DMRG Tutorial

James Vance
github.com/jnvance
jvance@ictp.it · jnvance@up.edu.ph

DMRG.x is a **distributed-memory** implementation of the **Density Matrix Renormalization Group** (DMRG) algorithm based on the **PETSc and SLEPc libraries**

Documentation:	https://**DMRGx.rtfd.io**
Source Code:	https://**github.com/jnvance/**DMRG.x
Tutorial Resources:	https://**github.com/jnvance/**DMRGTutorial

Contents of this tutorial

- DMRG algorithm
- Intro to parallel programming with MPI
- PETSc and SLEPc libraries
- Using the DMRG.x application

DMRG Algorithm

Traditional DMRG and its computational aspects

Main Reference:

Feiguin, Adrian E. "The Density Matrix Renormalization Group and its time-dependent variants." *AIP Conference Proceedings*. Vol. 1419. No. 1. AIP, 2011. [link]

Density Matrix Renormalization Group

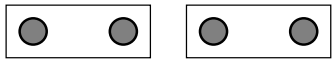
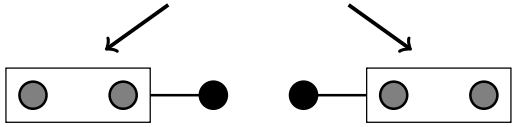
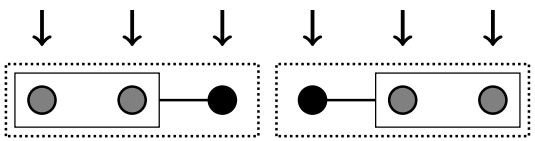
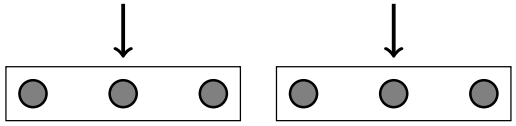
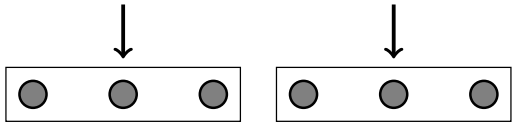
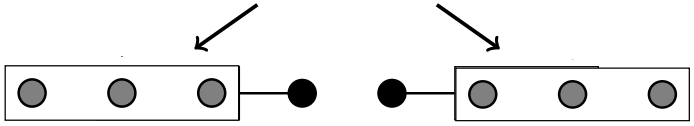
- DMRG is one of the most powerful numerical algorithms used to study the ground state properties of many-body systems
- Developed in 1992 by Steven White to overcome problems in applying NRG to quantum lattice systems
- Currently being used in many problems in physics and quantum chemistry

Advantages of using DMRG

- Can treat large systems with high precision with less computational resources than exact diagonalization

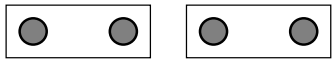
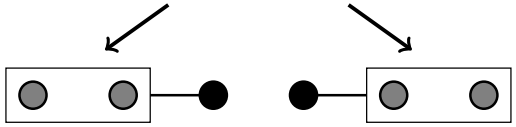
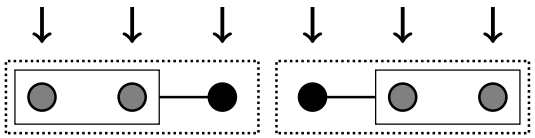
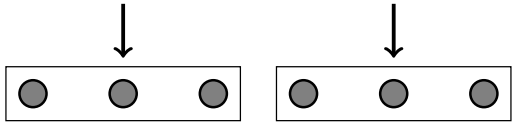
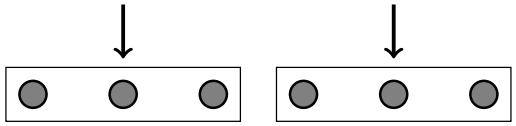
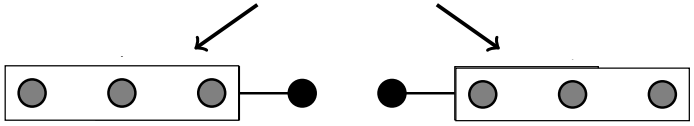
- Unlike quantum Monte Carlo methods, it does not suffer from the negative sign problem in frustrated and fermionic systems

The Infinite-size DMRG Algorithm

		length	basis size
Given two blocks and all of its operators		N	m
Single DMRG Step			
Grow the blocks by **adding one site** (dimension d)		$N+1$	md
Form the superblock and **diagonalize** the Hamiltonian		$2N+2$	$\boldsymbol{m^2d^2}$
From the ground state, determine the **reduced density matrices** (RDMs) of each block		$N+1$	$\boldsymbol{md}$
Truncate the basis using the top $\boldsymbol{D_R}$ eigenstates of the RDM		$N+1$	$\boldsymbol{m}$
⋮		$N+2$	md

Reduced basis

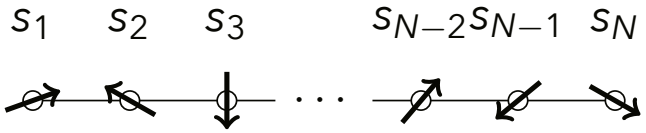
The Infinite-size DMRG Algorithm
From a **computational** perspective

		length	basis size
Setting up a **matrix representation**		N	m
Single DMRG Step			
Implementation of a **Kronecker product** routine		$N+1$	md
Diagonalization of a large **sparse matrix**		$2N+2$	$\boldsymbol{m^2d^2}$
Construction and **eigendecomposition** of the reduced density matrices		$N+1$	$\boldsymbol{md}$
		Reduced basis	
Rotation of the operator matrices to a truncated basis		$N+1$	$\boldsymbol{m}$
⋮		$N+2$	md

Matrix Representation

Illustrated with the Spin-1/2 Heisenberg Hamiltonian

1D spin chain:



Heisenberg Hamiltonian:

$$\hat{H} = \sum_{i=1}^{N-1} \hat{\mathbf{S}}_i \cdot \hat{\mathbf{S}}_{i+1}$$

$$\hat{H} = \sum_{i=1}^{N-1} \hat{S}_i^z \hat{S}_{i+1}^z + \frac{1}{2}\left[\hat{S}_i^+ \hat{S}_{i+1}^- + \hat{S}_i^- \hat{S}_{i+1}^+\right]$$

where

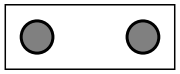
$$S^z = \begin{pmatrix} 1/2 & 0 \\ 0 & -1/2 \end{pmatrix}, S^+ = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix}, S^- = \begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix},$$

Feiguin, Adrian E. "The Density Matrix Renormalization Group and its time-dependent variants." AIP Conference Proceedings. Vol. 1419. No. 1. AIP, 2011.

Matrix Representation

Illustrated with the Spin-1/2 Heisenberg Hamiltonian

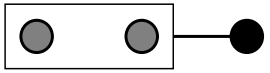
- Two spins:



$$\hat{H} = \hat{S}_1^z \hat{S}_2^z + \frac{1}{2}\left[\hat{S}_1^+ \hat{S}_2^- + \hat{S}_1^- \hat{S}_2^+\right]$$

$$H_2 = S^z \otimes S^z + \frac{1}{2}\left[S^+ \otimes S^- + S^- \otimes S^+\right]$$

- Adding a single site:



$$H_3 = H_2 \otimes \mathbb{1}_2 + \tilde{S}_2^z \otimes S^z + \frac{1}{2}\left[\tilde{S}_2^+ \otimes S^- + \tilde{S}_2^- \otimes S^+\right]$$

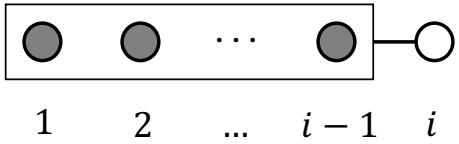
$$\tilde{S}_2^z = \mathbb{1}_2 \otimes S^z, \quad \tilde{S}_2^\pm = \mathbb{1}_2 \otimes S^\pm$$

Feiguin, Adrian E. "The Density Matrix Renormalization Group and its time-dependent variants." AIP Conference Proceedings. Vol. 1419. No. 1. AIP, 2011.

Matrix Representation

Illustrated with the Spin-1/2 Heisenberg Hamiltonian

- General recursion for adding an i^{th} site to a block of $i-1$ sites



$$H_i = H_{i-1} \otimes \mathbb{1}_2 + \tilde{S}^z_{i-1} \otimes S^z + \frac{1}{2}\left[\tilde{S}^+_{i-1} \otimes S^- + \tilde{S}^-_{i-1} \otimes S^+\right]$$

where

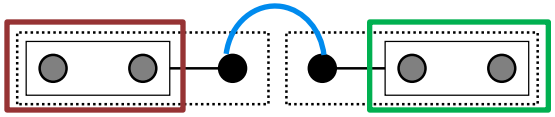
$$H_2 = S^z \otimes S^z + \frac{1}{2}\left[S^+ \otimes S^- + S^- \otimes S^+\right]$$

$$\tilde{S}^z_{i-1} = \mathbb{1}_{2^{i-2}} \otimes S^z, \qquad \tilde{S}^\pm_{i-1} = \mathbb{1}_{2^{i-2}} \otimes S^\pm$$

Feiguin, Adrian E. "The Density Matrix Renormalization Group and its time-dependent variants." AIP Conference Proceedings. Vol. 1419. No. 1. AIP, 2011.

