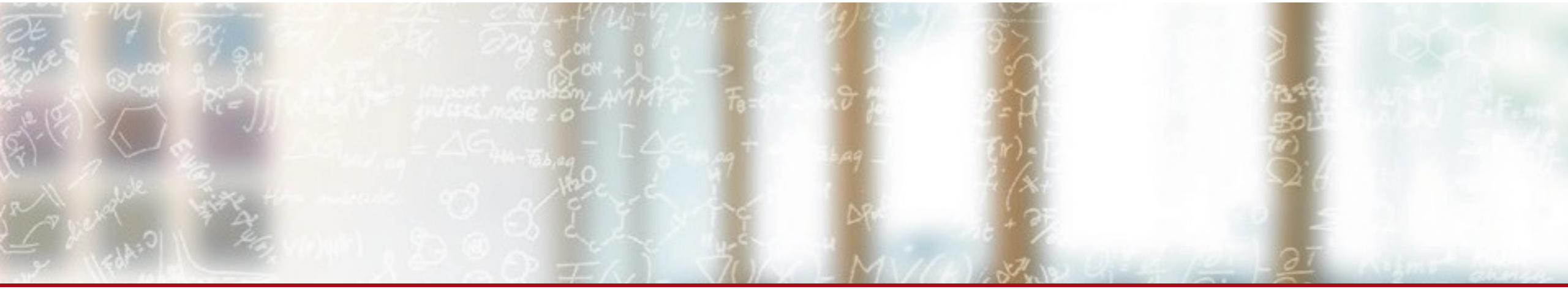
CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich



GPU acceleration of plane-wave codes using SIRIUS library

MATERIAL SCIENCE CODES ON INNOVATIVE HPC ARCHITECHTURES: TARGETING EXASCALE

Anton Kozhevnikov, CSCS

December 05, 2017

CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich

Introduction

Hybrid supercomputers at CSCS

Tödi
AMD Opteron + NVIDIA K20X
393 Teraflops

October 2011

Hybrid supercomputers at CSCS

Tödi
AMD Opteron + NVIDIA K20X
393 Teraflops

Piz Daint
Intel Sandy Bridge + NVIDIA K20X
7.787 Petaflops

October 2011

Novembre 2013

Hybrid supercomputers at CSCS

Tödi
AMD Opteron + NVIDIA K20X
393 Teraflops

Piz Daint
Intel Sandy Bridge + NVIDIA K20X
7.787 Petaflops

Piz Daint
Intel Haswell + NVIDIA P100
25.326 Petaflops

October 2011

Novembre 2013

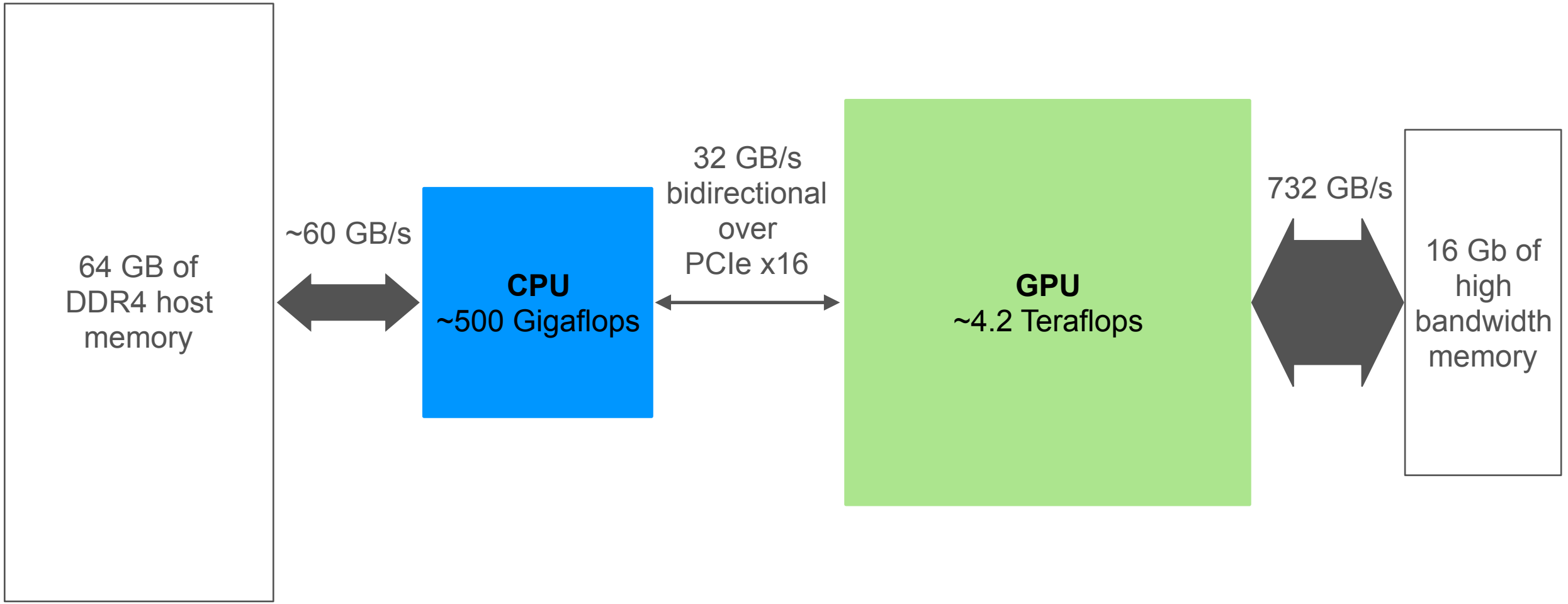
Novembre 2016

Piz Daint: #3 supercomputer in the world

Cray XC50, 5320 nodes

Intel Xeon E5-2690v3 12C, 2.6GHz, 64GB + NVIDIA Tesla P100 16GB
4.761 Teraflops / node

Piz Daint node layout



Porting codes to GPUs

No magic “silver bullet” exists!

Porting codes to GPUs

No magic “silver bullet” exists!

Usual steps in porting codes to GPUs

Porting codes to GPUs

No magic “silver bullet” exists!

Usual steps in porting codes to GPUs

- cleanup and refactor the code
- (possibly) change the data layout
- fully utilize CPU threads and prepare code for node-level parallelization
- move compute-intensive kernels to GPUs

Porting codes to GPUs

- CUDA (C / C++ / Fortran)

```
__global__ void add_pw_ekin_gpu_kernel(int num_gvec__,
                                       double alpha__,
                                       double const* pw_ekin__,
                                       cuDoubleComplex const* phi__,
                                       cuDoubleComplex const* vphi__,
                                       cuDoubleComplex* hphi__)
{
    int ig = blockIdx.x * blockDim.x + threadIdx.x;
    if (ig < num_gvec__) {
        cuDoubleComplex z1 = cuCadd(vphi__[ig], make_cuDoubleComplex(alpha__ * pw_ekin__[ig] * phi__[ig].x,
                                                                     alpha__ * pw_ekin__[ig] * phi__[ig].y));
        hphi__[ig] = cuCadd(hphi__[ig], z1);
    }
}
```

- OpenCL

```
__kernel void vector_add(const int n, __global float *a, __global float *b, __global float *c) {
  const int i = get_global_id(0);
  if (i < n) {
    c[i] = a[i] + b[i];
  }
}
```

- OpenACC

```
      acc = 0
      !$acc parallel present(x)
      !$acc loop reduction(+:acc)
      do i = 1, N
          acc = acc + x(i) * x(i)
      enddo
      !$acc end parallel
      call mpi_allreduce(acc, accglobal, 1, MPI_DOUBLE, MPI_SUM, MPI_COMM_WORLD, err)
```

- OpenMP 4.0

```
#pragma omp target data map(tofrom: x[0:n],y[0:n])
{
    #pragma omp target
    #pragma omp for
    for (int i = 0; i < n; i++)
        y[i] += a * x[i];
}
```

Porting codes to GPUs

- CUDA (C / C++ / Fortran)

```
__global__ void add_pw_ekin_gpu_kernel(int num_gvec__,
                                       double alpha__,
                                       double const* pw_ekin__,
                                       cuDoubleComplex const* p
                                       cuDoubleComplex const* v
                                       cuDoubleComplex* hphi__)
{
    int ig = blockIdx.x * blockDim.x + threadIdx.x;
    if (ig < num_gvec__) {
        cuDoubleComplex z1 = cuCadd(vphi__[ig], make_cuDoubleCo

        hphi__[ig] = cuCadd(hphi__[ig], z1);
    }
}
```

- OpenCL

```
__kernel void vector_add(const int n, __global float *a, __global float *b, __global float *c) {
    id(0);
```

- OpenACC

```
      acc = 0
      !$acc parallel present(x)
      !$acc loop reduction(+:acc)
      do i = 1, N
          acc = acc + x(i) * x(i)
      enddo
      !$acc end parallel
      call mpi_allreduce(acc, accglobal, 1, MPI_DOUBLE, MPI_SUM, MPI_COMM_WORLD, err)
```

4.0

```
arget data map(tofrom: x[0:n],y[0:n])

mp target
#pragma omp for
for (int i = 0; i < n; i++)
    y[i] += a * x[i];
}
```

Electronic-structure codes

Electronic structure codes

Basis functions for KS states / Atomic potential treatment	Periodic Bloch functions (plane-waves or similar)	Localized orbitals
Full-potential	FLEUR Wien2K Exciting Elk	FHI-aims FPLO
Pseudo-potential	VASP CPMD Quantum ESPRESSO Abinit Qbox	CP2K SIESTA OpenMX

Atomic total energies with LAPW

	Hydrogen	Helium	Lithium	Beryllium	Boron	Carbon	Nirogen	Oxygen	Fluorine	Neon
NIST	-0.478671	-2.834836	-7.343957	-14.447209			-54.136799			-128.233481
LAPW	-0.478671	-2.834835	-7.343958	-14.447209	-24.356062	-37.470324	-54.136792	-74.531333	-99.114324	-128.233477
MADNESS	-0.478671	-2.834836	-7.343957	-14.447209	-24.356065	-37.470329	-54.136798	-74.531345	-99.114902	-128.233481
NWCHEM					-24.356064	-37.470328	-54.136799	-74.531344	-99.114901	
	$1s^1$	$1s^2$	$2s^1$	$2s^2$	$2s^2p^1$	$2s^2p^2$	$2s^2p^3$	$2s^2p^4$	$2s^2p^5$	$2s^2p^6$

Linearized augmented plane-wave method (LAPW):

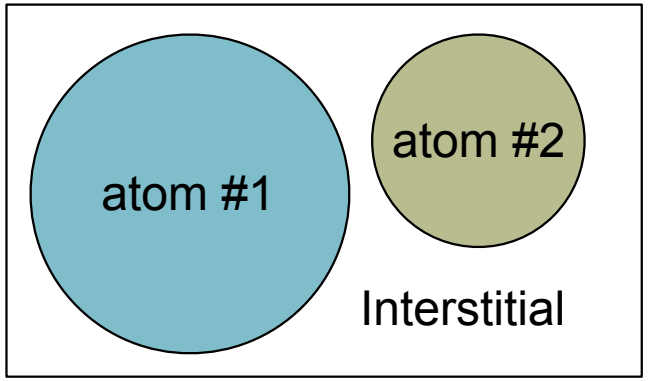
- provides a very high accuracy of the DFT total energy
- designed for crystalline solids
- considered as a gold standard for electronic structure simulations

Delta DFT codes effort

Code	Version	Basis	Electron treatment	Δ-value	Authors
WIEN2k	13.1	LAPW/APW+lo	all-electron	0 meV/atom	S. Cottenier [16]
FHI-aims	081213	tier2 numerical orbitals	all-electron (relativistic atomic_zora scalar)	0.2 meV/atom	ASE [2,16]
Exciting	development version	LAPW+xlo	all-electron	0.2 meV/atom	Exciting [10,16]
Elk	3.1.5	APW+lo	all-electron	0.3 meV/atom	Elk [14,16]
Quantum ESPRESSO	5.1	plane waves	SSSP Accuracy (mixed NC/US/PAW potential library)	0.3 meV/atom	QuantumESPRESSO [12,16]
FHI-aims	081213	tier2 numerical orbitals	all-electron (relativistic zora scalar 1e-12)	0.3 meV/atom	ASE [2]
VASP	5.2.12	plane waves	PAW 2015 GW-ready (5.4)	0.3 meV/atom	K. Lejaeghere [16]
ABINIT	7.8.2	plane waves	PAW JTH v1.0	0.4 meV/atom	F. Jollet and M. Torrent
FLEUR	0.26	LAPW (+lo)	all-electron	0.4 meV/atom	FLEUR [9,16]

Full-potential linearized augmented plane-wave method

- Unit cell is partitioned into “muffin-tin” spheres and interstitial region
- Inside MT spheres spherical harmonic expansion is used
- In the interstitial region functions are expanded in plane-waves

