CBMS Conference on Fast Direct Solvers

Dartmouth College

June 23 – June 27, 2014

Lecture 11: Concluding remarks

Gunnar Martinsson

The University of Colorado at Boulder

Research support by:

Solution operators of elliptic PDEs are *very* nice

Recall from Lecture 1 that the solution operator of a linear elliptic PDE such as, e.g.,

$$
\begin{cases} A\,u(\boldsymbol{x}) = g(\boldsymbol{x}), & \boldsymbol{x} \in \Omega, \\ u(\boldsymbol{x}) = 0, & \boldsymbol{x} \in \Gamma, \end{cases} \tag{1}
$$

(constant coefficients, or not) takes the form

$$
u(\boldsymbol{x}) = [\mathcal{G}g](\boldsymbol{x}) = \int_\Omega G(\boldsymbol{x},\boldsymbol{y})\,g(\boldsymbol{y})\,dA(\boldsymbol{y}), \qquad \boldsymbol{x} \in \Omega.
$$

We have seen a range of algorithms for numerically building approximations to $\mathcal{G}$ in different contexts; these algorithms have to deal with the fact that $\mathcal{G}$ is a *global operator,* which is approximated by a *dense matrix.* They require a fair amount of storage.

The upside is that the operator $\mathcal{G}$ is typically a *very benign* mathematical object. For instance, we can often split the operator as

$$
\mathcal{G} = \mathcal{G}_{\rm near} + \mathcal{G}_{\rm far},
$$

where

- $\mathcal{G}_{\rm near}$ is typically moderately smoothing, and *strictly local.*
  Algorithms for applying and storing $\mathcal{G}_{\rm near}$ need essentially no communication.
- $\mathcal{G}_{\rm far}$ is global, but *strongly smoothing* $\rightarrow$ it blurs information over distance.

Solution operators of elliptic PDEs are *very* nice

Example: Consider the Poisson equation

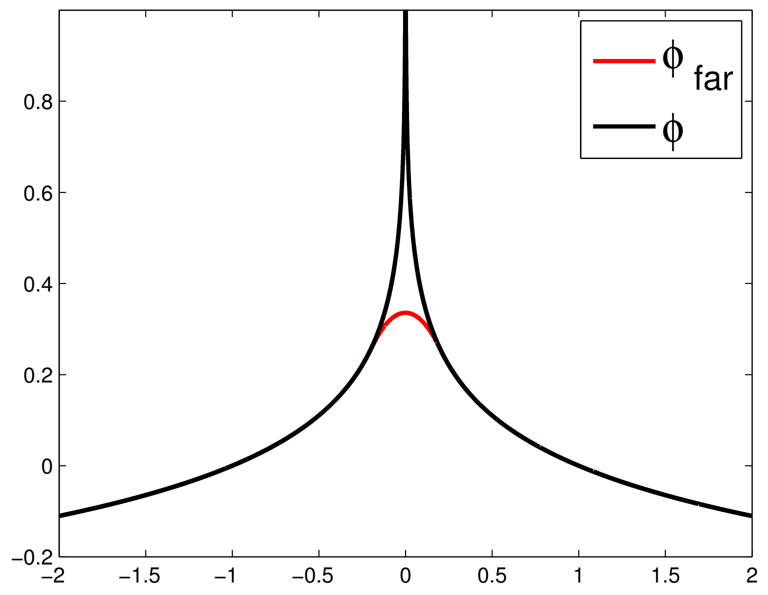
$$\begin{cases} -\Delta u(\boldsymbol{x}) = g(\boldsymbol{x}), & \boldsymbol{x} \in \mathbb{R}^3 \\ \lim_{|\boldsymbol{x}|\to\infty} |u(\boldsymbol{x})| = 0 & \end{cases}$$

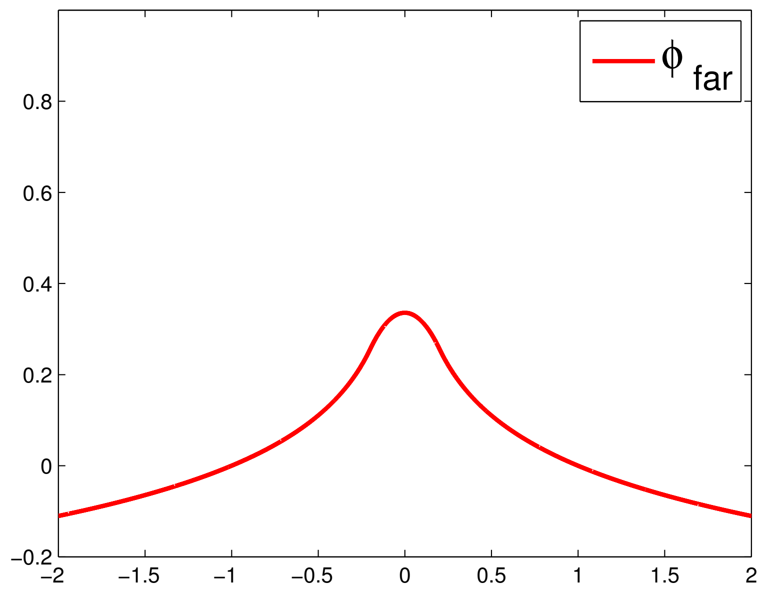
The solution operator takes the form

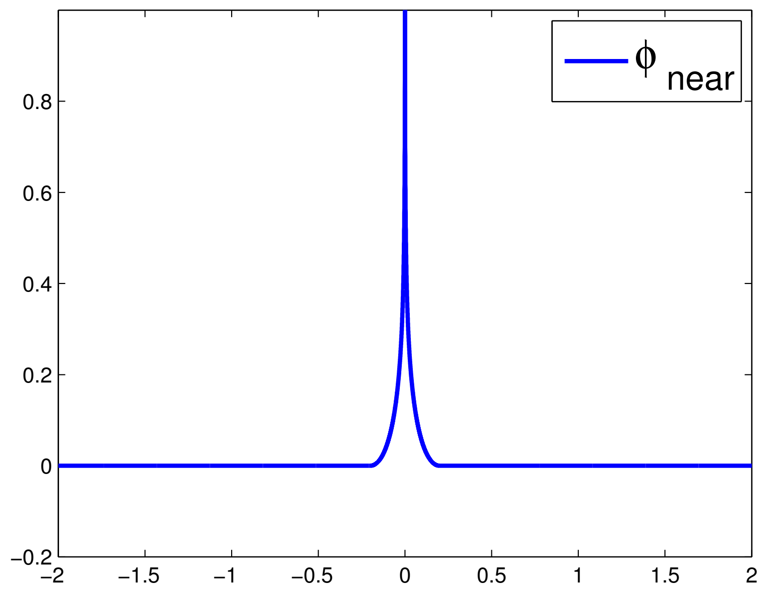
$$u(\boldsymbol{x}) = [\mathcal{G}g](\boldsymbol{x}) = [\phi * g](\boldsymbol{x}) = \int_{\mathbb{R}^3} \phi(\boldsymbol{x}-\boldsymbol{y})\, g(\boldsymbol{y})\, d\boldsymbol{y}, \qquad \boldsymbol{x} \in \mathbb{R}^3,$$

where $\phi(\boldsymbol{x}) = -\dfrac{1}{4\pi|\boldsymbol{x}|}$ is the fundamental solution. Now let us *smooth* ϕ slightly,

$$\phi \qquad = \qquad \phi_{\rm far} \qquad + \qquad \phi_{\rm near}$$







C^∞ function

locally supported

Solution operators of elliptic PDEs are *very* nice

Example: Consider the Poisson equation

$$\begin{cases} -\Delta u(\boldsymbol{x}) = g(\boldsymbol{x}), & \boldsymbol{x} \in \mathbb{R}^3 \\ \lim_{|\boldsymbol{x}|\to\infty} |u(\boldsymbol{x})| = 0 \end{cases}$$

The solution operator takes the form

$$u(\boldsymbol{x}) = [\mathcal{G}g](\boldsymbol{x}) = [\phi * g](\boldsymbol{x}) = \int_{\mathbb{R}^3} \phi(\boldsymbol{x} - \boldsymbol{y})\, g(\boldsymbol{y})\, d\boldsymbol{y}, \qquad \boldsymbol{x} \in \mathbb{R}^3,$$

where $\phi(\boldsymbol{x}) = -\frac{1}{4\pi|\boldsymbol{x}|}$ is the fundamental solution.

Now let us *smooth* ϕ slightly, $\phi = \phi_{\rm far} + \phi_{\rm near}$.

