# MPI

So far: communication (point-to-point and collective)

This time: data packing

# Datatypes

Built-in datatypes:

**`MPI_CHAR`**, **`MPI_INT`**, **`MPI_FLOAT`**, ...

What if you want to talk about your own structures?

More common:

What if you want to talk about rows, columns, and blocks of a 2-D matrix?

# Matrices in C

```
int m[M][N]; // M rows, N columns
```

| | | | |
|---|---|---|---|
| m[0][0] | m[0][1] | … | m[0][N-1] |
| m[1][0] | m[1][1] | … | m[1][N-1] |
| … | … | … | … |
| m[M][0] | m[M][1] | … | m[M-1][N-1] |

=

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| m[0][0] | m[0][1] | … | m[0][N-1] | m[1][0] | m[1][1] | … | m[1][N-1] | … … | m[M-1][N-1] |

This is ***row-major*** layout

# Matrices in C

```
int m[M][N]; // M rows, N columns
```

| | | | |
|---|---|---|---|
| m[0][0] | m[0][1] | … | m[0][N-1] |
| m[1][0] | m[1][1] | … | m[1][N-1] |
| … | … | … | … |
| m[M][0] | m[M][1] | … | m[M-1][N-1] |

=

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| m[0][0] | m[0][1] | … | m[0][N-1] | m[1][0] | m[1][1] | … | m[1][N-1] | … … | m[M-1][N-1] |

This is ***row-major*** layout

# Matrices in C

```
int m[M][N]; // M rows, N columns
```

| | | | |
|---|---|---|---|
| m[0][0] | m[0][1] | … | m[0][N-1] |
| m[1][0] | m[1][1] | … | m[1][N-1] |
| … | … | … | … |
| m[M][0] | m[M][1] | … | m[M-1][N-1] |

=

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| m[0][0] | m[0][1] | … | m[0][N-1] | m[1][0] | m[1][1] | … | m[1][N-1] | … … | m[M-1][N-1] |

This is ***row-major*** layout

# Matrices in C

```
int m[M][N]; // M rows, N columns
```

| | | | |
|---|---|---|---|
| m[0][0] | m[0][1] | … | m[0][N-1] |
| m[1][0] | m[1][1] | … | m[1][N-1] |
| … | … | … | … |
| m[M][0] | m[M][1] | … | m[M-1][N-1] |

=

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| m[0][0] | m[0][1] | … | m[0][N-1] | m[1][0] | m[1][1] | … | m[1][N-1] | … … | m[M-1][N-1] |

This is ***row-major*** layout

# Sending Rows

Send row **`j`**:

```
MPI_Send(&m[j][0], N, MPI_INT, ...);
```

The same, but nicer:

```
MPI_Send(&m[j][0], 1, row, ...);
```

... if we define **`row`**

Even nicer:

```
MPI_Scatter(m, 1, row, ...);
```

# Defining Rows

```
MPI_Datatype row;

MPI_Type_contiguous(N, MPI_INT, &row);
MPI_Type_commit(&row);
```

# Sending Rows

Send column **i**:

```
for (j = 0; j < M; j++)
  col[j] = m[j][i];
MPI_Send(col, M, MPI_INT, ...);
```

The same, but nicer:

```
MPI_Send(&m[0][i], 1, one_col, ...);
```

... if we define **one_col**

Even nicer:

```
MPI_Scatter(m, 1, col, ...);
```

... if we define **col**

# Defining One Column

```
MPI_Datatype one_col;

MPI_Type_vector(M, 1, N, MPI_INT, &one_col);
MPI_Type_commit(&one_col);
```

# Defining A Scatter-Friendly Column

```
MPI_Datatype col;
int lens[2] = { 1, 1 };
MPI_Aint offsets[2] = { 0, sizeof(int) };
MPI_Datatype types[2] = { one_col, MPI_UB };

MPI_Type_struct(2, lens, offsets, types, &col);
MPI_Type_commit(&col);
```

# Blocks

A **T** by **T** sub-array of an **N** by **N** array:

```
MPI_Datatype block;

MPI_Type_vector(T, T, N, MPI_INT, &block);
MPI_Type_commit(&block);

MPI_Send(&m[jb*T][ib*T], 1, block, ...);
```

Can't make **block** scatter-friendly