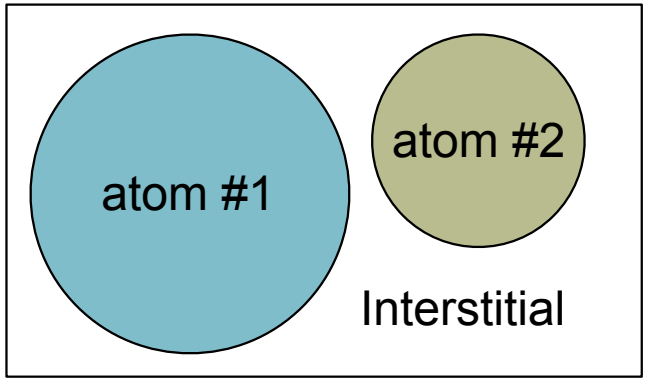


Full-potential linearized augmented plane-wave method

- Unit cell is partitioned into “muffin-tin” spheres and interstitial region
- Inside MT spheres spherical harmonic expansion is used
- In the interstitial region functions are expanded in plane-waves

Basis functions:

$$\varphi_{\mathbf{G}+\mathbf{k}}(\mathbf{r}) = \begin{cases} \sum_{\ell m} \sum_{\nu=1}^{O_\ell^\alpha} A_{\ell m \nu}^\alpha(\mathbf{G}+\mathbf{k}) u_{\ell\nu}^\alpha(r) Y_{\ell m}(\hat{\mathbf{r}}) & \mathbf{r} \in \mathrm{MT}\alpha \\ \frac{1}{\sqrt{\Omega}} e^{i(\mathbf{G}+\mathbf{k})\mathbf{r}} & \mathbf{r} \in \mathrm{I} \end{cases}$$

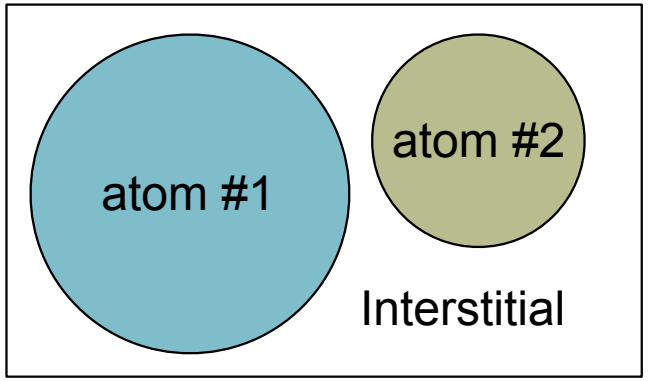


Full-potential linearized augmented plane-wave method

- Unit cell is partitioned into “muffin-tin” spheres and interstitial region
- Inside MT spheres spherical harmonic expansion is used
- In the interstitial region functions are expanded in plane-waves

Basis functions:

$$\varphi_{\mathbf{G}+\mathbf{k}}(\mathbf{r}) = \begin{cases} \sum_{\ell m} \sum_{\nu=1}^{O_\ell^\alpha} A_{\ell m \nu}^\alpha(\mathbf{G}+\mathbf{k}) u_{\ell\nu}^\alpha(r) Y_{\ell m}(\hat{\mathbf{r}}) & \mathbf{r} \in \mathrm{MT}\alpha \\ \frac{1}{\sqrt{\Omega}} e^{i(\mathbf{G}+\mathbf{k})\mathbf{r}} & \mathbf{r} \in \mathrm{I} \end{cases}$$



Potential and density:

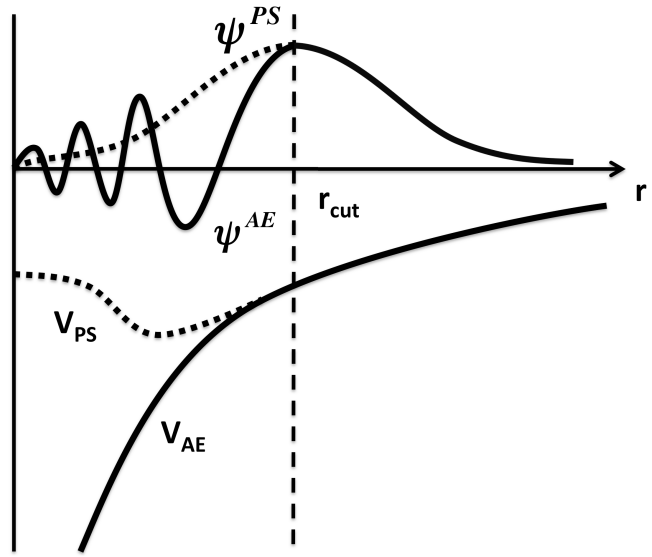
$$V(\mathbf{r}) = \begin{cases} \sum_{\ell m} V_{\ell m}^\alpha(r) Y_{\ell m}(\hat{\mathbf{r}}) & \mathbf{r} \in \mathrm{MT}\alpha \\ \sum_{\mathbf{G}} V(\mathbf{G}) e^{i\mathbf{G}\mathbf{r}} & \mathbf{r} \in \mathrm{I} \end{cases} \qquad \rho(\mathbf{r}) = \begin{cases} \sum_{\ell m} \rho_{\ell m}^\alpha(r) Y_{\ell m}(\hat{\mathbf{r}}) & \mathbf{r} \in \mathrm{MT}\alpha \\ \sum_{\mathbf{G}} \rho(\mathbf{G}) e^{i\mathbf{G}\mathbf{r}} & \mathbf{r} \in \mathrm{I} \end{cases}$$

Full-potential linearized augmented plane-wave method

- No approximation to atomic potential
- Core states are included
- Number of basis functions: ~100 / atom
- Number of valence states: ~15-20% of the total basis size
- Large condition number of the overlap matrix
- Full diagonalization of dense matrix is required (iterative subspace diagonalization schemes are not efficient)
- Atomic forces can be easily computed
- Stress tensor can’t be easily computed (N-point numerical scheme is required)

Pseudopotential plane-wave method

- Unit cell is mapped to a regular grid
- All functions are expanded in plane-waves
- Atomic potential is replaced by a pseudopotential $\hat{V}_{\mathrm{PS}} = V_{loc}(\mathbf{r}) + \sum_{\alpha} \sum_{\xi\xi'} |\beta_{\xi}^{\alpha}\rangle D_{\xi\xi'}^{\alpha} \langle\beta_{\xi'}^{\alpha}|$

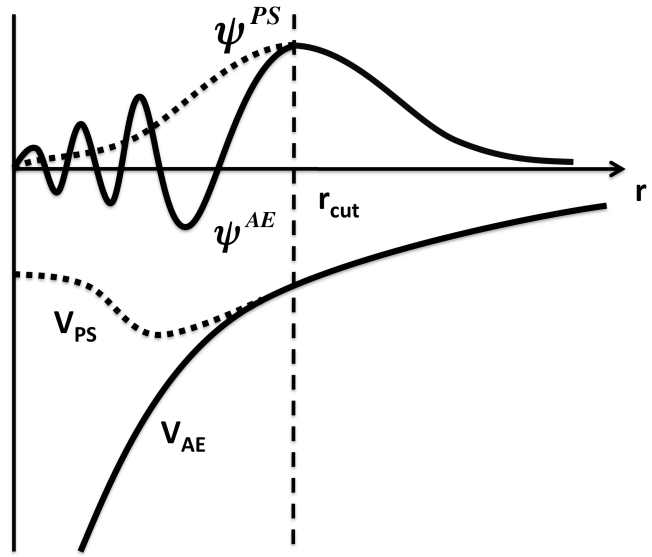


Pseudopotential plane-wave method

- Unit cell is mapped to a regular grid
- All functions are expanded in plane-waves
- Atomic potential is replaced by a pseudopotential $\hat{V}_{\mathrm{PS}} = V_{loc}(\mathbf{r}) + \sum_{\alpha} \sum_{\xi\xi'} |\beta_{\xi}^{\alpha}\rangle D_{\xi\xi'}^{\alpha} \langle\beta_{\xi'}^{\alpha}|$

Basis functions:

$$\varphi_{\mathbf{G}+\mathbf{k}}(\mathbf{r}) = \frac{1}{\sqrt{\Omega}} e^{i(\mathbf{G}+\mathbf{k})\mathbf{r}}$$

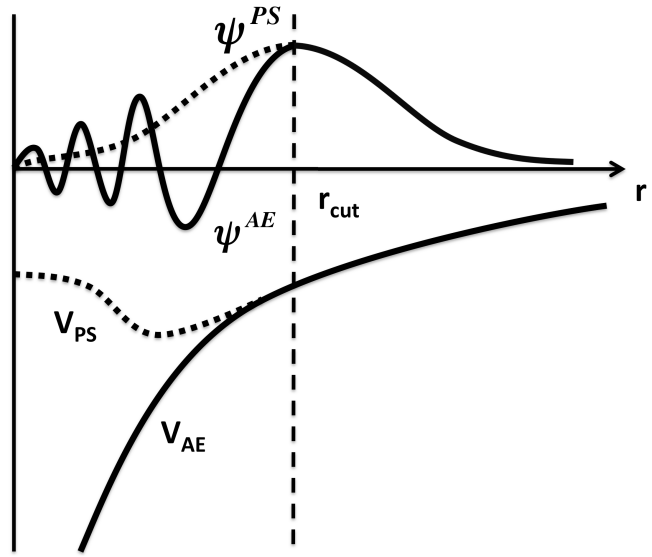


Pseudopotential plane-wave method

- Unit cell is mapped to a regular grid
- All functions are expanded in plane-waves
- Atomic potential is replaced by a pseudopotential $\hat{V}_{\mathrm{PS}} = V_{loc}(\mathbf{r}) + \sum_{\alpha} \sum_{\xi\xi'} |\beta_{\xi}^{\alpha}\rangle D_{\xi\xi'}^{\alpha} \langle\beta_{\xi'}^{\alpha}|$

Basis functions:

$$\varphi_{\mathbf{G}+\mathbf{k}}(\mathbf{r}) = \frac{1}{\sqrt{\Omega}} e^{i(\mathbf{G}+\mathbf{k})\mathbf{r}}$$



Potential and density:

$$V(\mathbf{r}) = \sum_{\mathbf{G}} V(\mathbf{G}) e^{i\mathbf{G}\mathbf{r}} \qquad \rho(\mathbf{r}) = \sum_{\mathbf{G}} \rho(\mathbf{G}) e^{i\mathbf{G}\mathbf{r}}$$

Pseudopotential plane-wave method

- Approximation to atomic potential
- Core states are excluded
- Number of basis functions: ~1000 / atom
- Number of valence states: ~0.001 - 0.01% of the total basis size
- Efficient iterative subspace diagonalization schemes exist
- Atomic forces can be easily computed
- Stress tensor can be easily computed

Common features of the FP-LAPW and PP-PW methods

- Definition of the unit cell (atoms, atom types, lattice vectors, symmetry operations, etc.)
- Definition of the reciprocal lattice, plane-wave cutoffs, **G** vectors, **G+k** vectors
- Definition of the wave-functions
- FFT driver
- Generation of the charge density on the regular grid
- Generation of the XC-potential
- Symmetrization of the density, potential and occupancy matrices
- Low-level numerics (spherical harmonics, Bessel functions, Gaunt coefficients, spline interpolation, Wigner D-matrix, linear algebra wrappers, etc.)

