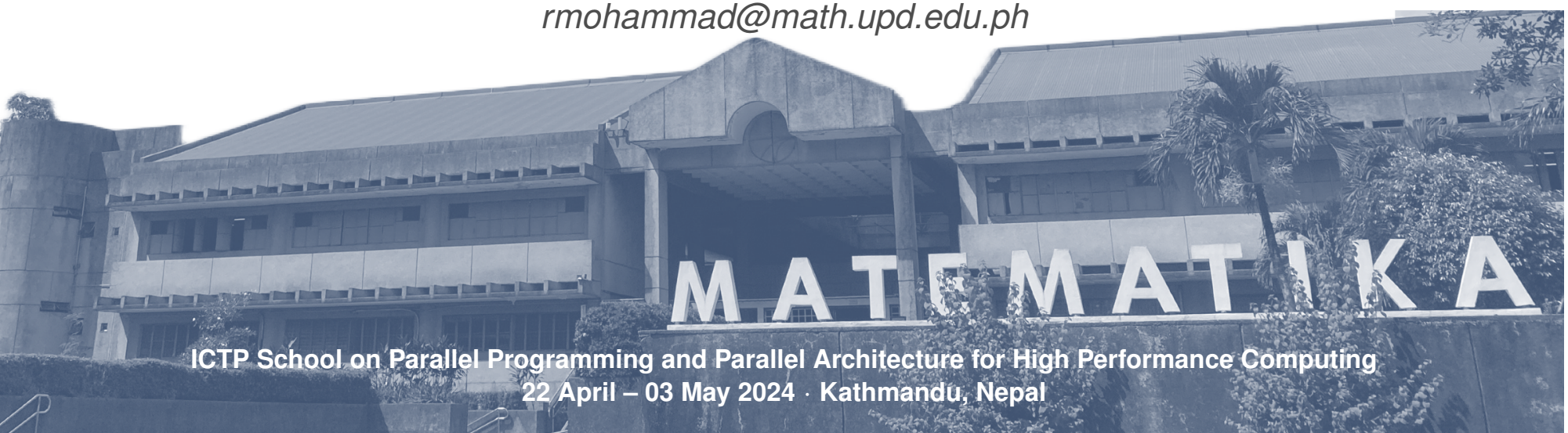
Parallelization of Jacobi Algorithm for Solving Laplace equation

...and my attempt at parallelization of my cell project

Rhudaina Mohammad
Numerical Analysis and Scientific Computing Group
Institute of Mathematics, University of the Philippines Diliman
rmohammad@math.upd.edu.ph



ICTP School on Parallel Programming and Parallel Architecture for High Performance Computing
22 April – 03 May 2024 · Kathmandu, Nepal

Cell Project: My First Attempt at Parallelization

Motivation for ICTP School

communications biology

ARTICLE

https://doi.org/10.1038/s42003-022-03174-6 OPEN

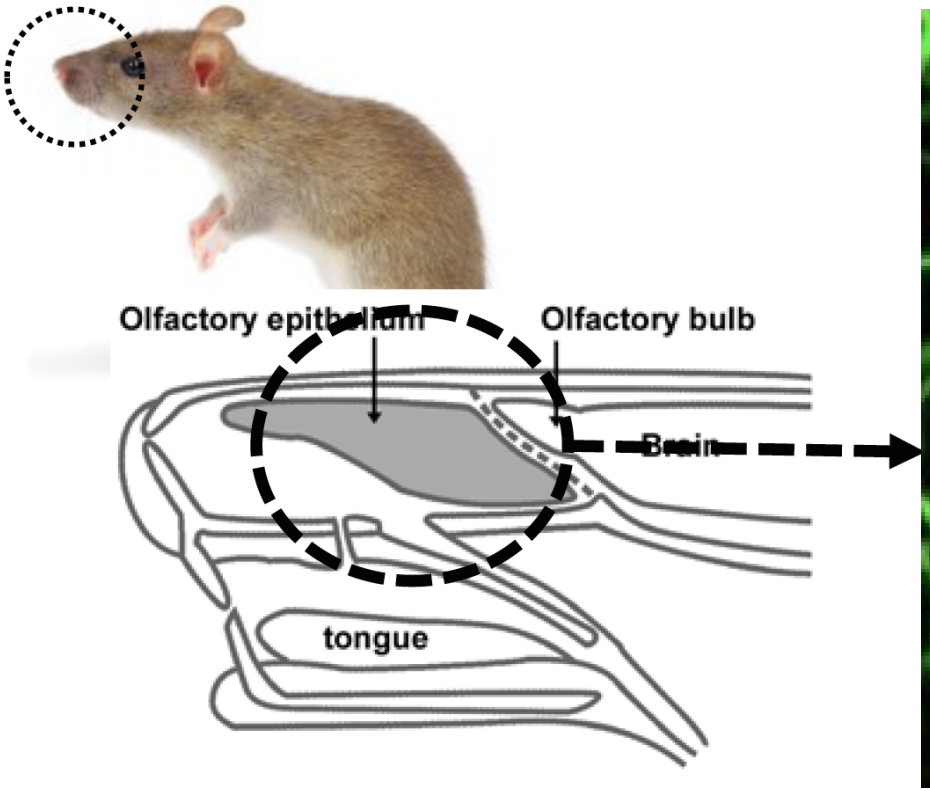
Check for updates

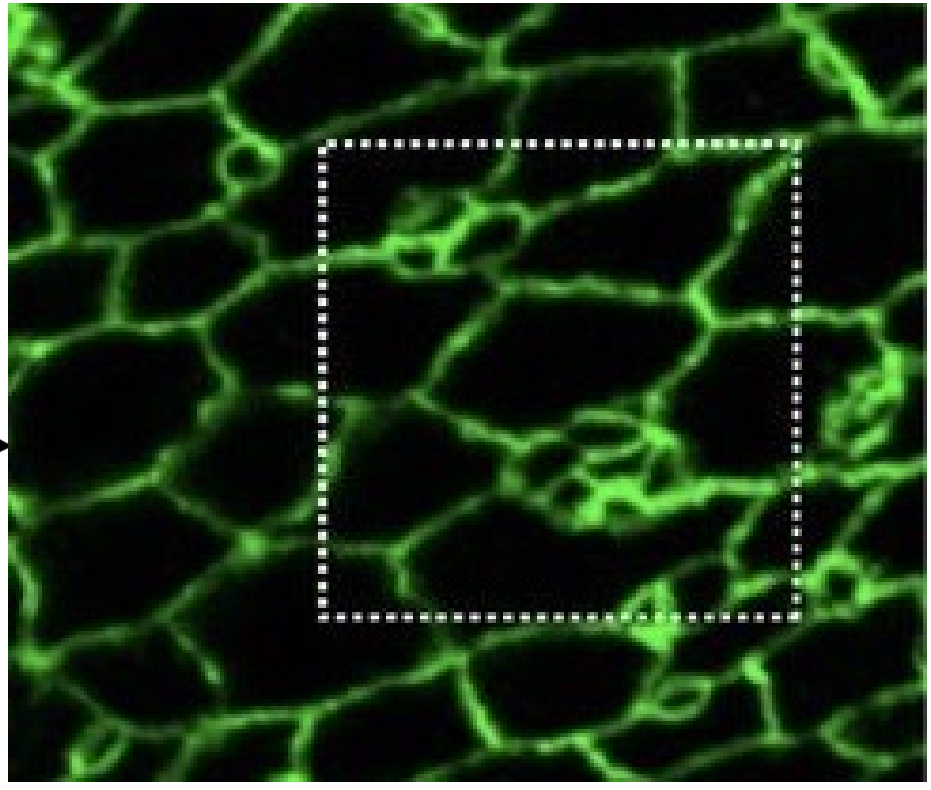
A numerical algorithm for modeling cellular rearrangements in tissue morphogenesis

Rhudaina Z. Mohammad [1,2], Hideki Murakawa [3], Karel Svadlenka [1,4✉] & Hideru Togashi [5,6]

- computational framework based on a level set formulation for tissue morphogenesis
- investigated cellular patterns observed in embryonic auditory and olfactory epithelium