Matrix Representation
of the superblock Hamiltonian

- Superblock:



$$\hat{H} = \hat{H}_{L,i+1} + \hat{H}_{R,i+2} + S^z_{i+1} S^z_{i+2} + \frac{1}{2}\left(S^+_{i+1} S^-_{i+2} + S^-_{i+1} S^+_{i+2}\right)$$

- Explicitly:

$$\begin{aligned} H &= H_{L,i+1} \otimes \mathbb{1}_{D_R \times 2} + \mathbb{1}_{D_L \times 2} \otimes H_{R,i+2} \\ &+ \tilde{S}^z_{L,i+1} \otimes \tilde{S}^z_{R,i+2} \\ &+ \frac{1}{2} \tilde{S}^+_{L,i+1} \otimes \tilde{S}^-_{R,i+2} \\ &+ \frac{1}{2} \tilde{S}^-_{L,i+1} \otimes \tilde{S}^+_{R,i+2} \end{aligned}$$

Feiguin, Adrian E. "The Density Matrix Renormalization Group and its time-dependent variants." AIP Conference Proceedings. Vol. 1419. No. 1. AIP, 2011.

Diagonalization
of the superblock Hamiltonian

- Solve the ground state of the superblock Hamiltonian:

$$H|\Psi\rangle = E|\Psi\rangle$$

- Iterative diagonalization using the Lanczos algorithm:

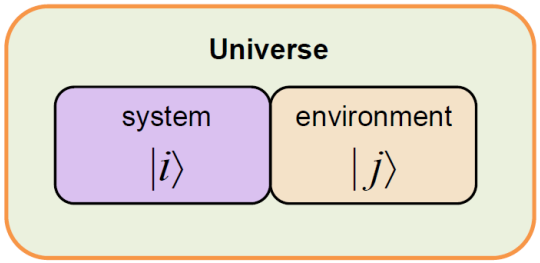
$$|\phi_{n+1}\rangle = \hat{H}|\phi_n\rangle - a_n|\phi_n\rangle - b_n^2|\phi_{n-1}\rangle,$$

$$a_n = \frac{\langle\phi_n|\hat{H}|\phi_n\rangle}{\langle\phi_n|\phi_n\rangle}, \qquad b_n^2 = \frac{\langle\phi_n|\phi_n\rangle}{\langle\phi_{n-1}|\phi_{n-1}\rangle},$$

- Only the matrix-vector product has to be known
- The large sparse matrix (size $D_L D_R d^2 \times D_L D_R d^2$) should not be constructed **(matrix-free approach)**

Feiguin, Adrian E. "The Density Matrix Renormalization Group and its time-dependent variants." AIP Conference Proceedings. Vol. 1419. No. 1. AIP, 2011.

Construction and Eigendecomposition
of the Reduced Density Matrices



- Bipartite system:
- Reduced density matrix of the "system" block

$$\hat{\rho}_A = \mathrm{Tr}_B |\Psi\rangle\langle\Psi|$$

$$\rho_{A\,ii'} = \langle i|\hat{\rho}_A|i'\rangle = \sum_j \langle i\,j|\Psi\rangle\langle\Psi|i'\,j\rangle = \sum_j \Psi_{i\,j}\Psi^*_{i'\,j}$$

- Properties of $\hat{\rho}_A$
  1. Hermitian
  2. Non-negative eigenvalues
  3. Unit-trace
  4. Eigenvectors form an orthonormal basis

New eigenvector basis:

$$\hat{\rho}_A = \sum_\alpha \omega_\alpha |\alpha\rangle\langle\alpha|;$$

with $\omega_\alpha \geq 0$ and $\sum_\alpha \omega_\alpha = 1$

Feiguin, Adrian E. "The Density Matrix Renormalization Group and its time-dependent variants." AIP Conference Proceedings. Vol. 1419. No. 1. AIP, 2011.

Density Matrix Truncation and Rotation
to the New Basis

- Eigendecomposition of the reduced density matrix

$$\hat{\rho}_A = \sum_{\alpha} \omega_\alpha |\alpha\rangle\langle\alpha|$$

- Keep at most m states with the largest eigenvalues ω_α forming a rectangular rotation operator U $(md \times m)$
- Rotate each block operator O_A to the truncated basis

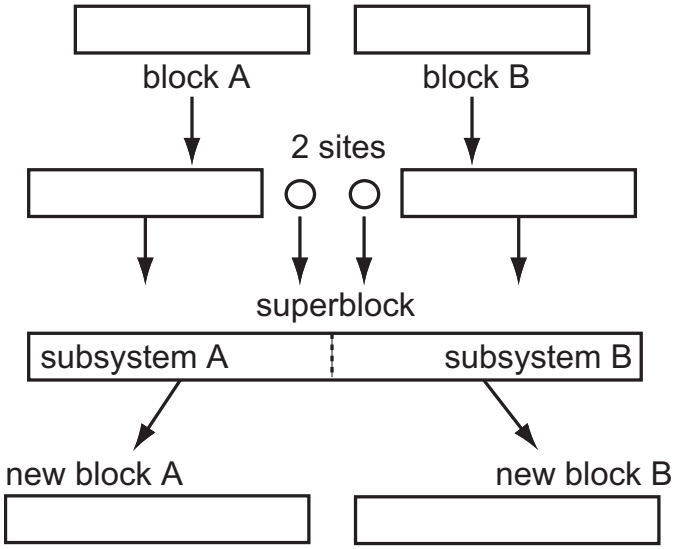
$$\tilde{O}_A = U^\dagger O_A U$$

$md \times md$

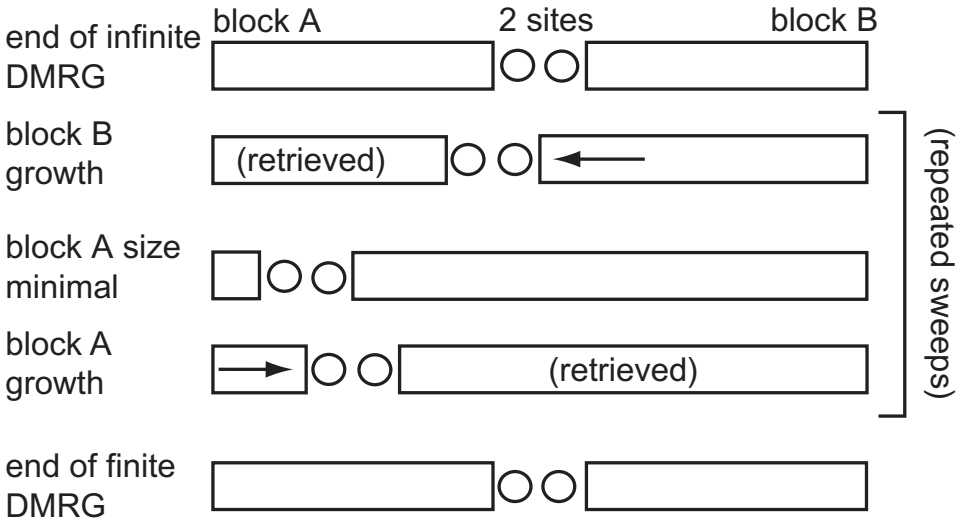
$m \times m$

Feiguin, Adrian E. "The Density Matrix Renormalization Group and its time-dependent variants." AIP Conference Proceedings. Vol. 1419. No. 1. AIP, 2011.

DMRG Algorithm



Infinite-size algorithm (warmup)



Finite-size algorithm (sweeps)

Schollwöck, Ulrich. "The density-matrix renormalization group in the age of matrix product states." Annals of Physics 326.1 (2011): 96-192.