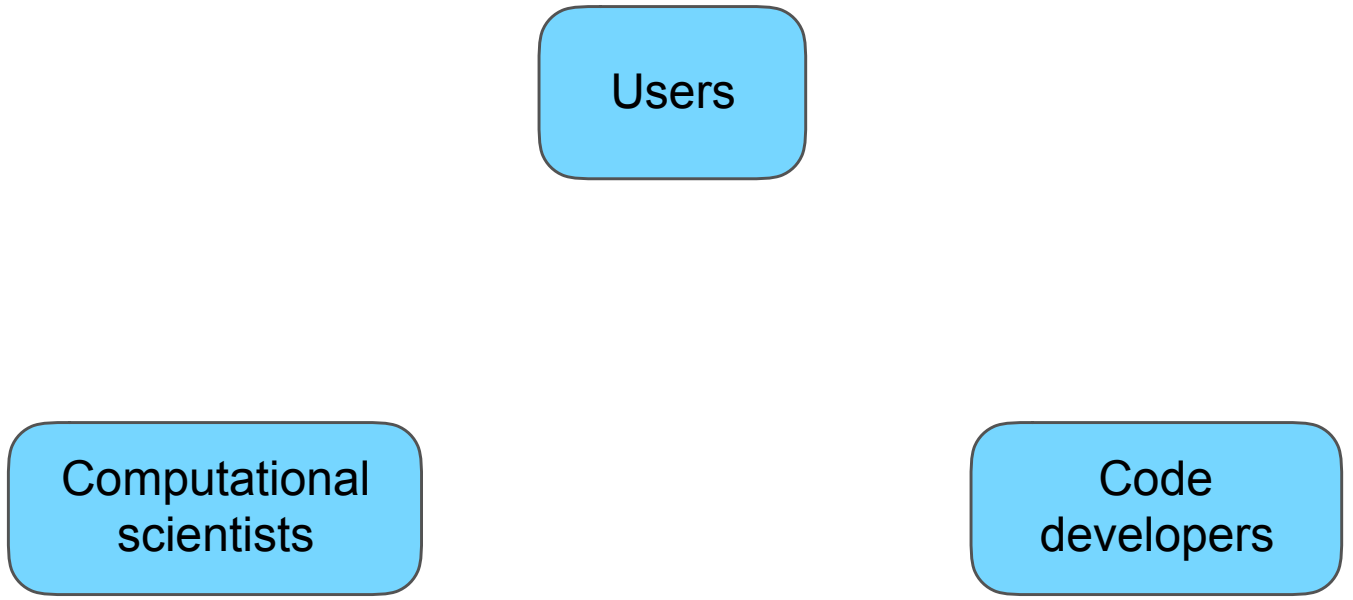
CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich

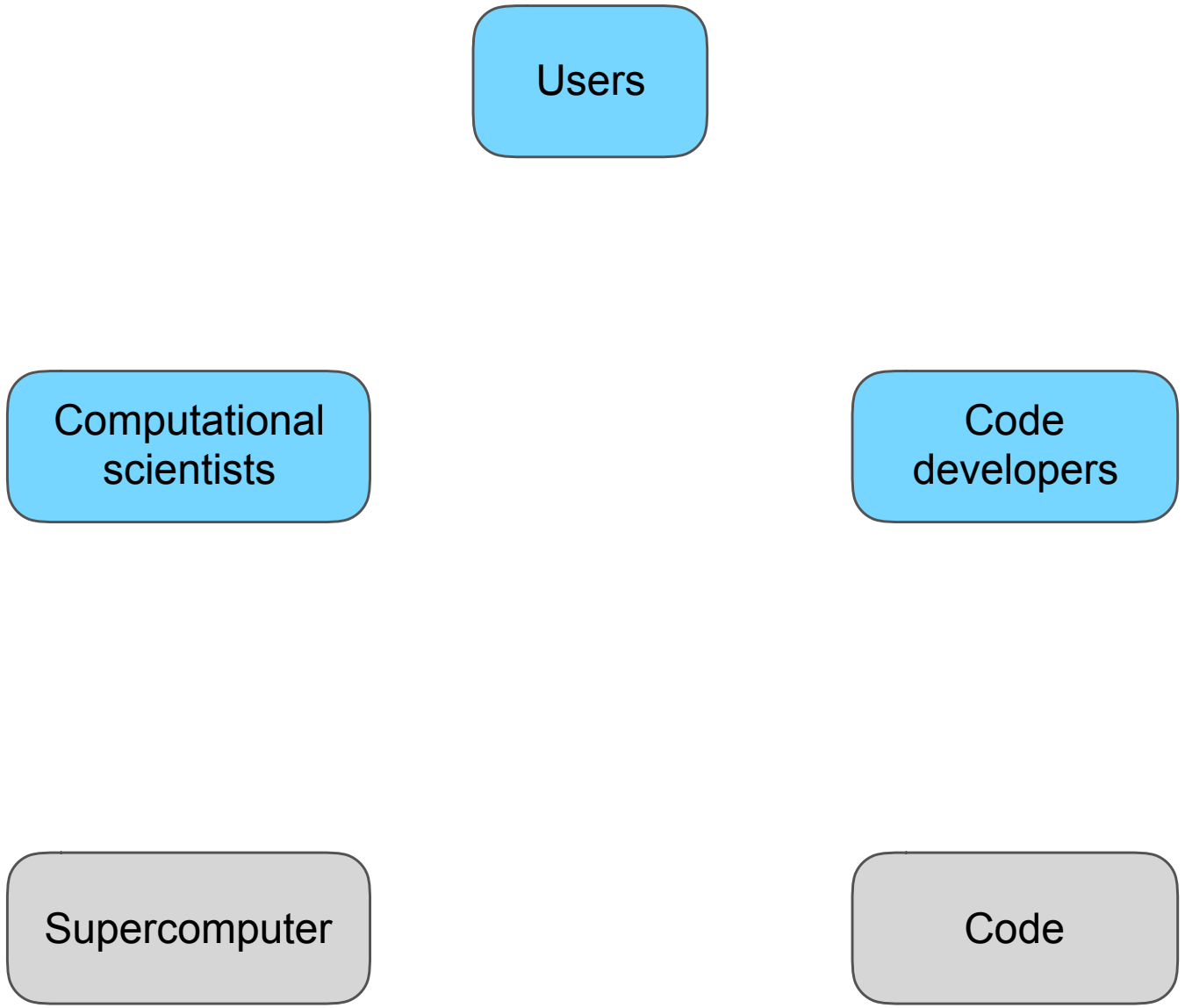
SIRIUS library

Motivation for a common domain specific library

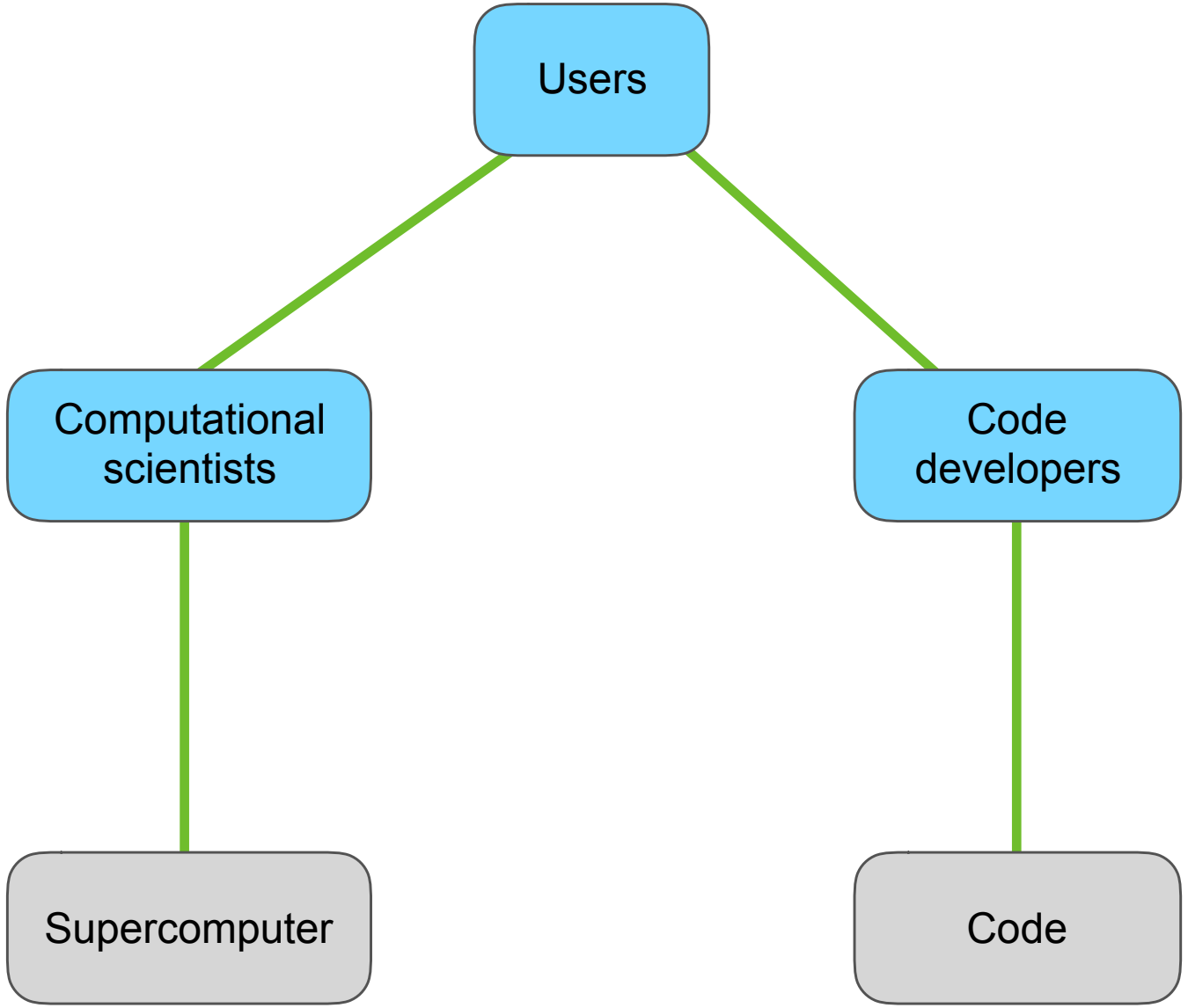
Motivation for a common domain specific library



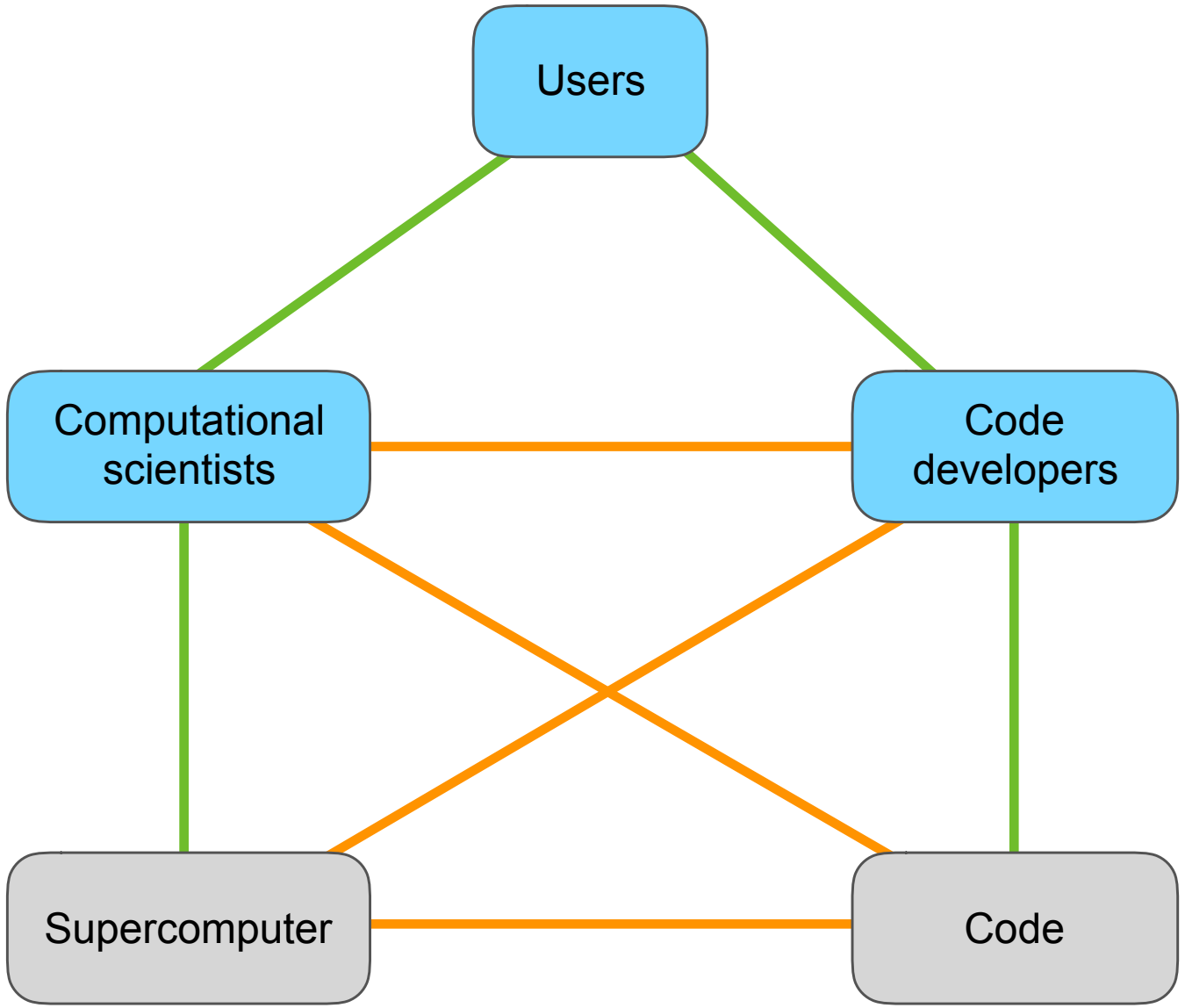
Motivation for a common domain specific library