Cellular Patterns in Sensory Epithelium

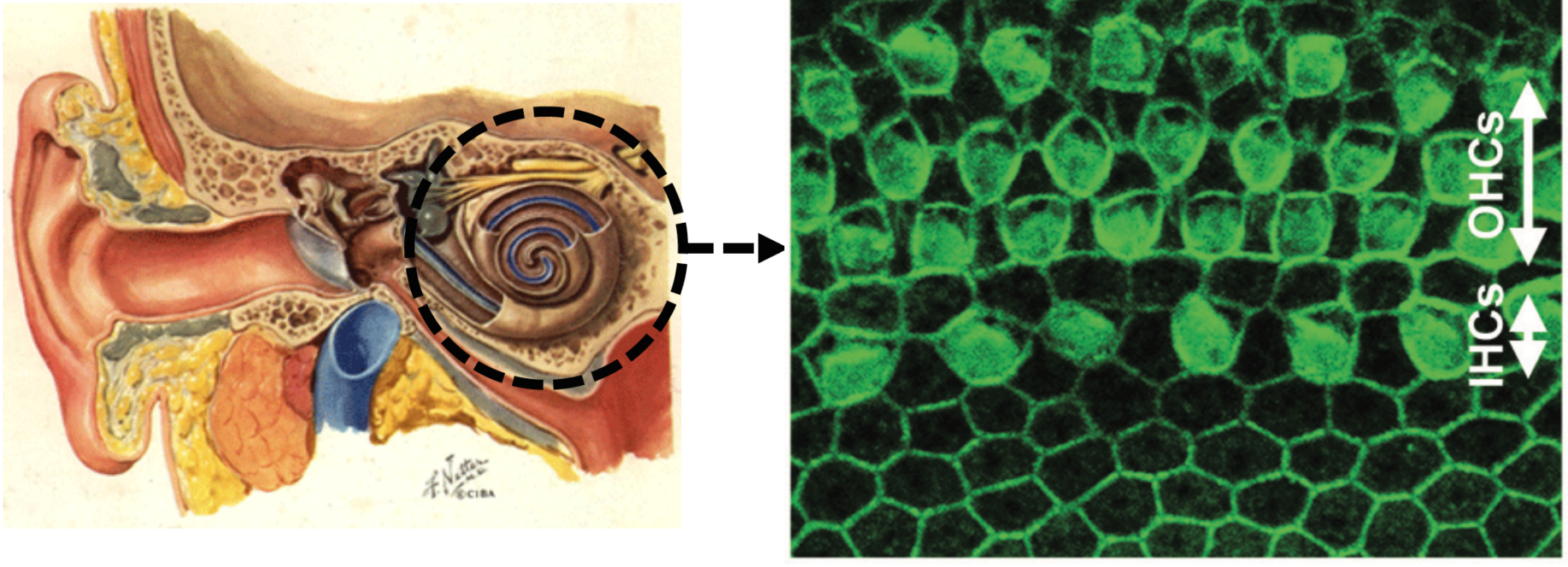




used with permission from H. Togashi (Kobe University)

Cellular patterning in the olfactory epithelium (OE), located inside the nasal cavity of a wild-type mouse at embryonic day 14

Cellular Patterns in Sensory Epithelium



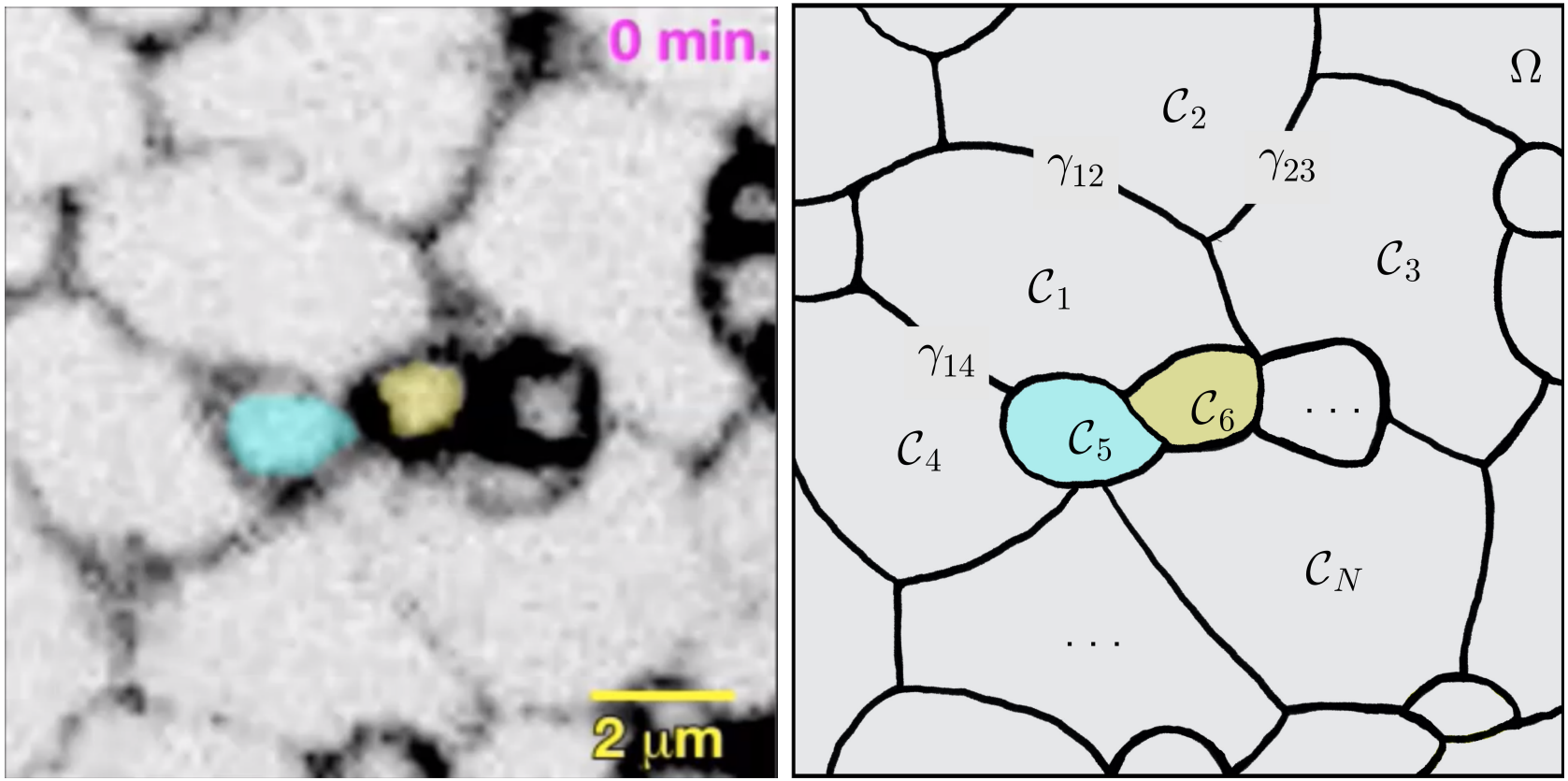
used with permission from H. Togashi (Kobe University)

Cellular patterning in the auditory epithelium (AE), located in the cochlea of a wild-type mouse at postnatal day 1

Cellular Patterns in Sensory Epithelium

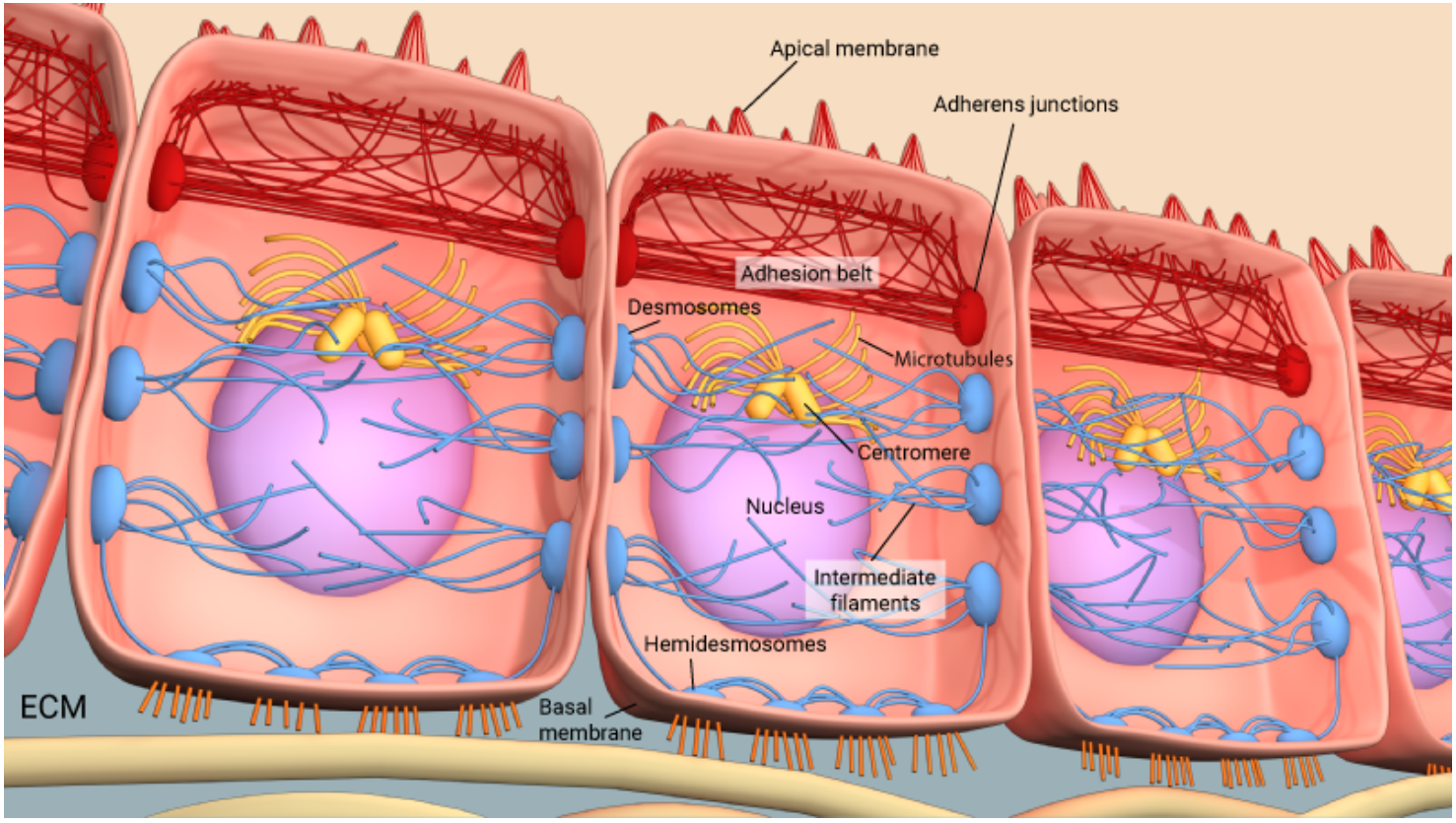
To analyze the effect of these factors in tissue morphogenesis, we assume