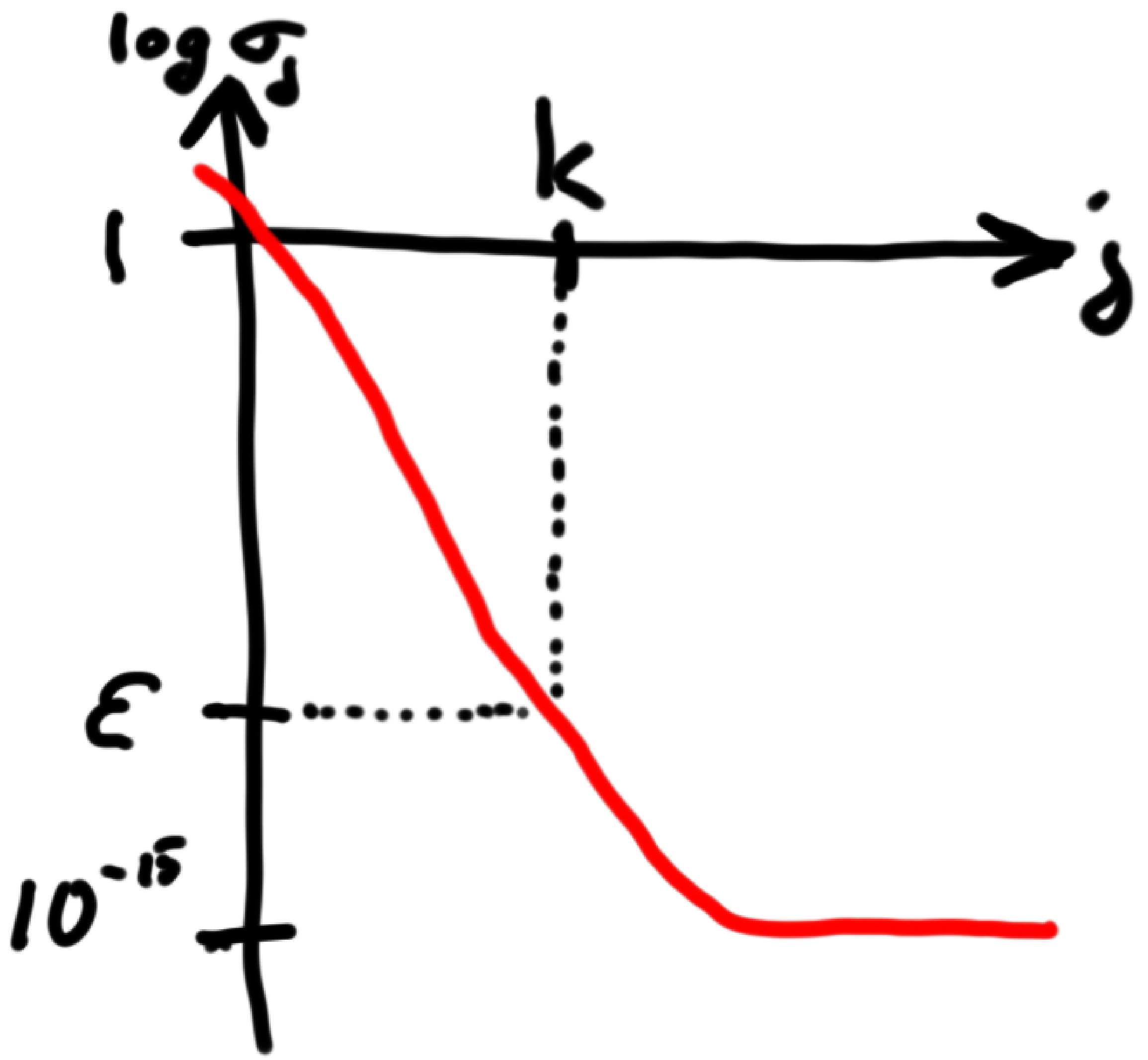
Then $\mathcal{G} = \mathcal{G}_{\rm far} + \mathcal{G}_{\rm near}$ where $\mathcal{G}_{\rm far}\, g = \phi_{\rm far} * g$ and $\mathcal{G}_{\rm far}\, g = \phi_{\rm near} * g$.

- The operator $\mathcal{G}_{\rm near}$ is slightly smoothing, but *local*.
- The operator $\mathcal{G}_{\rm far}$ is extremely smoothing, but *global*.

What does it mean that the solution operator $\mathcal{G}$ "blurs information"?

Let Ω_1 and Ω_2 be two "separated" subdomains, and let $\mathcal{G}_{12}$ denote the restriction of $\mathcal{G}$.

Then the singular values of $\mathcal{G}$ "typically" look like:

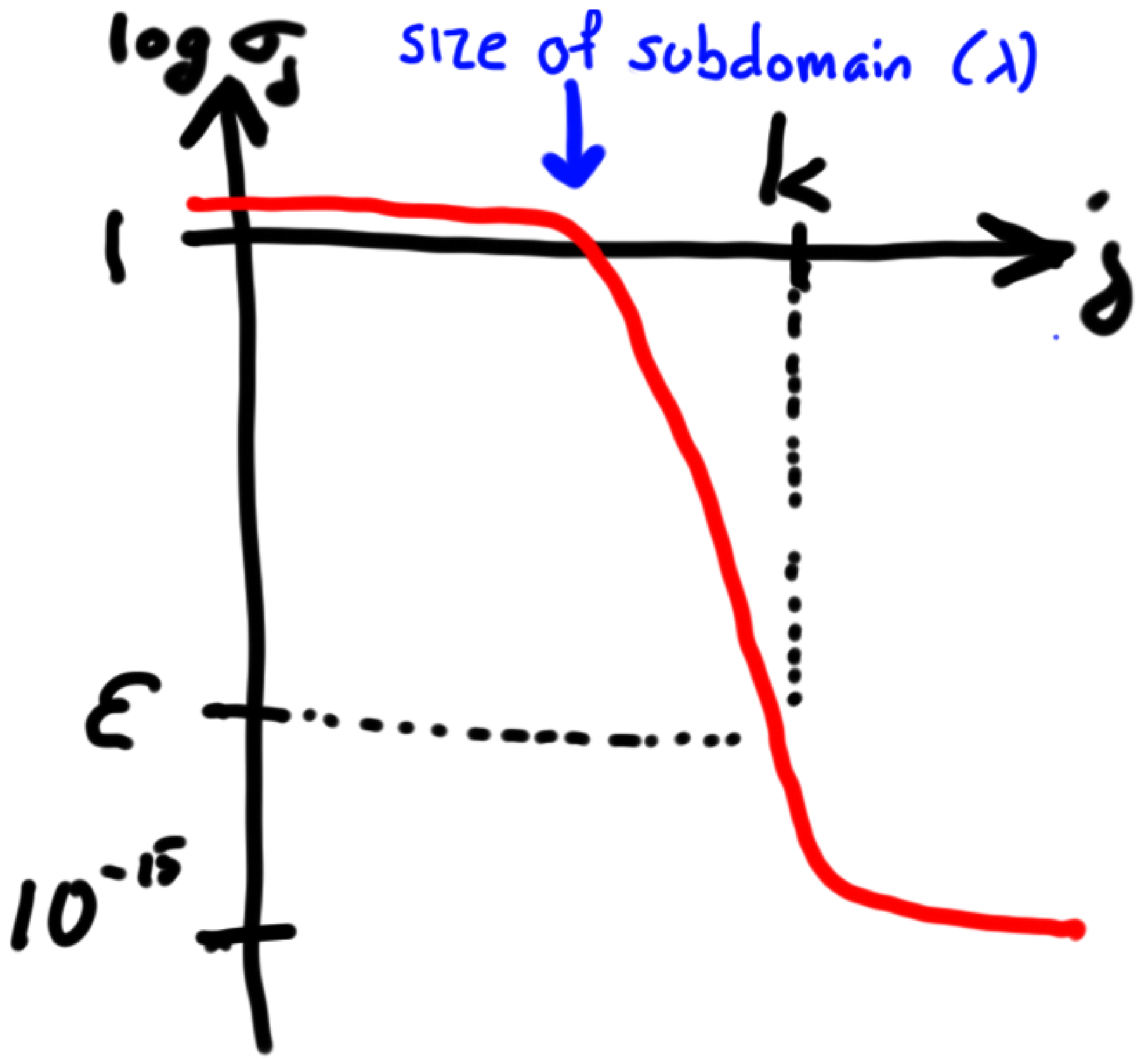


Typical singular values for problems with non-oscillatory kernels: Laplace, elasticity, Stokes, etc.

- $\mathcal{G}$ can be the inverse of a finite difference matrix.
- $\mathcal{G}$ can be a "frontal matrix" in nested dissection.
- $\mathcal{G}$ can be a DtN operator in the Hierarchical Poincaré-Steklov scheme.
- $\mathcal{G}$ can be either a boundary integral operator, or the inverse of such an operator.

What does it mean that the solution operator $\mathcal{G}$ "blurs information"?

Let Ω_1 and Ω_2 be two "separated" subdomains, and let $\mathcal{G}_{12}$ denote the restriction of $\mathcal{G}$. Then the singular values of $\mathcal{G}$ "typically" look like:



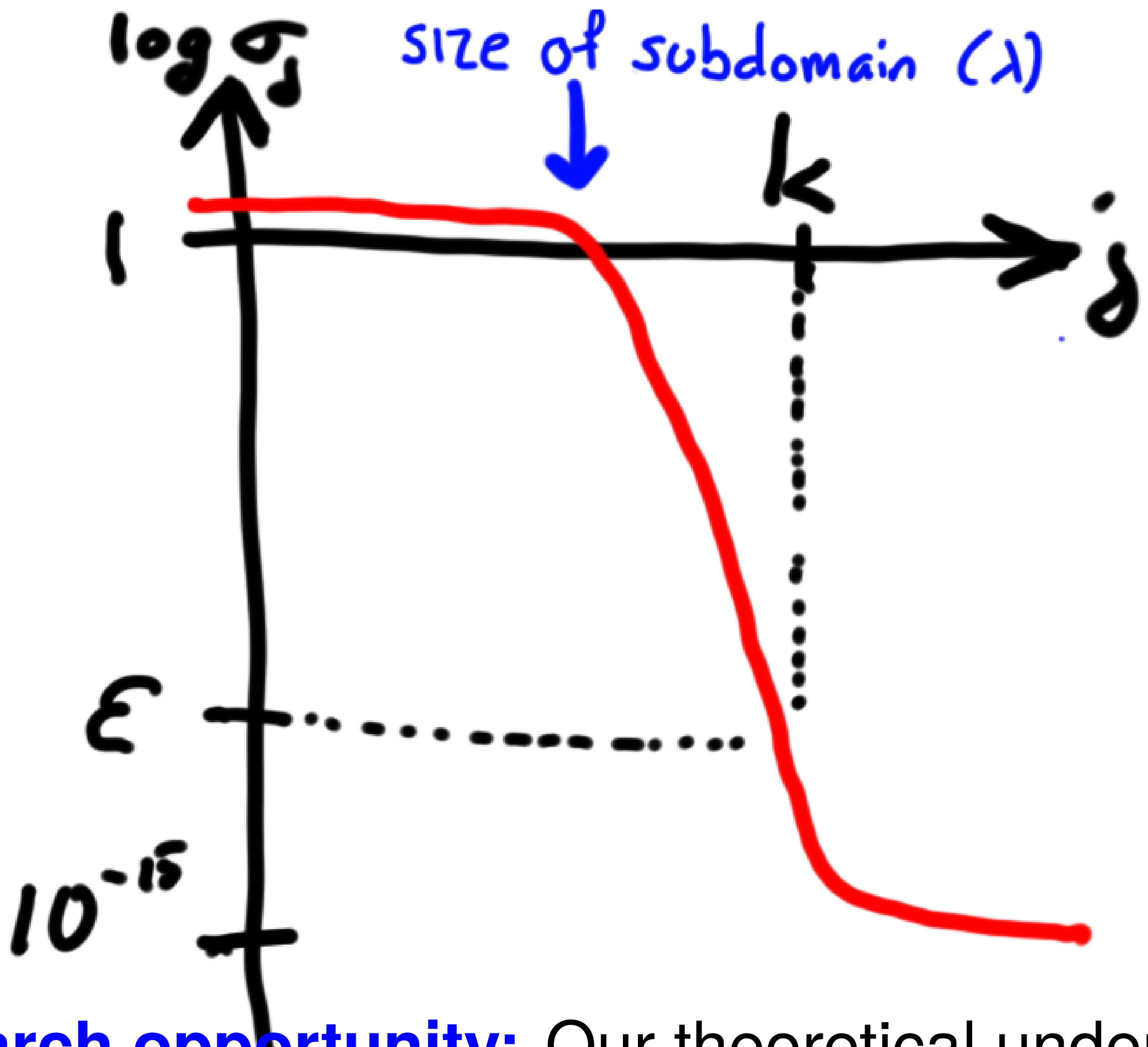
Typical singular values for problems with oscillatory kernels: Helmholtz, time-harmonic Maxwell, etc.

