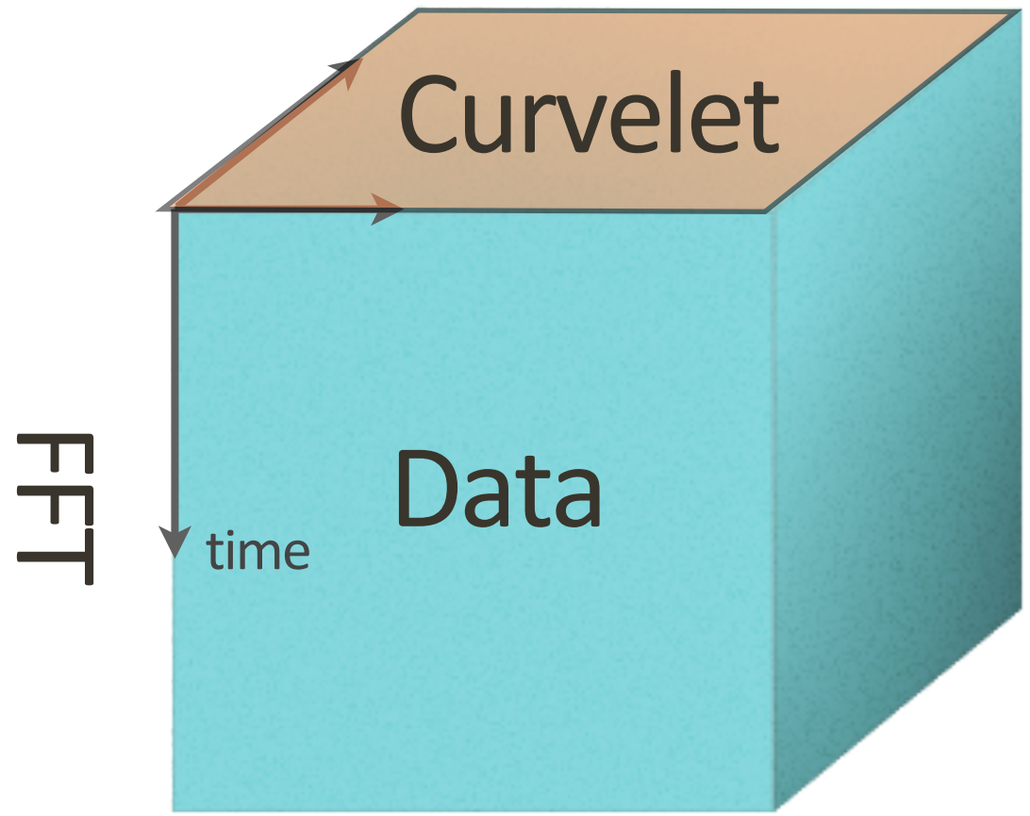
5D FFT



Lots of transposing the data (predictable communication)