- tissue evolves via a succession of quasi-equilibrium states, i.e, cell shapes are described by their instantaneous state of lowest energy
- no apoptosis occurs during the evolution



Microscopy image of mouse OE culture prepared at E14 [Katsunuma et al., 2016] and its schematic diagram. OCs are indicated by blue and yellow.

Cellular Patterns in Sensory Epithelium



Retrieved from https://www.mechanobio.info/epithelial-cells-junctions-2/

Epithelial cell-cell adhesions are mediated via adherens junctions (AJs), where nectins and cadherins cooperate and bind the intracellular proteins p120-catenin and β-catenin

Cellular Patterns in Sensory Epithelium

Consider an aggregate of cells as a bounded domain

$$\Omega = \mathcal{C}_1 \cup \cdots \cup \mathcal{C}_N \subset \mathbb{R}^d$$

partitioned into N cells where cell-cell junction

$$\gamma_{ij} := \mathcal{C}_i \cap \mathcal{C}_j = \partial\mathcal{C}_i \cap \partial\mathcal{C}_j$$

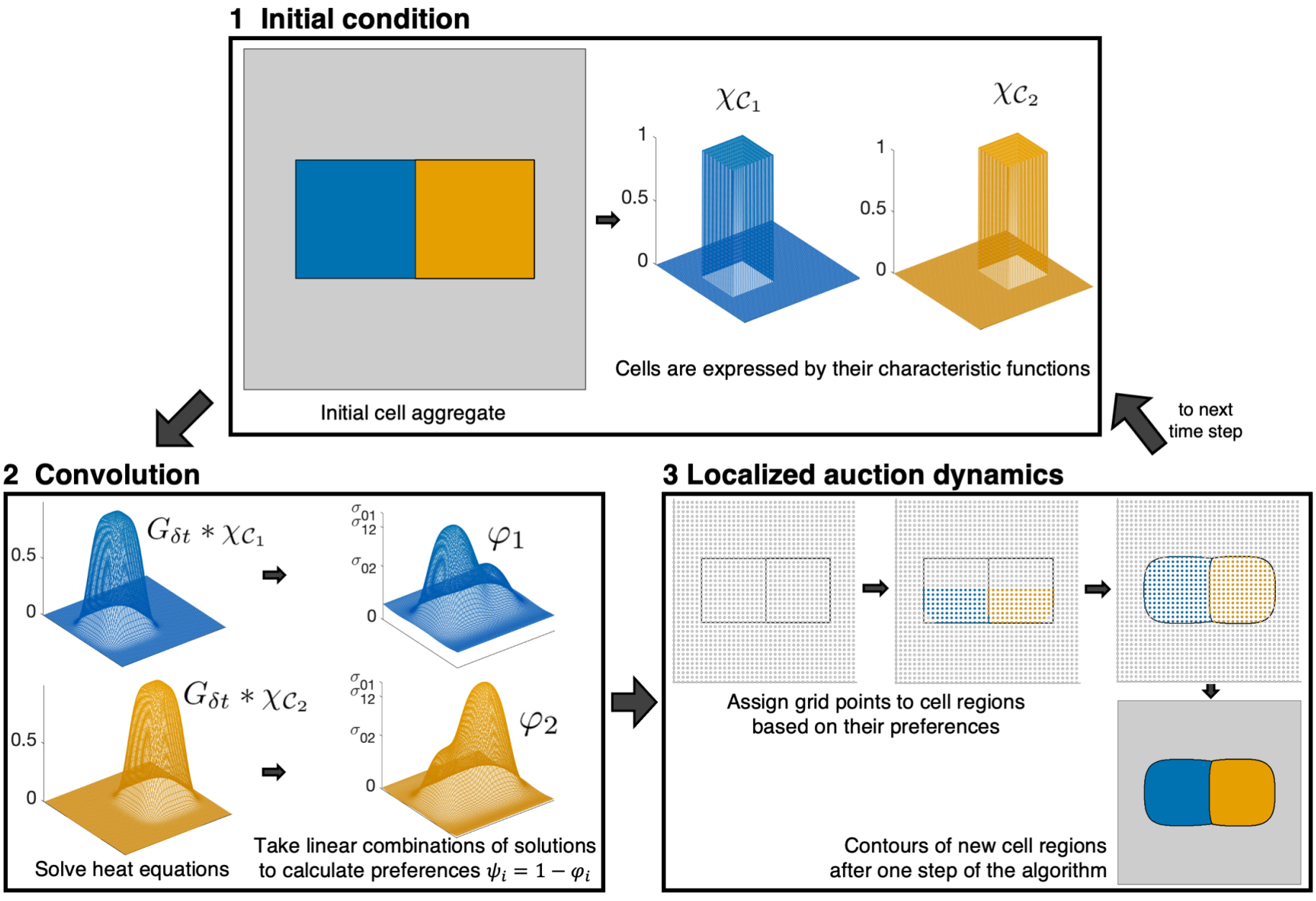
Take cellular rearrangement as L^2-gradient flow of a weighted surface energy

$$E(\mathcal{C}_1, \ldots, \mathcal{C}_N) = \sum_{i \neq j} \sigma_{ij} \int_{\gamma_{ij}} d\mathcal{H}^{d-1},$$