Motivation for a common domain specific library

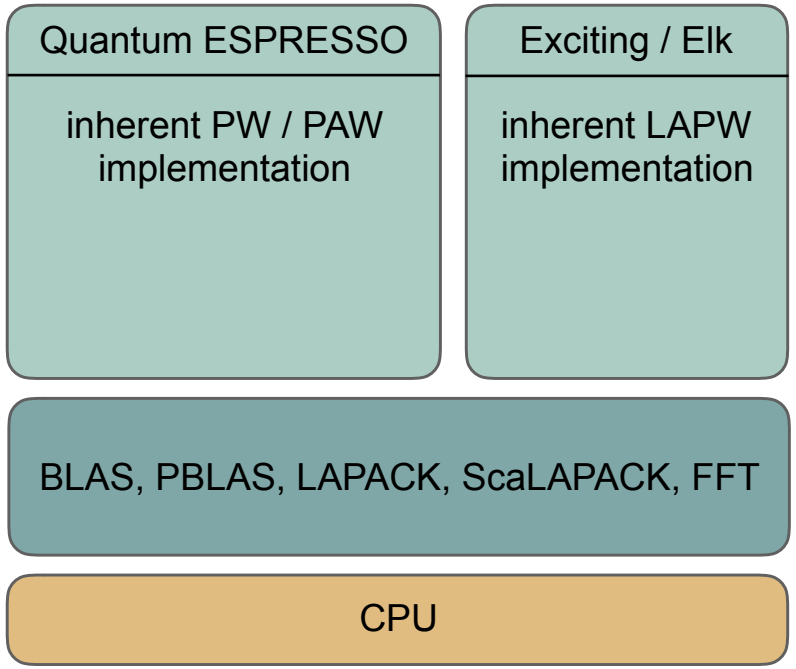


Motivation for a common domain specific library



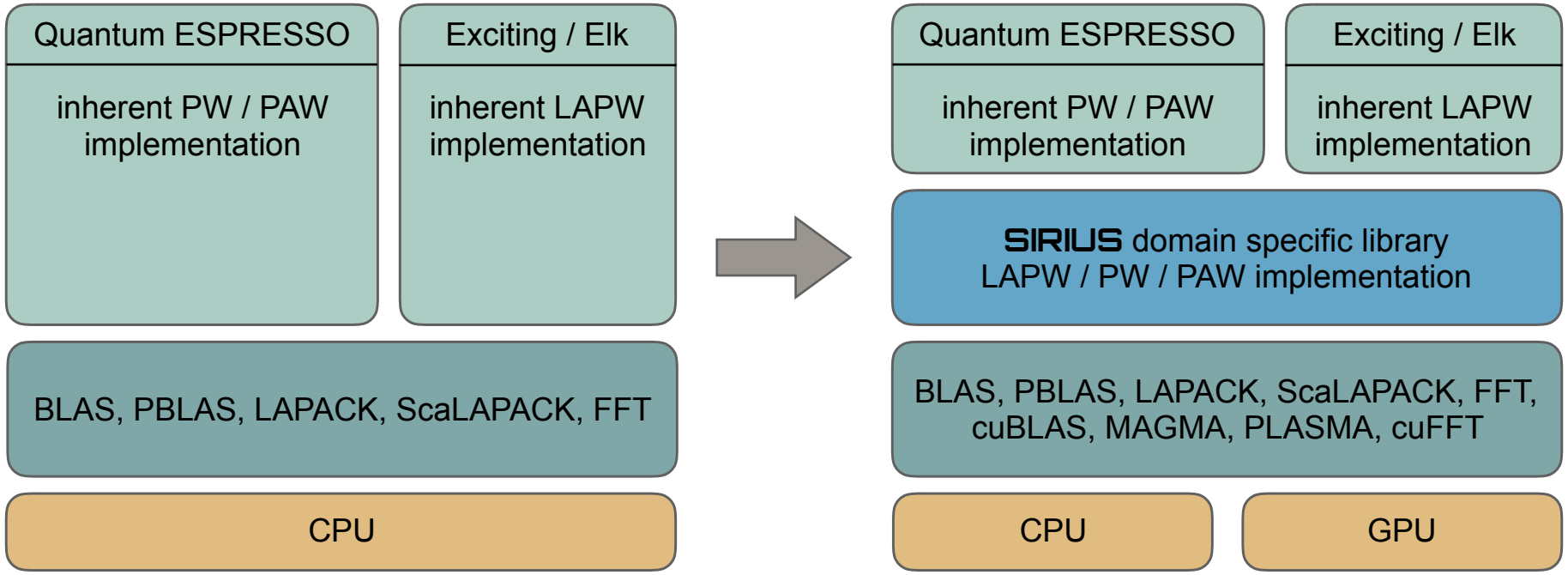
Motivation for a common domain specific library

Extend the legacy Fortran codes with the API calls to a domain-specific library which runs on GPUs and other novel architectures.

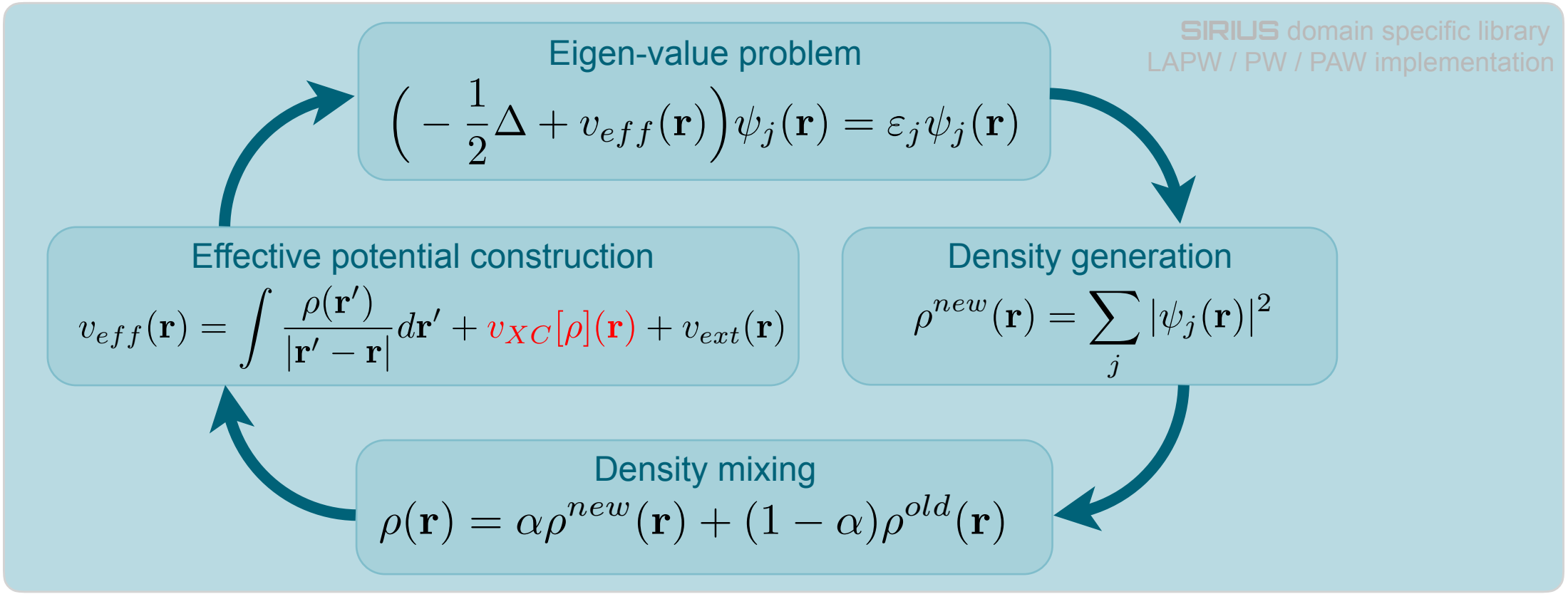


Motivation for a common domain specific library

Extend the legacy Fortran codes with the API calls to a domain-specific library which runs on GPUs and other novel architectures.



Where to draw the line?



Output:

wave-functions $\psi_j(\mathbf{r})$ and eigen energies ε_j

charge density $\rho(\mathbf{r})$ and magnetization $\mathbf{m}(\mathbf{r})$

total energy E_{tot}, atomic forces $\mathbf{F}_\alpha$ and stress tensor $\sigma_{\alpha\beta}$

SIRIUS library

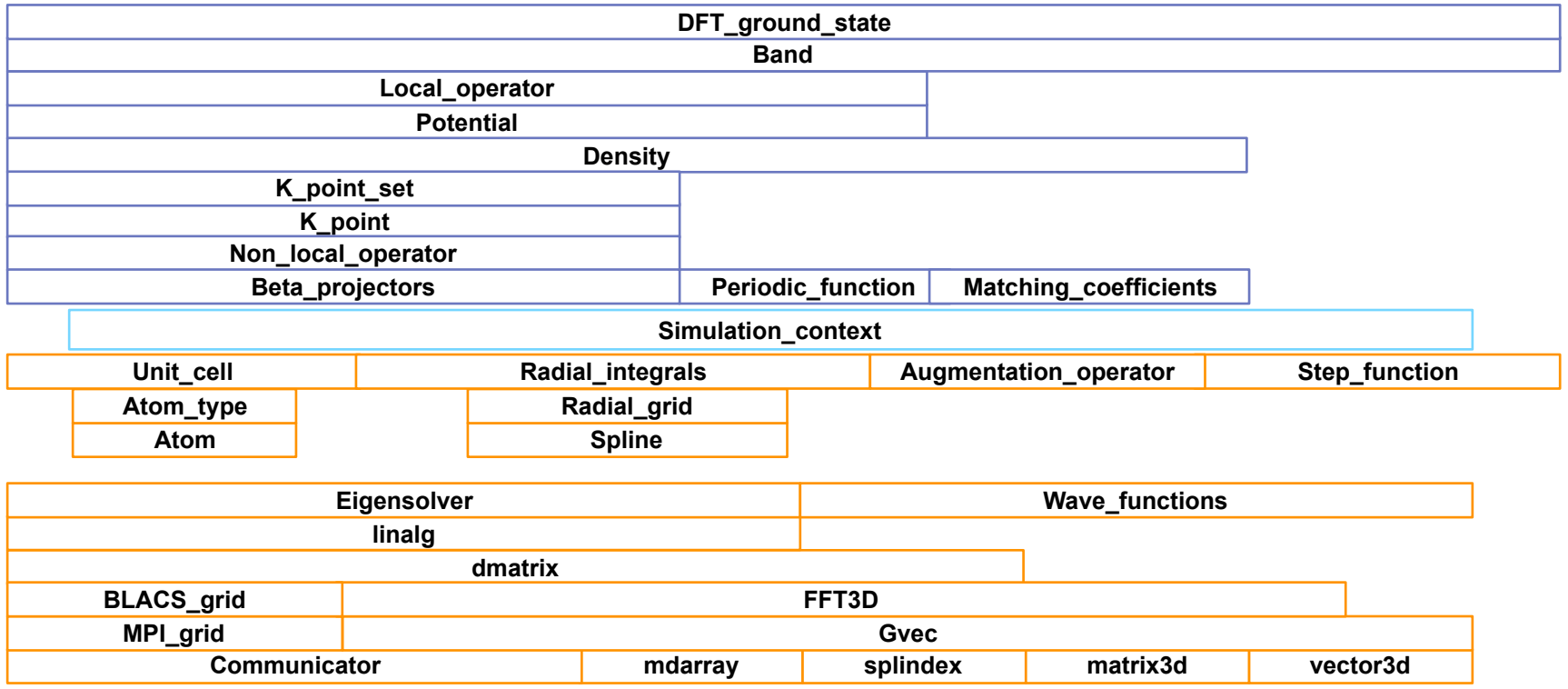
- full-potential (L)APW+lo
  - non-magnetic, collinear and non-collinear magnetic ground states
  - non-relativistic, ZORA and IORA valence solvers
  - Dirac solver for core states

- norm-conserving, ultrasoft and PAW pseudopotentials
  - non-magnetic, collinear and non-collinear magnetic ground states
  - spin-orbit correction
  - atomic forces
  - stress tensor
  - Gamma-point case

SIRIUS library

https://github.com/electronic-structure/SIRIUS

SIRIUS is a collection of classes that abstract away the different building blocks of PW and LAPW codes. The class composition hierarchy starts from the most primitive classes (**Communicator**, **mdarray**, etc.) and progresses towards several high-level classes (**DFT_ground_state**, **Band**, **Potential**, etc.). The code is written in C++11 with MPI, OpenMP and CUDA programming models.



Doxygen documentation

https://electronic-structure.github.io/SIRIUS-doc/

dRlm_dr()

```
static void sirius::SHT::dRlm_dr ( int                          lmax__,
                                   vector3d< double > &         r__,
                                   mdarray< double, 2 > &       data__
                                 )
```

inline static

Compute the derivatives of real spherical harmonics over the components of cartesian vector.