Semi-separable transforms

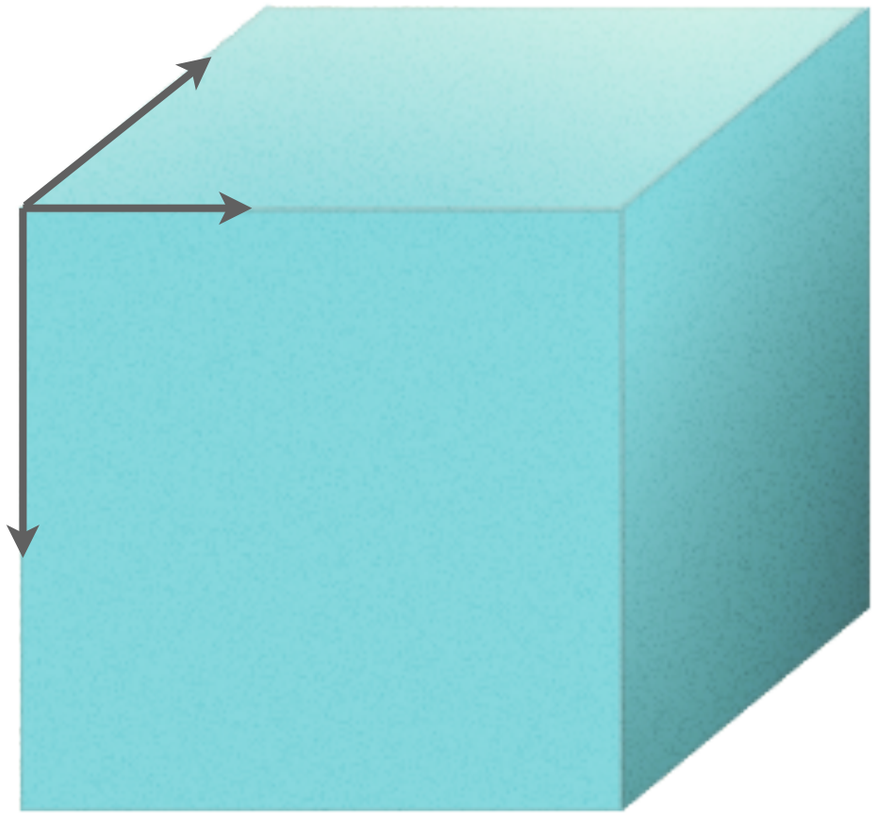
1D FFT x 2D Curvelet x 2D Curvelet



Some transposing of data + chaotic intercommunication

Non-separable transforms

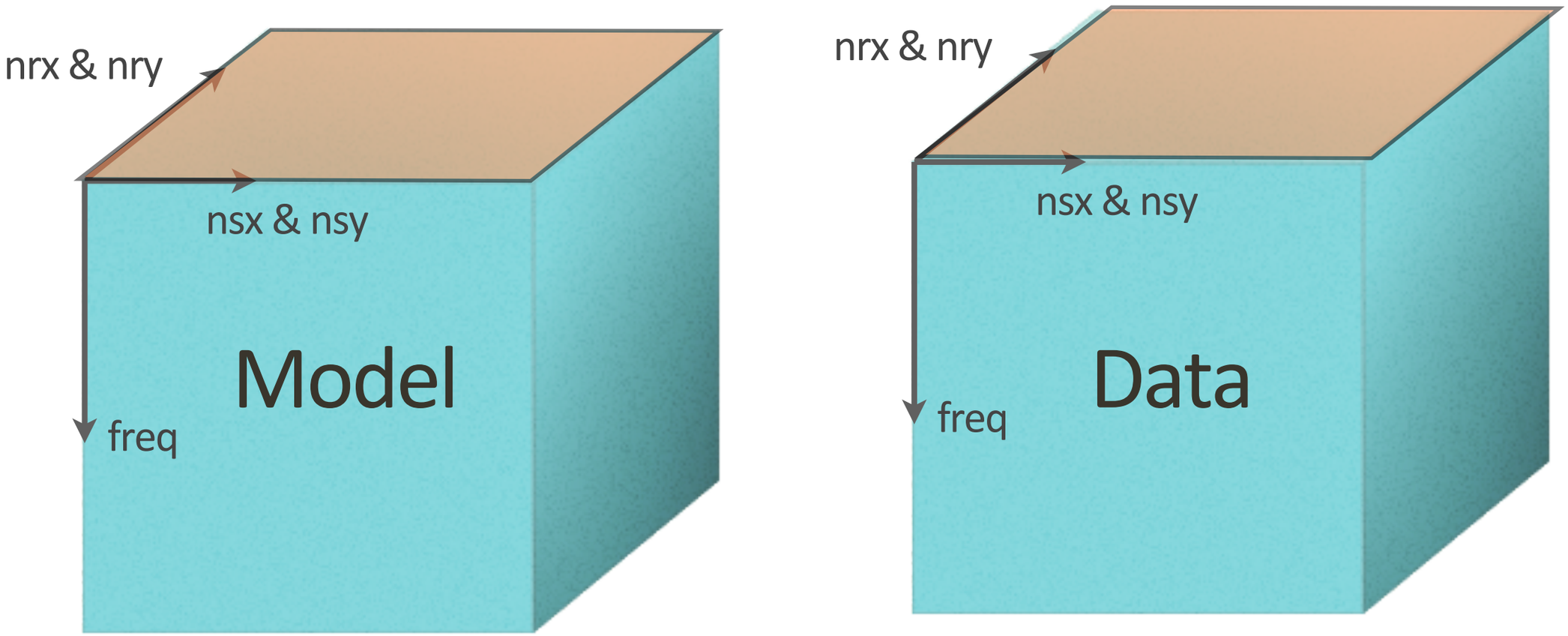
5D Curvelet?



Lots of transposing the data + chaotic intercommunication ^2

Binary wavefield operations

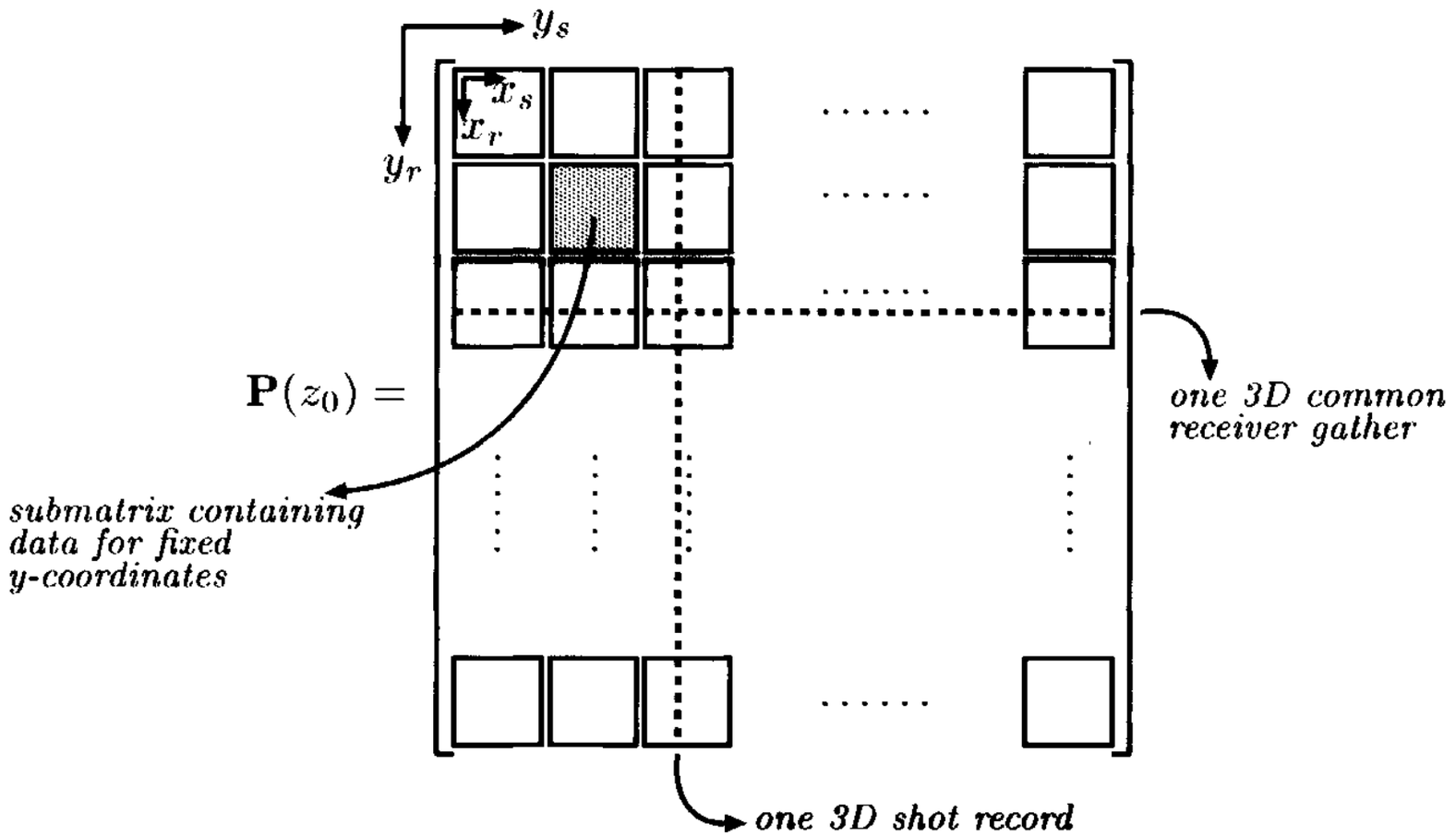
wavefield convolution/correlation



essentially involves *massive* matrix/tensor multiplications

Matrix multiplication

"Data Matrix"