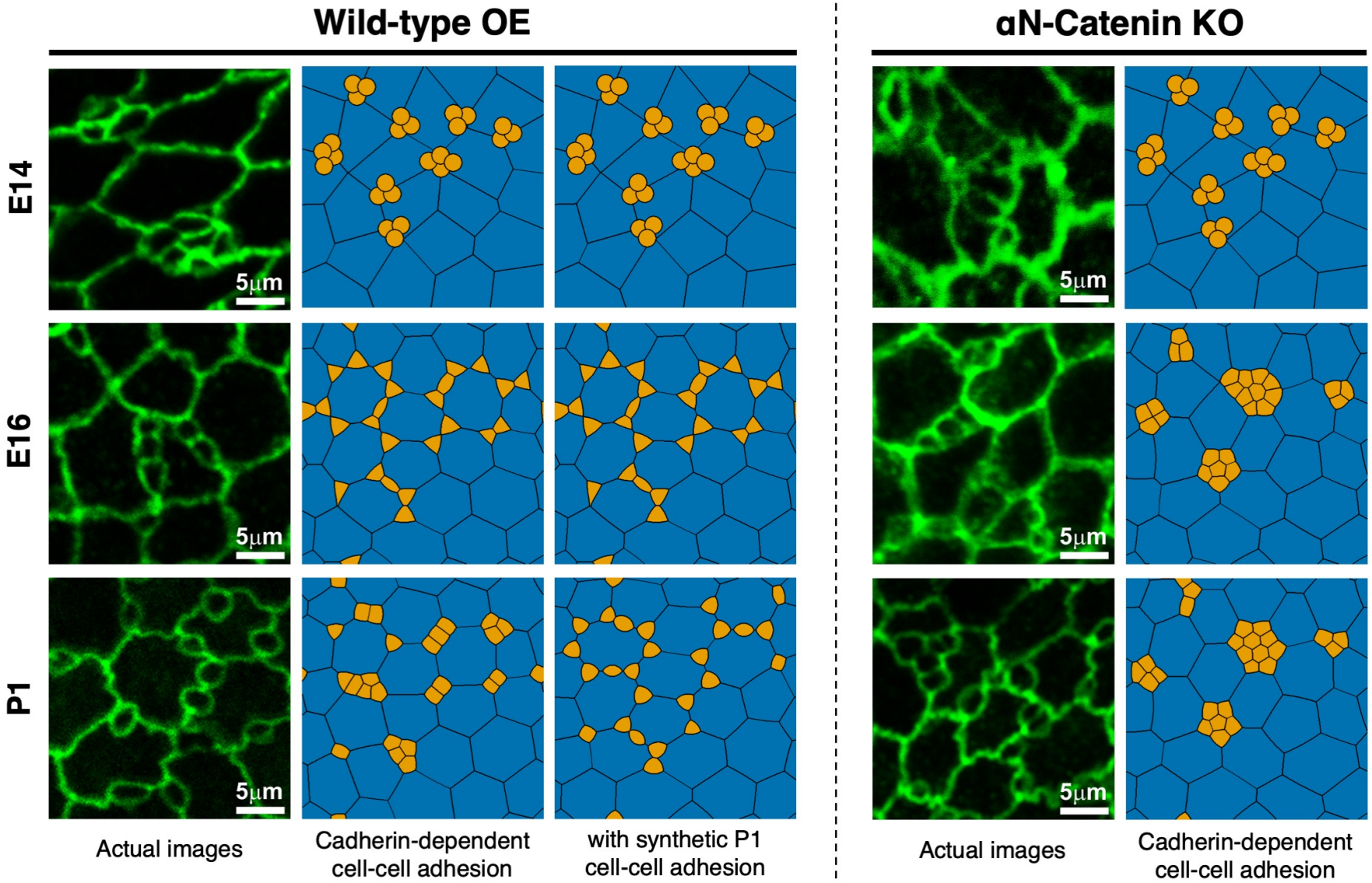
constrained by each cell's prescribed volume V_k^0 $(k = 1, 2, \ldots, N)$.

- Here, σ_{ij}'s are derived from measured values such as cell-cell adhesion (interfacial tension).

Proposed Algorithm

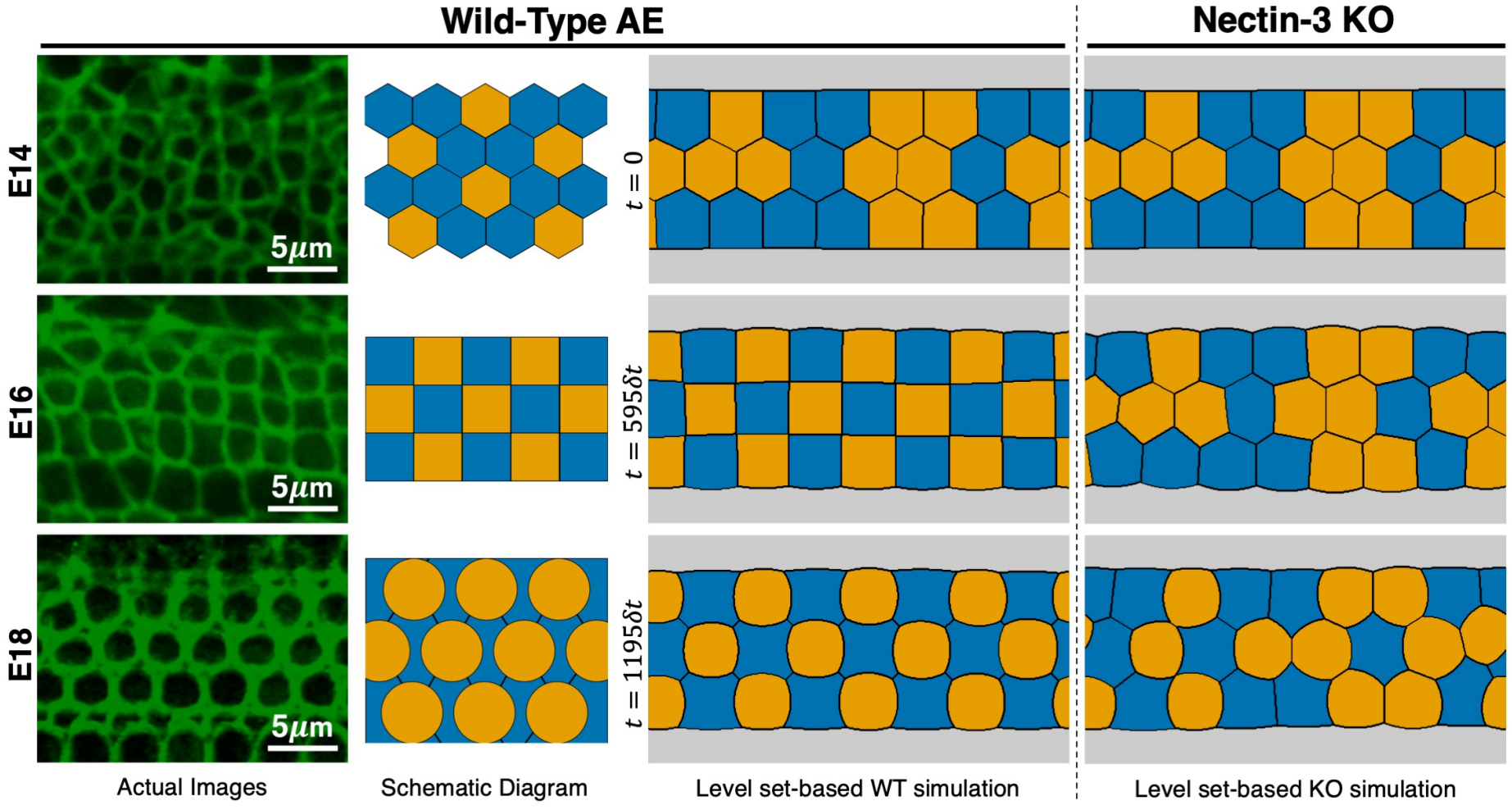


Developing OE Simulation



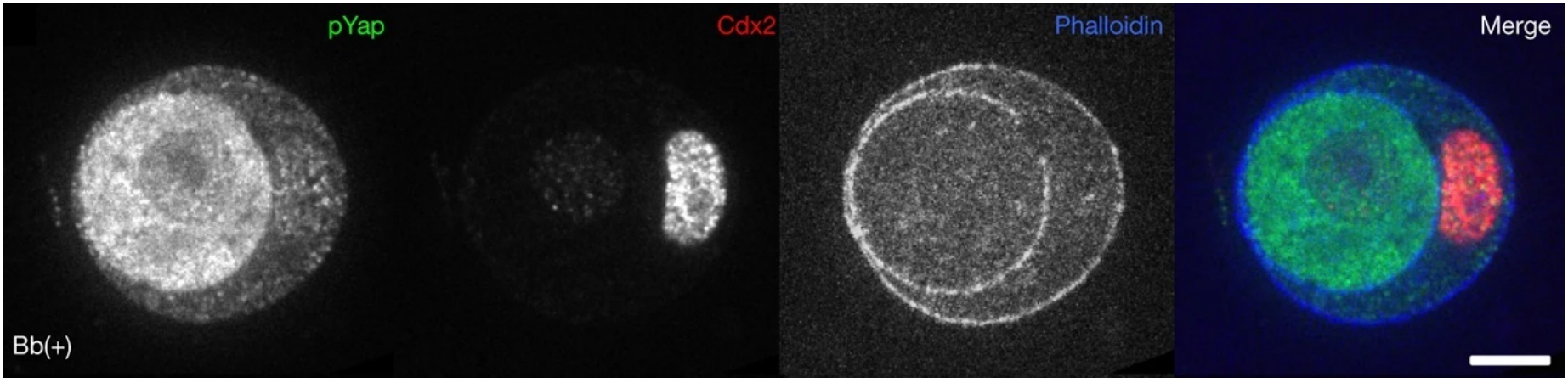
[Katsunuma et al., 2016]: during development, olfactory cells (OCs) and supporting cells (SCs) dynamically arrange themselves to form a mosaic