Sample codes and links

`https://github.com/jnvance/DMRGTutorial`

To download: **Clone or download**

◎ `git clone <url>`

◎ Download ZIP

SIMPLE-DMRG

Prototype Implementation in Python

Main Reference:

James R. Garrison, & Ryan V. Mishmash. (2017, November 29). simple-dmrg/simple-dmrg: Simple DMRG 1.0 (Version v1.0.0). Zenodo. http://doi.org/10.5281/zenodo.1068359

Prototype Implementation: SIMPLE-DMRG

- Developed by J. Garrison and R. Mishmash (UCSB) for the 2013 Trieste summer school on quantum spin liquids

- 1D DMRG code built on numpy and scipy functions:
  - `scipy.sparse.kron` – Kronecker product
  - `scipy.sparse.linalg.eigsh` – Lanczos routine

- Demonstration:
  - `simple_dmrg_02_finite_system.py`
  - `simple_dmrg_03_conserved_quantum_numbers.py`

Features

- Parallel: Multi-threading capabilities
  - if scipy is compiled with multithreaded BLAS/MKL

- Built for the 1D Heisenberg XXZ Hamiltonian but it is extensible to other models as well

Limitations

- 2D models often require larger m number of states kept after truncation
- Hamiltonian dimension ~ $m^2 d^2$
  - e.g. for spin-1/2 with m=2000 -> 16,000,000 states
- 2D DMRG requires parallel programming approach

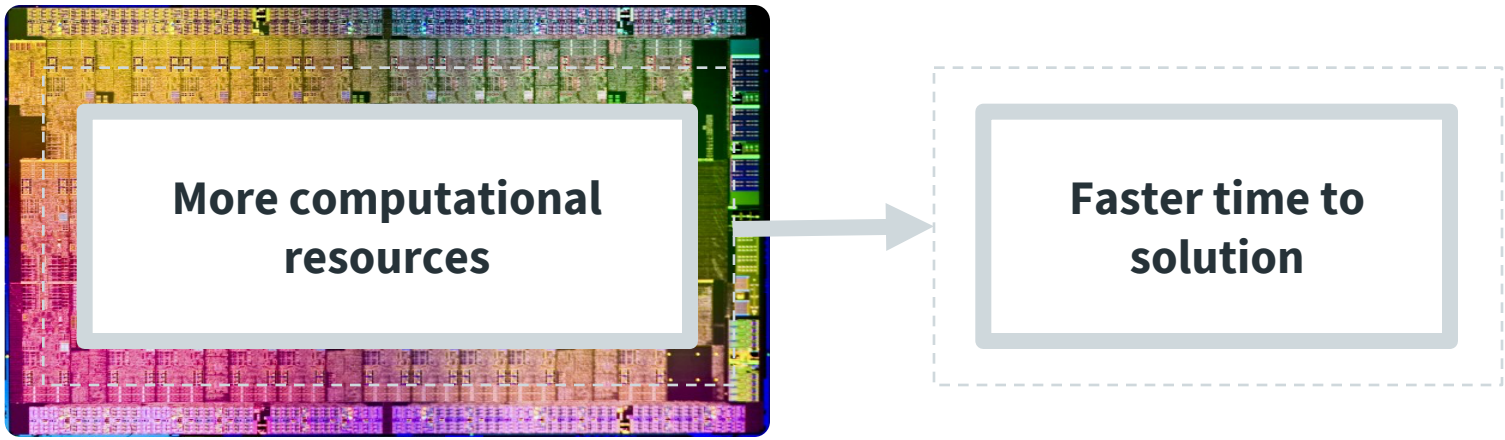
(Quick) Introduction to Parallel Computing

Main Reference:

Barney, Blaise. "Introduction to parallel computing." Lawrence Livermore National Laboratory 6.13 (2010): 10. https://computing.llnl.gov/tutorials/parallel_comp/

Parallel Computing

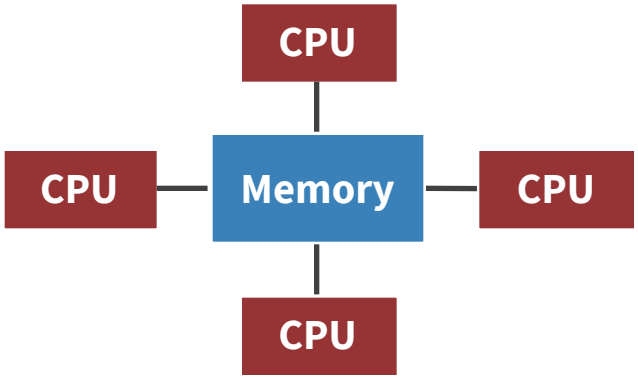
- A form of computation that performs many calculations simultaneously
- Instead of making CPUs faster, just **add more cores**
- Present in most modern computers and mobile devices having “multi-core” technology



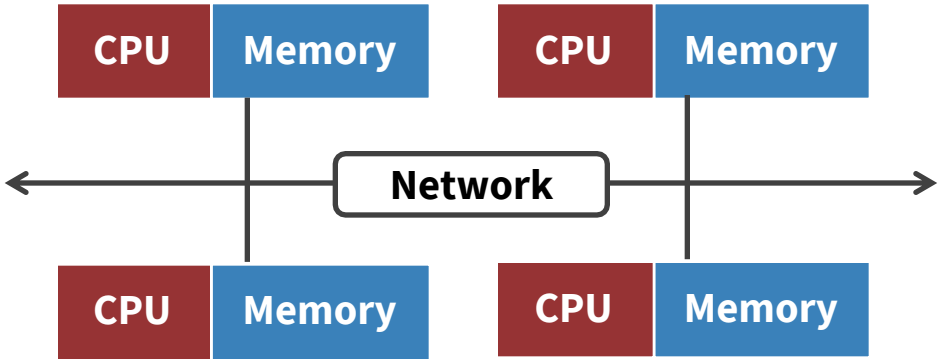
Intel Knights Landing: 72-core Xeon Phi supercomputing chip

Barney, Blaise. "Introduction to parallel computing." Lawrence Livermore National Laboratory 6.13 (2010): 10. https://computing.llnl.gov/tutorials/parallel_comp/