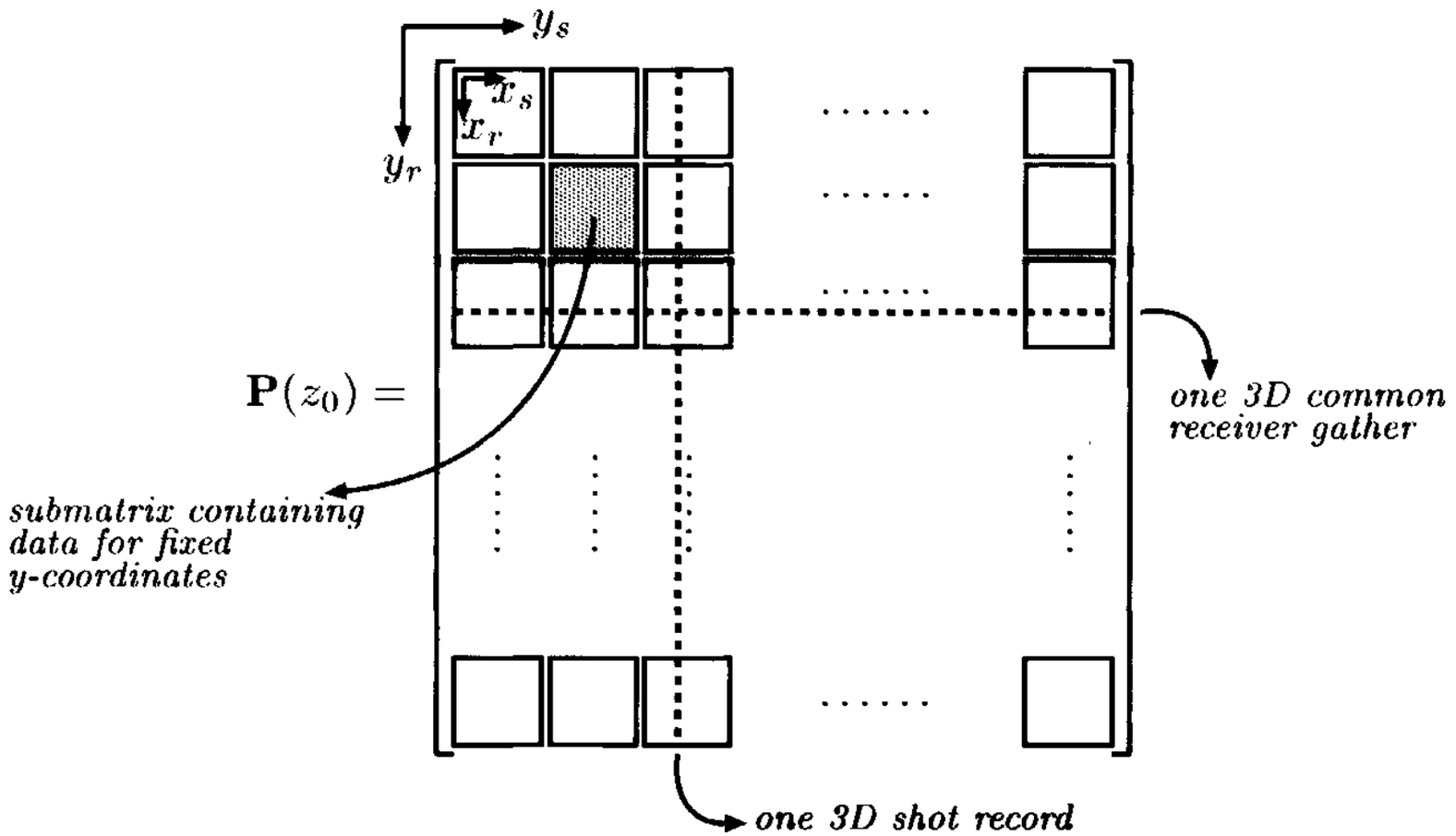


E.J. van Dedem 2002

size of matrix 120^4 in double precision 1.5 GB

Matrix multiplication

"Data Matrix"

