

CSC367 Parallel computing

Lecture 15: Improving the performance of scientific simulations

Some strategies to improve the performance of scientific simulations

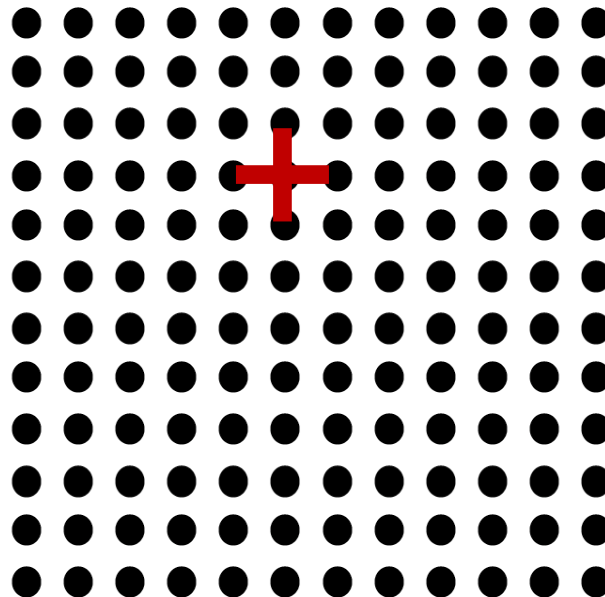
- Tune the surface to volume ratio
- Use ghost zones when necessary
- Approximate interactions if possible
- Compress data

Some strategies to improve the performance of scientific simulations

- Tune the surface to volume ratio
- Use ghost zones when necessary
- Approximate interactions if possible
- Compress data

Example: stencil computation on a mesh

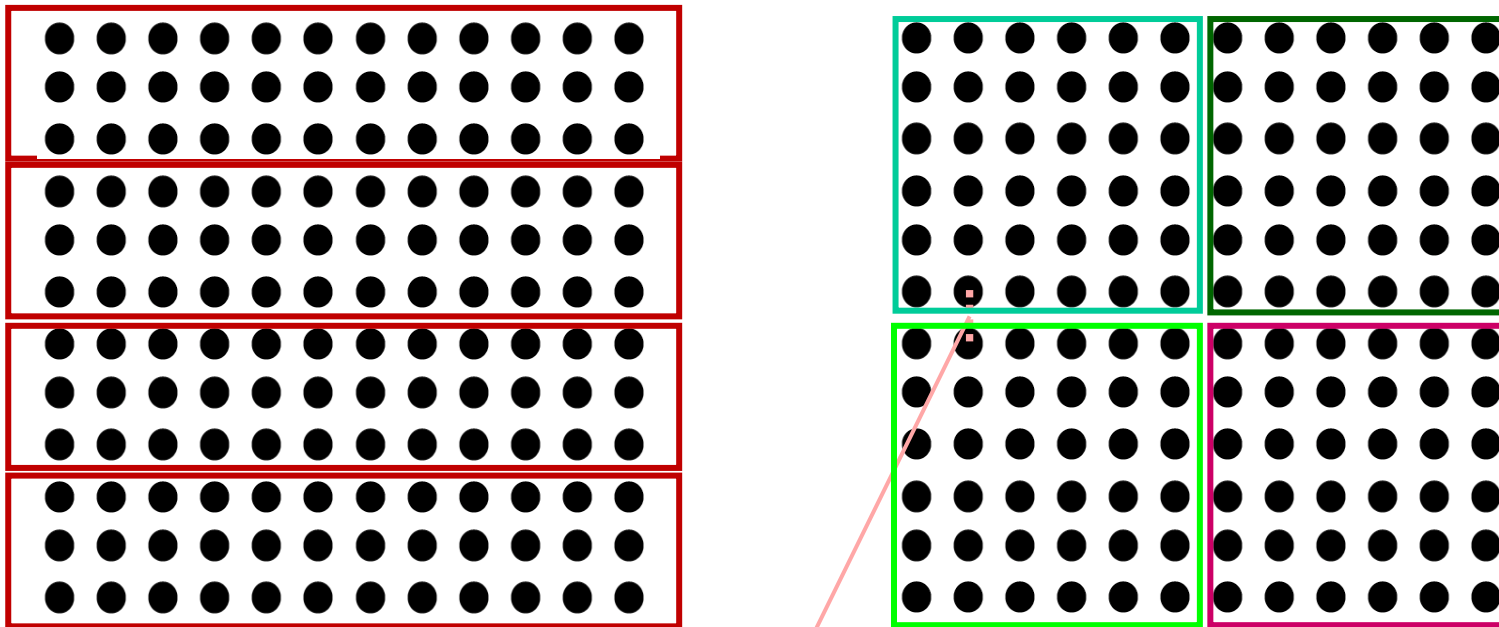
- Lets use an example to illustrate surface to volume ratio.
- Each circle is a mesh point and each cell in the mesh needs values from its neighbors to update the new mesh: This is known as a **stencil computation**



Example: stencil computation on a mesh

- Which of the partitionings below has less communication?
- What is the amount of edge crossing for each partitioning method if we have p partitions and an n by n grid?