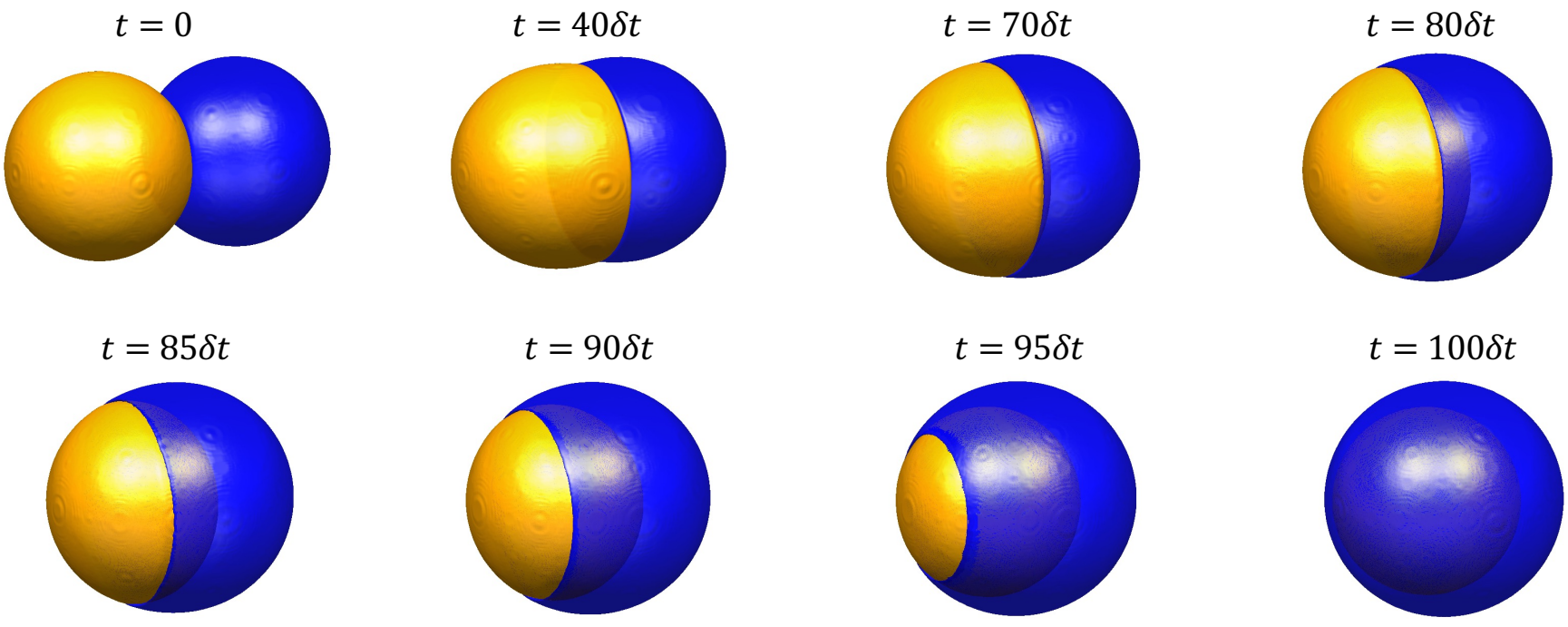
Developing AE Simulation



[Togashi 2016]: cellular rearrangement of AE from embryonic day 14 to 18

Embryo Morphogenesis

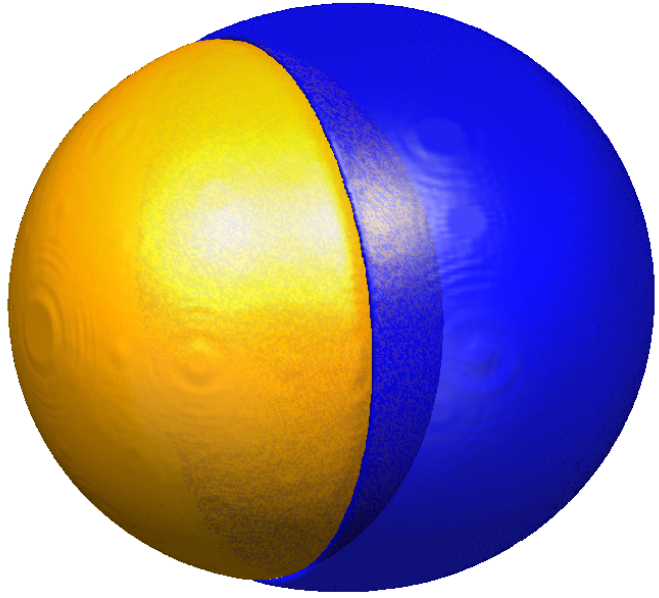




[Maitre et al. 2016]: embryo morphogenesis reveals that cells internalize only when differences in surface contractility exceed a predictable threshold

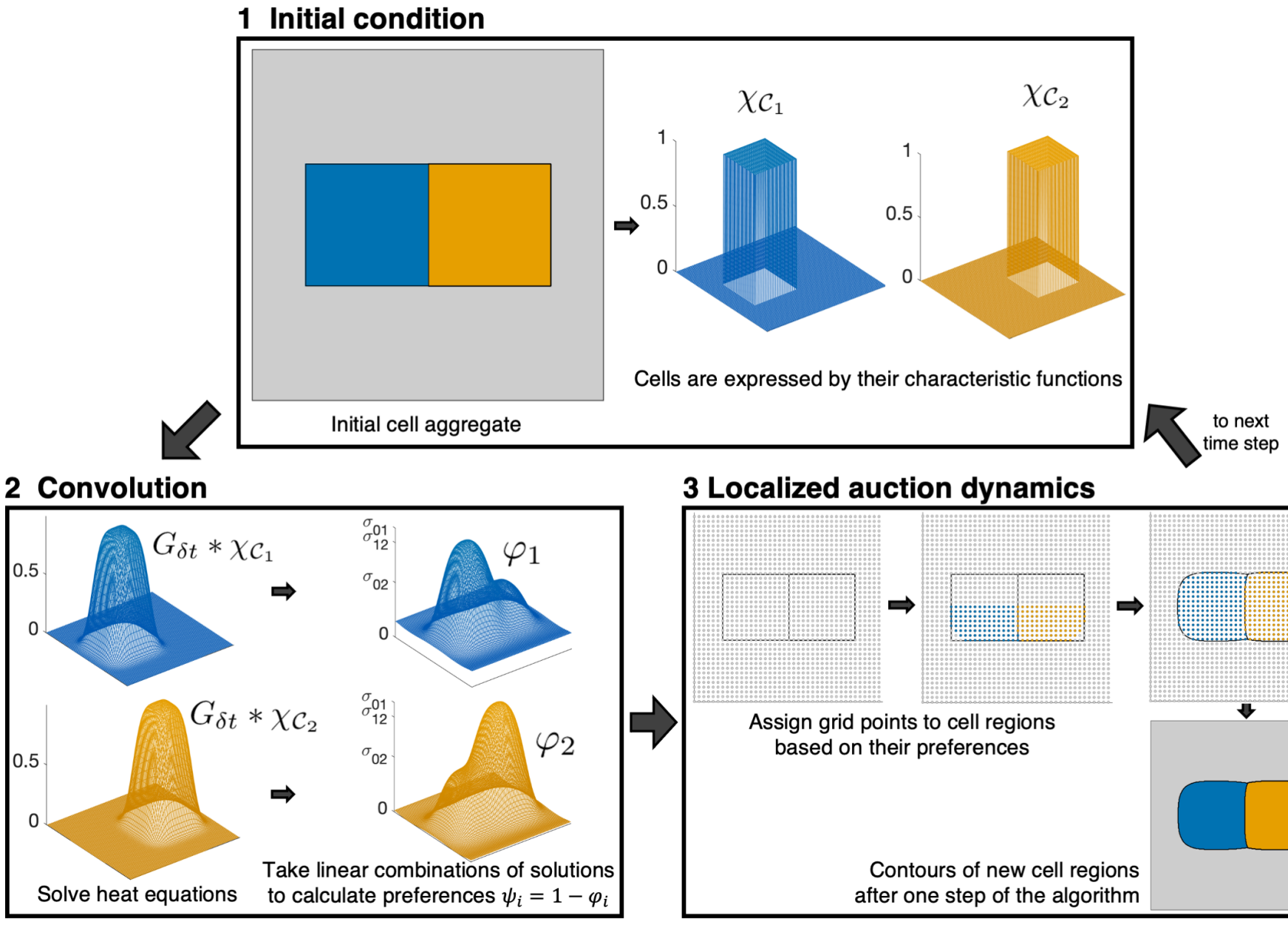
Embryo Morphogenesis

Level Set-based Scheme: **t = 0.400000**



$\sigma_{BB} = 1.00$ $\sigma_{BO} = 0.50$ $\sigma_{OO} = 1.00$ $\sigma_{BW} = 1.00$ $\sigma_{OW} = 1.36$ $\sigma_{WW} = 1.00$