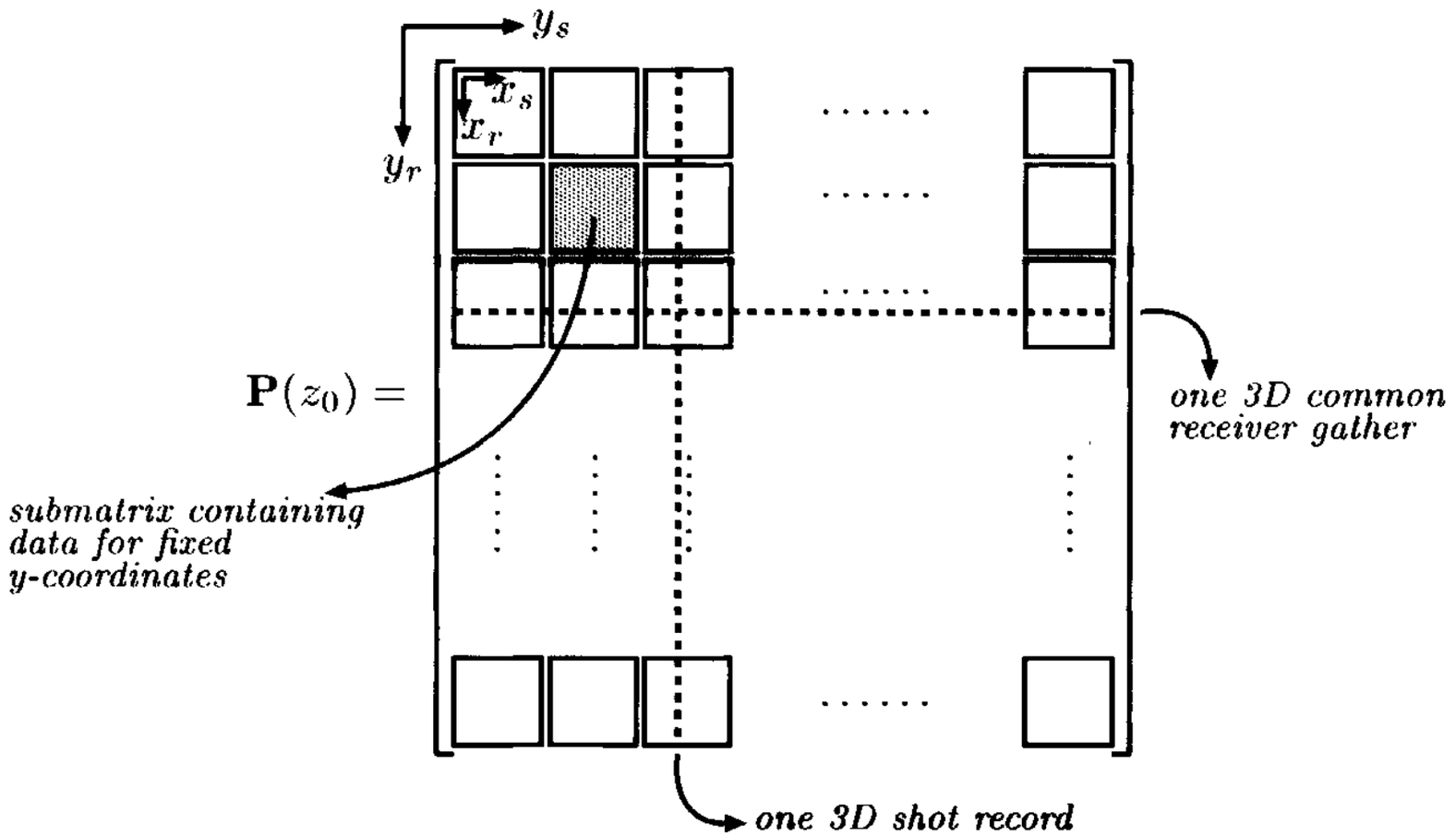


E.J. van Dedem 2002

size of matrix 250^4 in double precision 30 GB

Matrix multiplication

"Data Matrix"



E.J. van Dedem 2002

size of matrix 500^4 in double precision 465 GB

Expressing multi-dim actions

Most multi-dimensional actions can be expressed by the combination of

- Local operations
- Kronecker products
- “n-ary” Block-diagonal operators
- handling of data-side organization, communication, and metadata

Separable examples in Matlab/SPOT

Time-domain FFT

```
A = opKron(opDirac(nx*ny),opFFT(nt))
```

3D FFT

```
A = opKron(opFFT(ny),opFFT(nx),opFFT(nt))
```

1D FFT x 2D Curvelet

```
A = opKron(opCurvelet2D(nx,ny),opFFT(nt))
```

From serial to parallel

Time-domain FFT

```
A = oppKron2Lo(opDirac(nx*ny),opFFT(nt))
```

3D FFT

```
A = opKron2Lo(opFFT2(nx,ny),opFFT(nt))
```

1D FFT x 2D Curvelet

```
A = opKron2Lo(opCurvelet2D(nx,ny),opFFT(nt))
```

Non-separable examples

Frequency-dependent filtering

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

f is (distributed) array of frequencies

@filter(x,f) performs filter on x based on frequency f

Slice-wise matrix-matrix multiply

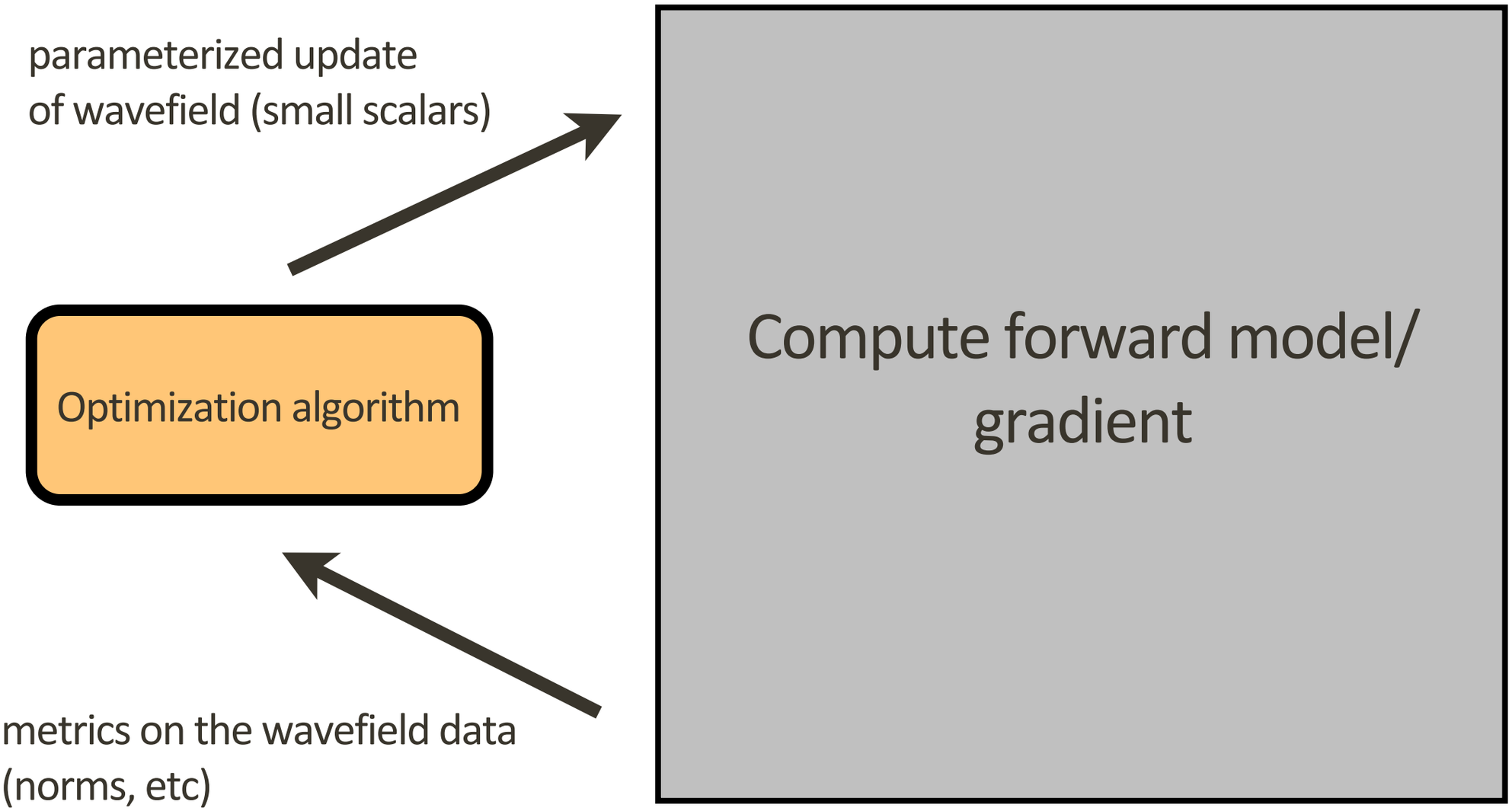
```
A = opNAryBlockDiag(MAT,@matmult)
```