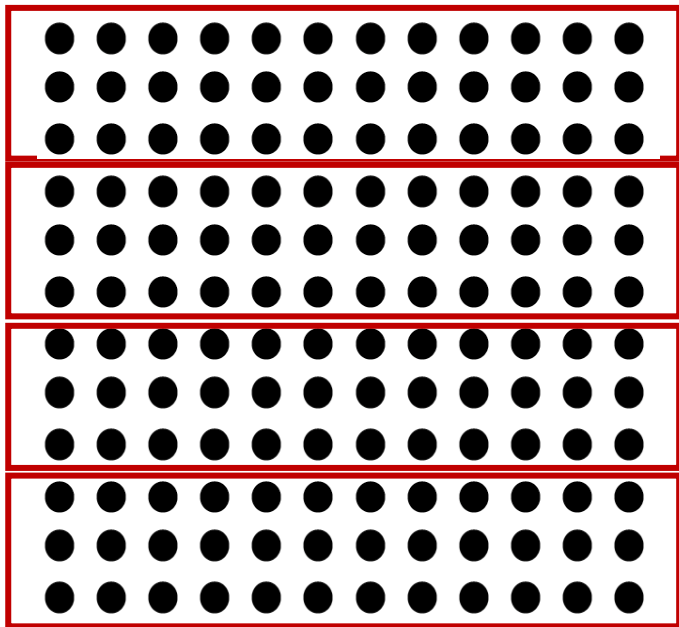
each process has 1 partition



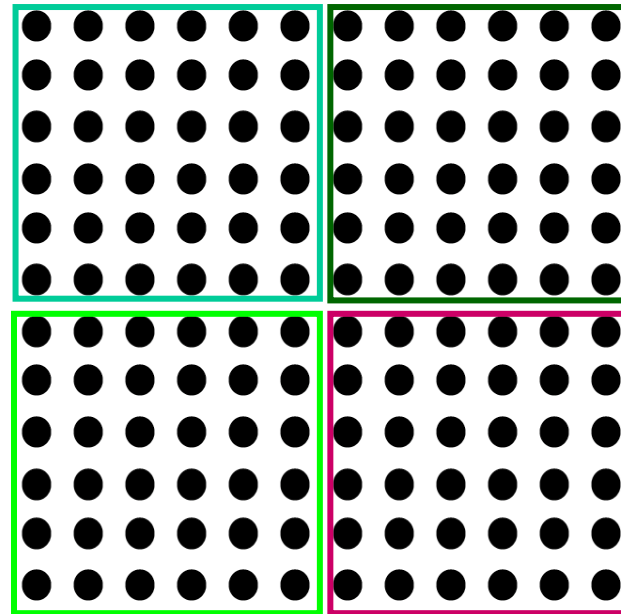
An edge crossing is where a communication edge is cut by the partitioning

Example: stencil computation on a mesh

- Which of the partitionings below has less communication?
- What is the amount of edge crossing for each partitioning method if we have p partitions and an n by n grid?



$n*(p-1)$
edge crossings



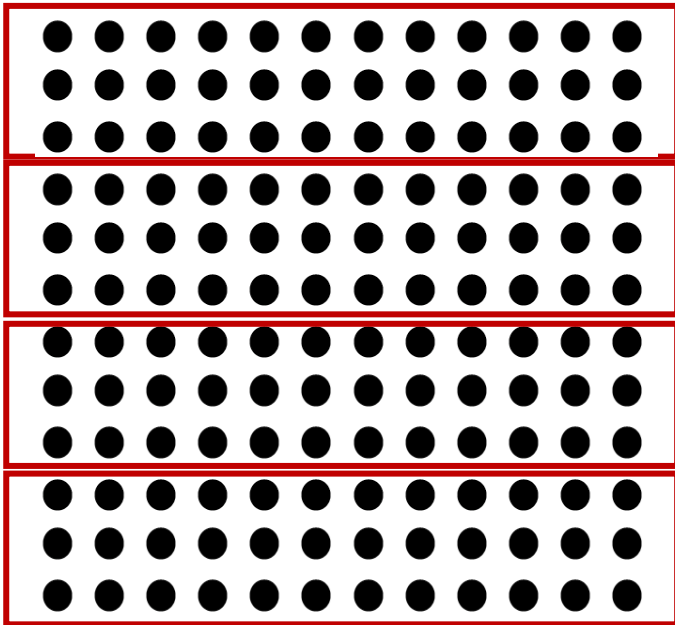
$2*n*(p^{1/2} - 1)$
edge crossings

Example: stencil computation on a mesh

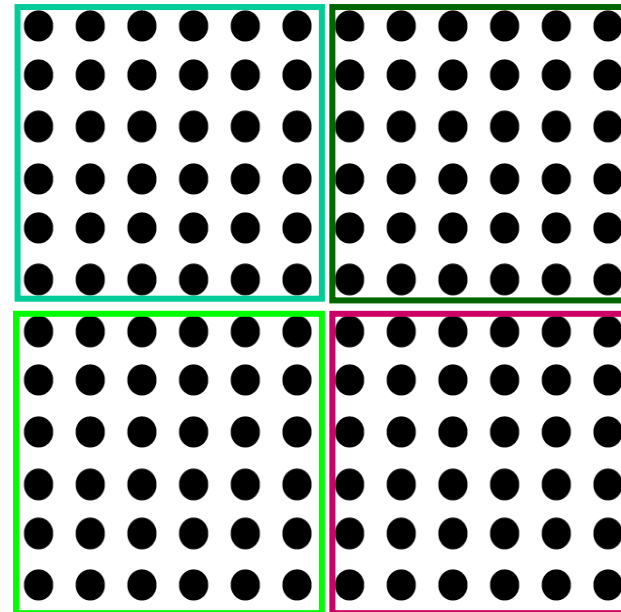
- **Volume** is defined as the amount of computation that happens per partition
- **Surface** is defined as the average edge crossings per partition
- Which scenario below has a **lower surface to volume ratio?**

want lower surface! reduce communication!

surface : communication cross process/node
volume : computation on 1 process/node