Parallel Computing Memory Architectures

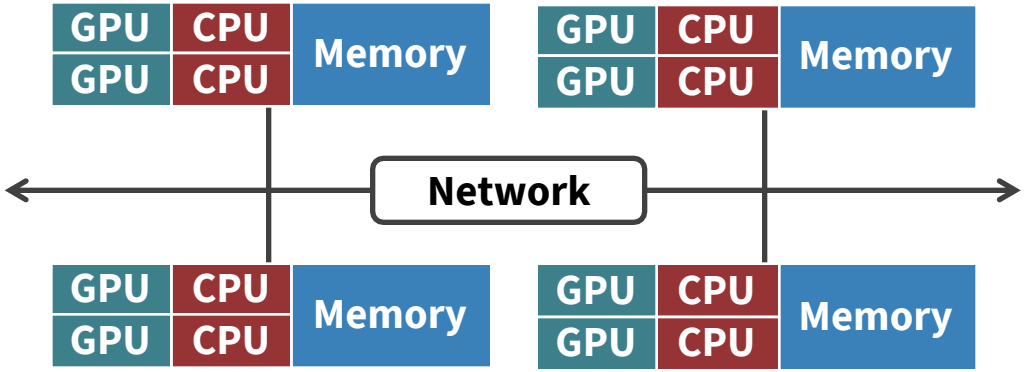


▲ Shared Memory
Multithreading, OpenMP

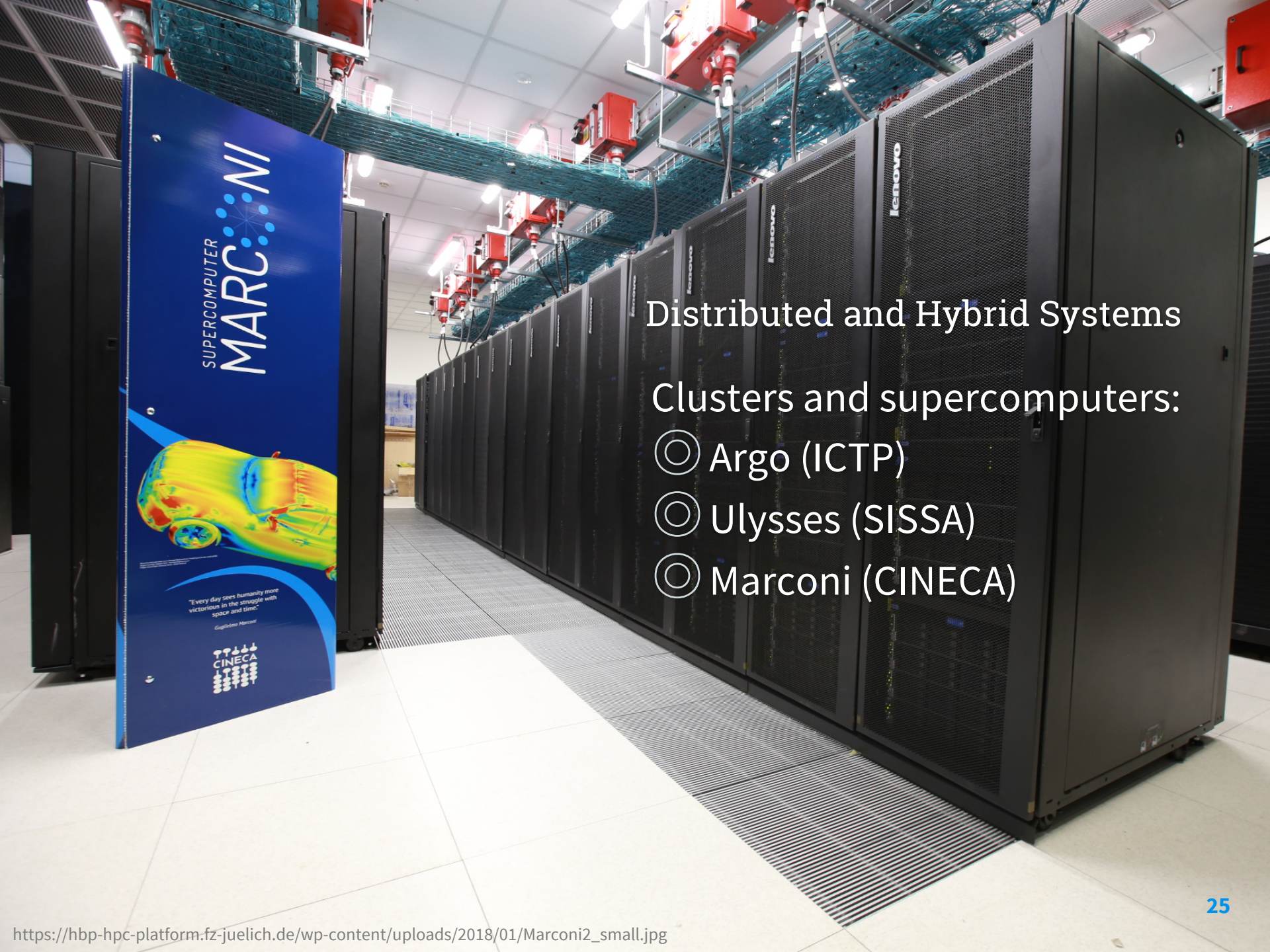


▲ Distributed Memory
Message Passing Interface

Hybrid ▶
OpenMP-MPI,
CUDA



Barney, Blaise. "Introduction to parallel computing." Lawrence Livermore National Laboratory 6.13 (2010): 10. https://computing.llnl.gov/tutorials/parallel_comp/



Distributed and Hybrid Systems

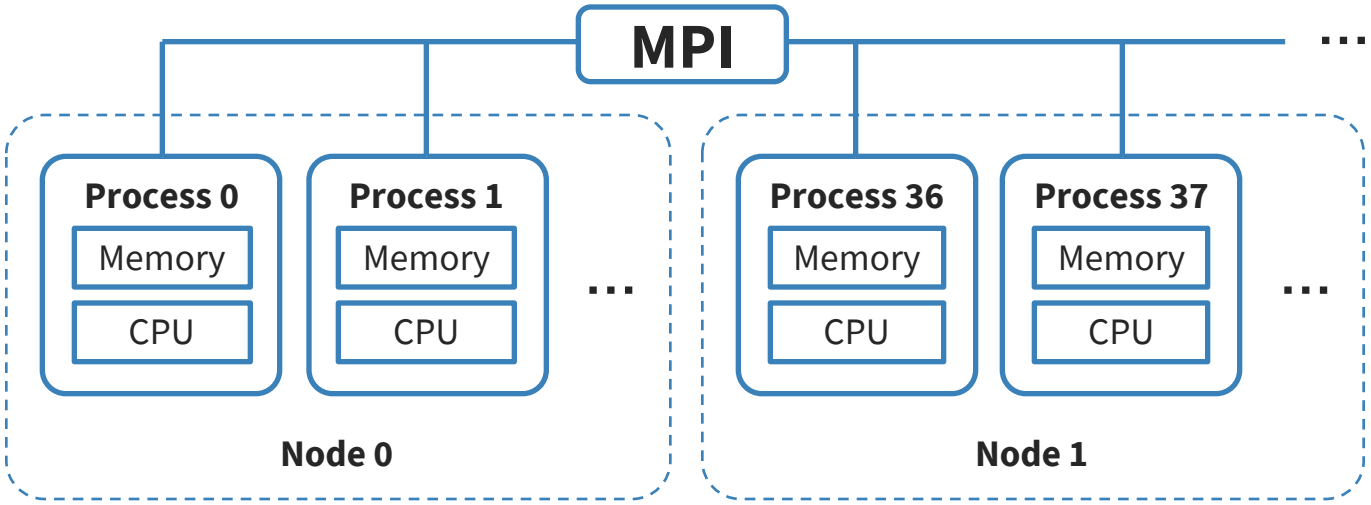
Clusters and supercomputers:

- Argo (ICTP)
- Ulysses (SISSA)
- Marconi (CINECA)

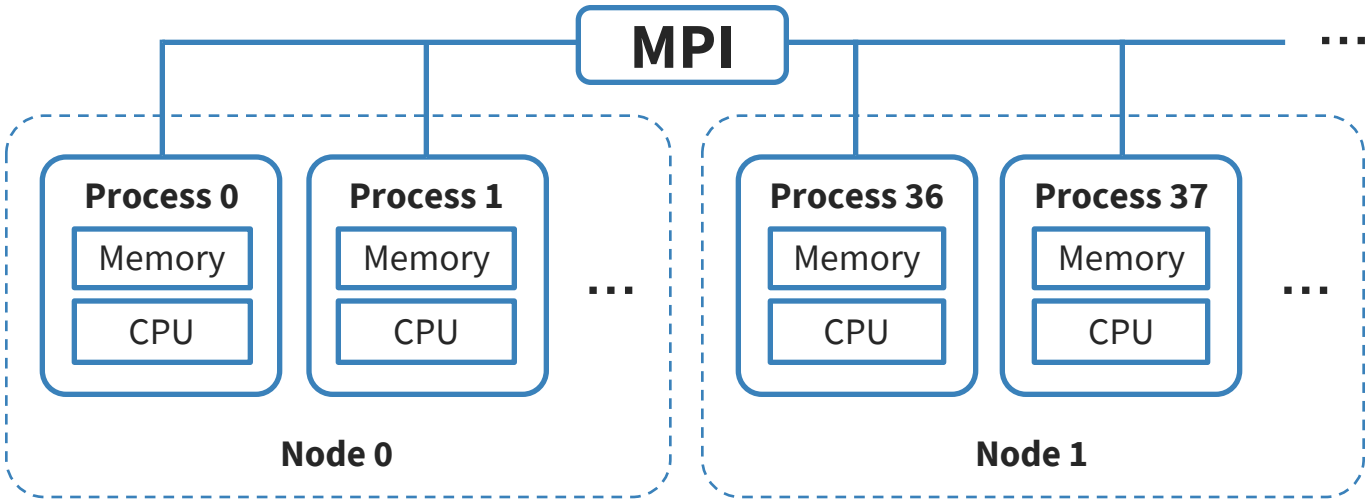
https://hbp-hpc-platform.fz-juelich.de/wp-content/uploads/2018/01/Marconi2_small.jpg

Parallel Programming with **Message Passing Interface** (MPI)

- MPI is the “de facto” industry standard for distributed systems due to its portability, efficiency, and flexibility
- Runs on *distributed*, *shared* and *hybrid* platforms
- Works on a **distributed programming model:**



Parallel Programming with **Message Passing Interface** (MPI)



- Every process has access only to its own data
- MPI facilitates the communication among processes, whether on the same or different nodes

Some MPI Basics

- **MPI communicator** - group of processes that communicate with one another (e.g. `MPI_COMM_WORLD`)
- **rank** – unique id of each process in the communicator
- **size** – total number of processes in the communicator

- **Types of MPI communication:**
  - Point-to-point
  - Collective

Sample MPI Code

01-hello/hello.c

Compiling the code:

```
$ mpicc hello.c -o hello.x
```

Running the program:

```
 $ mpirun -np <NPROCS> <EXEC> <ARGS>
```

e.g.

```
 $ mpirun -np 4 ./hello.x
```

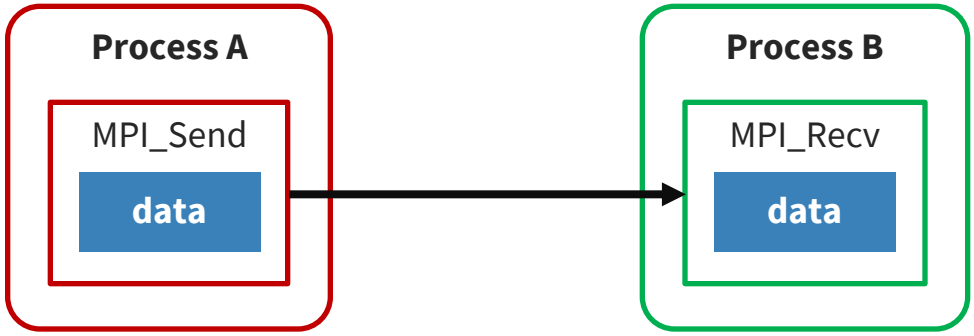
```c
#include<mpi.h>
#include<stdio.h>

int main(int argc, char **argv)
{
    int rank, size;
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    /*---- Body of the MPI program ----*/

    printf("Hello from process %d of %d.\n", rank, size);

    /*--------------------------------*/
    MPI_Finalize();
    return(0);
}
```