- $\mathcal{G}$ can be the inverse of a finite difference matrix.
- $\mathcal{G}$ can be a "frontal matrix" in nested dissection.
- $\mathcal{G}$ can be a DtN operator in the Hierarchical Poincaré-Steklov scheme.
- $\mathcal{G}$ can be either a boundary integral operator, or the inverse of such an operator.

What does it mean that the solution operator $\mathcal{G}$ "blurs information"?

Let Ω_1 and Ω_2 be two "separated" subdomains, and let $\mathcal{G}_{12}$ denote the restriction of $\mathcal{G}$.

Then the singular values of $\mathcal{G}$ "typically" look like:



Typical singular values for problems with oscillatory kernels: Helmholtz, time-harmonic Maxwell, etc.

Research opportunity: Our theoretical understanding of this decay is very incomplete.

We have good analysis of extremely simple situations, e.g., when $\mathcal{G}$ is simply convolution by a known fundamental solution.

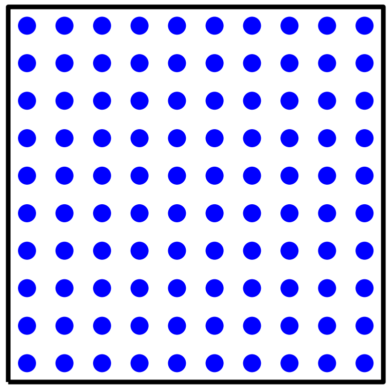
We base algorithms of heuristics, which makes *adaptivity* and *validation* essential.

Let the algorithm figure out the decay rate, and warn the user if it is not as expected.

In case decay is slow, the problem should be slower execution time, not loss of accuracy.

A common pattern: Hierarchical domain decomposition

Consider a PDE $Au = f$ defined on a square $\Omega = [0, 1]$. Put a grid on the square.

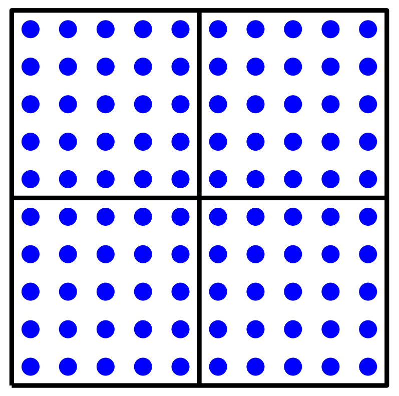


The original grid.

A common pattern: Hierarchical domain decomposition

Consider a PDE $Au = f$ defined on a square $\Omega = [0, 1]$. Put a grid on the square.

Split the domain into “small” patches we call “leaves” (they will be organized in a tree).



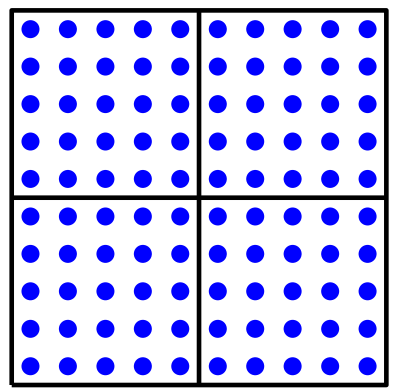
The original grid.

A common pattern: Hierarchical domain decomposition

Consider a PDE $Au = f$ defined on a square $\Omega = [0, 1]$. Put a grid on the square.

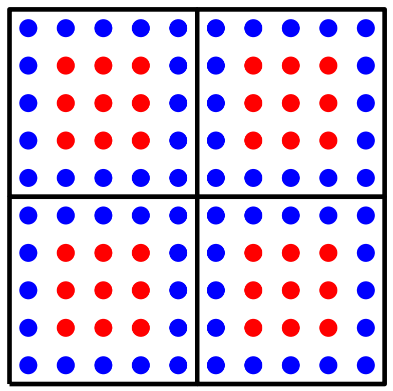
Split the domain into “small” patches we call “leaves” (they will be organized in a tree).

On each leaf, compute by “brute force” a local solution operator (e.g. a DtN operator).
This eliminates “internal” grid points from the computation. (“Static condensation.”)



The original grid.

$\xrightarrow{(1)}$



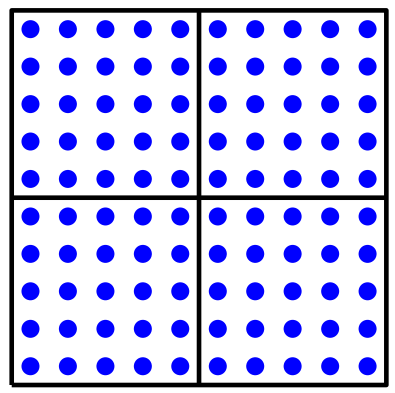
Leaves reduced.

A common pattern: Hierarchical domain decomposition

Consider a PDE $Au = f$ defined on a square $\Omega = [0, 1]$. Put a grid on the square.

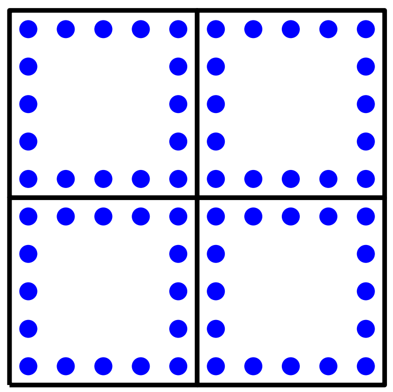
Split the domain into “small” patches we call “leaves” (they will be organized in a tree).

On each leaf, compute by “brute force” a local solution operator (e.g. a DtN operator).
This eliminates “internal” grid points from the computation. (“Static condensation.”)



The original grid.

$\xrightarrow{(1)}$



Leaves reduced.

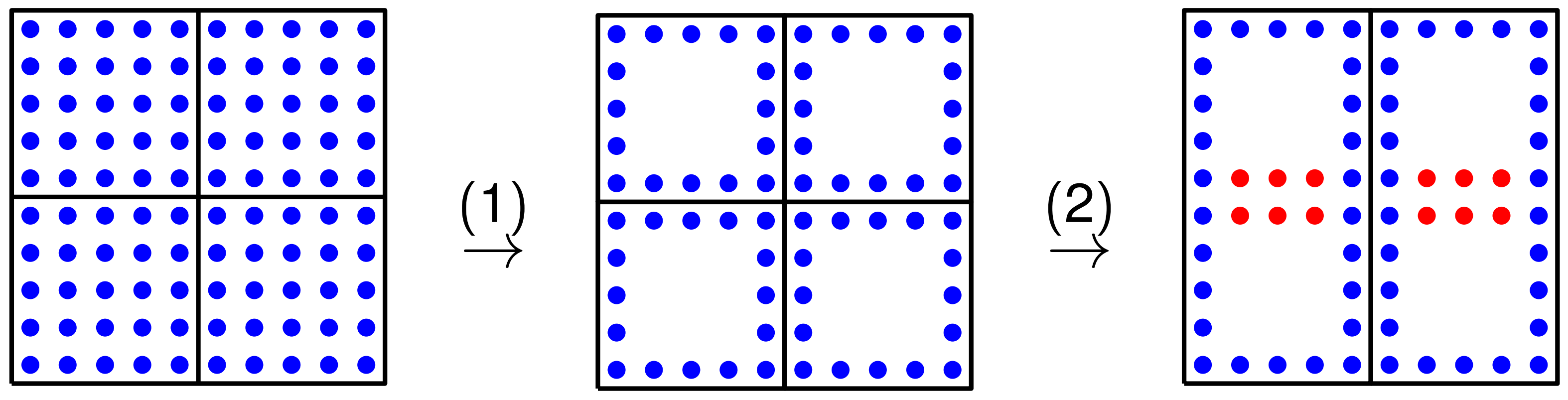
A common pattern: Hierarchical domain decomposition

Consider a PDE $Au = f$ defined on a square $\Omega = [0, 1]$. Put a grid on the square.

Split the domain into “small” patches we call “leaves” (they will be organized in a tree).

On each leaf, compute by “brute force” a local solution operator (e.g. a DtN operator).
This eliminates “internal” grid points from the computation. (“Static condensation.”)

Merge the leaves in pairs of two.



The original grid. *Leaves reduced.* *After merge.*

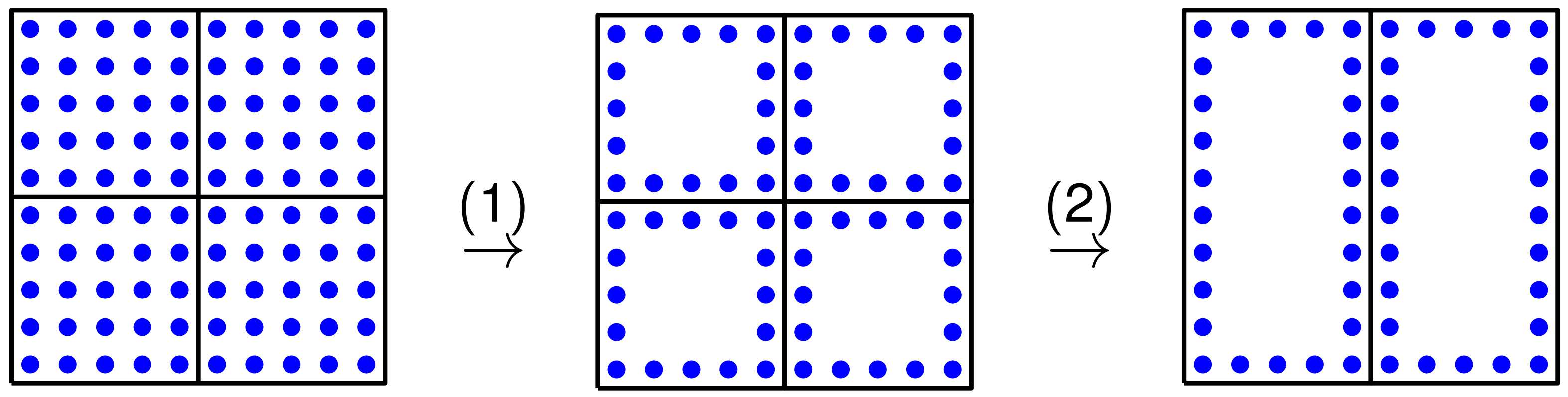
A common pattern: Hierarchical domain decomposition

Consider a PDE *Au* $= f$ defined on a square $\Omega = [0,1]$. Put a grid on the square.