The following derivative is computed:

$$\frac{\partial R_{\ell m}(\theta_r,\phi_r)}{\partial r_\mu} = \frac{\partial R_{\ell m}(\theta_r,\phi_r)}{\partial \theta_r}\frac{\partial \theta_r}{\partial r_\mu} + \frac{\partial R_{\ell m}(\theta_r,\phi_r)}{\partial \phi_r}\frac{\partial \phi_r}{\partial r_\mu}$$

The derivatives of angles are:

$$\frac{\partial \theta_r}{\partial r_x} = \frac{\cos(\phi_r)\cos(\theta_r)}{r}$$
$$\frac{\partial \theta_r}{\partial r_y} = \frac{\cos(\theta_r)\sin(\phi_r)}{r}$$
$$\frac{\partial \theta_r}{\partial r_z} = -\frac{\sin(\theta_r)}{r}$$

and

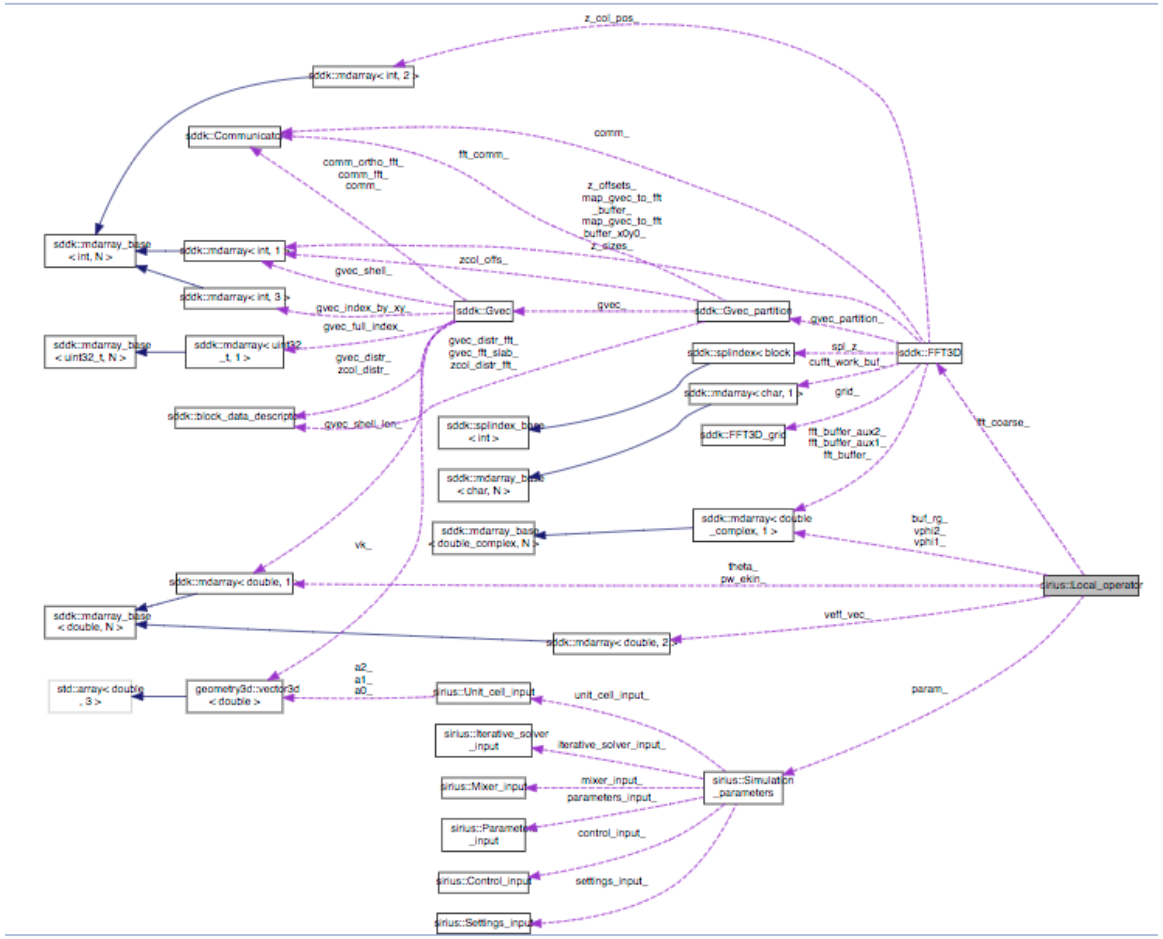
$$\frac{\partial \phi_r}{\partial r_x} = -\frac{\sin(\phi_r)}{\sin(\theta_r)r}$$
$$\frac{\partial \phi_r}{\partial r_y} = \frac{\cos(\phi_r)}{\sin(\theta_r)r}$$
$$\frac{\partial \phi_r}{\partial r_z} = 0$$

The derivative of ϕ has discontinuities at $\theta = 0, \theta = \pi$. This, however, is not a problem, because multiplication by the the derivative of $R_{\ell m}$ removes it.
The following functions have to be hardcoded:

$$\frac{\partial R_{\ell m}(\theta,\phi)}{\partial \theta}$$
$$\frac{\partial R_{\ell m}(\theta,\phi)}{\partial \phi}\frac{1}{\sin(\theta)}$$

Mathematica script for spherical harmonic derivatives:

```
Rlm[l_, m_, th_, ph_] :=
 If[m > 0, Sqrt[2]*ComplexExpand[Re[SphericalHarmonicY[l, m, th, ph]]],
   If[m < 0, Sqrt[2]*ComplexExpand[Im[SphericalHarmonicY[l, m, th, ph]]],
     If[m == 0, ComplexExpand[Re[SphericalHarmonicY[l, 0, th, ph]]]]
   ]
 ]
Do[Print[FullSimplify[D[Rlm[l, m, theta, phi], theta]]], {l, 0, 4}, {m, -l, l}]
Do[Print[FullSimplify[TrigExpand[D[Rlm[l, m, theta, phi], phi]/Sin[theta]]]], {l, 0, 4}, {m, -l, l}]
```



Potential class

- Generate LDA / GGA exchange-correlation potential from the density

$$v^{XC}(\mathbf{r}) = \frac{\delta E^{XC}[\rho(\mathbf{r}), \mathbf{m}(\mathbf{r})]}{\delta \rho(\mathbf{r})} \qquad \mathbf{B}^{XC}(\mathbf{r}) = \frac{\delta E^{XC}[\rho(\mathbf{r}), \mathbf{m}(\mathbf{r})]}{\delta \mathbf{m}(\mathbf{r})}$$

- Generate Hartree potential

$$v^{H}(\mathbf{r}) = \int \frac{\rho(\mathbf{r}')}{|\mathbf{r} - \mathbf{r}'|} d\mathbf{r}'$$

- Generate local part of pseudo-potential

$$V_{loc}(\mathbf{G}) = \frac{1}{V} \sum_{\mathbf{T},\alpha} \int e^{-i\mathbf{G}\mathbf{r}} V_{loc}^{\alpha}(\mathbf{r} - \mathbf{T} - \tau_{\alpha}) d\mathbf{r}$$

- Generate D-operator matrix

$$D_{\xi\xi'}^{\alpha} = \int V(\mathbf{r}) Q_{\xi\xi'}^{\alpha}(\mathbf{r}) d\mathbf{r}$$

Density class

- Generate charge density and magnetization from the valence wave-functions

$$\boldsymbol{\rho}(\mathbf{r}) = \frac{1}{2}\Big(\mathbf{I}\rho(\mathbf{r}) + \boldsymbol{\sigma}\mathbf{m}(\mathbf{r})\Big) = \frac{1}{2}\begin{pmatrix} \rho(\mathbf{r}) + m_z(\mathbf{r}) & m_x(\mathbf{r}) - im_y(\mathbf{r}) \\ m_x(\mathbf{r}) + im_y(\mathbf{r}) & \rho(\mathbf{r}) - m_z(\mathbf{r}) \end{pmatrix} = \sum_j^{occ} \begin{pmatrix} \psi_j^{\uparrow *}(\mathbf{r})\psi_j^{\uparrow}(\mathbf{r}) & \psi_j^{\downarrow *}(\mathbf{r})\psi_j^{\uparrow}(\mathbf{r}) \\ \psi_j^{\uparrow *}(\mathbf{r})\psi_j^{\downarrow}(\mathbf{r}) & \psi_j^{\downarrow *}(\mathbf{r})\psi_j^{\downarrow}(\mathbf{r}) \end{pmatrix}$$

- Generate core charge density (full-potential case)
- Generate density matrix

$$d^{\alpha}_{\xi\xi'} = \langle \beta^{\alpha}_{\xi} | \hat{N} | \beta^{\alpha}_{\xi'} \rangle = \sum_{j\mathbf{k}} \langle \beta^{\alpha}_{\xi} | \Psi_{j\mathbf{k}} \rangle n_{j\mathbf{k}} \langle \Psi_{j\mathbf{k}} | \beta^{\alpha}_{\xi'} \rangle$$

- Augment charge density $\tilde{\rho}(\mathbf{G}) = \sum_{\alpha} \sum_{\xi\xi'} d^{\alpha}_{\xi\xi'} Q^{\alpha}_{\xi'\xi}(\mathbf{G})$

- Symmetrize density and magnetization
- Mix density and magnetization

Band class

- Setup and solves the Kohn-Sham eigen-value problem

$$\hat{\mathbf{H}}|\Psi\rangle = \varepsilon|\Psi\rangle$$

  - Full-potential LAPW case: $\mathbf{H}x=\varepsilon\mathbf{S}x$, direct diagonalization of dense matrix

  - Pseudopotential PW case: $\mathbf{H}x=\varepsilon x$ (norm-conserving pseudo) or $\mathbf{H}x=\varepsilon\mathbf{S}x$ (ultrasoft pseudo), iterative diagonalization of dense matrix
    - Conjugate gradient
    - LOBPCG
    - RMM-DIIS
    - Chebyshev filtering method
    - Davidson algorithm

Band class

- Setup and solves the Kohn-Sham eigen-value problem

$$\hat{\mathbf{H}}|\Psi\rangle = \varepsilon|\Psi\rangle$$

  - Full-potential LAPW case: $\mathbf{H}x=\varepsilon\mathbf{S}x$, direct diagonalization of dense matrix

  - Pseudopotential PW case: $\mathbf{H}x=\varepsilon x$ (norm-conserving pseudo) or $\mathbf{H}x=\varepsilon\mathbf{S}x$ (ultrasoft pseudo), iterative diagonalization of dense matrix
    - Conjugate gradient
    - LOBPCG
    - RMM-DIIS
    - Chebyshev filtering method
    - Davidson algorithm

All solvers were tried, Davidson method works the best

Davidson iterative solver

- We want to solve eigen-value problem $\mathbf{H}\tilde{\psi}_j = \varepsilon_j \tilde{\psi}_j$

Davidson iterative solver

- We want to solve eigen-value problem $\mathbf{H}\tilde{\psi}_j = \varepsilon_j \tilde{\psi}_j$

- We know how to apply Hamiltonian to the wave-functions

$$\tilde{h}_{\psi_j} = \mathbf{H}\tilde{\psi}_j = \int e^{-i\mathbf{G}\mathbf{r}} \hat{H}\psi_j(\mathbf{r})d\mathbf{r} \qquad \hat{H} = -\frac{1}{2}\Delta + v_{eff}(\mathbf{r}) + \sum_{\alpha}\sum_{\xi\xi'} |\beta_{\xi}^{\alpha}\rangle D_{\xi\xi'}^{\alpha} \langle\beta_{\xi'}^{\alpha}|$$

Davidson iterative solver

- We want to solve eigen-value problem $\mathbf{H}\tilde{\psi}_j = \varepsilon_j \tilde{\psi}_j$

- We know how to apply Hamiltonian to the wave-functions

$$\tilde{h}_{\psi_j} = \mathbf{H}\tilde{\psi}_j = \int e^{-i\mathbf{G}\mathbf{r}} \hat{H}\psi_j(\mathbf{r})d\mathbf{r} \qquad \hat{H} = -\frac{1}{2}\Delta + v_{eff}(\mathbf{r}) + \sum_{\alpha}\sum_{\xi\xi'} |\beta_{\xi}^{\alpha}\rangle D_{\xi\xi'}^{\alpha} \langle\beta_{\xi'}^{\alpha}|$$

Key idea of the Davidson iterative solver: start with a subspace spanned by a guess to ψ_j and expand it with preconditioned residuals.