Point-to-Point Communication



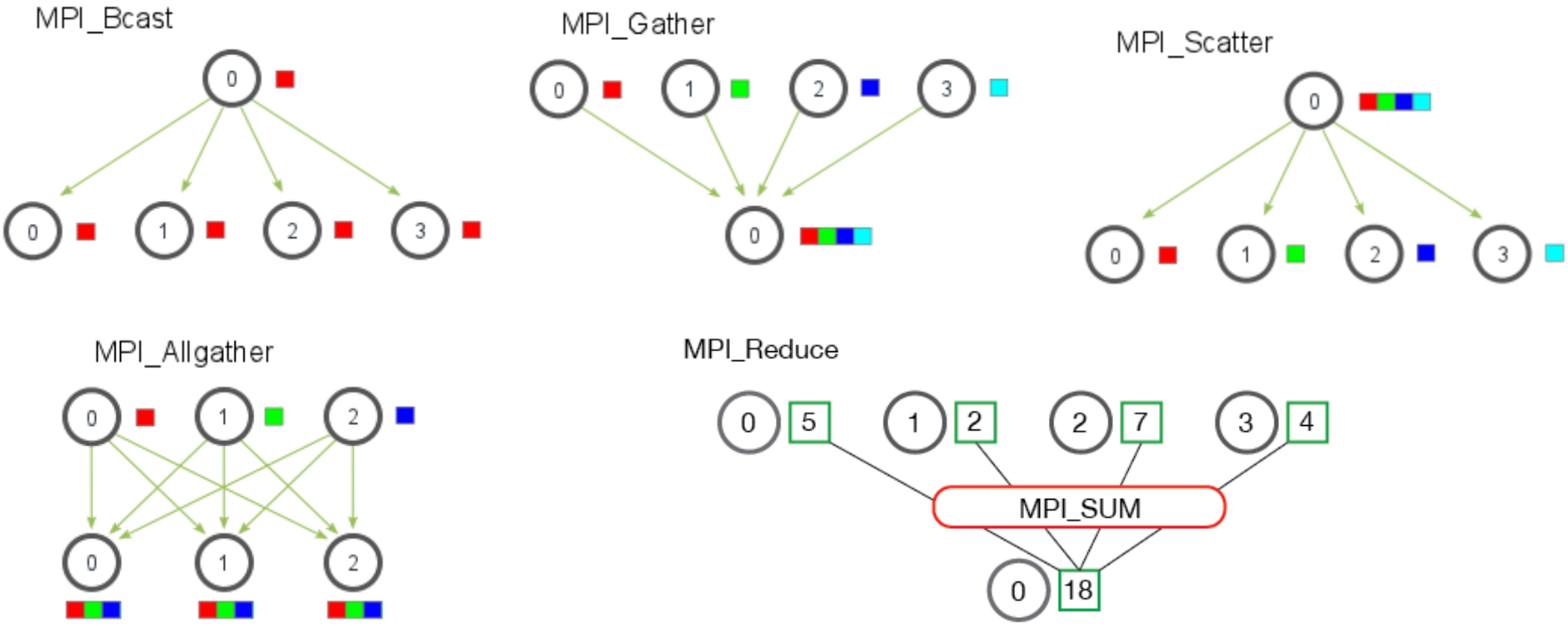

```
MPI_Send( void* data, int count, MPI_Datatype datatype,
          int destination, int tag, MPI_Comm communicator)
MPI_Recv( void* data, int count, MPI_Datatype datatype,
          int source, int tag, MPI_Comm communicator,
          MPI_Status* status)
```

Sample MPI Code

02-send-recv/main.c

```
10      /*---- Body of the MPI program ----*/
11
12      if(size < 2){printf("Needs at least 2 procs.\n");return(1);}
13
14      int number;
15      if (rank == 0) {
16          number = -1;
17          MPI_Send(&number, 1, MPI_INT, 1, 0, MPI_COMM_WORLD);
18      } else if (rank == 1) {
19          MPI_Recv(&number, 1, MPI_INT, 0, 0, MPI_COMM_WORLD,
20                   MPI_STATUS_IGNORE);
21          printf("Process 1 received number %d from process 0\n",
22                 number);
23      }
24
25      /*---------------------------------*/
```

Collective Communication




```
MPI_Reduce( void* send_data, void* recv_data, int count,
            MPI_Datatype datatype, MPI_Op op, int root,
            MPI_Comm communicator)
```


Figures taken from : http://mpitutorial.com/tutorials/mpi-reduce-and-allreduce/

Sample MPI Code

03-reduce/main.c

```
10      /*---- Body of the MPI program ----*/
11      /* Each rank r calculates 2*rank+1 */
12      int rank_val = 2*rank + 1;
13      printf("[%d] Val: %-5d\n", rank, rank_val);
14
15      /* Get the sum of all values from all ranks */
16      int sum;
17      MPI_Reduce(&rank_val, &sum, 1, MPI_INT, MPI_SUM, 0,
18                 MPI_COMM_WORLD);
19      printf("[%d] Sum: %-5d\n", rank, sum);
20
21      /* Assert that sum=size^2 */
22      if(rank==0 && sum!=size*size){
23          printf("Sum should be %d.\n",size*size);
24          return(1);
25      }
26      /*-------------------------------*/
```

Parallel Linear Algebra

- Recall the Lanczos algorithm

$$|\phi_{n+1}\rangle = \hat{H}|\phi_n\rangle - a_n|\phi_n\rangle - b_n^2|\phi_{n-1}\rangle,$$

$$a_n = \frac{\langle\phi_n|\hat{H}|\phi_n\rangle}{\langle\phi_n|\phi_n\rangle}, \qquad b_n^2 = \frac{\langle\phi_n|\phi_n\rangle}{\langle\phi_{n-1}|\phi_{n-1}\rangle},$$

- Large sparse matrix-vector multiplication
- Since the Hamiltonian is very large, we want to parallelize this calculation

Two options:

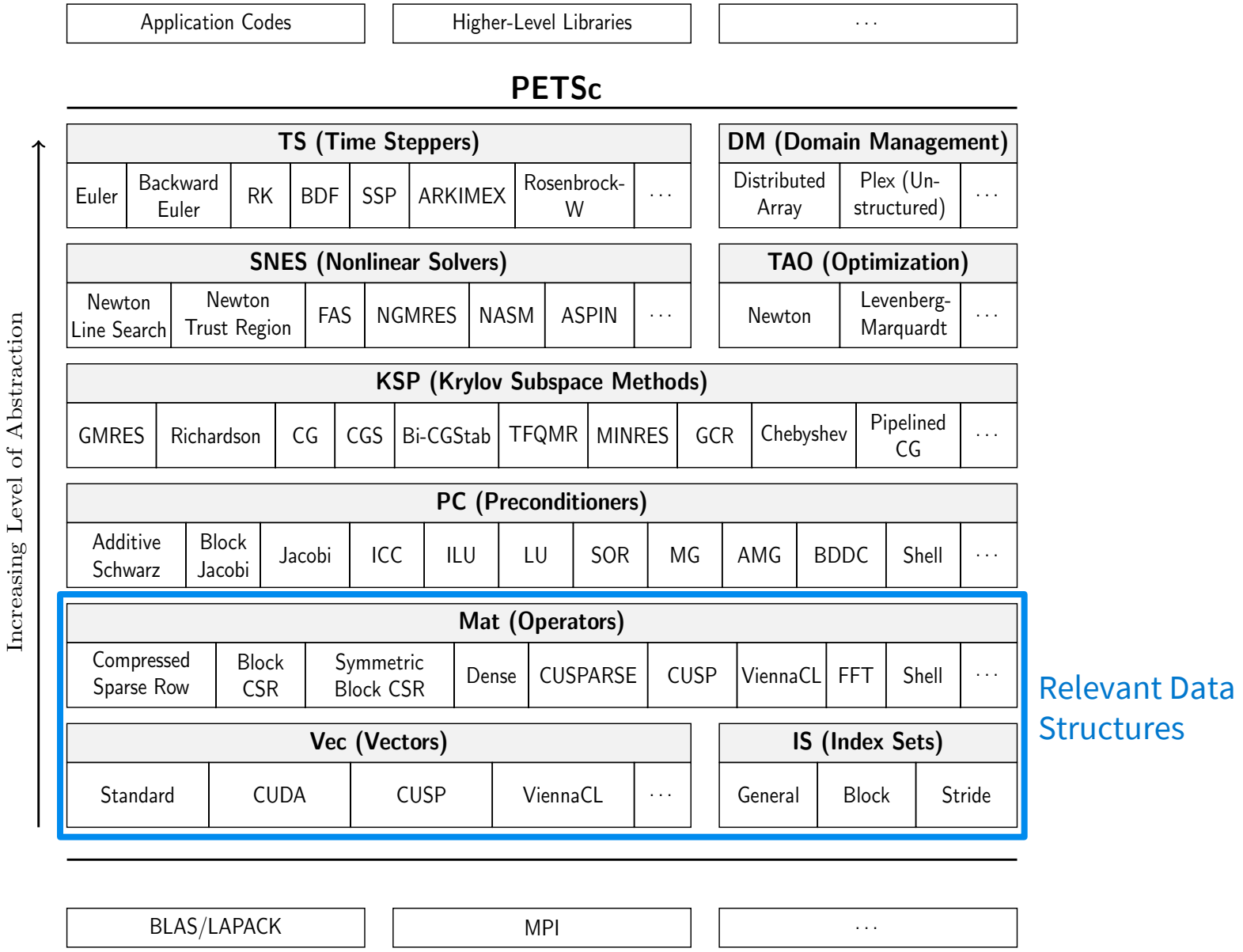
- Write the parallel algorithms from scratch
- Use third-party libraries