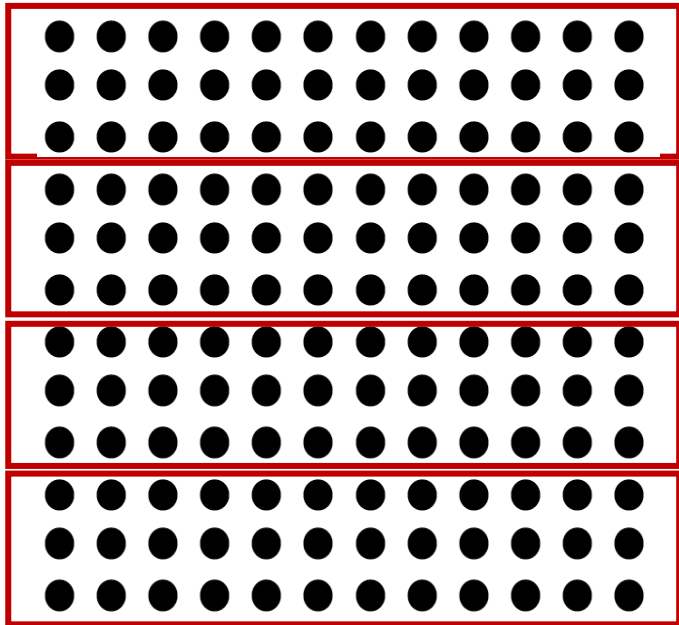
$n \cdot (p-1)$
edge crossings



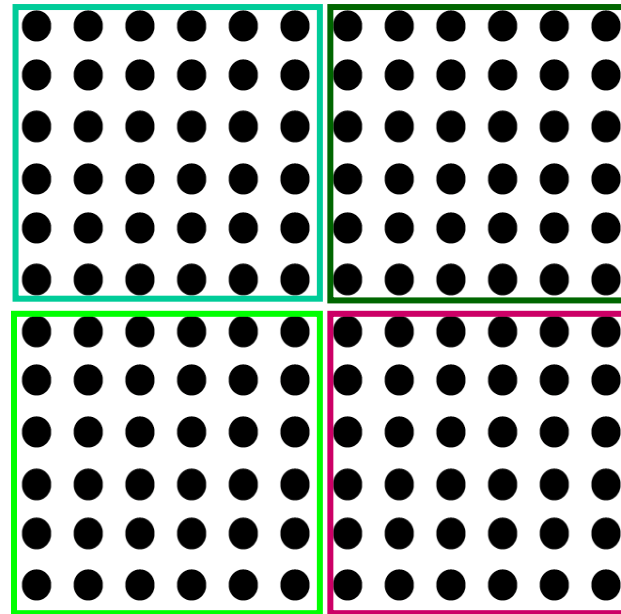
$2 \cdot n \cdot (p^{1/2} - 1)$
edge crossings

Example: stencil computation on a mesh

- The second scenario (block partitioning) has a lower surface to volume ratio which means the ratio of communication to computation is lower.
- A lower surface to volume ratio **might** lead to better performance if working on distributed platforms. Try both and observe!



$n \cdot (p - 1)$
edge crossings

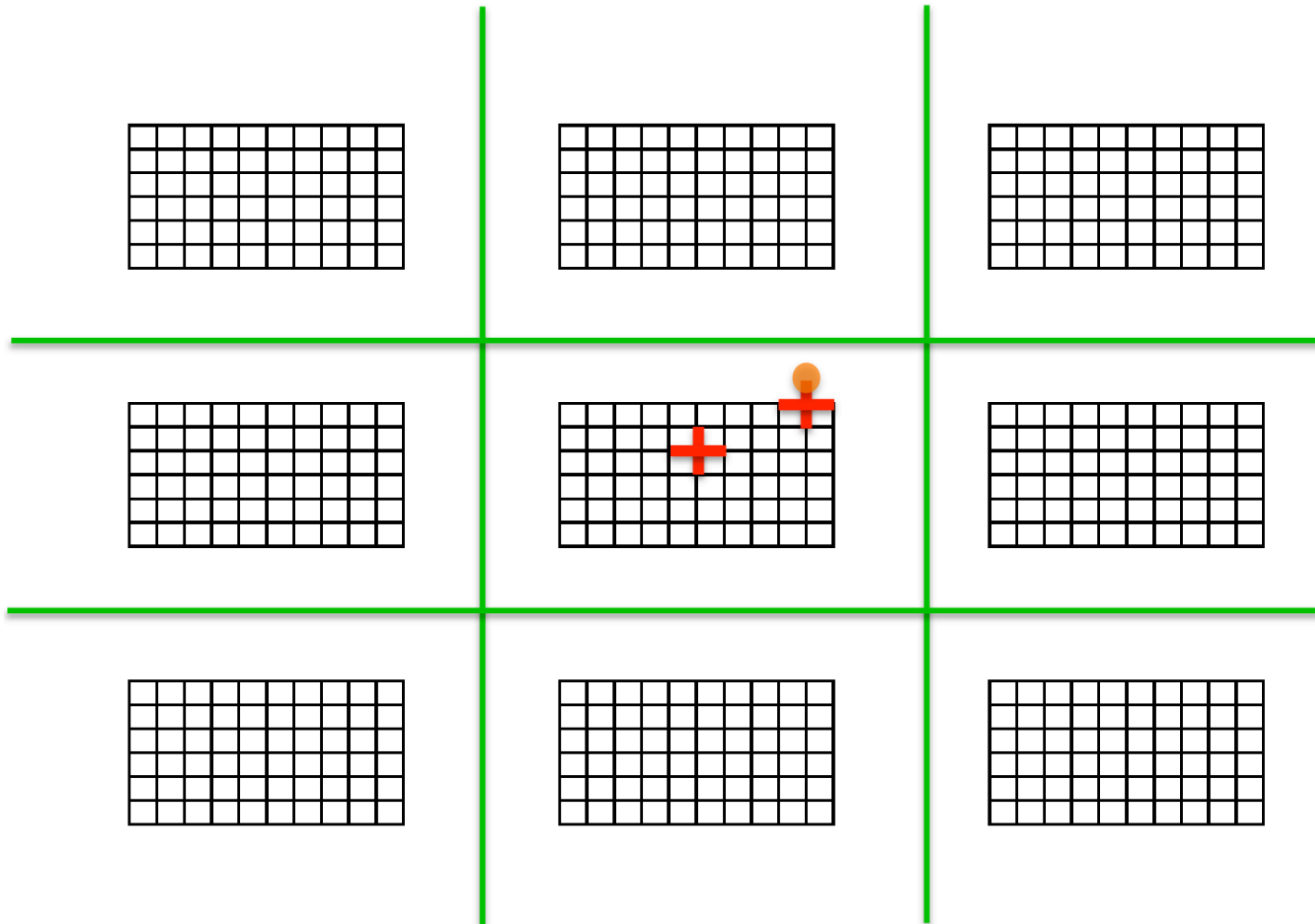


$2 \cdot n \cdot (p^{1/2} - 1)$
edge crossings

Some strategies to improve the performance of scientific simulations

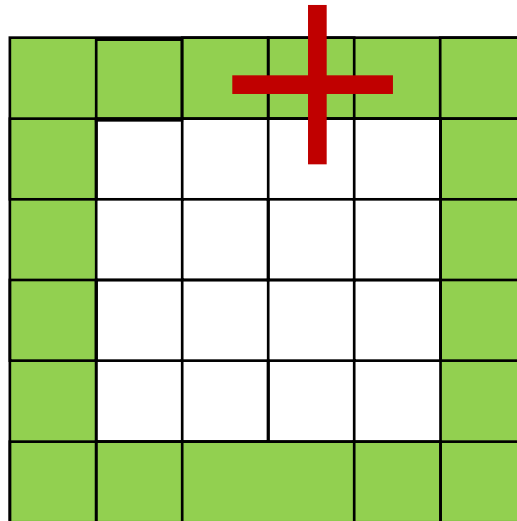
- Tune the surface to volume ratio
- Use ghost zones when necessary
- Approximate interactions if possible
- Compress data