Split the domain into “small” patches we call “leaves” (they will be organized in a tree).

On each leaf, compute by “brute force” a local solution operator (e.g. a DtN operator). This eliminates “internal” grid points from the computation. (“Static condensation.”)

Merge the leaves in pairs of two. For each pair, compute a local solution operator by combining the solution operators of the two leaves.



The original grid. *Leaves reduced.* *After merge.*

A common pattern: Hierarchical domain decomposition

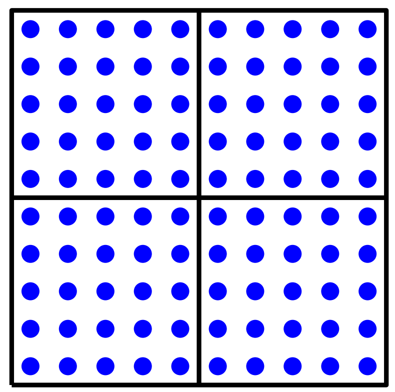
Consider a PDE $Au = f$ defined on a square $\Omega = [0, 1]$. Put a grid on the square.

Split the domain into “small” patches we call “leaves” (they will be organized in a tree).

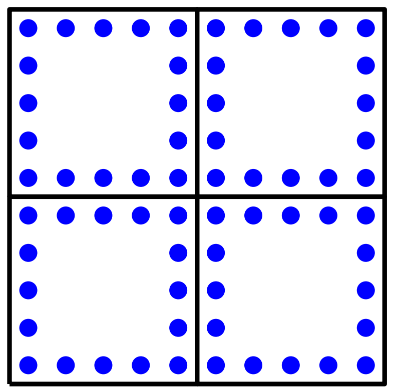
On each leaf, compute by “brute force” a local solution operator (e.g. a DtN operator). This eliminates “internal” grid points from the computation. (“Static condensation.”)

Merge the leaves in pairs of two. For each pair, compute a local solution operator by combining the solution operators of the two leaves.

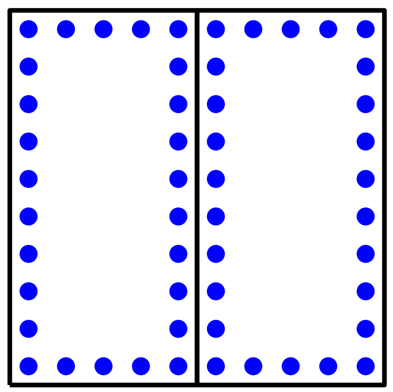
Continue merging by pairs, organizing the domain in a tree of patches.


The original grid.

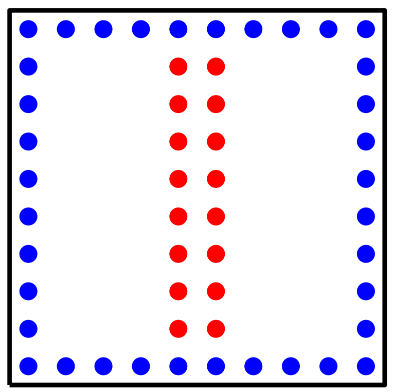
$\xrightarrow{(1)}$


Leaves reduced.

$\xrightarrow{(2)}$


After merge.

$\xrightarrow{(3)}$


After merge.

A common pattern: Hierarchical domain decomposition

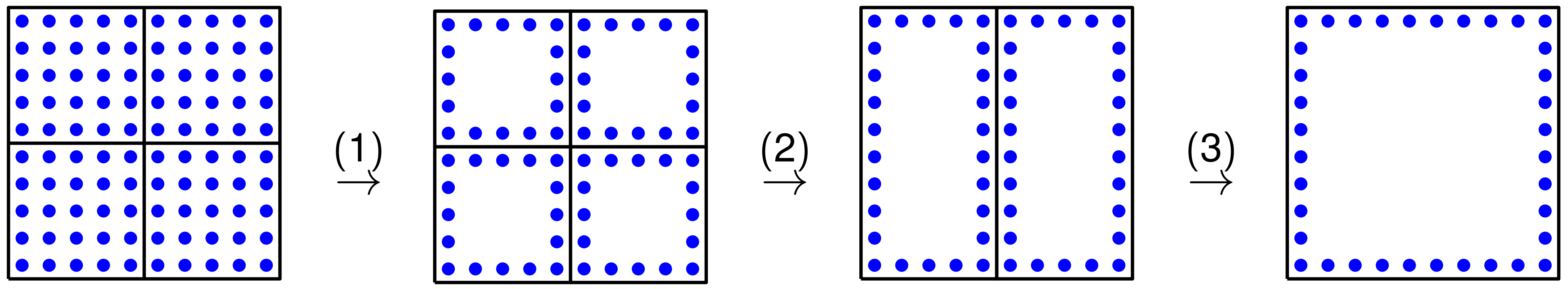
Consider a PDE $Au = f$ defined on a square $\Omega = [0, 1]$. Put a grid on the square.

Split the domain into “small” patches we call “leaves” (they will be organized in a tree).

On each leaf, compute by “brute force” a local solution operator (e.g. a DtN operator). This eliminates “internal” grid points from the computation. (“Static condensation.”)

Merge the leaves in pairs of two. For each pair, compute a local solution operator by combining the solution operators of the two leaves.

Continue merging by pairs, organizing the domain in a tree of patches.



The original grid. *Leaves reduced.* *After merge.* *After merge.*

A common pattern: Hierarchical domain decomposition

Consider a PDE $Au = f$ defined on a square $\Omega = [0, 1]$. Put a grid on the square.

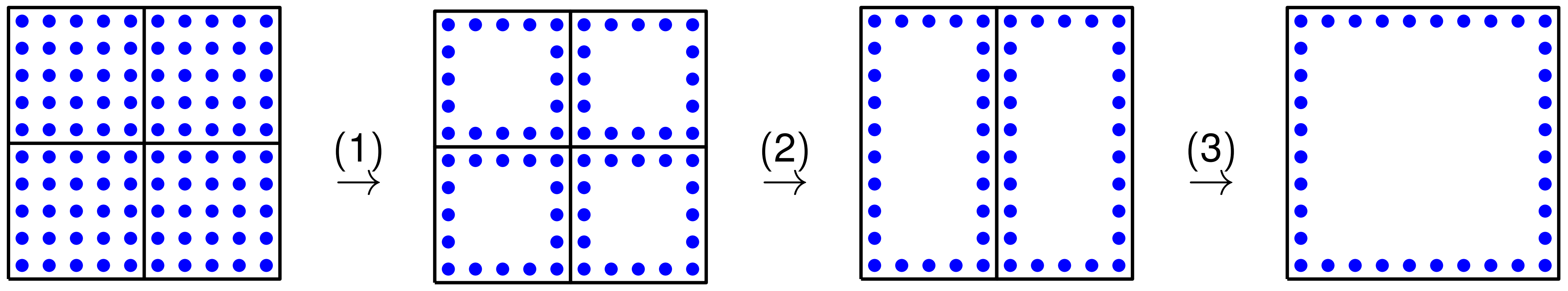
Split the domain into “small” patches we call “leaves” (they will be organized in a tree).

On each leaf, compute by “brute force” a local solution operator (e.g. a DtN operator). This eliminates “internal” grid points from the computation. (“Static condensation.”)

Merge the leaves in pairs of two. For each pair, compute a local solution operator by combining the solution operators of the two leaves.

Continue merging by pairs, organizing the domain in a tree of patches.

When you reach the top, invert/factor/apply the solution operator for the entire domain.



The original grid. *Leaves reduced.* *After merge.* *After merge.*

A common pattern: Hierarchical domain decomposition

Consider a PDE $Au = f$ defined on a square $\Omega = [0, 1]$. Put a grid on the square.

Split the domain into “small” patches we call “leaves” (they will be organized in a tree).

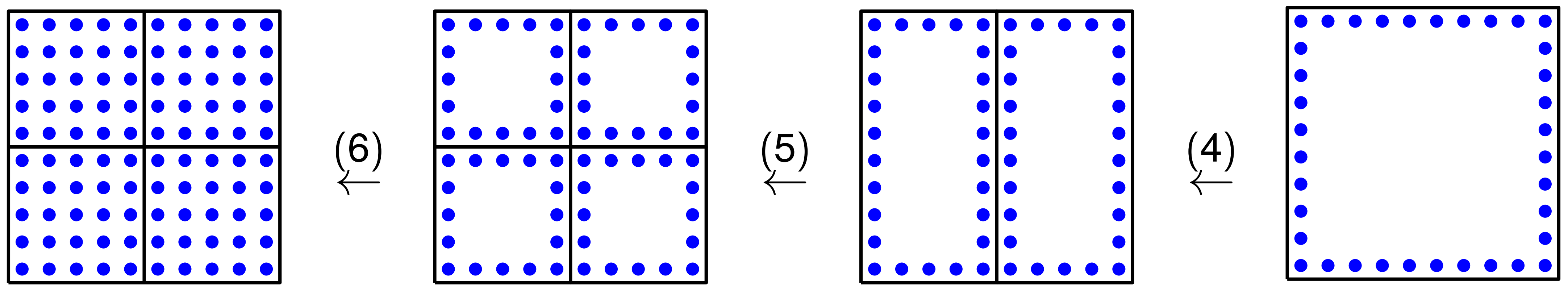
On each leaf, compute by “brute force” a local solution operator (e.g. a DtN operator). This eliminates “internal” grid points from the computation. (“Static condensation.”)

Merge the leaves in pairs of two. For each pair, compute a local solution operator by combining the solution operators of the two leaves.

Continue merging by pairs, organizing the domain in a tree of patches.

When you reach the top, invert/factor/apply the solution operator for the entire domain.

Then reconstruct the solution at all internal points via a downwards pass.