MAT is 3D array distributed over the “slice” dim

@matmult(x,mat) performs mat-mult between x and mat

(relies on conventions for correct array permutation)

Loop-driven inversion



Message-passing to master? BSP-style coordination?

Expressed in Matlab

Operators can be directly used with algorithms that only rely on A*x and A'*x

```
data = distributed(data);

lsqr(A,data);
spgl1(A,data);
```

Key: Inversion algorithm is largely oblivious to underlying operator and data structure implementation

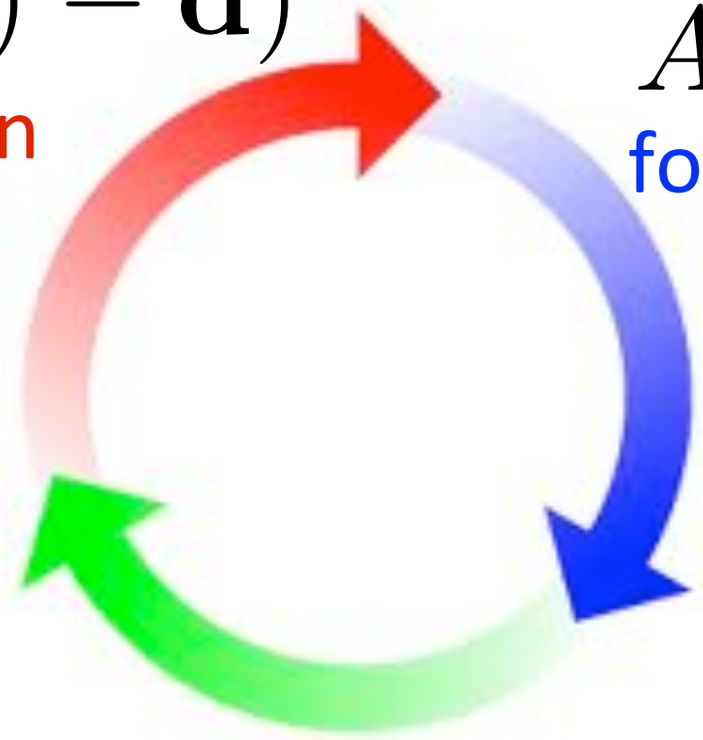
3D frequency domain FWI from an HPC perspective

Tristan van Leeuwen

SLIM
University of British Columbia

$$\min_{\mathbf{m}} \rho(F(\mathbf{m}) - \mathbf{d})$$

formulation



$$A(\mathbf{m})\mathbf{u} = \mathbf{q}$$

forward modelling

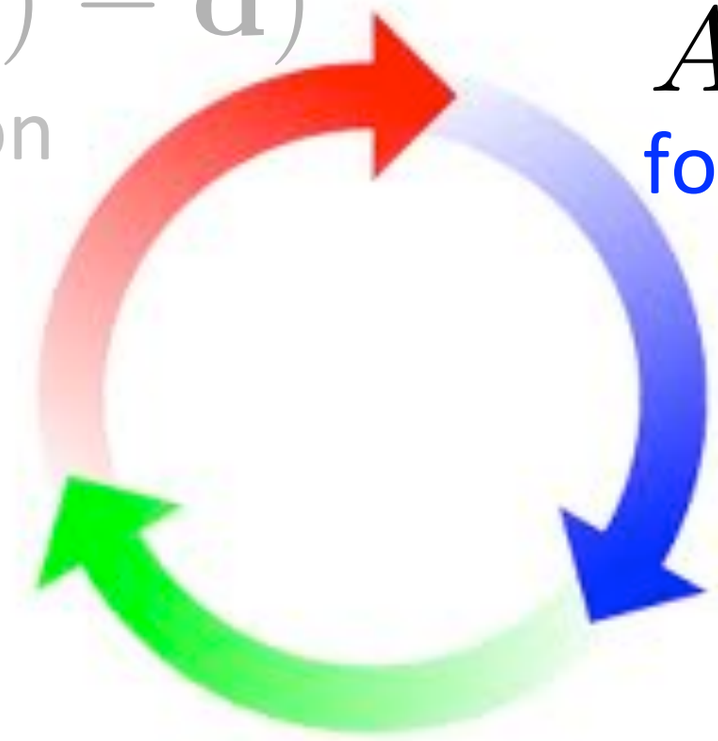
$$\mathbf{m}_{k+1} = \mathbf{m}_k + \alpha_k \mathbf{s}_k$$

optimization strategies

computational framework

$$\min_{\mathbf{m}} \rho(F(\mathbf{m}) - \mathbf{d})$$

formulation



$$A(\mathbf{m})\mathbf{u} = \mathbf{q}$$

forward modelling

$$\mathbf{m}_{k+1} = \mathbf{m}_k + \alpha_k \mathbf{s}_k$$

optimization strategies

computational framework

Forward modelling

We model the data in the frequency domain:

$$A(\omega, \mathbf{m})\mathbf{u}_i = \mathbf{q}_i$$
$$\mathbf{d}_i = P\mathbf{u}_i$$