Davidson iterative solver: application of the Hamiltonian

- Application of the Laplace operator (kinetic energy)

$$-\frac{1}{2}\Delta\psi_j(\mathbf{r}) = \sum_{\mathbf{G}}\Big(-\frac{1}{2}\Delta e^{i\mathbf{G}\mathbf{r}}\Big)\tilde{\psi}_j(\mathbf{G}) = \sum_{\mathbf{G}} e^{i\mathbf{G}\mathbf{r}}\frac{G^2}{2}\tilde{\psi}_j(\mathbf{G})$$

- Application of the local part of potential

$$\tilde{\psi}_j(\mathbf{G}) \xrightarrow{FFT^{-1}} \psi_j(\mathbf{r}) \rightarrow v_{eff}(\mathbf{r})\psi_j(\mathbf{r}) \xrightarrow{FFT} \tilde{h}_{\psi_j}(\mathbf{G})$$

- Application of the non-local local part of potential

$$\sum_{\alpha\xi}\beta^{\alpha}_{\xi}(\mathbf{G})\sum_{\xi'}D^{\alpha}_{\xi\xi'}\sum_{\mathbf{G}'}\beta^{\alpha*}_{\xi'}(\mathbf{G}')\tilde{\psi}_j(\mathbf{G}') \rightarrow \tilde{h}_{\psi_j}(\mathbf{G})$$

zgemm#1

zgemm#2

zgemm#3

Davidson iterative solver: algorithm

- Initialize the trial basis set: $\tilde{\phi}_j^0 \Leftarrow \tilde{\psi}_j$

Davidson iterative solver: algorithm

- Initialize the trial basis set: $\tilde{\phi}_j^0 \Leftarrow \tilde{\psi}_j$
- Apply Hamiltonian to the basis functions: $\tilde{h}_{\phi_j^m} = \mathbf{H}\tilde{\phi}_j^m$

Davidson iterative solver: algorithm

- Initialize the trial basis set: $\tilde{\phi}_j^0 \Leftarrow \tilde{\psi}_j$
- Apply Hamiltonian to the basis functions: $\tilde{h}_{\phi_j^m} = \mathbf{H}\tilde{\phi}_j^m$
- Compute reduced Hamiltonian matrix: $h_{jj'}^m = \sum_{\mathbf{G}} \tilde{\phi}_j^{m*}(\mathbf{G})\tilde{h}_{\phi_{j'}}^m(\mathbf{G})$

Davidson iterative solver: algorithm

- Initialize the trial basis set: $\tilde{\phi}_j^0 \Leftarrow \tilde{\psi}_j$
- Apply Hamiltonian to the basis functions: $\tilde{h}_{\phi_j^m} = \mathbf{H}\tilde{\phi}_j^m$
- Compute reduced Hamiltonian matrix: $h_{jj'}^m = \sum_{\mathbf{G}} \tilde{\phi}_j^{m*}(\mathbf{G})\tilde{h}_{\phi_{j'}}^m(\mathbf{G})$
- Diagonalize reduced Hamiltonian matrix and get N lowest eigen pairs: $\mathbf{h}^m\mathbf{Z}^m = \epsilon_j\mathbf{Z}^m$

Davidson iterative solver: algorithm

- Initialize the trial basis set: $\tilde{\phi}_j^0 \Leftarrow \tilde{\psi}_j$
- Apply Hamiltonian to the basis functions: $\tilde{h}_{\phi_j^m} = \mathbf{H}\tilde{\phi}_j^m$
- Compute reduced Hamiltonian matrix: $h_{jj'}^m = \sum_{\mathbf{G}} \tilde{\phi}_j^{m*}(\mathbf{G})\tilde{h}_{\phi_{j'}}^m(\mathbf{G})$
- Diagonalize reduced Hamiltonian matrix and get N lowest eigen pairs: $\mathbf{h}^m\mathbf{Z}^m = \epsilon_j\mathbf{Z}^m$
- Compute residuals ($R_j = \hat{H}\psi_j - \epsilon_j\psi_j$): $\tilde{R}_j^m = \tilde{h}_{\phi_j^m}\mathbf{Z}^m - \epsilon_j\tilde{\phi}_j^m\mathbf{Z}^m$

Davidson iterative solver: algorithm

- Initialize the trial basis set: $\tilde{\phi}_j^0 \Leftarrow \tilde{\psi}_j$
- Apply Hamiltonian to the basis functions: $\tilde{h}_{\phi_j^m} = \mathbf{H}\tilde{\phi}_j^m$
- Compute reduced Hamiltonian matrix: $h_{jj'}^m = \sum_{\mathbf{G}} \tilde{\phi}_j^{m*}(\mathbf{G})\tilde{h}_{\phi_{j'}}^m(\mathbf{G})$
- Diagonalize reduced Hamiltonian matrix and get N lowest eigen pairs: $\mathbf{h}^m\mathbf{Z}^m = \epsilon_j\mathbf{Z}^m$
- Compute residuals ($R_j = \hat{H}\psi_j - \epsilon_j\psi_j$): $\tilde{R}_j^m = \tilde{h}_{\phi_j^m}\mathbf{Z}^m - \epsilon_j\tilde{\phi}_j^m\mathbf{Z}^m$
- Apply preconditioner to the unconverged residuals, orthogonalize and add the resulting functions to the basis: $\{\tilde{\phi}_j^{m+1}\} = \{\tilde{\phi}_j^m\} \bigoplus \{P\tilde{R}_j^m\}$

Davidson iterative solver: algorithm

- Initialize the trial basis set: $\tilde{\phi}_j^0 \Leftarrow \tilde{\psi}_j$
- Apply Hamiltonian to the basis functions: $\tilde{h}_{\phi_j^m} = \mathbf{H}\tilde{\phi}_j^m$
- Compute reduced Hamiltonian matrix: $h_{jj'}^m = \sum_{\mathbf{G}} \tilde{\phi}_j^{m*}(\mathbf{G})\tilde{h}_{\phi_{j'}}^m(\mathbf{G})$
- Diagonalize reduced Hamiltonian matrix and get N lowest eigen pairs: $\mathbf{h}^m\mathbf{Z}^m = \epsilon_j\mathbf{Z}^m$
- Compute residuals ($R_j = \hat{H}\psi_j - \epsilon_j\psi_j$): $\tilde{R}_j^m = \tilde{h}_{\phi_j^m}\mathbf{Z}^m - \epsilon_j\tilde{\phi}_j^m\mathbf{Z}^m$
- Apply preconditioner to the unconverged residuals, orthogonalize and add the resulting functions to the basis: $\{\tilde{\phi}_j^{m+1}\} = \{\tilde{\phi}_j^m\} \bigoplus \{P\tilde{R}_j^m\}$
- Recompute the wave-functions: $\tilde{\psi}_j = \tilde{\phi}_j^m\mathbf{Z}^m$

Davidson iterative solver: algorithm

- Initialize the trial basis set: $\tilde{\phi}_j^0 \Leftarrow \tilde{\psi}_j$
- Apply Hamiltonian to the basis functions: $\tilde{h}_{\phi_j^m} = \mathbf{H}\tilde{\phi}_j^m$
- Compute reduced Hamiltonian matrix: $h_{jj'}^m = \sum_{\mathbf{G}} \tilde{\phi}_j^{m*}(\mathbf{G})\tilde{h}_{\phi_{j'}}^m(\mathbf{G})$
- Diagonalize reduced Hamiltonian matrix and get N lowest eigen pairs: $\mathbf{h}^m\mathbf{Z}^m = \epsilon_j\mathbf{Z}^m$
- Compute residuals ($R_j = \hat{H}\psi_j - \epsilon_j\psi_j$): $\tilde{R}_j^m = \tilde{h}_{\phi_j^m}\mathbf{Z}^m - \epsilon_j\tilde{\phi}_j^m\mathbf{Z}^m$
- Apply preconditioner to the unconverged residuals, orthogonalize and add the resulting functions to the basis: $\{\tilde{\phi}_j^{m+1}\} = \{\tilde{\phi}_j^m\} \bigoplus \{P\tilde{R}_j^m\}$
- Recompute the wave-functions: $\tilde{\psi}_j = \tilde{\phi}_j^m\mathbf{Z}^m$

```
=== QE output ===
Dynamical RAM for wfc: 2.50 MB
Dynamical RAM for <psi|beta>: 2.81 MB
Dynamical RAM for psi: 10.02 MB
Dynamical RAM for hpsi: 10.02 MB
Dynamical RAM for spsi: 10.02 MB
============================
```

and psi is $\tilde{\phi}_j$ here

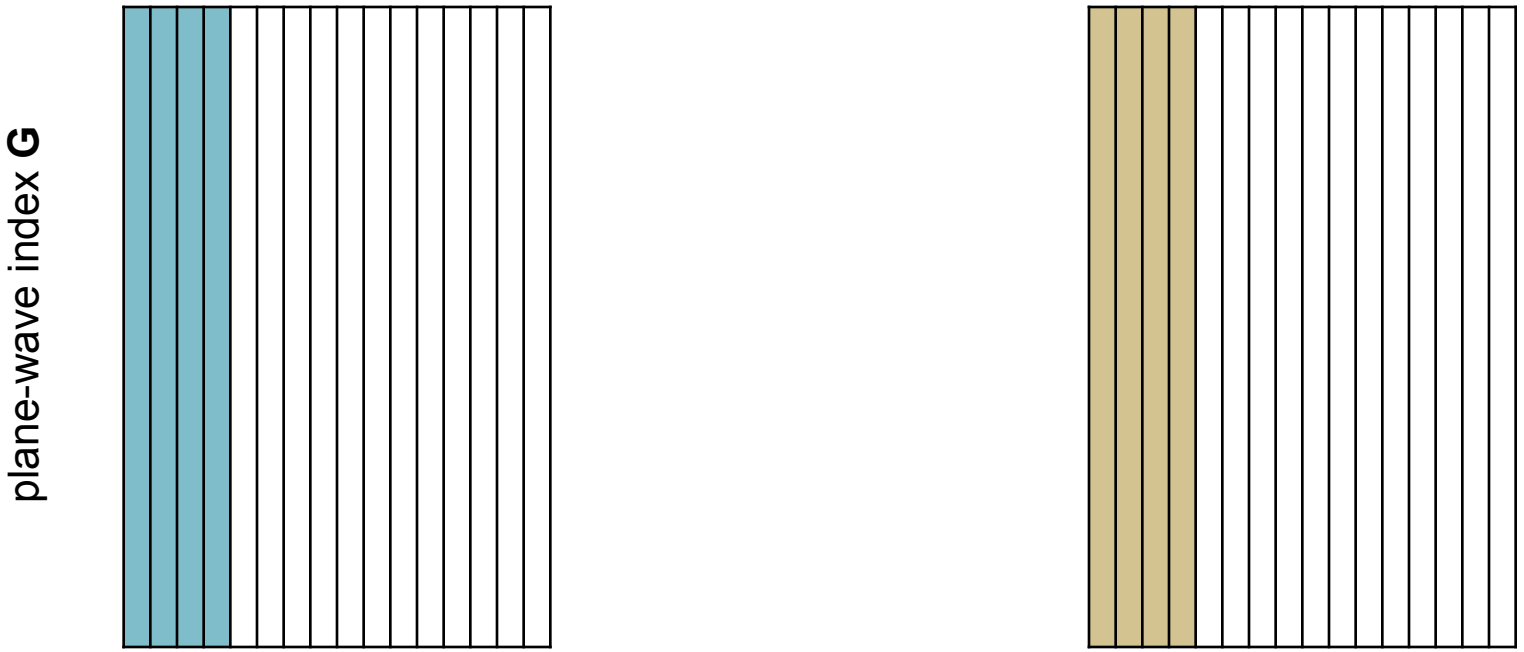
diago_david_ndim	INTEGER
Default:	4
diago_thr_init	REAL
diago_full_acc	LOGICAL