Communication pattern in stencil code



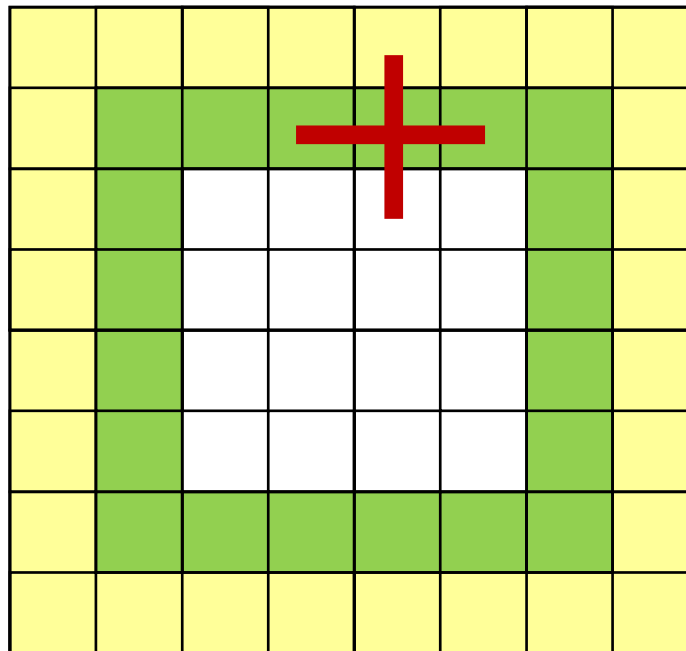
Redundant “Ghost” nodes in stencil computations

- Each partition needs information from its neighbors to calculate the values belonging to the green region (boundary elements)



Redundant “Ghost” elements in stencil computations

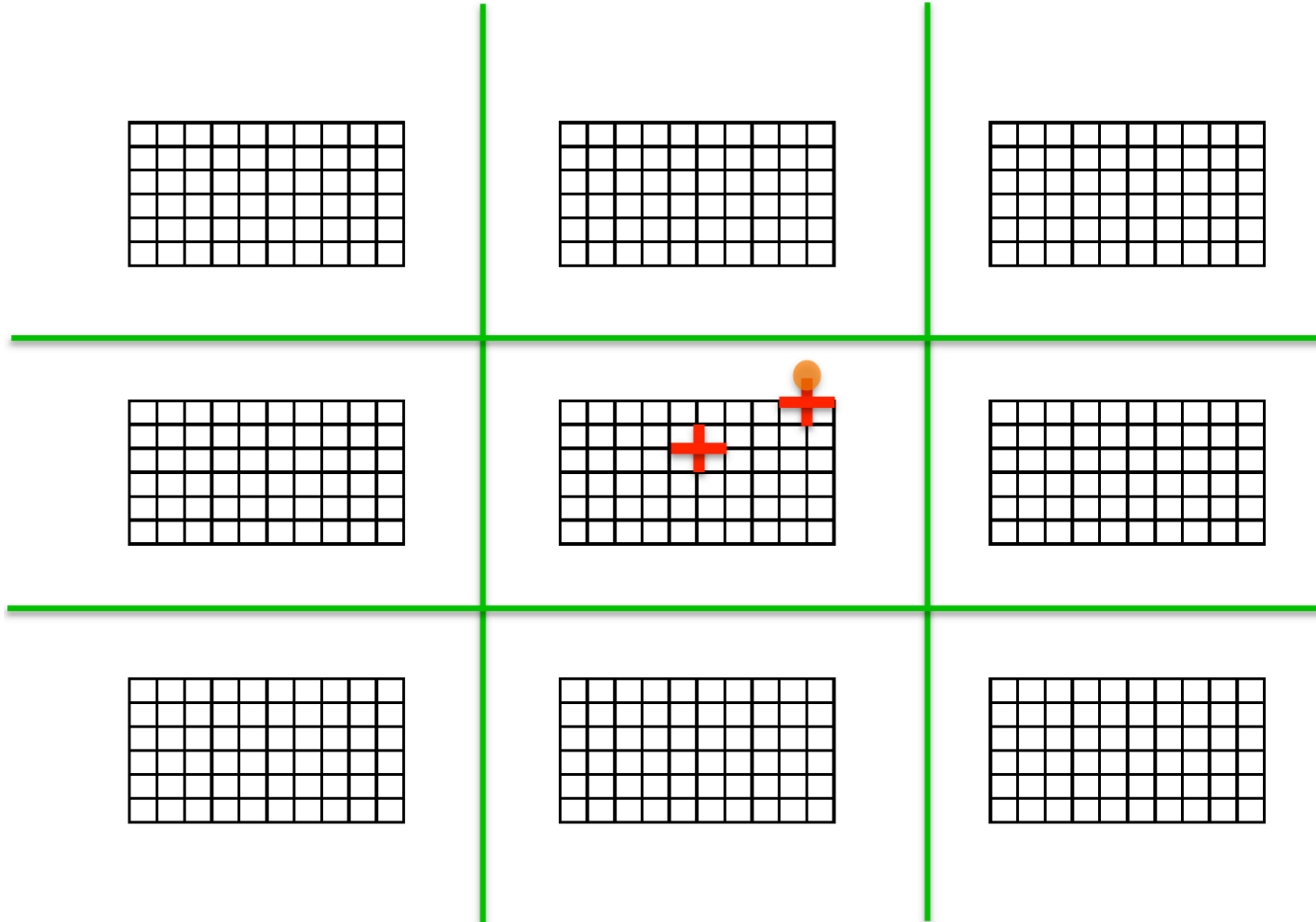
- We can copy redundant “ghost” or “halo” elements to each partition to enable the computations for that partition to proceed.
- The yellow region is the ghost zone!