Attempt at Computational Acceleration

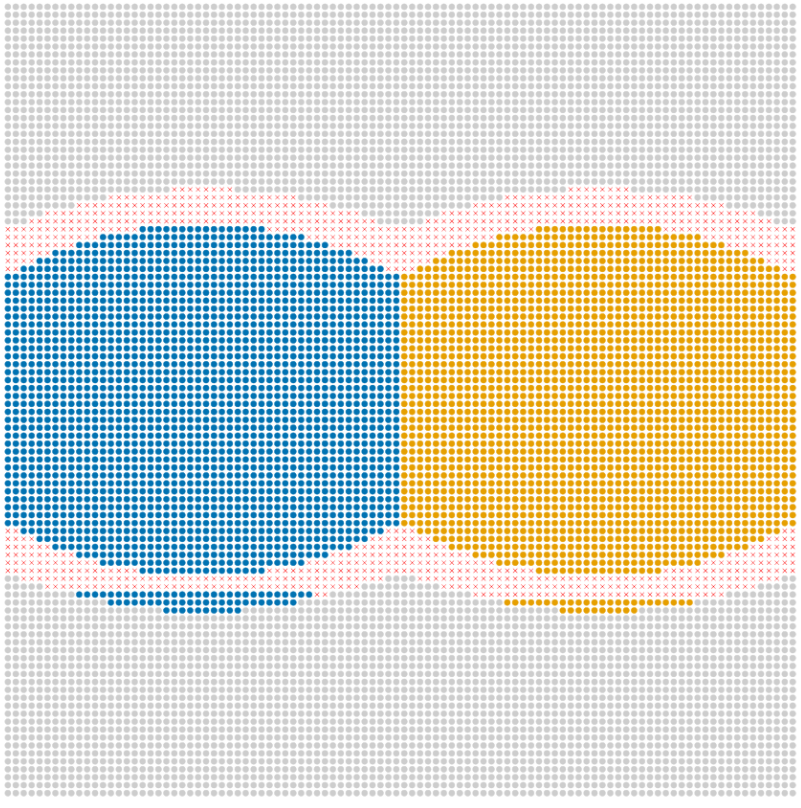


Attempt at Computational Acceleration

```
    // Define FFTW plans
    int nthreads = 1;
#if defined(_OPENMP)
    nthreads = omp_get_max_threads();
    printf("Running FFTW %d OpenMP threads...\n", nthreads);
#endif
    int normCoef = nbCells*nbNodes;
    int nbCmplxNodes = ((int)(nbNodesByCol/2)+1)*nbNodesByRow;
    if (dim==3) nbCmplxNodes *= nbNodesByRow;
    int *nbPerDim = (int *) malloc((dim+1)*sizeof(int));
    for (int i = 1; i < dim+1; i++) nbPerDim[i] = nbNodesByRow;
    nbPerDim[0] = nbCells;
    if (dim==2) nbPerDim[2] = nbNodesByCol;

    double freqStepX = (nbNodesByCol-1.)/nbNodesByCol;
    double freqStepY = (nbNodesByCol-1.)/nbNodesByRow;

#if defined(_OPENMP)
    int status = fftw_init_threads();
    if (status==0){
      printf("FFTW error in thread initialization.\n");
    }
    fftw_plan_with_nthreads(nthreads);
#endif
    fftw_complex *ft_phi = fftw_malloc(nbCells*nbCmplxNodes*sizeof(fftw_complex));
    fftw_plan forward;
    if (dim==2) forward = fftw_plan_dft_r2c_3d(nbCells, nbNodesByRow, nbNodesByCol, phi, ft_phi, FFTW_MEASURE);
    if (dim==3) forward = fftw_plan_dft_r2c(4, nbPerDim, phi, ft_phi, FFTW_MEASURE);
    fftw_plan backward;
    if (dim==2) backward = fftw_plan_dft_c2r_3d(nbCells, nbNodesByRow, nbNodesByCol, ft_phi, phi, FFTW_MEASURE);
    if (dim==3) backward = fftw_plan_dft_c2r(4, nbPerDim, ft_phi, phi, FFTW_MEASURE);

    // Export initial configuration to output
    export_solution(0, nbNodes, isAssigned);
    display_data(dim, nbNodes, nbCells, nbTypes, dt, typeTension);
```

Auction Dynamics

Auction Dynamics



assigned nodes: 09098/10000

node: 2539

preferred cell : C_1

bid on C_1: 0.02032

*assign to C_1

Cell volume

C_0(gray) : 5000/5000

C_1(blue) : 2064/2500

C_2(orange) : 2034/2500

Cell price

p_0: 0.14838

p_1: 0.00000

p_2: 0.00000

Jacobi Algorithm for Solving Laplace Equation

Jacobi Algorithm

Consider the Laplace equation:

$$\begin{cases} \Delta u = 0, & \text{in } [a,b]^2 \\ u = 0, & \text{on } \{x=b\} \cup \{y=b\} \\ u = -\frac{100}{b-a}(y-a) + 100, & \text{on } \{x=a\} \\ u = -\frac{100}{b-a}(x-a) + 100, & \text{on } \{y=a\} \end{cases}$$

Jacobi Algorithm

- Discretize the domain into $(n+2)^2$ points (x_i, y_j) where

$$a = x_0 < x_1 < \cdots < x_n < x_{n+1} = b$$

$$a = y_0 < y_1 < \cdots < y_n < y_{n+1} = b$$

- On a regular grid, we can approximate the Laplacian

$$\Delta u = u_{xx} + u_{yy}$$

using central difference method

$$u_{xx}(x,y) \approx \frac{u(x+h,y) - 2u(x,y) + u(x-h,y)}{h^2}$$

and

$$u_{yy}(x,y) \approx \frac{u(x,y+h) - 2u(x,y) + u(x,y-h)}{h^2}$$

Jacobi Algorithm

- Denoting $u_{i,j} = u(x_i, y_j)$, we get

$$\Delta u \approx \frac{u_{i+1,j} + u_{i-1,j} - 4u_{i,j} + u_{i,j+1} + u_{i,j-1}}{h^2}$$

- Since $\Delta u = 0$, we have

$$u_{i,j} \approx \frac{u_{i+1,j} + u_{i-1,j} + u_{i,j+1} + u_{i,j-1}}{4}$$

- Starting from an initial guess u^0, we generate a sequence of approximate solutions

$$u_{i,j}^{k+1} = \frac{u_{i+1,j}^k + u_{i-1,j}^k + u_{i,j+1}^k + u_{i,j-1}^k}{4}, \quad k = 0, 1, 2 \ldots$$

  such that $u^k \to u$, the solution of the Laplace equation as $k \to \infty$.