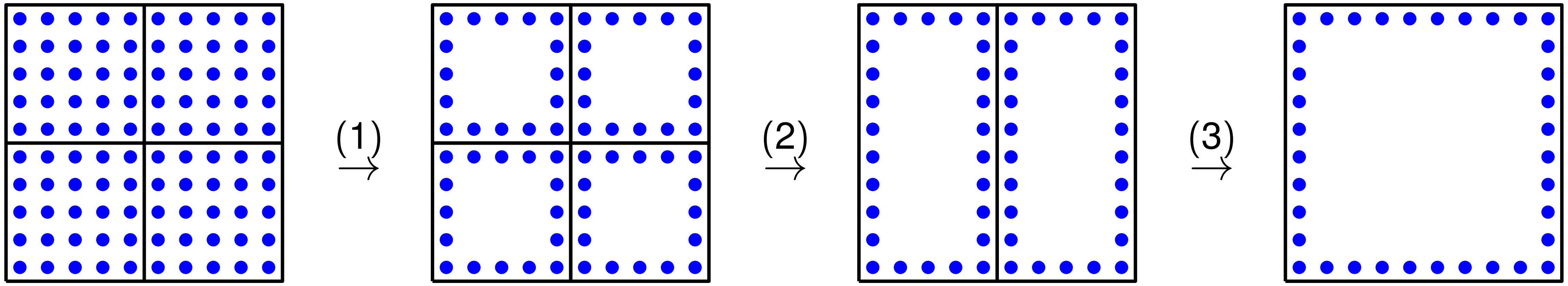


Full solution. *Solve.* *Solve.* *Top level solve.*

The idea of hierarchical domain decomposition

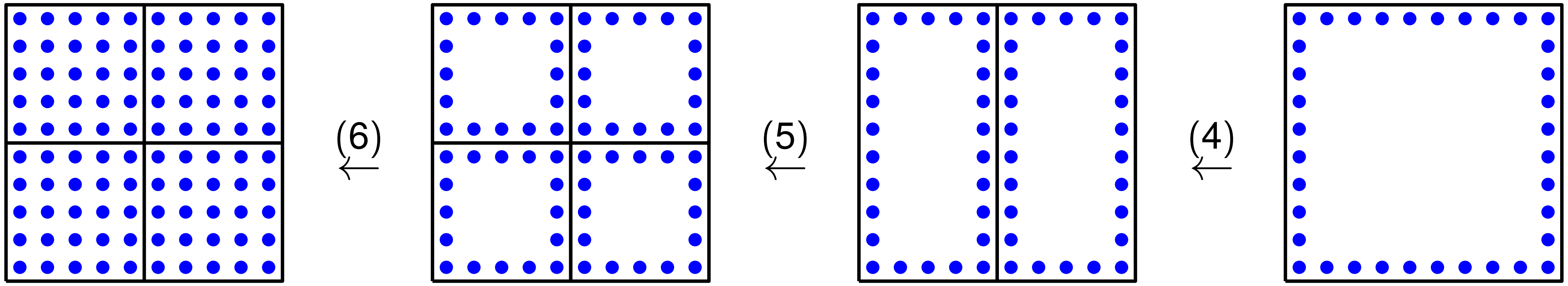
All direct solvers we have described follow the same basic pattern:

Upwards pass — build all solution operators:



The original grid. *Leaves reduced.* *After merge.* *After merge.*

Downwards pass — solve for a particular data function (*very* fast!):



Full solution. *Solve.* *Solve.* *Top level solve.*

Some local *"solution operators"* are built in the upwards pass — first on the leaves via a brute force computation, then by merging them in pairs.

Examples of “local solution operators”:

Sparse matrices from FD/FEM discretization:

- Local solution operators are, e.g., discrete analogs of Dirichlet-to-Neumann operators.
- Strictly boundary-to-boundary due to the fact that the discretized operators are local.
- The “width” increases as you increase the order.

Hierarchical Poincaré-Steklov scheme:

- Conceptually based on merging of “continuum operators.”
- Strictly boundary-to-boundary.
- Very thin interfaces regardless of order.

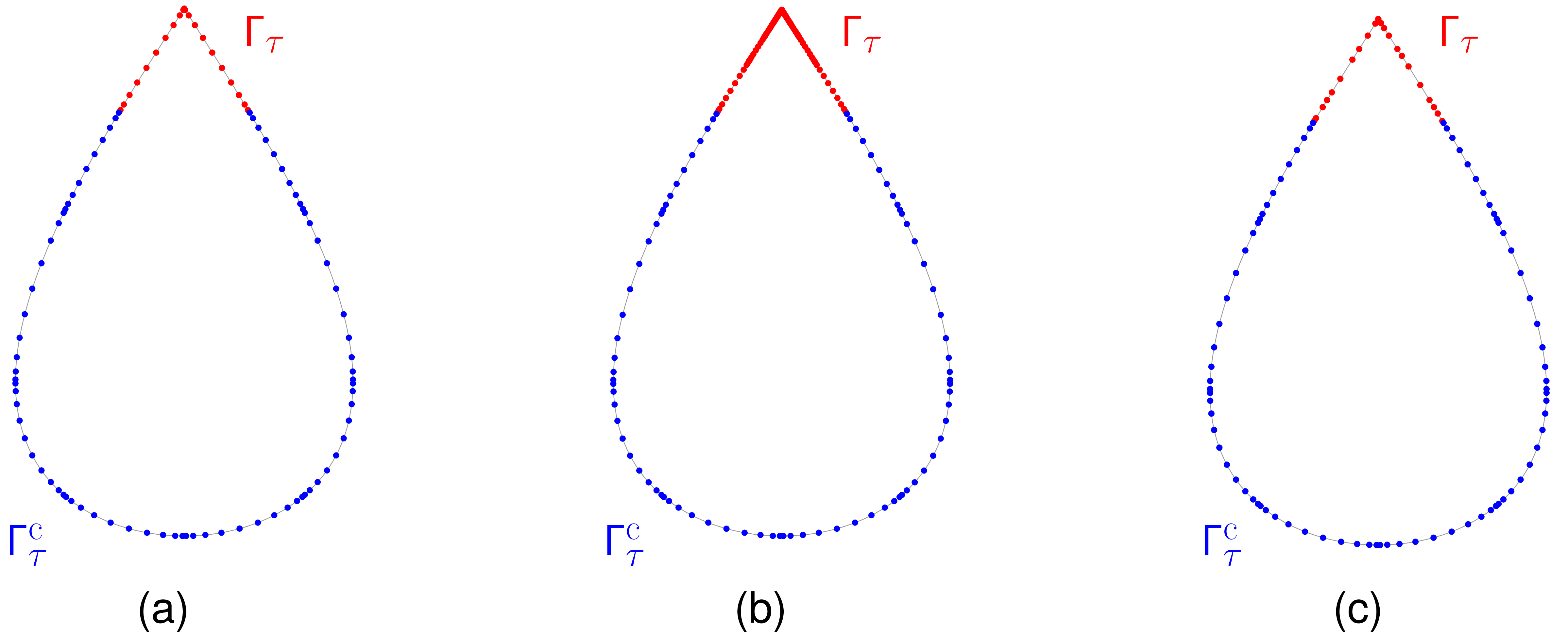
Dense matrices from Boundary Integral Equations formulations:

- Local solution operators are akin to scattering matrices.
- Typically engage interior points.
- High order quadratures can lead to thicker layers at the interfaces.
- Effective compression algorithms are essential, e.g., “Nyström trick”.

In all cases, the local solution operators compactly encode all the information about the sub-domain that is needed to solve the global problem.

Example of efficient encoding of internal information: Corner refinement

Consider a domain Γ discretized with Nyström based on a panel-based method using 10-point Gaussian quadrature. Suppose further that Γ has a corner, causing singularities in the layer potentials.



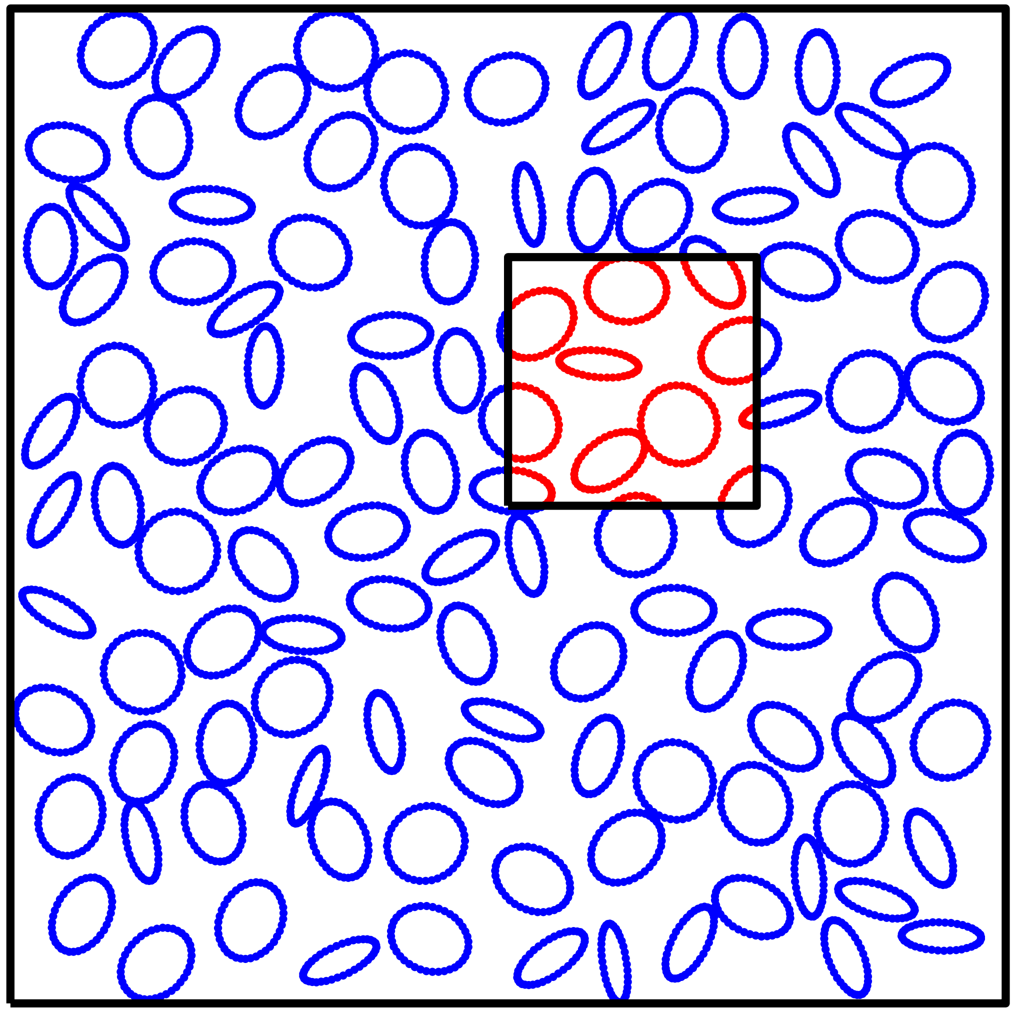
(a) Ignoring the singularity at the corner gives a small system but poor accuracy.