PETSc and SLEPc

Overview

The **Portable, Extensible Toolkit for Scientific Computation** (PETSc) is a suite of **data structures and routines** for the scalable (parallel) solution of scientific applications modeled by PDEs

Website: www.mcs.anl.gov/petsc/



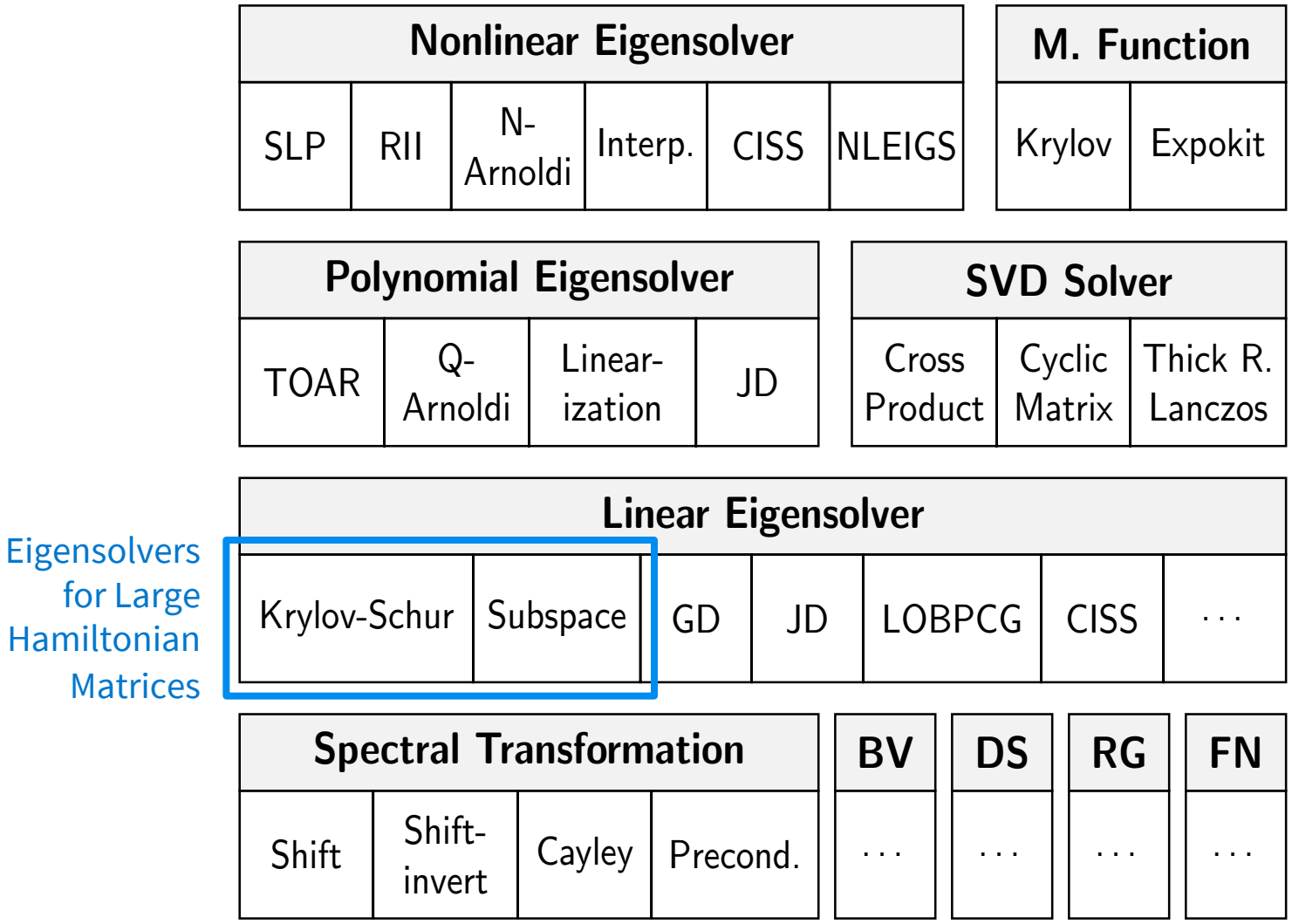
Relevant Data Structures

Numerical libraries of PETSc

The **Scalable Library for Eigenvalue Problem Computations** (SLEPc) is a software library for the solution of large scale **sparse eigenvalue problems** on parallel computers.

Website: slepc.upv.es/

SLEPc



Eigensolvers for Large Hamiltonian Matrices

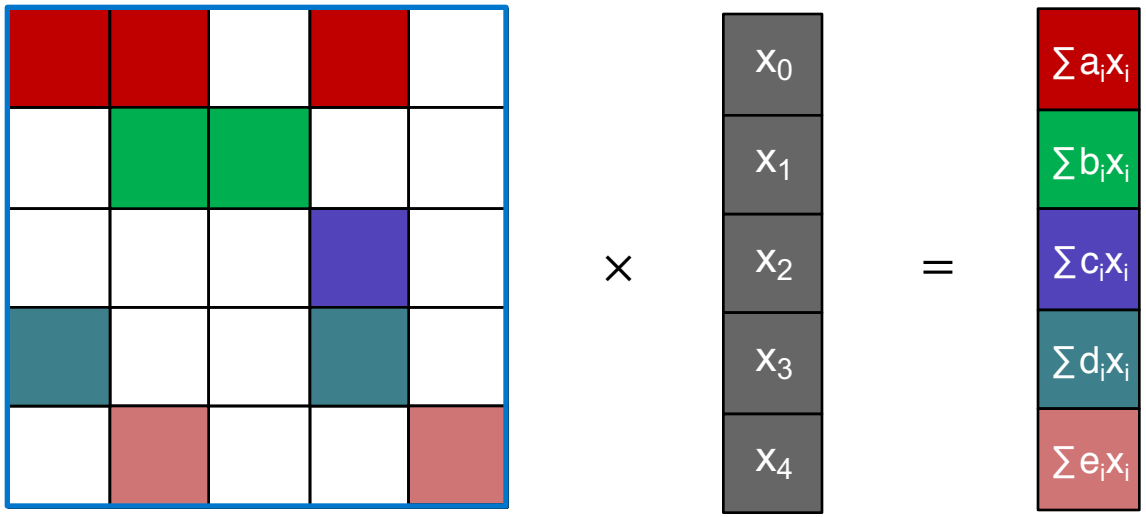
Numerical components of SLEPc

Relevant Features of SLEPc/PETSc

- Implemented **parallel** data structures and solvers
  - **Vectors** (Vec)
  - **Matrices** (Mat) – Operators and Hamiltonian
  - **Eigenvalue Problem Solver** (EPS) – finding the ground state of the superblock Hamiltonian
- Portability
  - **Write code once** for different machines, integer and floating point precision, real/complex scalars
  - Interfaces to BLAS/LAPACK routines
- Parallel Efficiency
  - Implemented algorithms are already tried and tested for efficiency (no need to code from scratch)

Sparse Matrix-Vector Multiplication

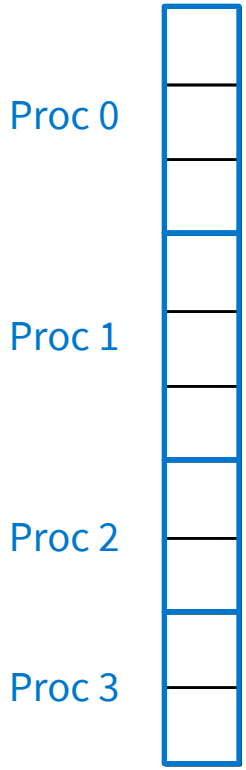
Logical layout:



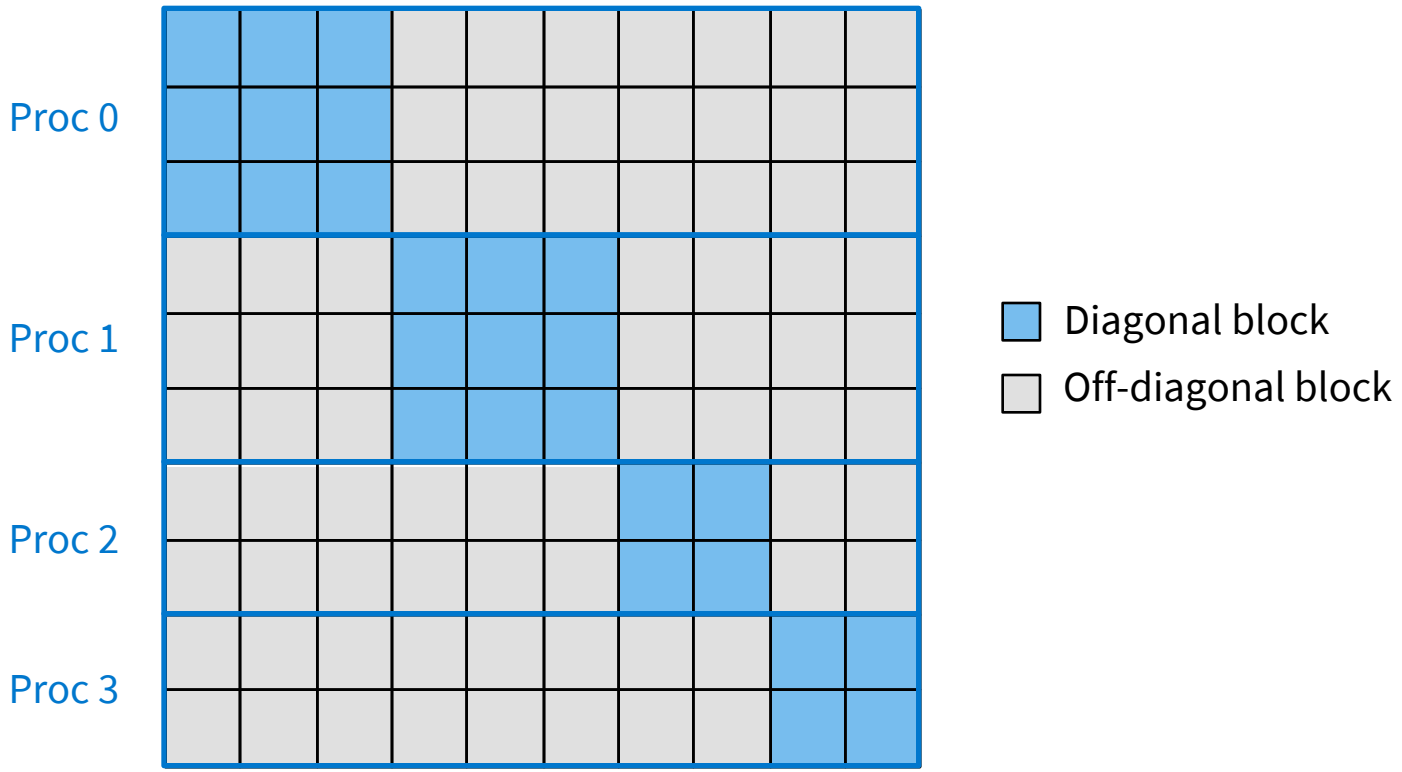
Compressed sparse row (CSR):

Parallel Linear Algebra with PETSc

- Distributed Vectors

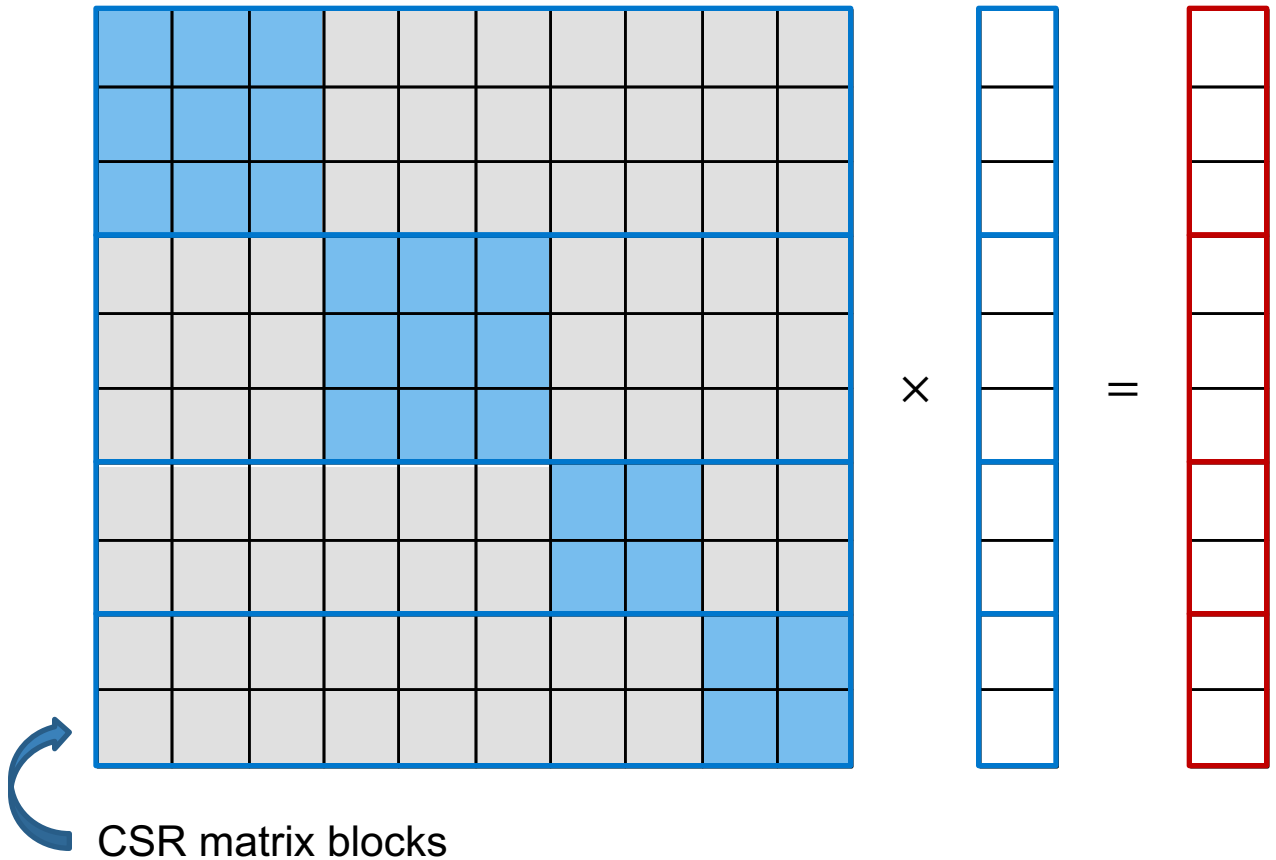


Row-Distributed Matrices

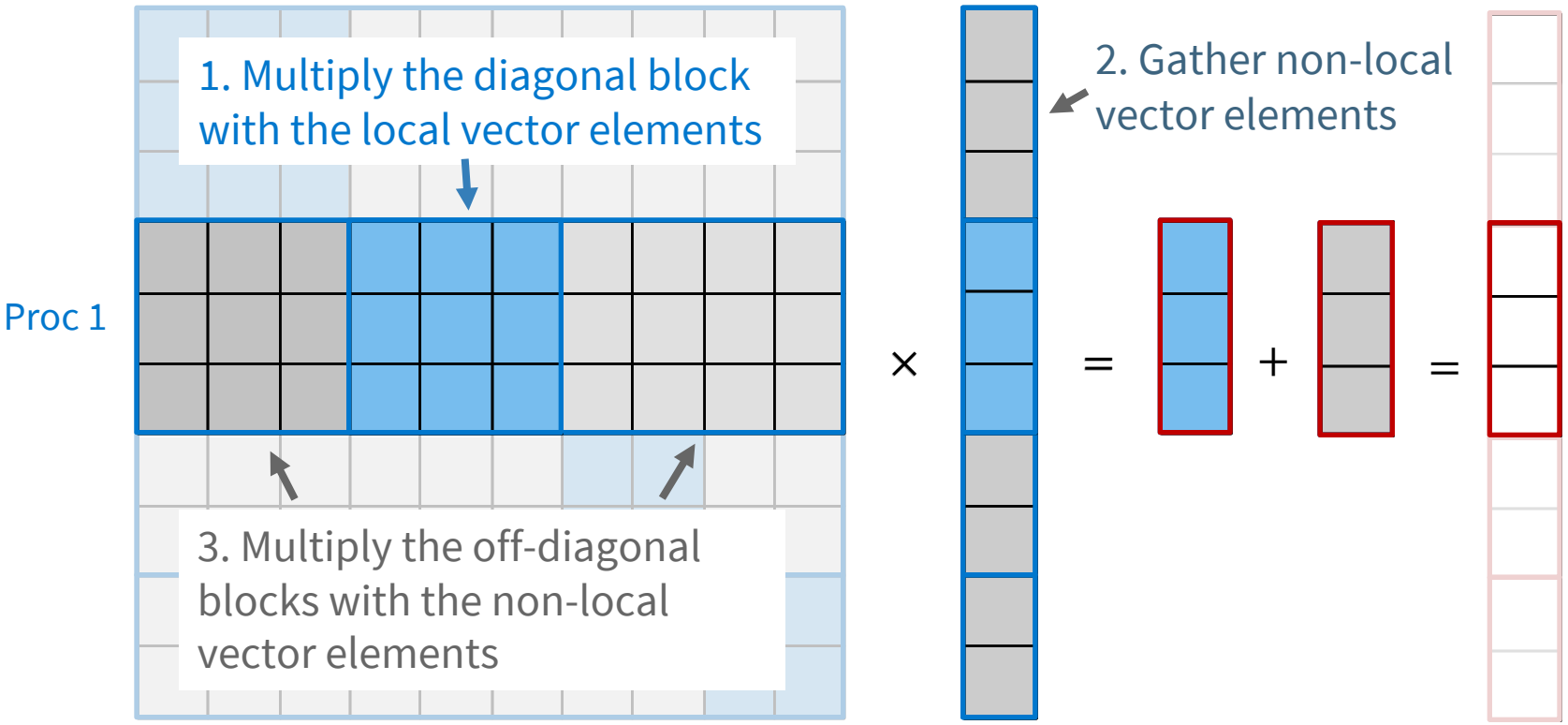


Parallel Linear Algebra with PETSc

- Parallel Sparse Matrix-Vector Multiplication



Parallel Sparse Matrix-Vector Multiplication



Finding the Ground State with SLEPc

- Iterative solution using Krylov subspace methods

$$\mathcal{K}_m(A, v) \equiv \text{span}\left\{v, Av, A^2v, \ldots, A^{m-1}v\right\}$$

- Built-in solvers:
  - Arnoldi
  - Lanczos
  - Krylov-Schur
  - etc.