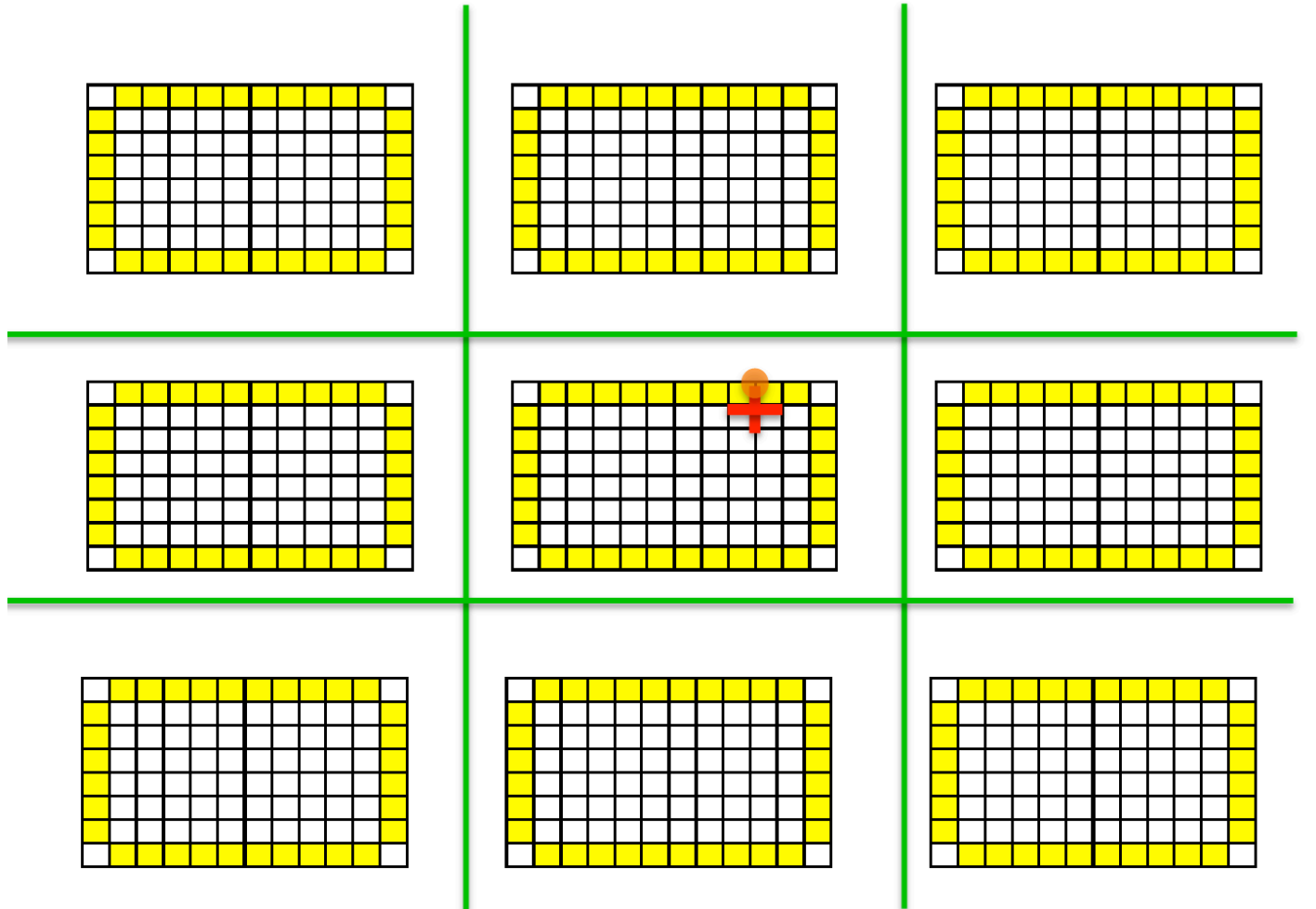
To compute green

Copy yellow

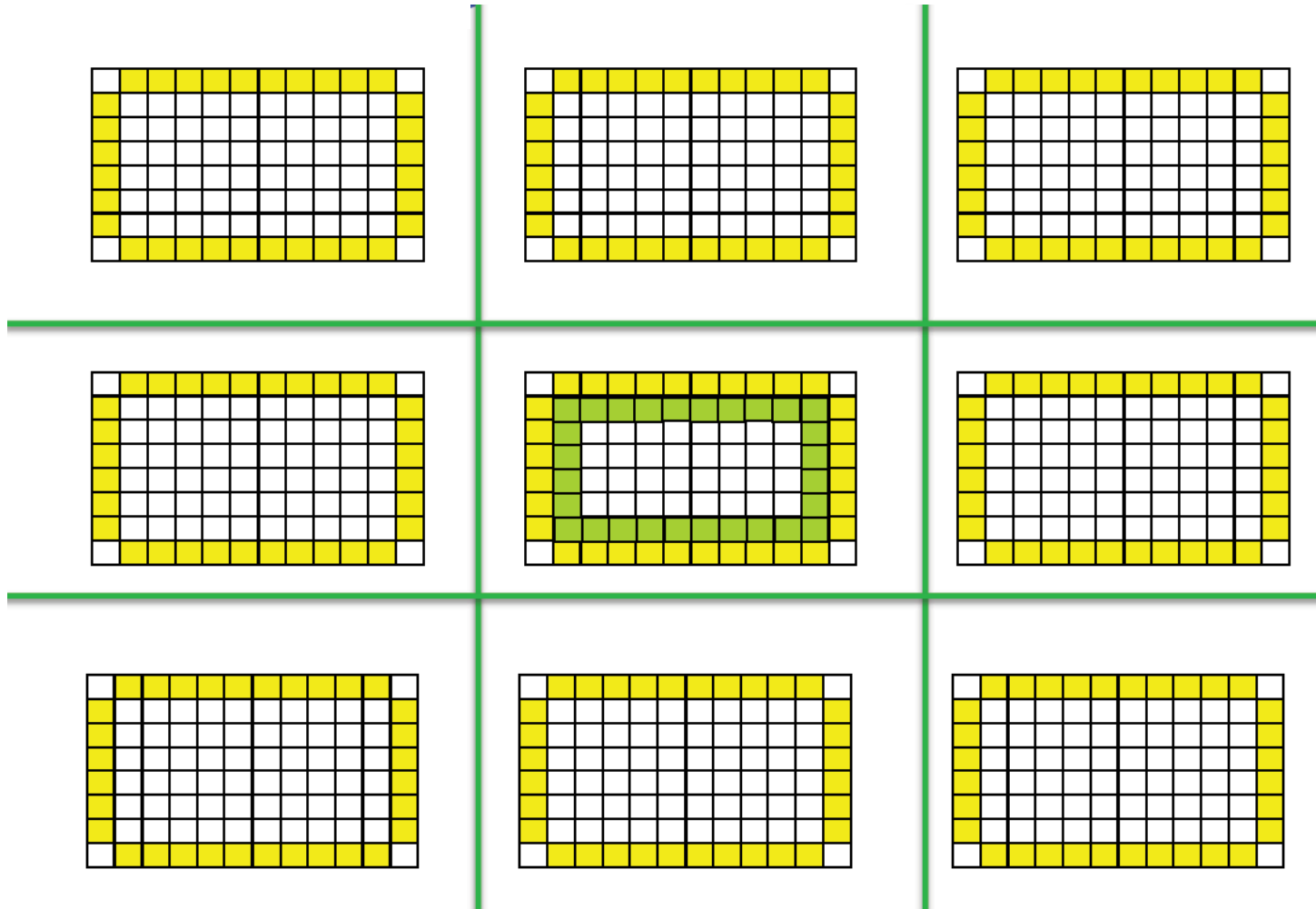
The default partitioning



Our partitions after copying ghost zones



Computation with the ghost zones



“Ghost” zones in computations

- In the previous algorithm ghosting required extra cells to be communicated between processors. This can happen before the computation initiates but you have to include it in your timing!
- Use non-blocking MPI operations for “halo” exchanges if possible to better overlap communication with computation! (hint: halos might help with your project’s MPI section)
- In some algorithms ghosting can lead to extra/redundant computation compared to the implementation that does not have ghost zones: remember that computation is often cheaper than communication so practitioners often prefer to do more compute if it helps to reduce the overall communication.
- Size of ghost region (and redundant computation) depends on network/memory speed vs. computation (flops) speed.

Some strategies to improve the performance of scientific simulations

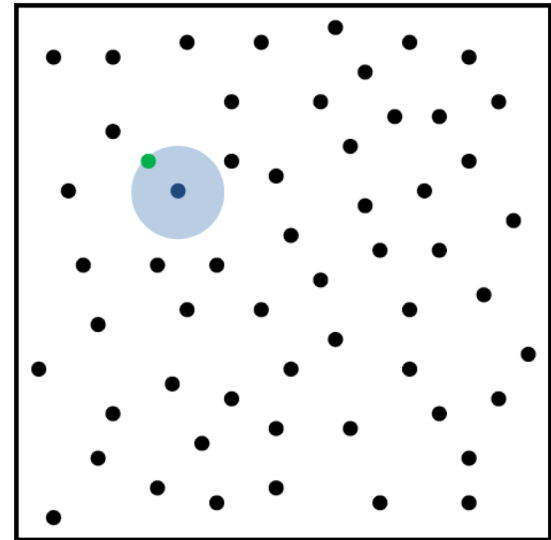
- Tune the surface to volume ratio
- Use ghost zones when necessary
- Approximate interactions if possible
- Compress data

Simulations with interactions!

- A large class of simulations in physics and in machine learning compute point/particle/data interactions. Examples include:
 - When masses are influencing each other with gravitation interactions: astronomy simulations, particles simulations, etc.
 - When electric charges interact.
 - Machine learning algorithms where data points inside a cluster have stronger correlation (“interactions”) vs loosely-coupled data points from different clusters.

Approximate interactions!

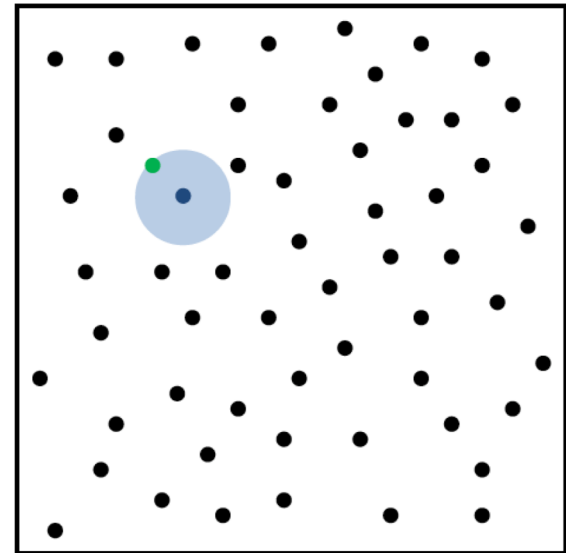
- It is often impossible or very expensive to compute all interactions in these simulations. The algorithm might not even be parallel!
- Example: Particles simulation
 - All particles interact with each other (can move each other around)
 - Simple algorithm: check every pair for collision: $O(n^2)$
 - Very expensive to calculate!



Interactions in nearby particles

ignore points that are far away

- However, particles that are located more than a cutoff radius of each other have little to no force to move one another!
- If the far-field interactions are ignored because they are small, we can optimize the simulation to only calculate interactions of particles in a "local neighborhood"
- The course project asks you to make this assumption.
- Note that approximating interactions comes at the cost of a less accurate solution so the loss in accuracy has to be acceptable by the practitioners in that domain!