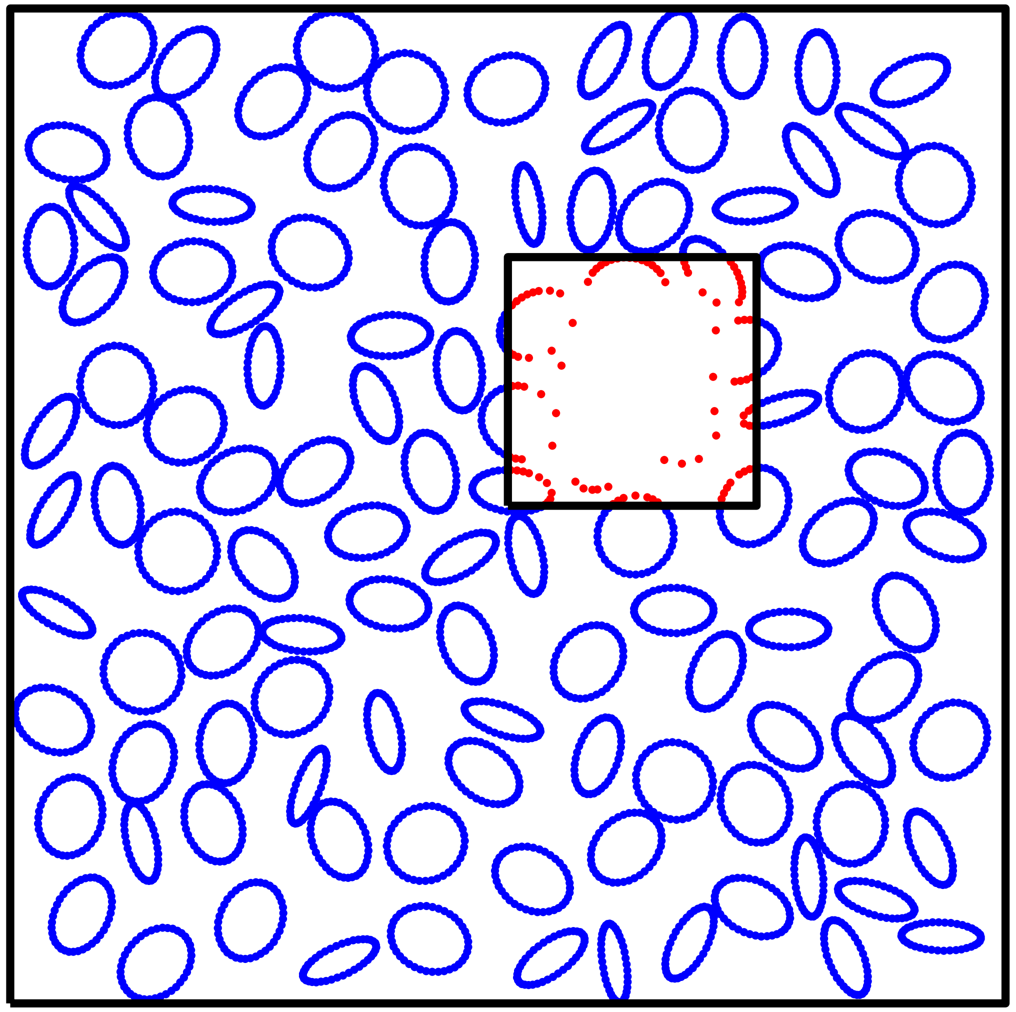
(b) Refining the grid near the corner gives great accuracy, but N gets very large.

(c) After local "compression" in which we compute a compressed scattering matrix for Γ_τ, we get the same accuracy as the discretization in (b), using as many nodes as in (a)!

Example of efficient encoding of internal information: Composite materials

Consider a two-phase material with some complicated "micro-structure":


I_1 in red, I_2 in blue.

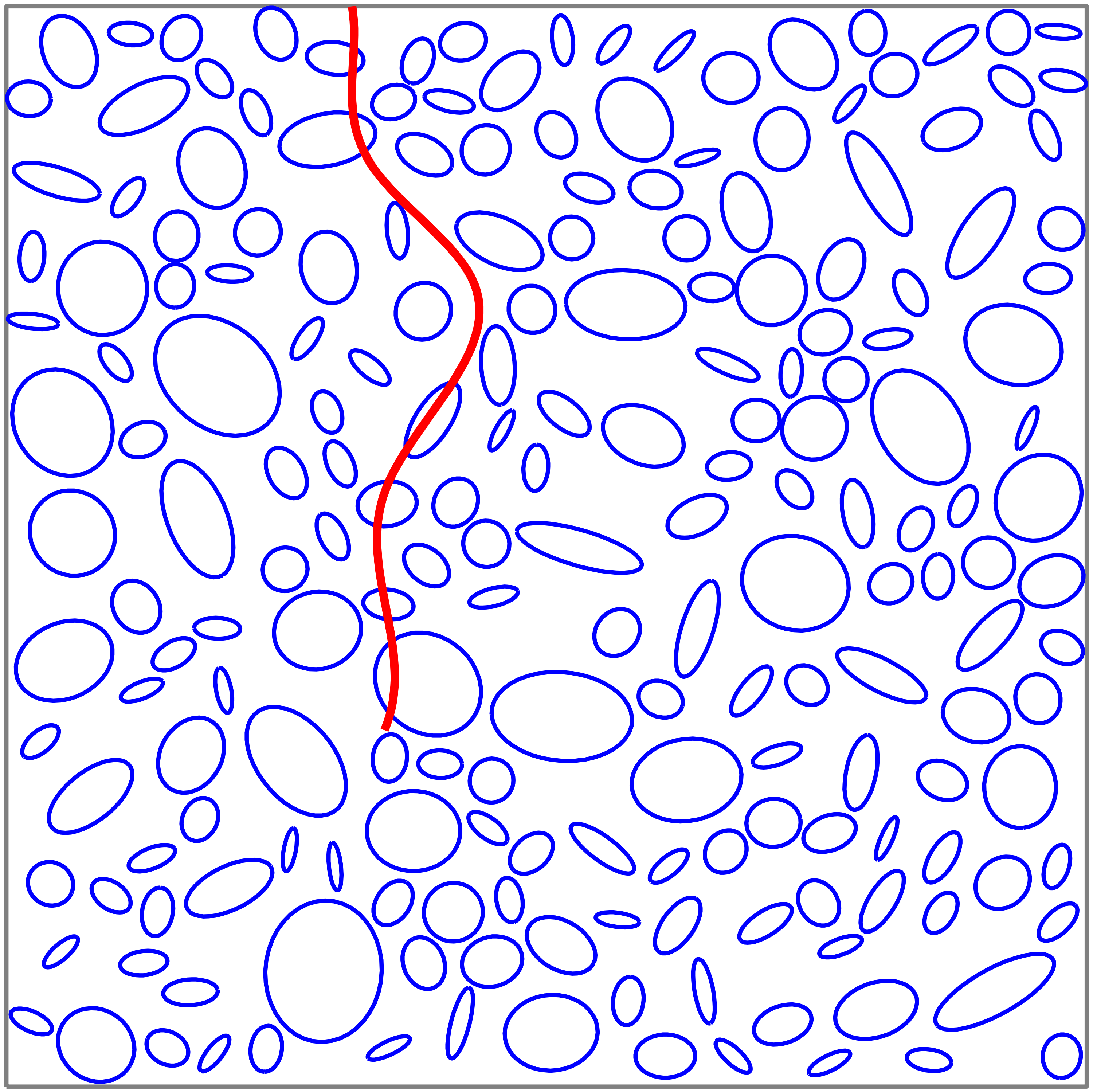

$\tilde{I}_1$ in red, I_2 in blue.

The local solution operator encodes all the internal geometry.

Excellent tool for model reduction!

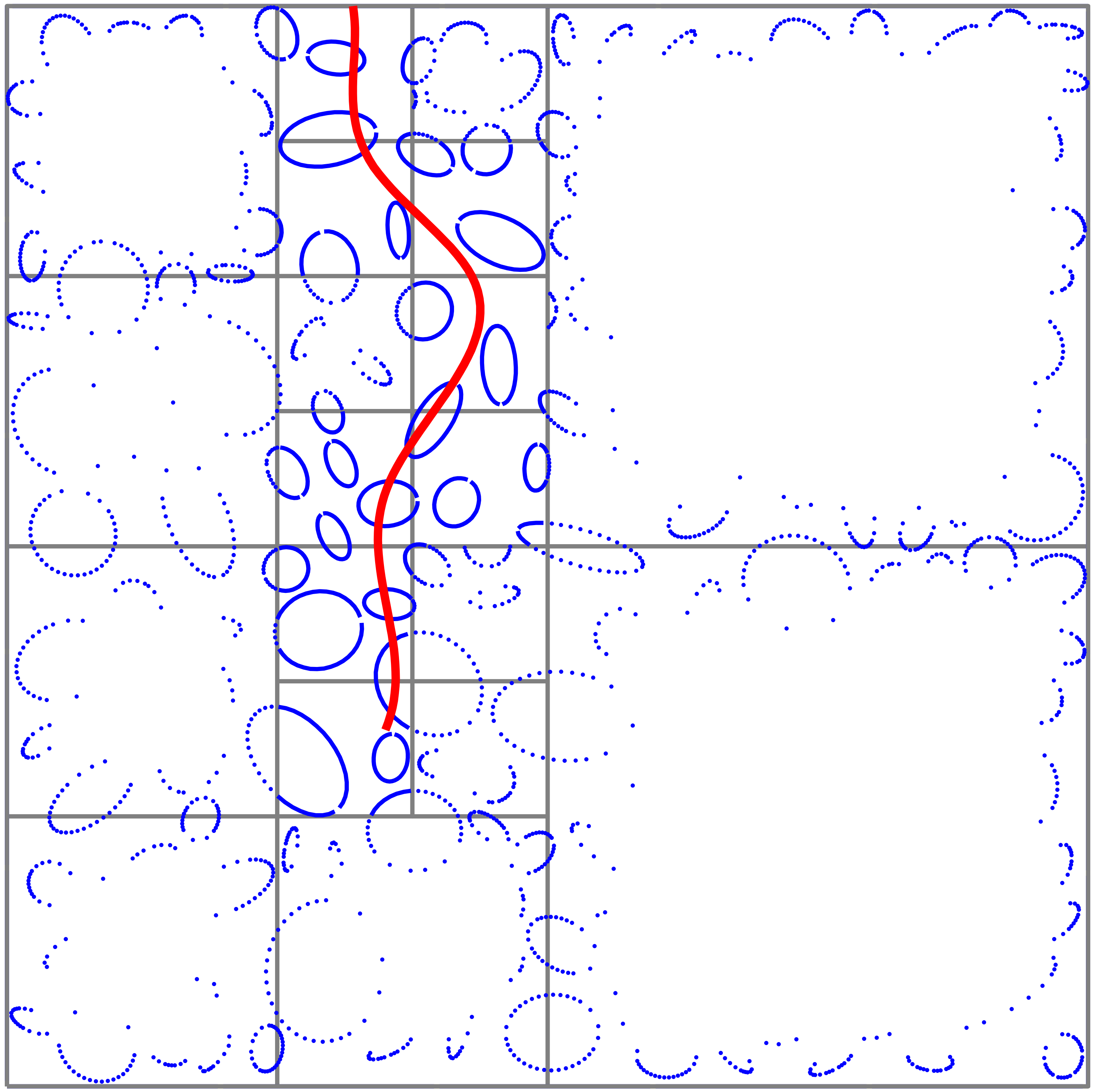
Example of efficient encoding of internal information: Composite materials

Consider modeling of a crack propagating through a composite material.