ω - angular frequency

$\mathbf{m}$ - gridded model parameters $\sim (N_x)^d$

$\mathbf{u}_i, \mathbf{q}_i$ - gridded wavefield and source

A - finite-difference matrix, large bandwidth

$\mathbf{d}_i$ - data $\sim (N_r)^{d-1}$

i - source index $\sim (N_s)^{d-1}$

Forward modelling

Typical problemsize

- receivers $N_r \sim 10^3$
- sources $N_s \sim 10^2$
- gridpoints $N_x \sim 10^3$

Storage and handling (3D FWI)

- data: $N_r^2 \times N_s^2 \sim 10^{10}$ (1 freq. distributed over sources)
- model: $N_x^3 \sim 10^9$ (domain decomposition)
- # of gridpoints scales linearly with frequency

Forward modelling

Direct factorization:

- large bandwidth
- only efficient when re-used for many sources

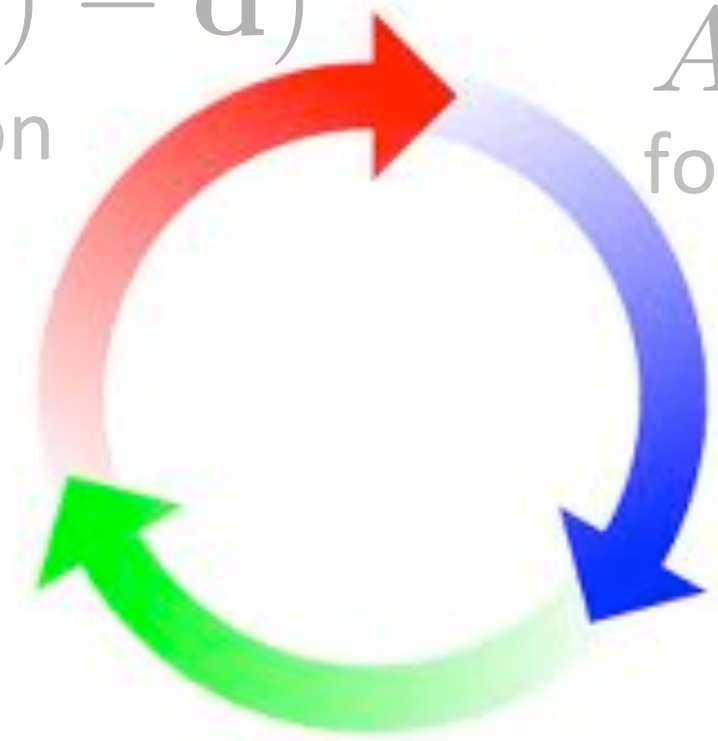
Iterative solvers:

- low memory use (only matrix)
- many sweeps through matrix
- communication due to domain decomposition
- work scales linearly with # sources

use low-level language for performance-critical operations

$$\min_{\mathbf{m}} \rho(F(\mathbf{m}) - \mathbf{d})$$

formulation



$$A(\mathbf{m})\mathbf{u} = \mathbf{q}$$

forward modelling

$$\mathbf{m}_{k+1} = \mathbf{m}_k + \alpha_k \mathbf{s}_k$$

optimization strategies

computational framework

Optimization

Variants of gradient-descent:

- Complicated chains for simple operations (norms, gradient, action of Jacobian)
- most work spent in function evaluation and gradient calculation

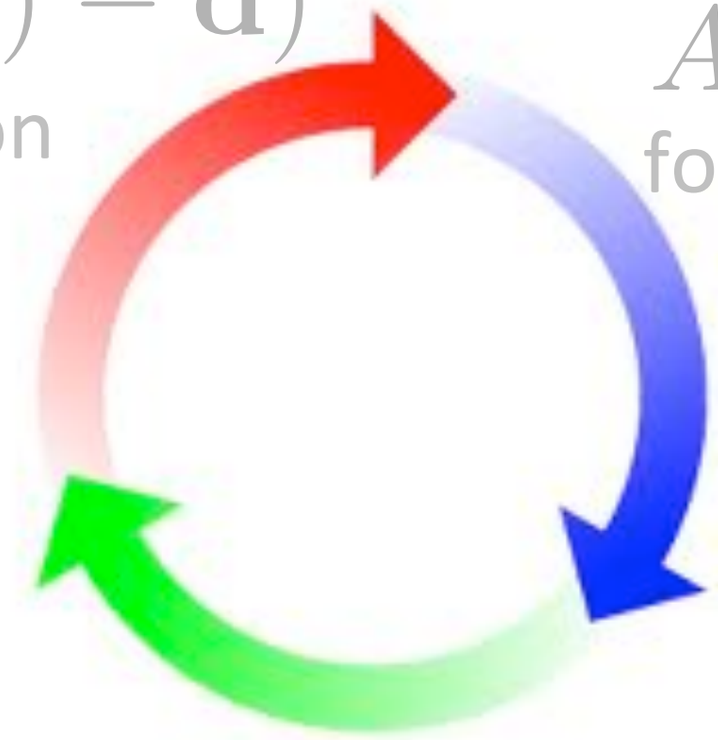
best expressed in high-level language

$$\min_{\mathbf{m}} \rho(F(\mathbf{m}) - \mathbf{d})$$

formulation

$$A(\mathbf{m})\mathbf{u} = \mathbf{q}$$

forward modelling



$$\mathbf{m}_{k+1} = \mathbf{m}_k + \alpha_k \mathbf{s}_k$$

optimization strategies

computational framework

Computational framework

grids,units, ...

coordinate free

PDE solver

misc. operators:
- interpolation
- FFT
- splines
-...

non-linear operator

linear operator

functional

optimization toolbox:
- linear least-squares
- quasi-newton
- ...