Jacobi Algorithm: 1D Decomposition

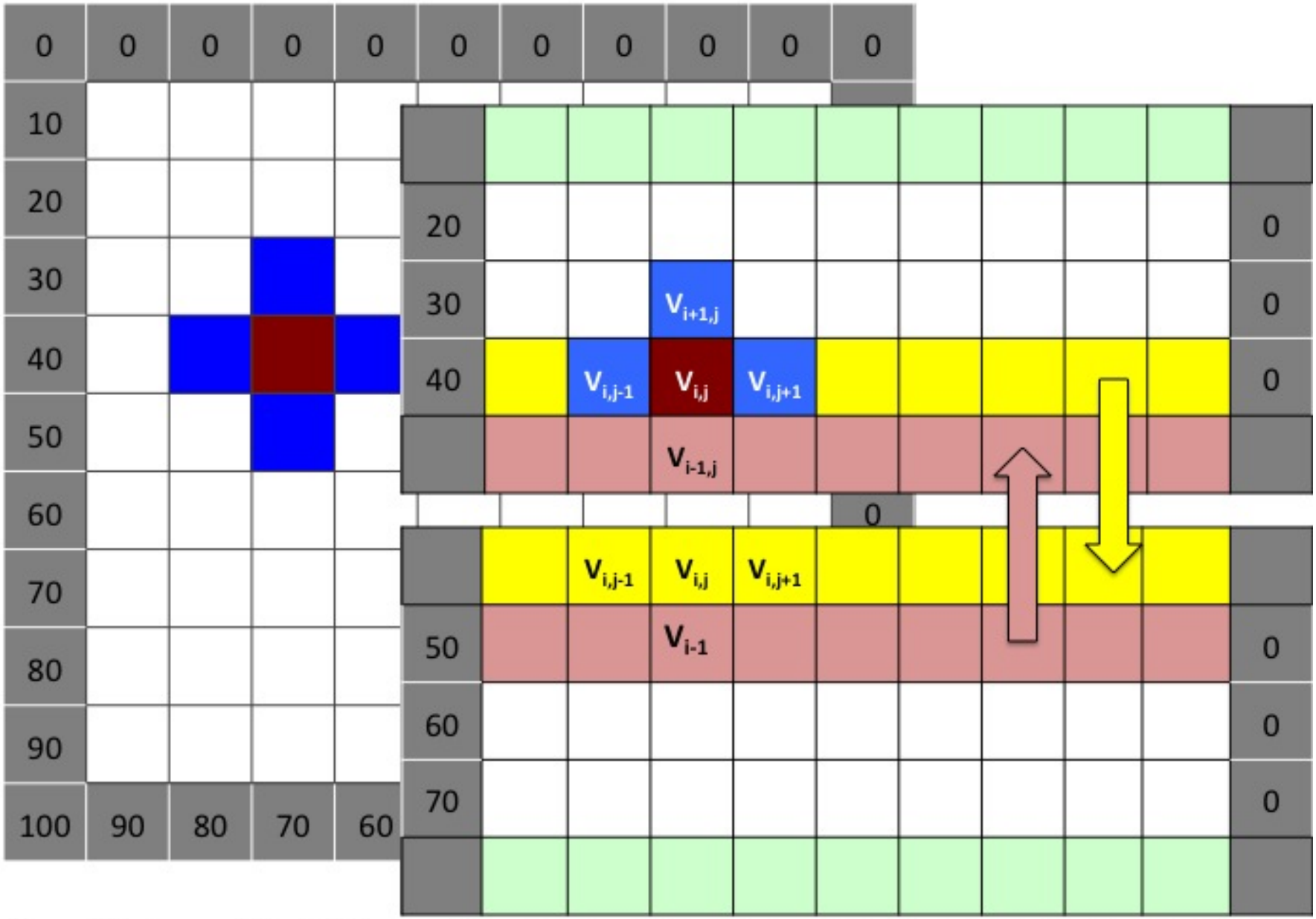


Figure 2: A diagram of the 1-D decomposition of the Jacobi Relaxation for Solving the Laplace's Equation, showing that the boundary elements that need to be communicated between processors.

Jacobi Algorithm: 1D Decomposition

```
int dim = atoi(argv[1]);
int max_iter = atoi(argv[2]);

int me, npes, n_loc, rest, offset=0;
MPI_Init( &argc, &argv );
MPI_Comm_rank( MPI_COMM_WORLD, &me );
MPI_Comm_size( MPI_COMM_WORLD, &npes );
MPI_Barrier(MPI_COMM_WORLD);  //synchronize all processes

double t_start, t_end, t_elapsed;
t_start = MPI_Wtime();

n_loc = dim/npes + 2;
rest = dim % npes;
if( me < rest ) n_loc += 1;
else offset = rest;

double* matrix = NULL;
matrix = (double *)  malloc(sizeof(double) * dim * n_loc);
if (matrix == NULL){
  printf("[ERROR] Check memory allocation for the matrix.\n");
  exit(-1);
}
memset(matrix, 0.5, sizeof(double) * dim * n_loc);
```

Jacobi Algorithm: 1D Decomposition

```
void communicate(double *matrix, int n_loc, int dim, int npes, int me){
  // Send up and send down
  if(me != 0){
    // send up the 2nd row (idx = 1) of previous processor (tag = 1)
    MPI_Send(matrix + dim, dim, MPI_DOUBLE, me-1, 1, MPI_COMM_WORLD);
  }
  if(me!= npes-1){
    // send down the second-to-the-last row (idx = n_loc-2) of next processor (tag = 2)
    MPI_Send(matrix + (n_loc-2)*dim, dim, MPI_DOUBLE, me+1, 2, MPI_COMM_WORLD);
  }

  // Receive from above and and from below
  if(me!= 0){
    // receive from above and update 1st row (tag = 2)
    MPI_Recv(matrix, dim, MPI_DOUBLE, me-1, 2, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
  }
  if(me!= npes-1){
    // receive from below and update last row (tag = 1)
    MPI_Recv(matrix + (n_loc-1)*dim, dim, MPI_DOUBLE, me+1, 1, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
  }
}
```

Jacobi Algorithm: 1D Decomposition

```
double err_chunk = 0;
for (int i = 1; i < n_loc-1; i++){
  for(int j = 0; j < dim; j++){
    // stencil west
    if (j == 0) west = (me*(n_loc-2) + i + offset) * increment;
    else        west = matrix[i*dim + j-1];
    // stencil east
    if (j == dim-1) east = 0;
    else            east = matrix[i*dim + j+1];

    new_mat[i*dim + j] = 0.25*( matrix[(i-1)*dim + j] + matrix[(i+1)*dim + j]
    + west + east );

    err_chunk += pow(new_mat[i*dim +j] - matrix[i*dim +j], 2);
  }
}
```

Jacobi Algorithm: 1D Decomposition

```
double west, east;
double increment = 100.0 / (dim+1);
for (size_t it = 0; it < max_iter; it++) {
  double err_chunk = 0;
  for (int i = 1; i < n_loc-1; i++){

  // Replace matrix values by new_mat values
  for (int i=1; i < n_loc-1; i++){
    for(int j=0; j < dim; j++){
      matrix[i*(dim) + j] = new_mat[i*(dim) + j ];
    }
  }
  communicate(matrix, n_loc, dim, npes, me);

  double error = 0;
  MPI_Allreduce(&err_chunk, &error, 1, MPI_DOUBLE, MPI_SUM, MPI_COMM_WORLD);
  if (me == 0) {
    printf("it %zu:\t %.7f\n", it, sqrt(error) );
    fprintf(file, "%.7f\n", sqrt(error) );
    if (sqrt(error) < TOL) break;
  }
}
```