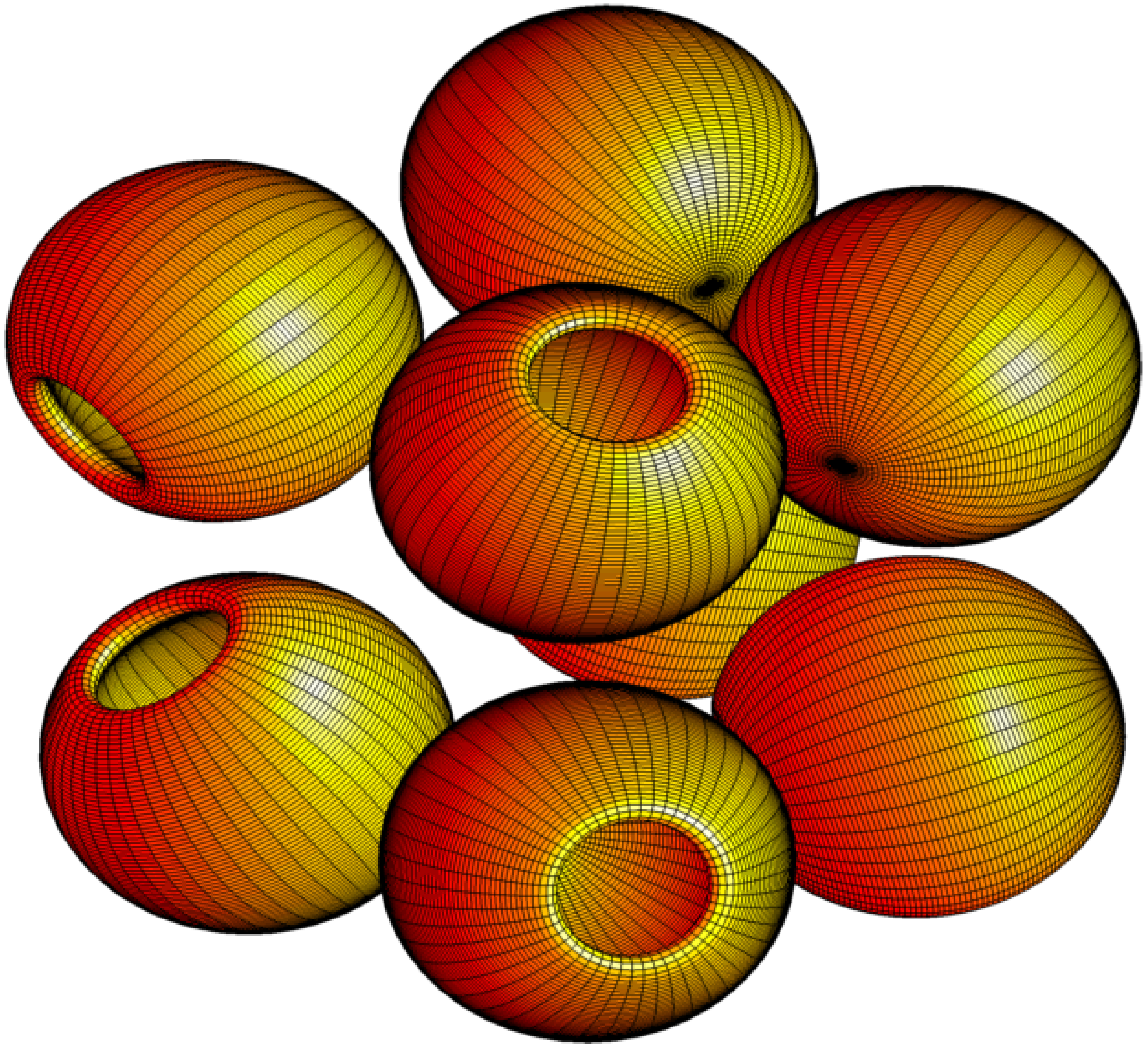


Example of efficient encoding of internal information: Composite materials

For any box not touched by the crack, we use the local solution operator!