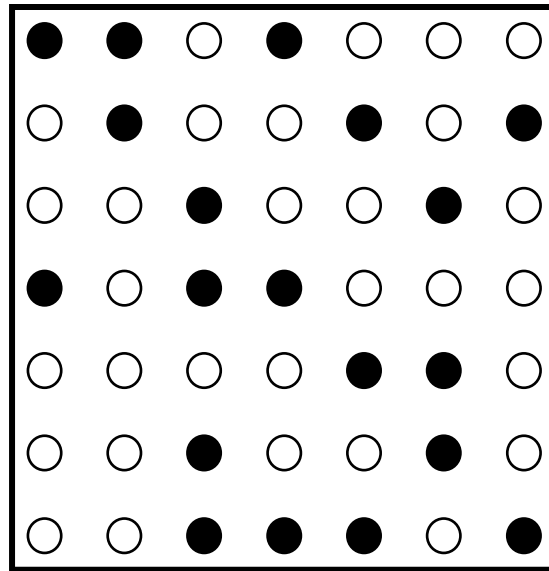


Some strategies to improve the performance of scientific simulations

- Tune the surface to volume ratio
- Use ghost zones when necessary
- Approximate interactions if possible
- Compress data

Compressing data

- Many scientific codes work with matrices that are sparse.
- A sparse matrix is a matrix that has very few non-zeros and all of its other elements are zero!
- The below is a sparse matrix where each dark circle is a non-zero and all white circles are zeros.

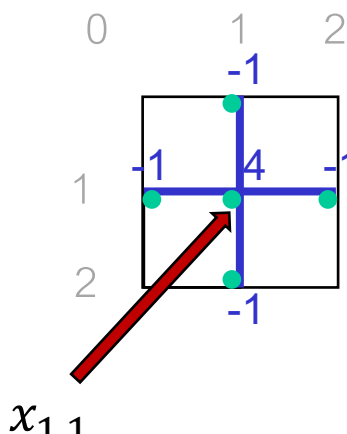


Sparse A

Stencil computations and sparse matrix vector multiplication

An example of sparse matrix multiplications used in scientific codes is when stencil computations are used in their algebraic form.

The below shows how a stencil update can be done using a sparse matrix-vector multiply. The solution vector \mathbf{y} is a vector that has all the new grid values!


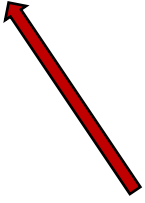
$$\mathbf{y} = \begin{pmatrix} 4 & -1 & & -1 & & & \\ -1 & 4 & -1 & & -1 & & \\ & -1 & 4 & & -1 & & \\ -1 & & & 4 & -1 & -1 & \\ & -1 & -1 & 4 & -1 & -1 & \\ & & -1 & -1 & 4 & & -1 \\ & & & -1 & & 4 & -1 \\ & & & & -1 & -1 & 4 \end{pmatrix} \times \begin{pmatrix} x_{0,0} \\ x_{0,1} \\ x_{0,2} \\ x_{1,0} \\ x_{1,1} \\ x_{1,2} \\ x_{2,0} \\ x_{2,1} \\ x_{2,2} \end{pmatrix}$$


\mathbf{A} is the stencil matrix and is sparse!

The vector \mathbf{x} stores the initial grid point values

Compressing sparse data structures for better performance

- For a very large large grid, the matrix A can become very large and sometimes it might not even fit in memory!
- As we know if data is too large to fit in caches/memory the code will run slow!

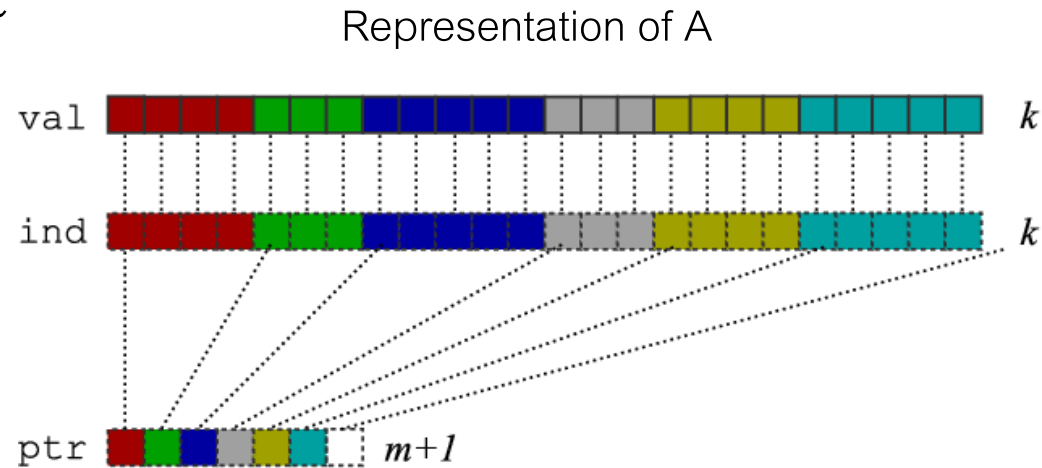
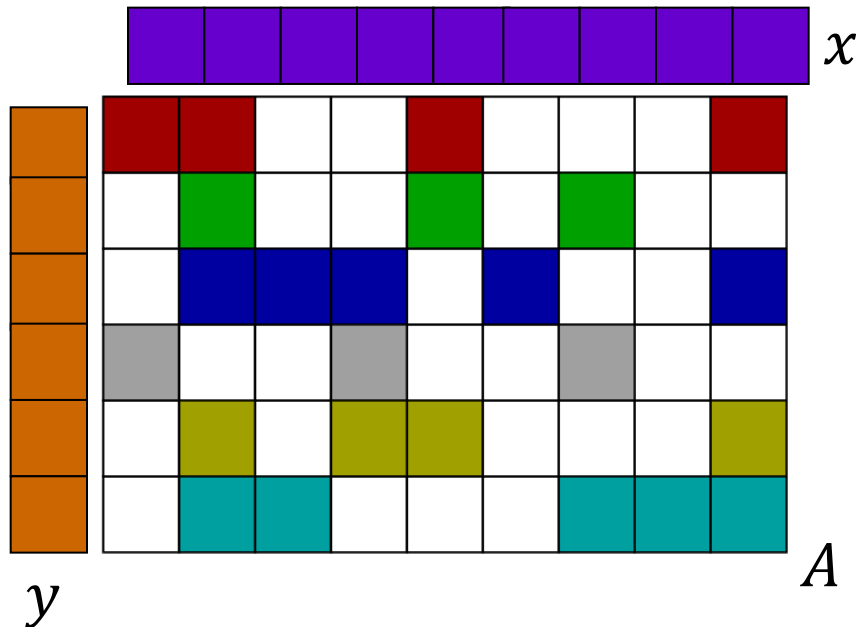
$$y = \begin{pmatrix} 4 & -1 & & -1 & & & \\ -1 & 4 & -1 & & -1 & & \\ & -1 & 4 & & & -1 & \\ -1 & & & 4 & -1 & & -1 \\ & -1 & & -1 & 4 & -1 & \\ & & -1 & & -1 & 4 & \\ & & & -1 & & 4 & -1 \\ & & & & -1 & & 4 \end{pmatrix} \times \begin{pmatrix} x_{0,0} \\ x_{0,1} \\ x_{0,2} \\ x_{1,0} \\ x_{1,1} \\ x_{1,2} \\ x_{2,0} \\ x_{2,1} \\ x_{2,2} \end{pmatrix}$$



A is the stencil matrix and is sparse!