Computational framework

Interface modelling and optimization software through object-oriented framework:

- implementation in parallel matlab
- use linear operator toolbox (P)SPOT
- include unit-tests (Jacobian, adjoint, gradient)

[TVL ’12]

Computational framework

Modelling

```
[D,J] = F(m,Q,model)
```

D - data cube as Matlab distributed array

`J` - Jacobian as SPOT operator

`m` - model as Matlab array

`Q` - source functions [q1, q2, ...]

`model` - struct containing acquisition setup etc.

Matlab framework

Action of the Jacobian

```
D    = J*dm;
% calls J.multiply(dm,1)
dmt = J'*D;
% calls J.multiply(D,-1)
```

Least-squares inversion

```
% construct operator
J  = opDF(m0,Q,model);
% call standard LSQR
dm = LSQR(J,D);
```

Matlab example

misfit

```
function [f,g] = Phi(m,Q,D,model)

[Dt,Jt]   = F(m,Q,model);
[f,df]    = penalty(Dt-D);
g         = Jt'*df;
```

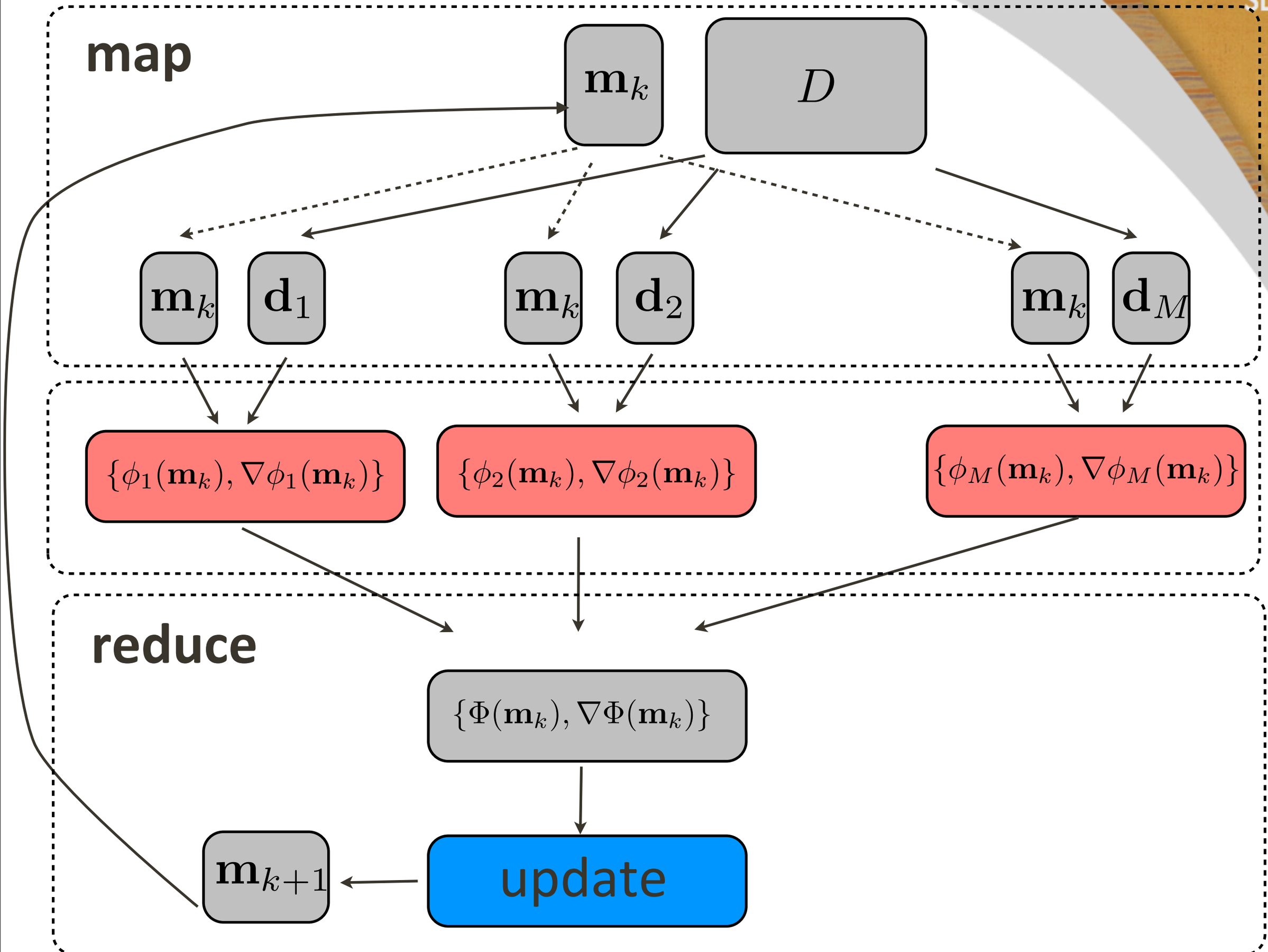
Optimization

```
% function handle
fh = @(m)misfit(m,Q,D,model);

% optimization
mn = lbfgs(fh,m0);
```