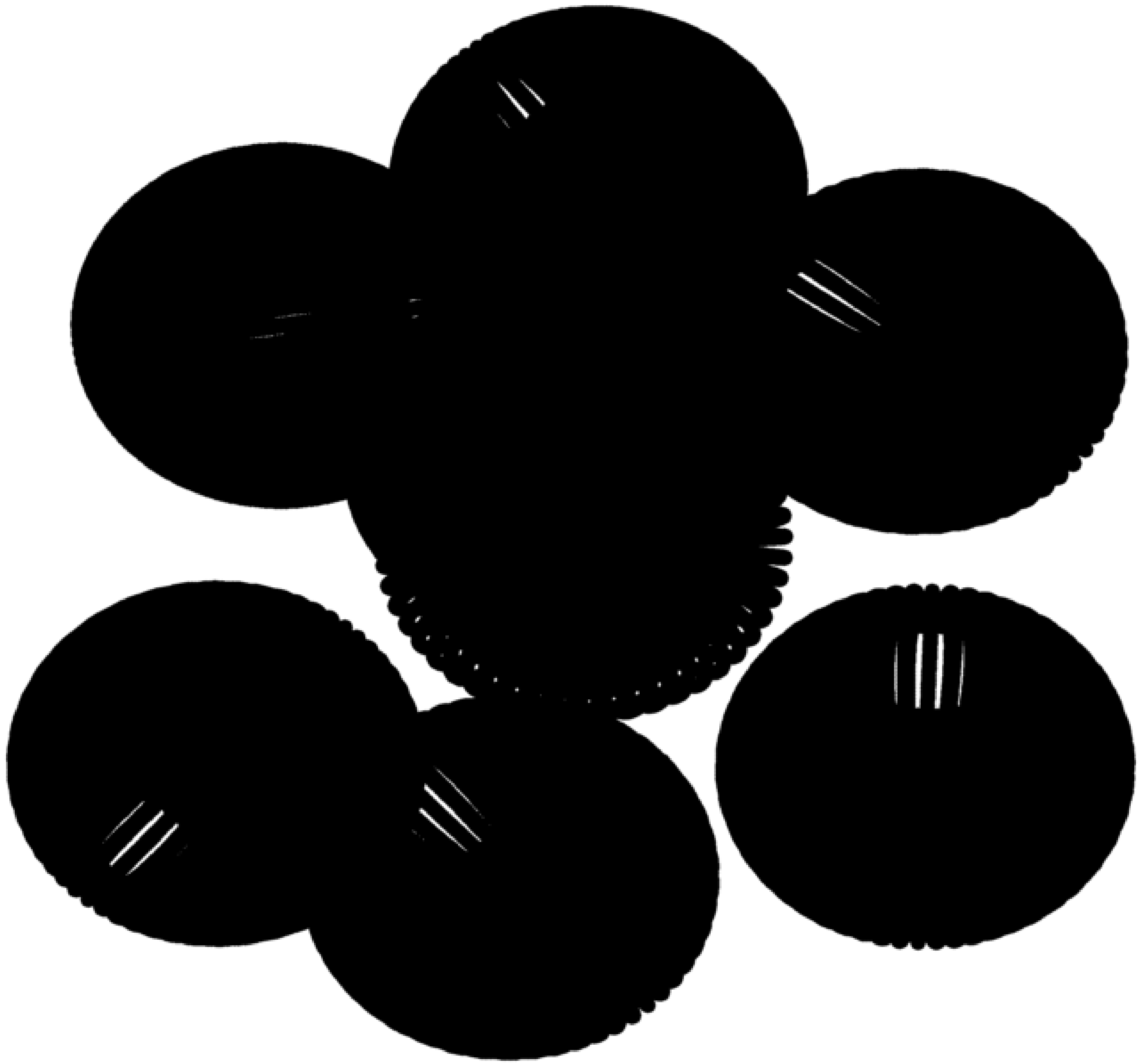


Example of efficient encoding of internal information: Multibody scattering



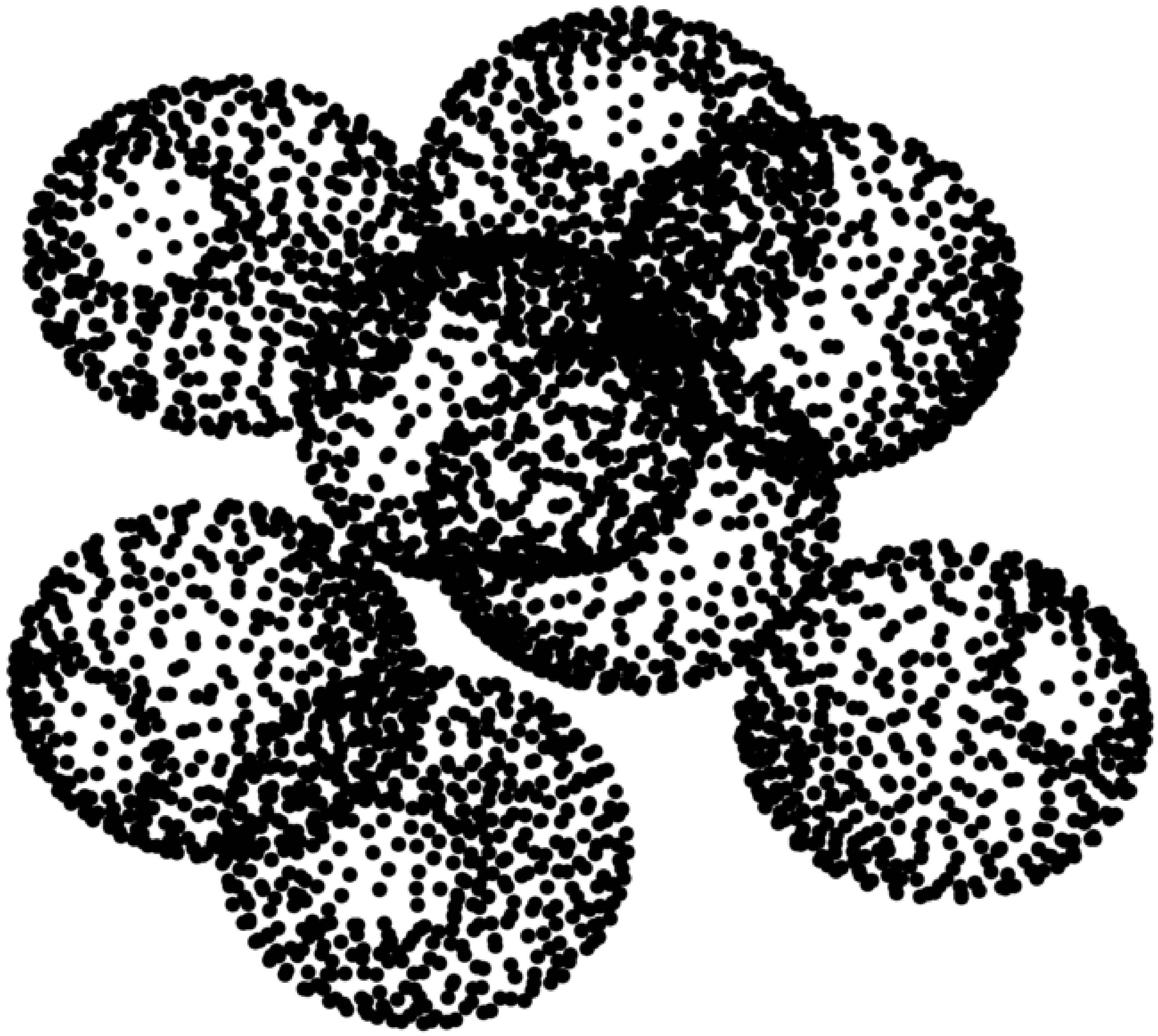
Consider scattering from some complicated multibody domain.

Example of efficient encoding of internal information: Multibody scattering



There are lots of discretization nodes involved. Very computationally intense!

Example of efficient encoding of internal information: Multibody scattering



After local compression of each scatter, the problem is much more tractable.

Hybrid **schemes are possible**

In direct solvers, processing the top levels, associated with the largest subdomains, is usually the most expensive part. Consider a problem involving N discretization points that are "volume filling". Then:

$$\text{Number of points surviving at the top (2D):}\ \sim N^{1/2}$$
$$\text{Number of points surviving at the top (3D):}\ \sim N^{2/3}$$

Using dense matrix algebra is expensive.
Using structured matrix algebra requires substantial coding work.

Observation: You do not *need* to complete the upwards pass all the way to the root.

Once the number of points surviving in a domain starts to get cumbersome, you can switch to an iterative solver.

The use of *interpolative decompositions* makes it very easy to use, e.g., the FMM to accelerate application of the compressed coefficient matrix.

Excellent tool for:
- Compressing corners.
- Multibody scattering.
- Two-scale problems (compress the fine scale).
- etc.

Using randomized sampling to accelerate matrix computations

Most fast algorithms based on rank-structured matrices spend a high percentage of the run time on computing low-rank approximations (partial SVDs, partial IDs, partial LU decompositions, etc).

These computations can be accelerated using methods based on randomized sampling.

- These methods interact with the matrix only via matrix-matrix multiplication,

$$\mathbf{R} \mapsto \mathbf{A}\mathbf{R}$$

$$\mathbf{R} \mapsto \mathbf{A}^*\mathbf{R}$$

$\rightarrow$ having *indirect* access to the matrix is fine!

- Complexity for computing a rank-k approximation to an $m \times n$ matrix $\mathbf{A}$ can be reduced from $O(mnk)$ to $O(mn\log(k))$.

- Particularly accurate for matrices whose singular values decay rapidly

$\rightarrow$ precisely our situation!

- Excellent for computing *interpolative decompositions.*

Using randomized sampling to accelerate *structured* matrix computations

Randomized sampling can also be used to find efficient representations of a *rank-structured matrix* $\mathbf{A}$ (its $\mathcal{S}$-matrix / $\mathcal{H}$-matrix / HBS-matrix / . . . representation).

Again, these methods interact with the matrix only via matrix-matrix multiplications:

$$\mathbf{R} \mapsto \mathbf{AR}$$

$$\mathbf{R} \mapsto \mathbf{A}^*\mathbf{R}$$

These methods can be used to *greatly simplify the coding*:

- Suppose $\mathbf{B}$ and $\mathbf{C}$ are given rank-structured matrices and you seek $\mathbf{A} = \mathbf{BC}$...
- Suppose you seek a rank-structured representation of a matrix $\mathbf{A} = \mathbf{B} + \mathbf{UC}^{-1}\mathbf{V}^*$ where $\mathbf{C}$ is small ...
- Suppose that you have a legacy iterative (rapidly convergent!) solver for some part of the problem...

- P.G. Martinsson, “A fast randomized algorithm for computing a Hierarchically Semi-Separable representation of a matrix”. *SIMAX*, **32**(4), pp. 1251–1274, 2011. (Preprint 2008.)
- “Fast construction of hierarchical matrix representation from matrix-vector multiplication”, L. Lin, J. Lu, L. Ying., *J. of Computational Physics*, **230**(10), 2011.
- “Randomized Sparse Direct Solvers”, Jianlin Xia, SIMAX, **34**(1), 2013.
- ... current work ...

Error analysis

Many classical methods for numerically solving PDEs are built on local polynomial approximations. With mesh size "h" and local polynomial order "p", the error is $O(h^p)$ or some such.

In contrast, many of the methods described in these lectures are not necessarily "convergent". Instead, they are "merely" highly accurate. You specify a fixed tolerance, and then expansions, quadratures, etc, are custom built for this tolerance.

The theoretical error analysis for direct solvers is currently unsatisfactory.
This makes *error estimation* critically important.

Luckily, validation becomes quite simple once you can compute solutions to 10 digits accuracy (or better). You can, for instance, differentiate computed solutions and directly compute residuals.

Fast direct solvers (like most “fast” methods) scale poorly with dimension

Extremely efficient on 1D domains (integral transforms, BIEs in the plane, frontal matrices for 2D problems).

Efficient on 2D domains (Lippman-Schwinger in 2D, BIEs on surfaces, frontal matrices for 3D problems).

Not great for 3D domains (Lippman-Schwinger in 3D).

Observation: When using direct solvers, *dimension reduction techniques* become extremely appealing:

- Rewrite constant-coefficient problems as BIEs whenever possible!
- Use multi-frontal / nested dissection / HPS scheme for variable coefficient problems!

Hardware trends favor direct solvers

The principal drawback of direct solvers is that they require a lot of memory.
But, the memory does not necessarily need to be all that fast.

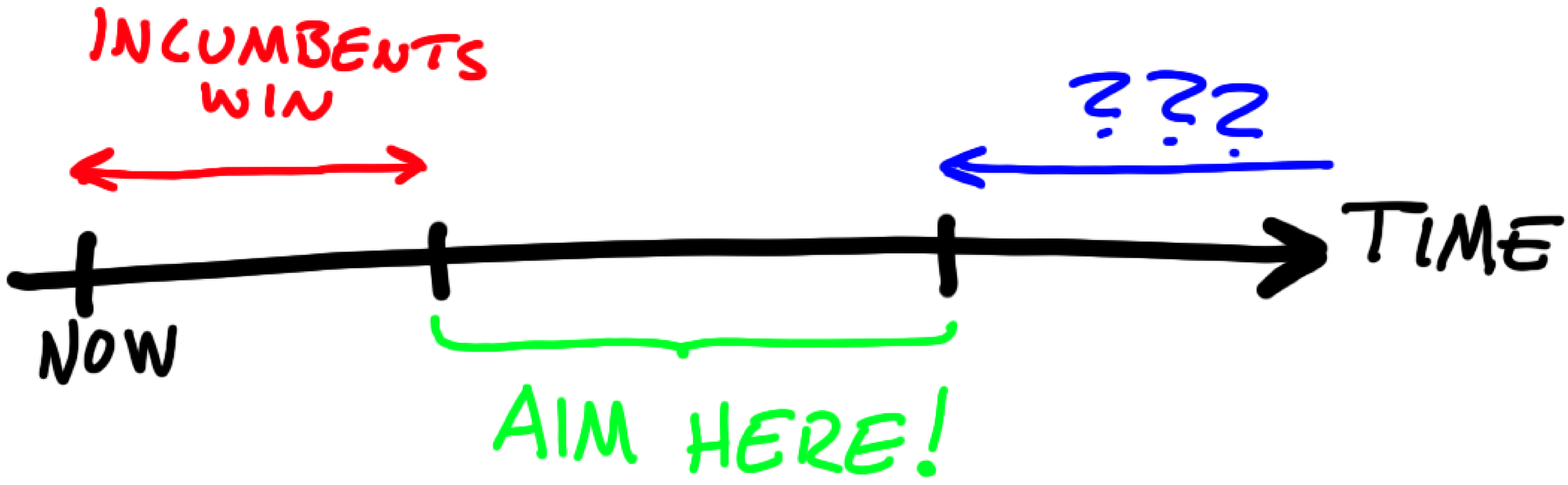
A principal advantage of direct solvers is that they are communication effective
$\rightarrow$ they parallelize well!

Observe that hardware trends work very much in favor of these methods:

- Multicore, parallel, and distributed computing is getting cheaper and cheaper.
- Communication is emerging as the key bottleneck.
- Storage keeps getting cheaper, especially “slow” storage such as flash memory.

Possibilities to pre-compute vast dictionaries and use table look-up.

Advice: In designing a research program concerning numerical methods for solving PDEs, aim for computers of the (reasonable) future.



Observe that the costs of developing code for handling 3D geometry is *high.*