Davidson iterative solver

- Initialize subspace basis functions and apply Hamiltonian

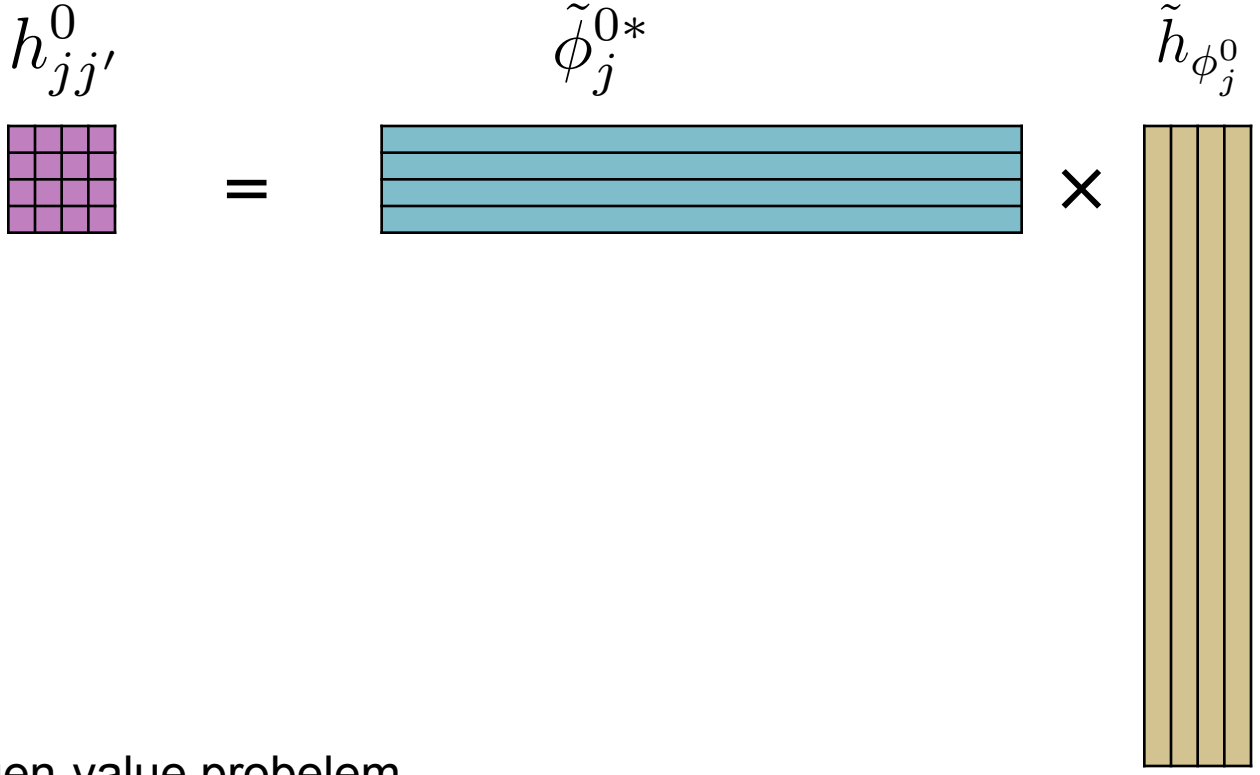
$\tilde{\phi}_j^0$

$\tilde{h}_{\phi_j^0} = \mathbf{H}\tilde{\phi}_j^0$

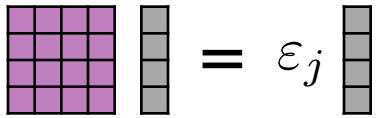


Davidson iterative solver

- Compute reduced Hamiltonian matrix

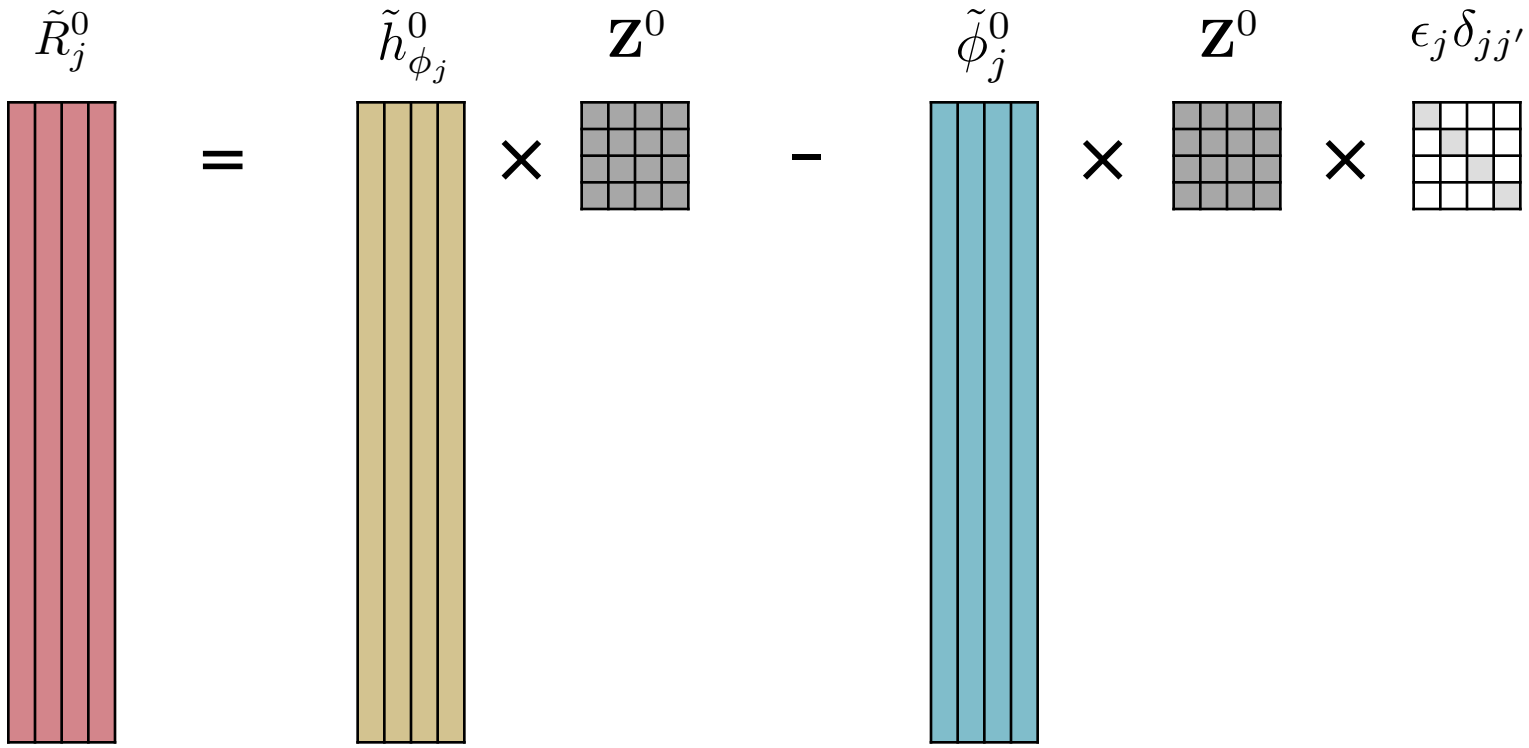


- Compute eigen-value probelem



Davidson iterative solver

- Compute residuals



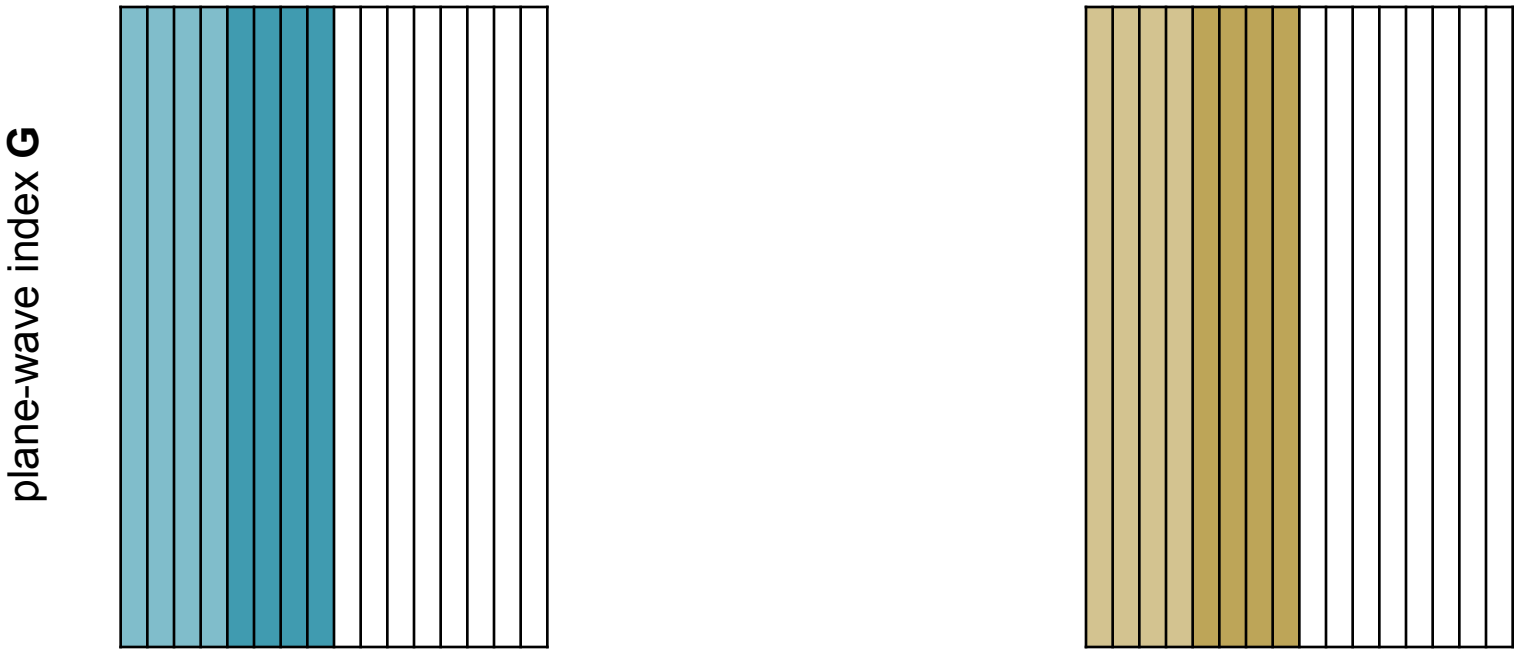
- Apply preconditioner: $P\tilde{R}^0_j(\mathbf{G}) = (H_{\mathbf{GG}} - \varepsilon_j)^{-1}\tilde{R}^0_j(\mathbf{G})$

Davidson iterative solver

- Expand variational space and apply Hamiltonian to new basis functions

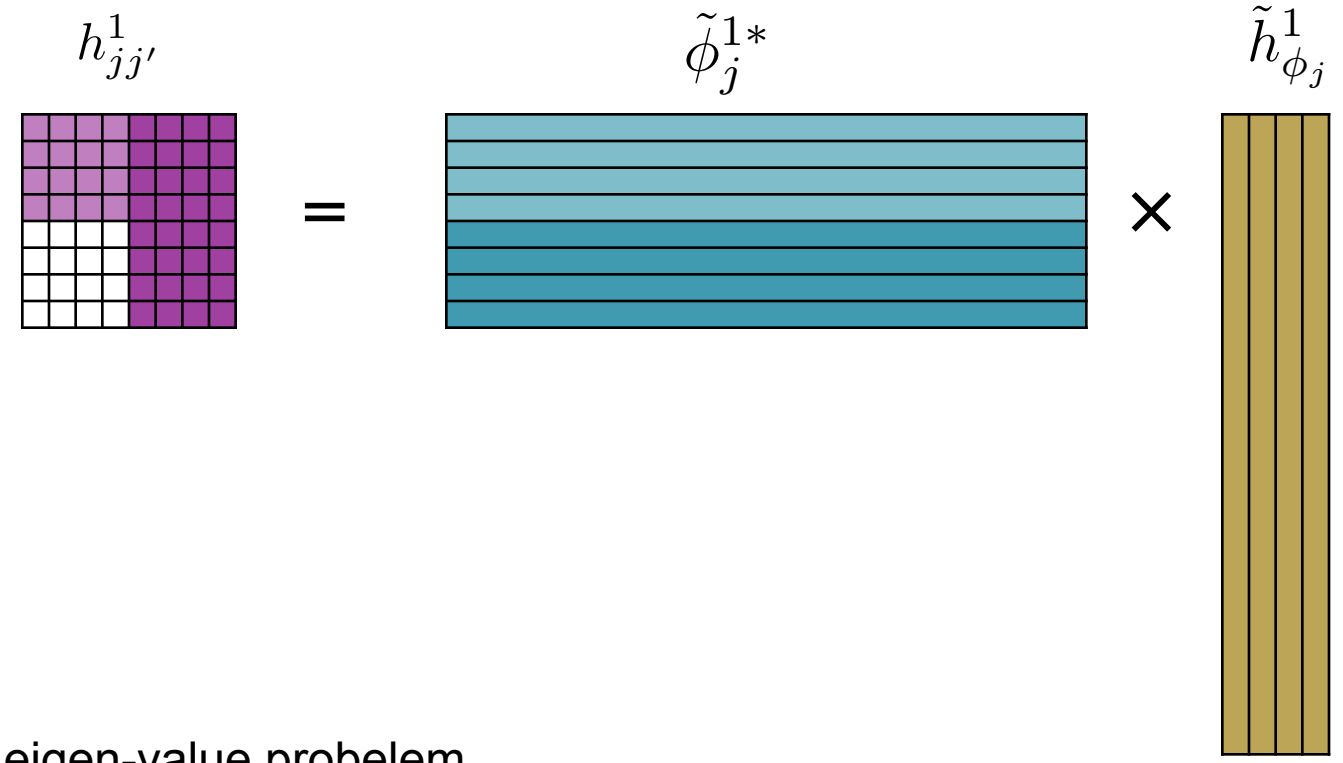
$$\tilde{\phi}_j^1 = \tilde{\phi}_j^0 \bigoplus P\tilde{R}_j^0$$