For Hermitian matrices, it is equivalent to the **thick-restarted Lanczos algorithm** (keeping vectors orthogonal to machine ϵ)

Roman, Jose E., et al. "SLEPc users manual." D. Sistemes Informatics i Computació, Universitat Politecnica de Valencia, Tech. Rep. DSIC-II/24/02-Revision 3 (2015).

Installing the libraries

- A step-by-step tutorial can be found at:

  http://dmrgx.rtfd.io

  under **Related Pages** > **Installation**

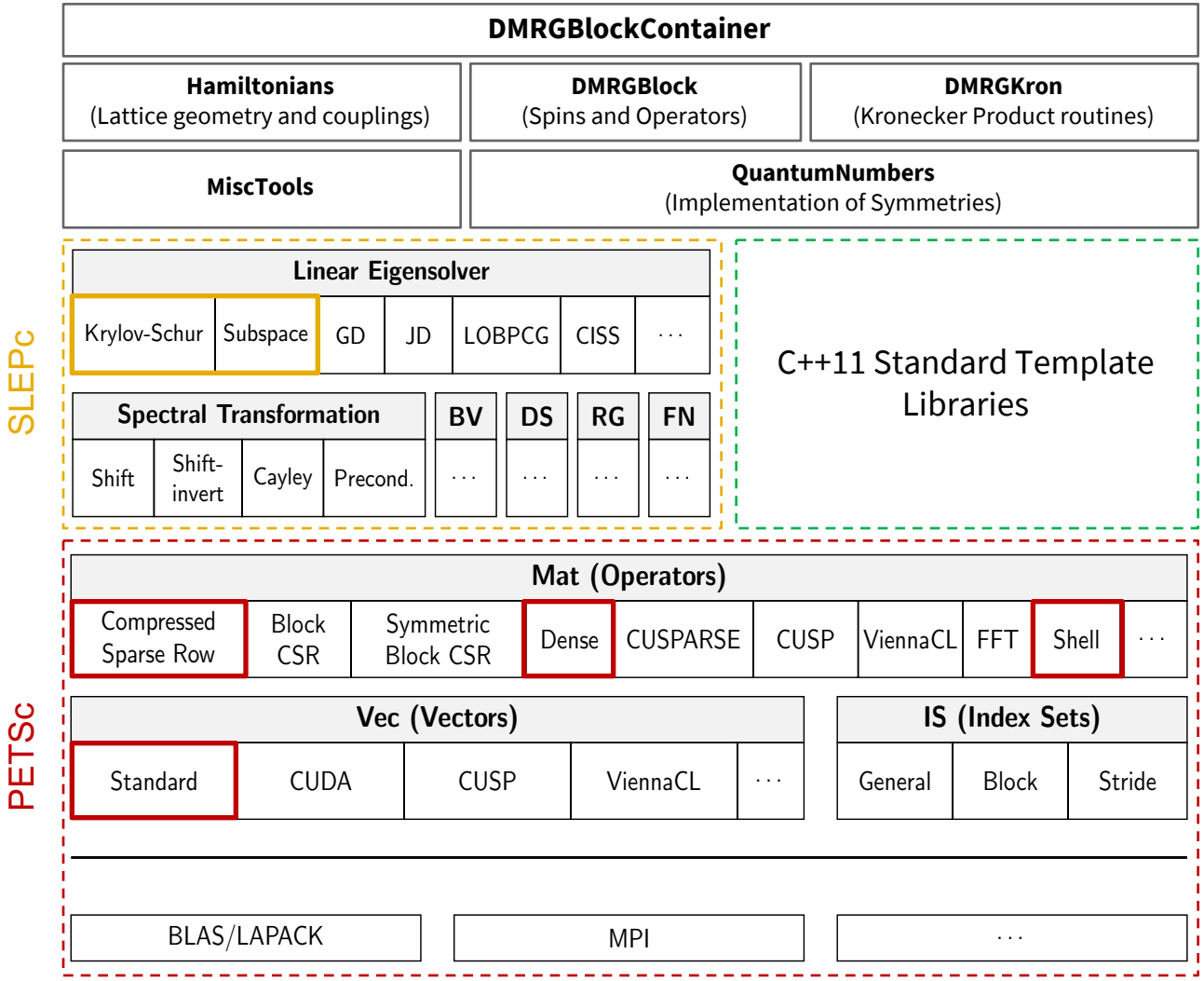
PETSc/SLEPc Exercises

- Hello World
- SLEPc Hands-on 1

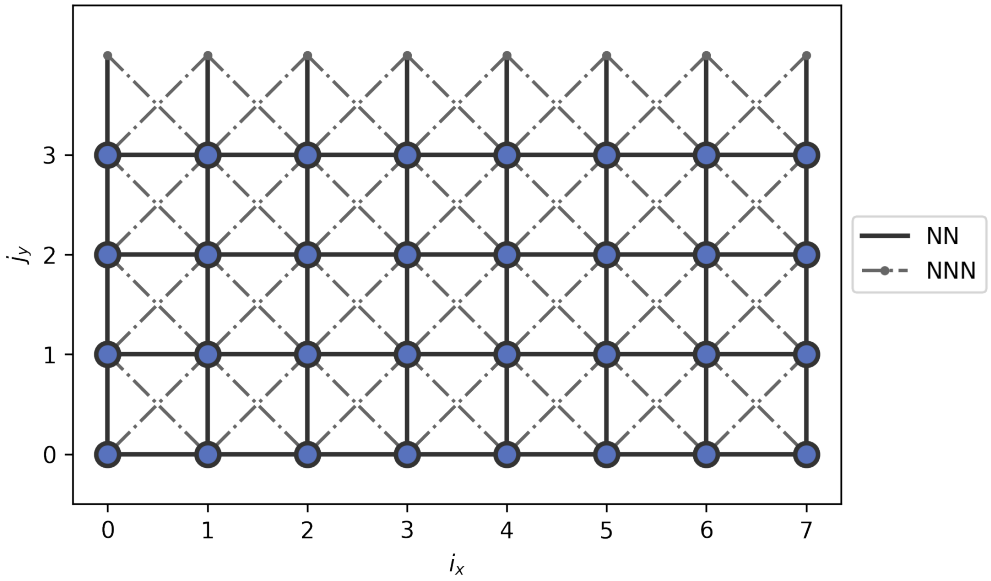
see SampleCode/02-petsc-slepc

DMRG.x

DMRG.x Modules and Libraries



Hamiltonian and Geometry



$$H = \sum_{\langle ij \rangle} \left(J_1 \left[\hat{S}_i^+ \hat{S}_j^- + \hat{S}_i^- \hat{S}_j^+ \right] + \Delta_1 \hat{S}_i^z \hat{S}_j^z \right)$$
$$+ \sum_{\langle\langle kl \rangle\rangle} \left(J_2 \left[\hat{S}_k^+ \hat{S}_l^- + \hat{S}_k^- \hat{S}_l^+ \right] + \Delta_2 \hat{S}_k^z \hat{S}_l^z \right)$$

2D-DMRG

Using a one-dimensional traversal

see 2D-DMRG.ppt

Implementation of U(1) Symmetries

◎ Since the Hamiltonian conserves total magnetization, we can look for the ground state at a specific total magnetization sector.

◎ The sectors are stored as lookup keys that map to a range of indices containing that array

$$S_j^z \rightarrow [i_{\text{start}}, i_{\text{end}} + 1)$$

◎ Then the Hamiltonian is built using sectors that satisfy

$$S_{j,A}^z + S_{k,B}^z = S_{\text{tot}}^z$$

Usage

- A step-by-step tutorial can be found at:

  http://dmrgx.rtfd.io

  under **Related Pages** > **Usage**

Post-Processing

Post-processing module written in Python

- DMRG.x/postproc/dmrg_postprocessing.py

Recommended interface: **jupyter notebook**

Credits

Special thanks to all the people who made and released these awesome resources for free:

- Presentation template by SlidesCarnival
- Photographs by Unsplash & Death to the Stock Photo (license)

More info on how to use this template at www.slidescarnival.com/help-use-presentation-template