Jacobi Algorithm: 1D Decomposition

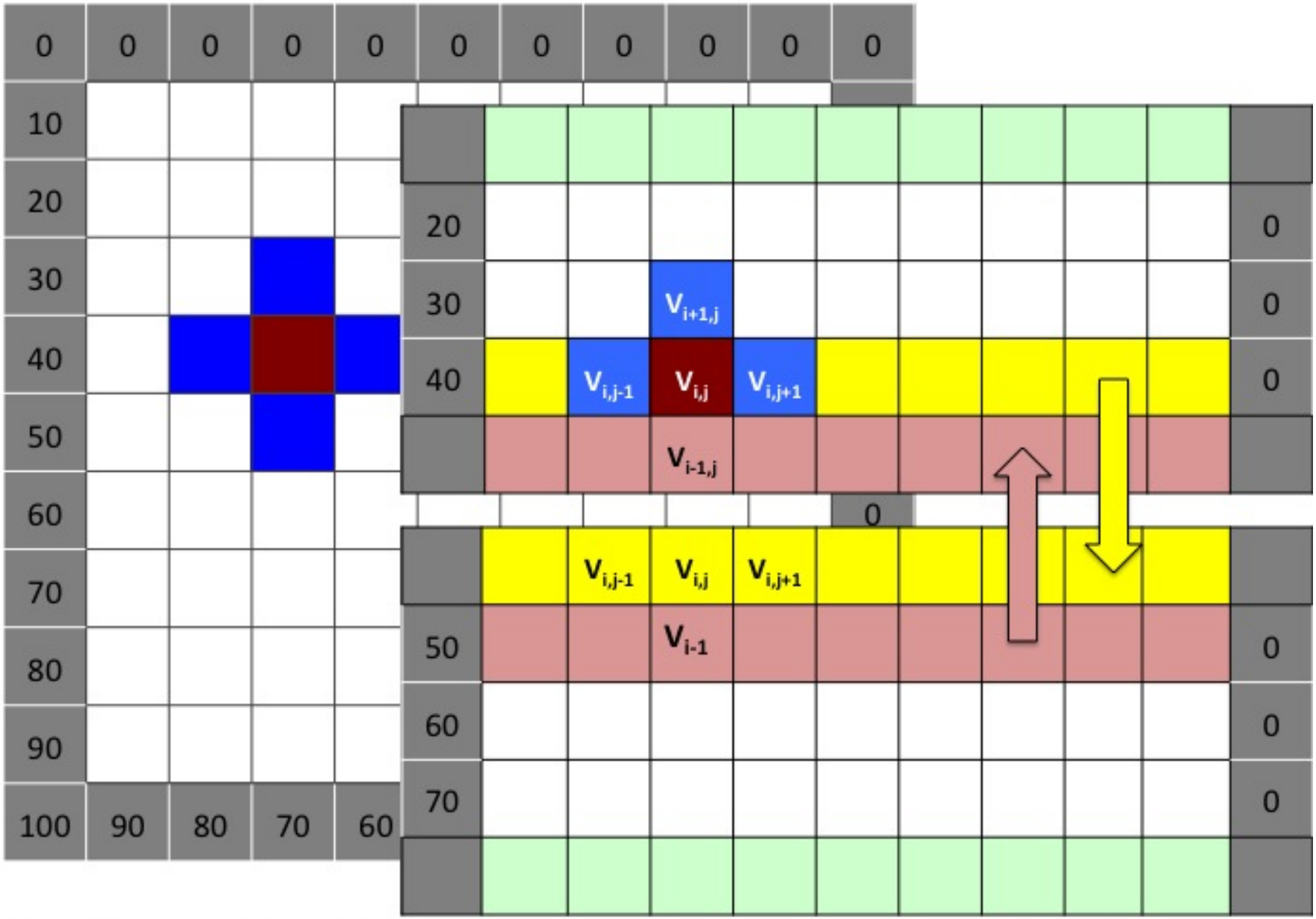
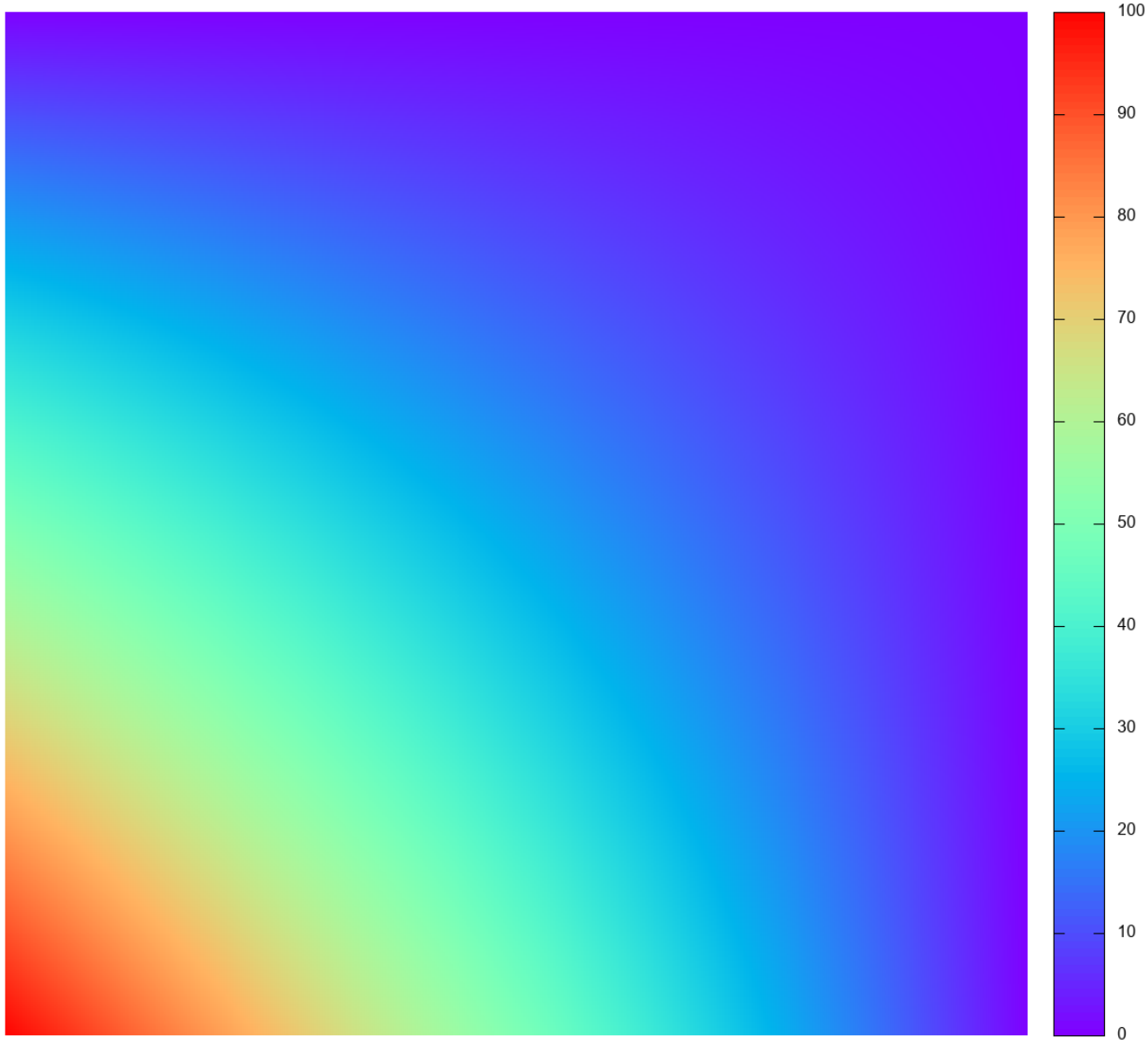
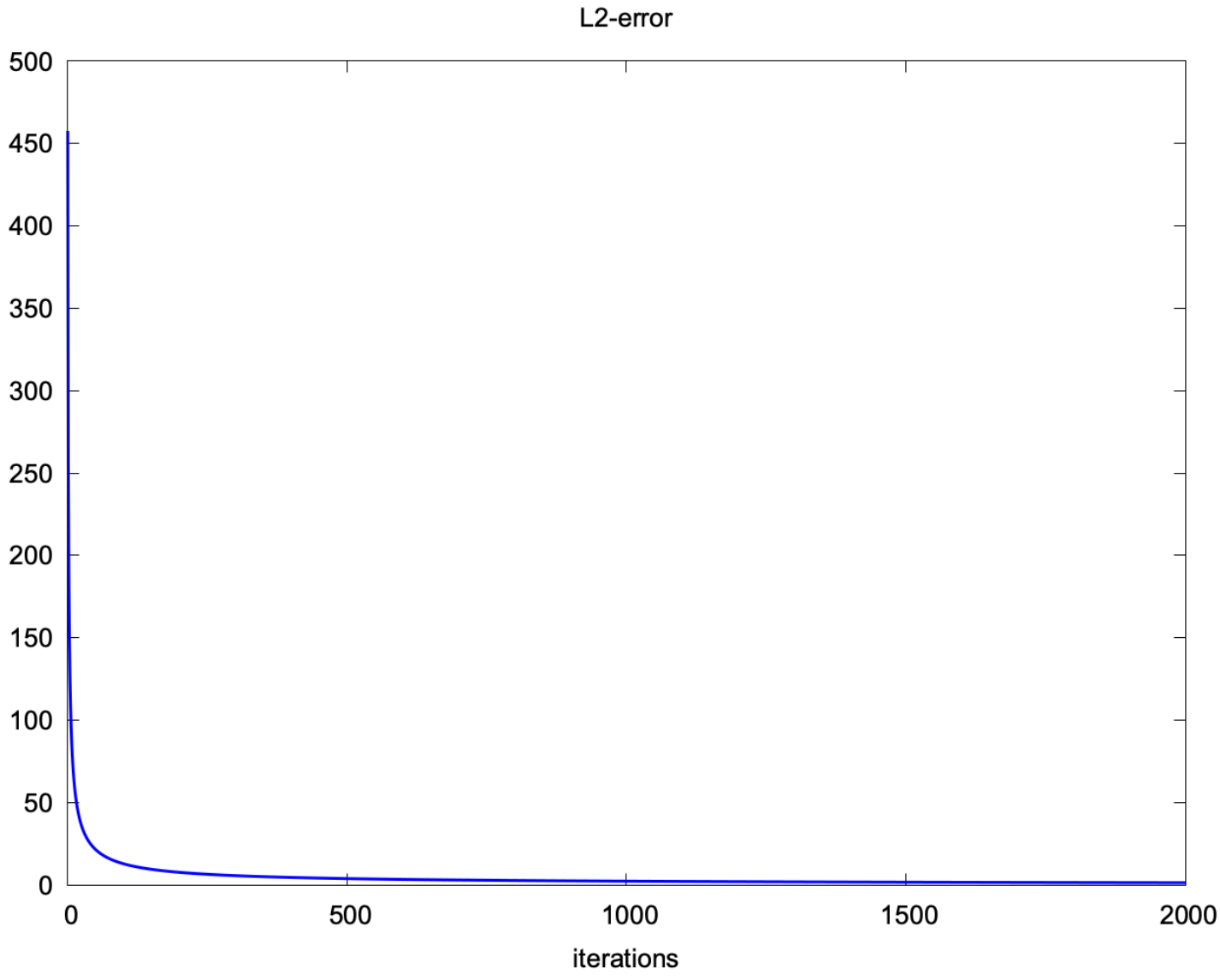


Figure 2: A diagram of the 1-D decomposition of the Jacobi Relaxation for Solving the Laplace's Equation, showing that the boundary elements that need to be communicated between processors.

Jacobi Algorithm: 1D Decomposition



Jacobi Algorithm: 1D Decomposition



Thank you!