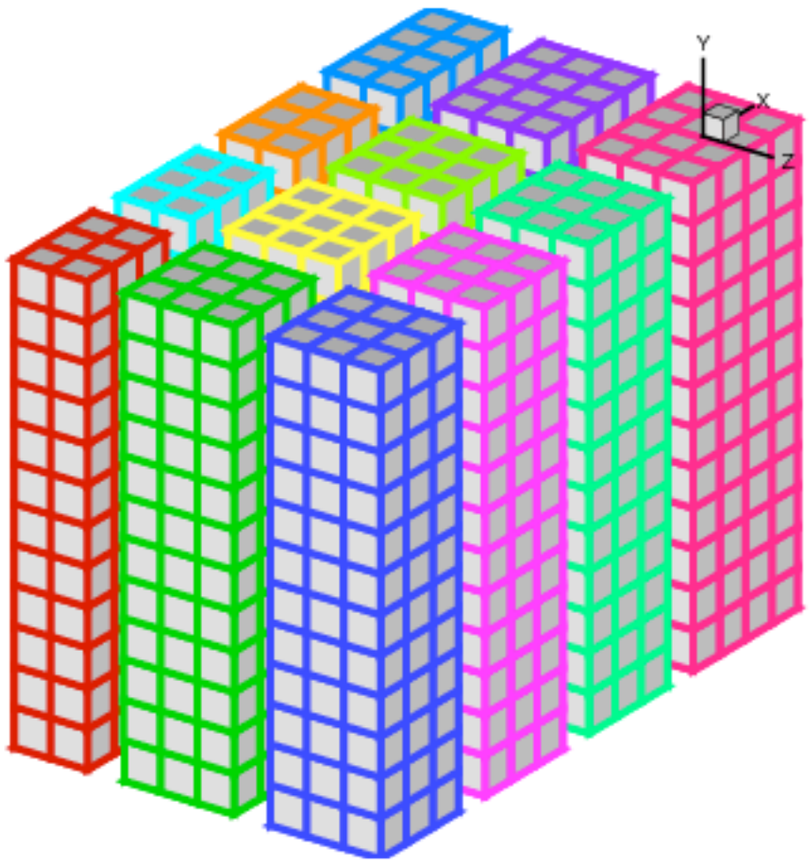
Matlab framework

- *Very easy* design and testing of algorithms
- Data-parallelism is handled by Matlab
- Future work aimed at (out-of-core) data abstractions in Matlab
- Code should scale to large problems, given enough workers.


map
$\mathbf{m}_k$
D
$\mathbf{m}_k$ $\mathbf{d}_1$
$\mathbf{m}_k$ $\mathbf{d}_2$
$\mathbf{m}_k$ $\mathbf{d}_M$
$\{\phi_1(\mathbf{m}_k), \nabla\phi_1(\mathbf{m}_k)\}$
$\{\phi_2(\mathbf{m}_k), \nabla\phi_2(\mathbf{m}_k)\}$
$\{\phi_M(\mathbf{m}_k), \nabla\phi_M(\mathbf{m}_k)\}$
reduce
$\{\Phi(\mathbf{m}_k), \nabla\Phi(\mathbf{m}_k)\}$
update
$\mathbf{m}_{k+1}$

$\{\phi_i(\mathbf{m}_k), \nabla\phi_i(\mathbf{m}_k)\}$

`Non-worker' code to compute misfit and gradient



$$A(\mathbf{m})\mathbf{u}_i = \mathbf{q}_i$$

$$A(\mathbf{m})^H\mathbf{v}_i = P_i^T\nabla\rho(\mathbf{d}_i - P_i\mathbf{u}_i)$$

$$\phi_i(\mathbf{m}) = \rho(\mathbf{d}_i - P_i\mathbf{u}_i)$$

$$\frac{\partial\phi_i}{\partial m_k} = \mathbf{u}_i^*\left(\frac{\partial A(\mathbf{m})}{\partial m_k}\right)^*\mathbf{v}_i$$

SLIM's perspective on HPC & big data

Felix J. Herrmann, Tim Lin, and Tristan van Leeuwen

Seismic Laboratory for Imaging and Modeling
the University of British Columbia

Challenges

Manageable & *flexible* software development *conducive* to

- making *progress* on exciting *science*
- without sacrificing *performance* and *scalability*

Guarantee *uptake* by *industry* w.r.t. *scalability* & *viable* license model

How to *close* the loop so workable *feedback* from *industry* makes it back into our *code* base...

Our approach

Exploit *structure* of wave-equation based inversion

- *data*-space *parallelism*–e.g., via MapReduce
- *model*-space *parallelism*–e.g., w/ *intra-node* parallel hardware
- operator *parallelism–via* Kronecker products etc.
  (permutations & contiguous, distributed, out-of-core)

Use *heterogeneous* object-oriented *development* environment

- encapsulating *low-level* code w/ *high-level* abstractions
- handling of *meta* data w/ data*containers* & *overloading*

Design principles
[*a la* Symes]

Hide/expose *meta* data w/ *data containers & overloading*
(sizes, domain/ranges; units, sample intervals, source locations, etc.)

Coordinate-free *vectorization/matrization*

- *setup optimization* problems w/ *meta*data propagation
  (tensors, reshapes, permutations, *parallel/out-of-core* data)

Domain-specific *passing* of *meta*data

- guarantee *physical* sense of PDE solves
- *computation* of proper *norms* for misfits–e.g., *physical* energy

Working prototypes

Linear *operator* abstractions with SPOT, pSPOT, oSPOT
(matrix-free vector calculus on *distributed* & *out-of-core* arrays)

- *robust* EPSI proved to be *scalable* – presented by Tim
- undergirds work on *hierarchical* Tucker – presented by Curt

Matrix-free framework for modelling & inversion–presented by Tristan
(*abstracted* Jacobians, gradients, norms, & *modularity* PDE solves)

- 2/3D *robust* q-Newton FWI, *modified* GN-FWI, *imaging* w/ multiples

Data abstractions with SeisDataContainer
(*in-out of core*, handling & propagation of *meta*data, support for IO)

Missing

Integration of *data* abstraction w/ PDE *solvers* & misfit *functionals*

- *handling* of *meta*data (easy)

Framework to handle *massive* data ("big data")

- *parallel* IO – including "transposes" (not so easy)
- MapReduce
  - Swift – *functional* parallelism (made progress)
  - Hadoop to explore *data* parallelism (?)
- GraphLab to explore *graph* parallelism (??)