The vector x stores the initial grid point values

Compress the sparse matrix

- A sparse matrix can be compressed in many ways.
- In the below we show the compressed sparse row (CSR) storage format.
- The *val* vector stores all the non-zero values in the matrix. The *ind* vector stores the column index of the non-zero values. The *ptr* vector points to the beginning of each row in the matrix in *val* and *ind*.
- Compression can significantly reduce the memory footprint of the sparse data.



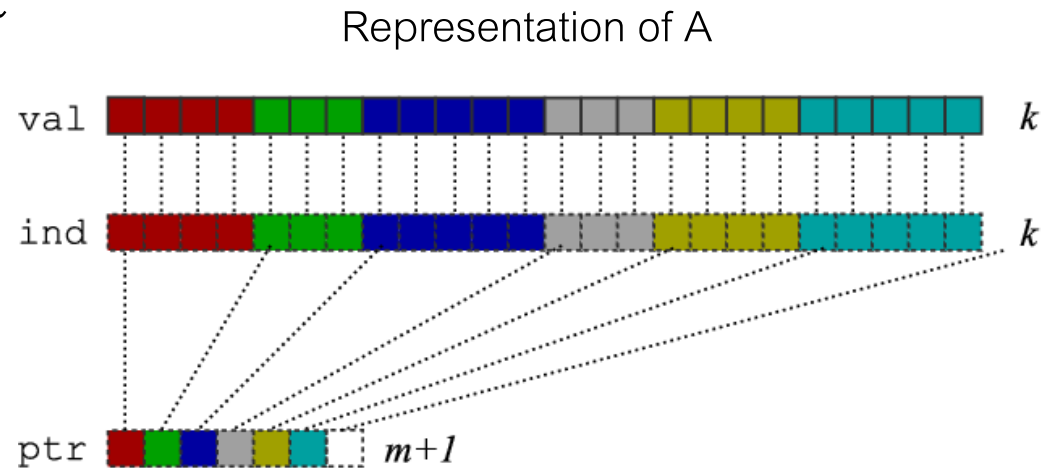
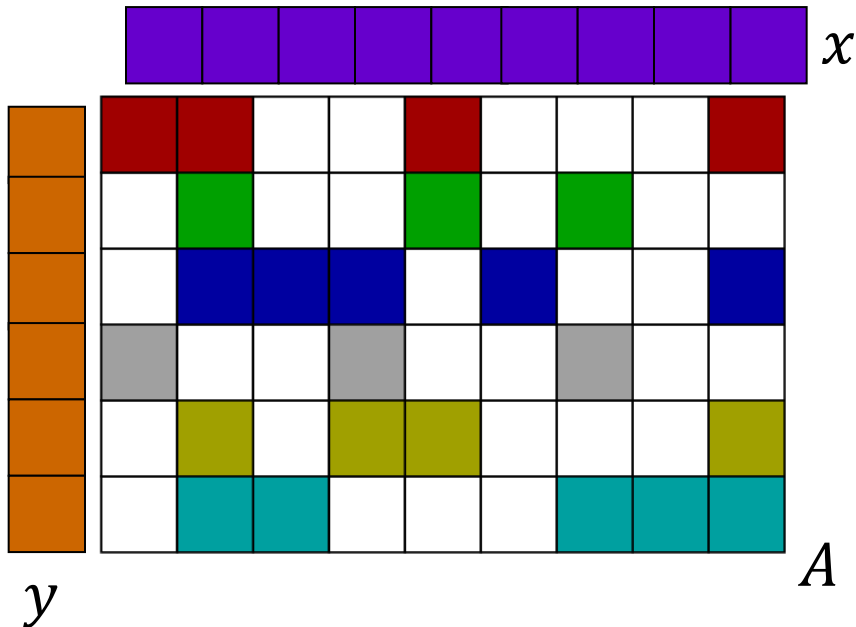
Sparse matrix multiplication in CSR format

Matrix-vector multiply kernel: $y = y + A x$

for each row i

for $k = \text{ptr}[i]$ to $\text{ptr}[i+1]-1$ do

$y[i] = y[i] + \text{val}[k] * x[\text{ind}[k]]$



Optimizations compressed sparse codes

- Compression helps to significantly reduce the memory and storage costs in matrix computations.
- However, optimizing sparse matrix codes (matrix methods that operate on sparse data) is much harder than optimizing dense operations.
- Indirections in the memory also handicap compilers for optimizing these codes.
- Some of the challenges in optimizing sparse codes are:
 - load imbalance (see the sparse matrix multiply example where each row has a different number of nonzeros).
 - Data is not well aligned in memory and we often need to use padding to enforce alignment.
 - The computation order and pattern is irregular so finding well-balanced tasks to run in parallel is challenging.

The mid-term test

- 15 % of the total grade
- Time: March 6 at 2pm
- Location: Exam center room EX 310
- Exam starts on the hour!
 - Plan to be there 10-15 minutes ahead
 - The exam is closed book, use of cell phones, calculators, etc is prohibited.

General tips

- Make sure you go over each topic and that you are at the very least comfortable with the material
- Read course slides, lecture, assignments & lab exercises carefully.
- Review assignments
- The test will have multiple choice questions and wrong answers will be penalized! So answer if you are reasonably sure.
- The test will have description questions which will test you on your understanding of concepts such as cache and memory access patterns, computational intensity, etc.
- Sadly .. no previous tests available, for obvious reasons
- Need help? Come to office hours today!