$$\tilde{h}_{\phi_j^1} = \mathbf{H}\tilde{\phi}_j^1$$

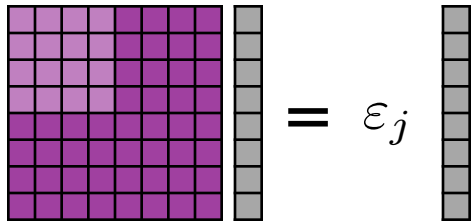


Davidson iterative solver

- Compute reduced Hamiltonian matrix

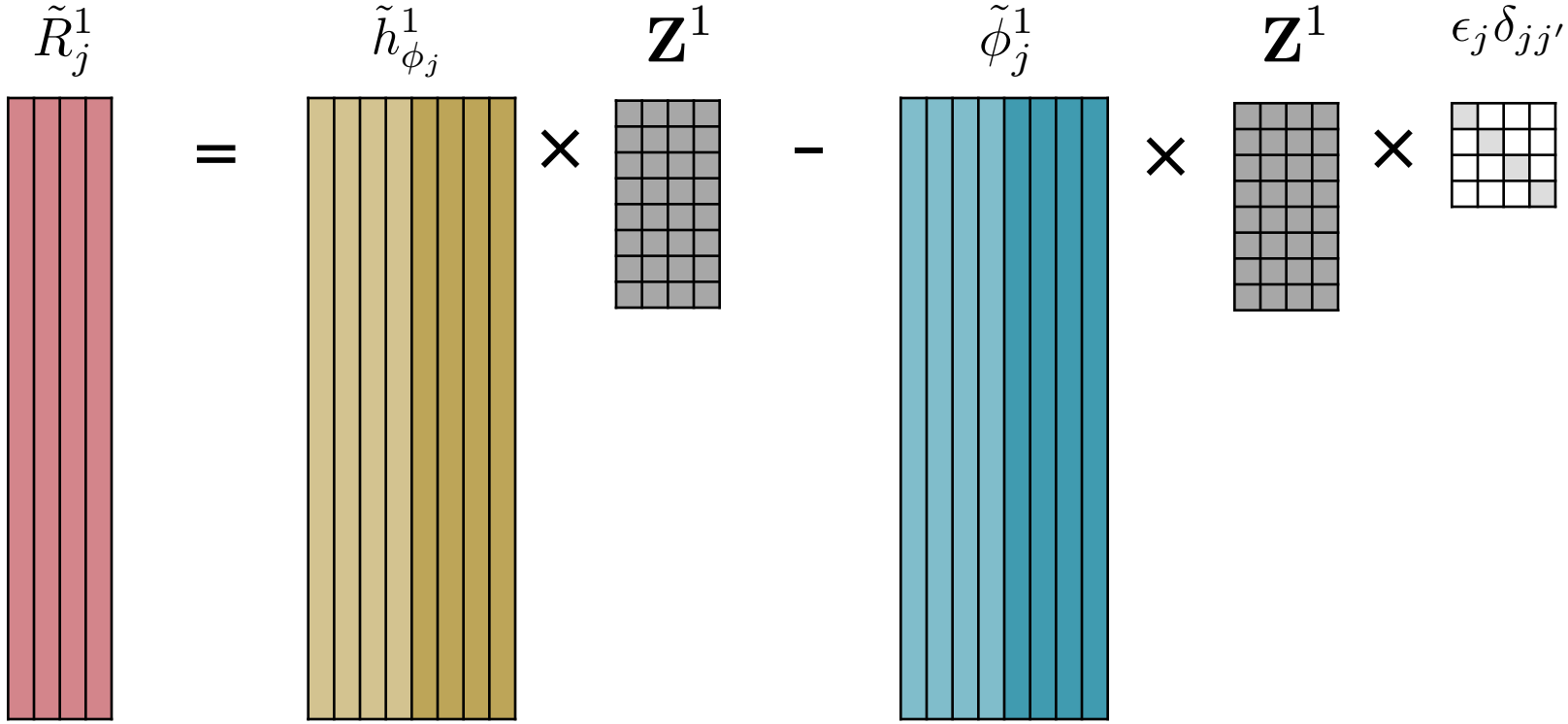


- Compute eigen-value probelem



Davidson iterative solver

- Compute residuals



- Apply preconditioner: $P\tilde{R}^1_j(\mathbf{G}) = (H_{\mathbf{GG}} - \varepsilon_j)^{-1}\tilde{R}^1_j(\mathbf{G})$

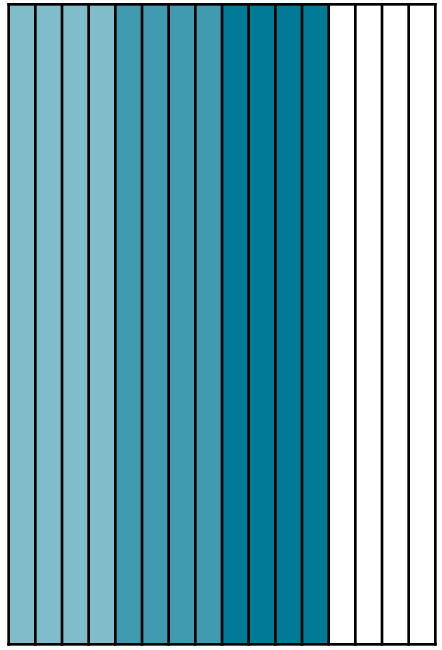
Davidson iterative solver

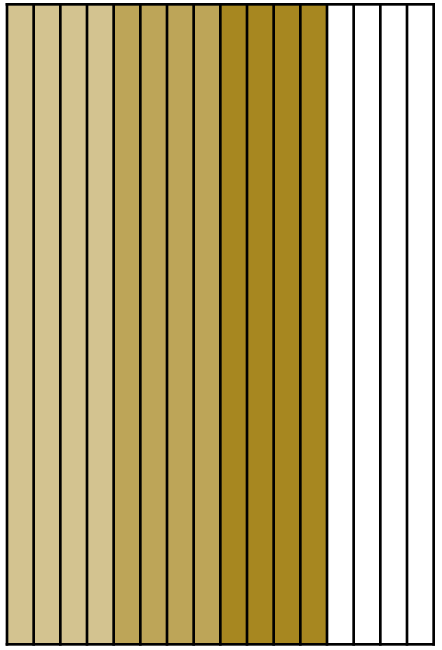
- Continue to expand variational space

$$\tilde{\phi}_j^2 = \tilde{\phi}_j^1 \bigoplus P\tilde{R}_j^1$$

$$\tilde{h}_{\phi_j^2} = \mathbf{H}\tilde{\phi}_j^2$$

plane-wave index **G**



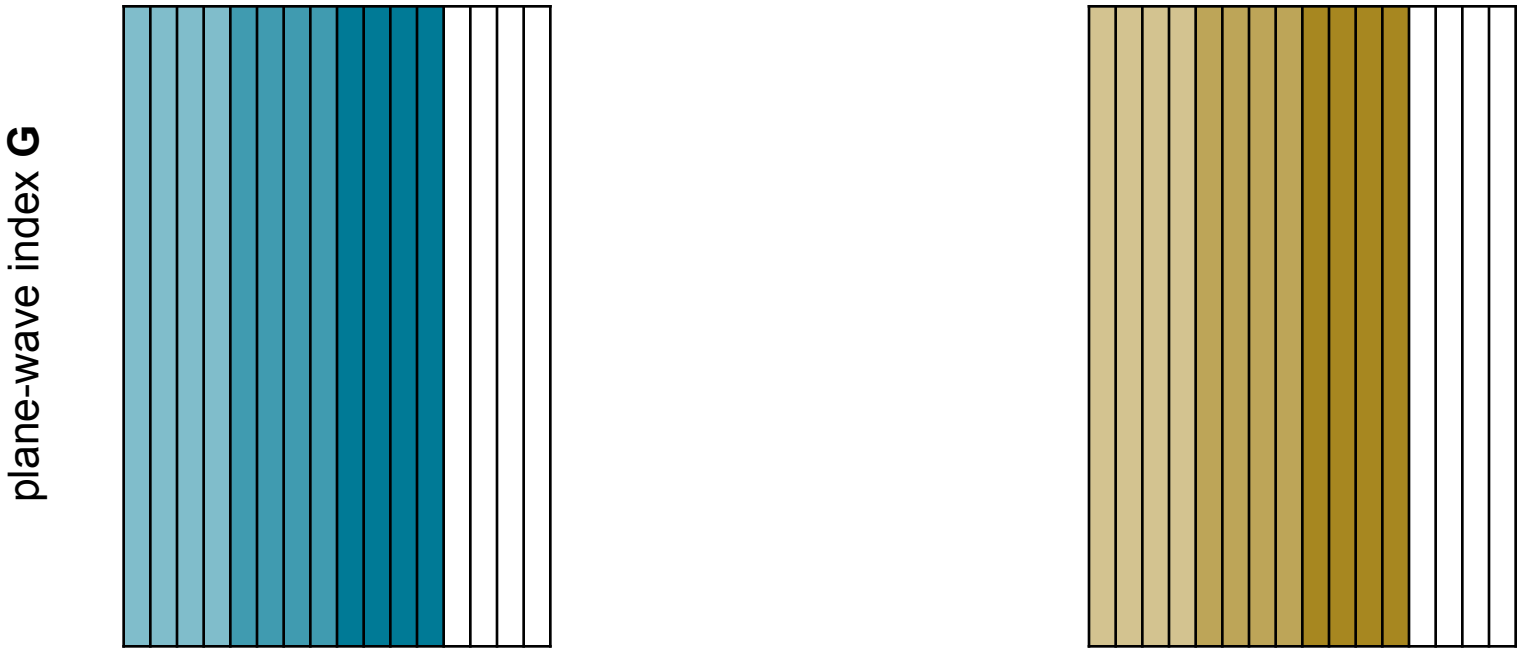


Davidson iterative solver

- Continue to expand variational space

$$\tilde{\phi}_j^2 = \tilde{\phi}_j^1 \bigoplus P\tilde{R}_j^1$$

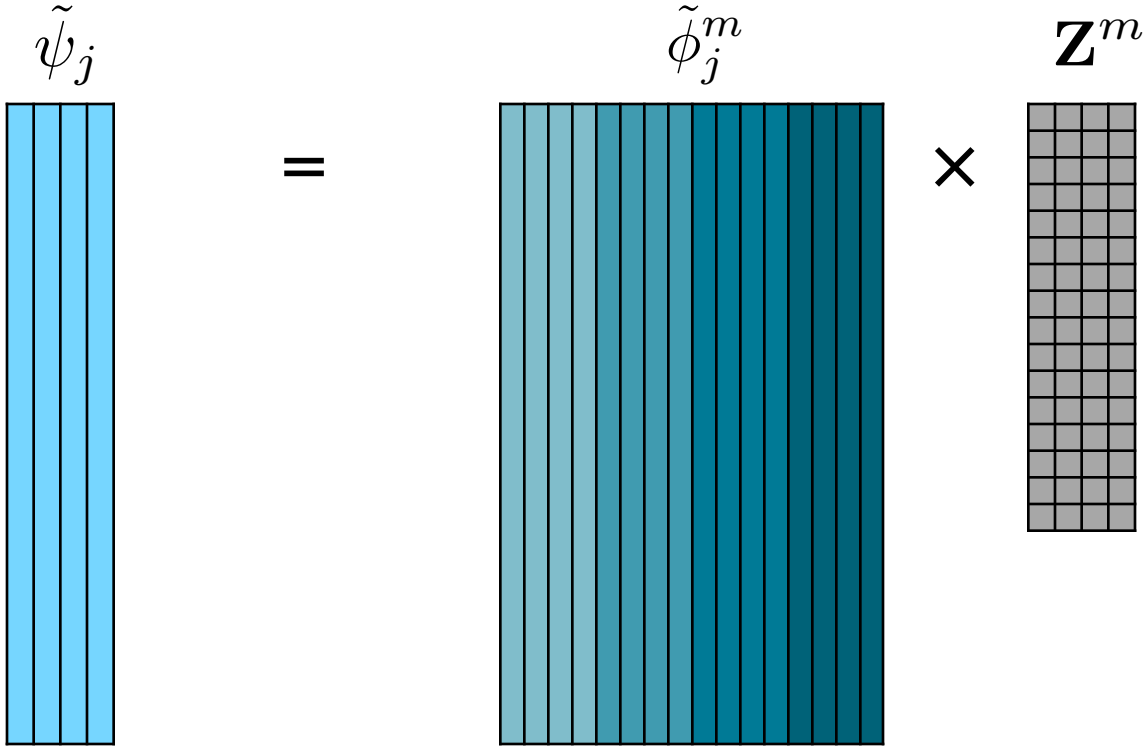
$$\tilde{h}_{\phi_j^2} = \mathbf{H}\tilde{\phi}_j^2$$



Iterate until the convergence (all residuals are zero) is reached

Davidson iterative solver

- Recompute the wave-functions



- Take ε_j from the last subspace diagonalization

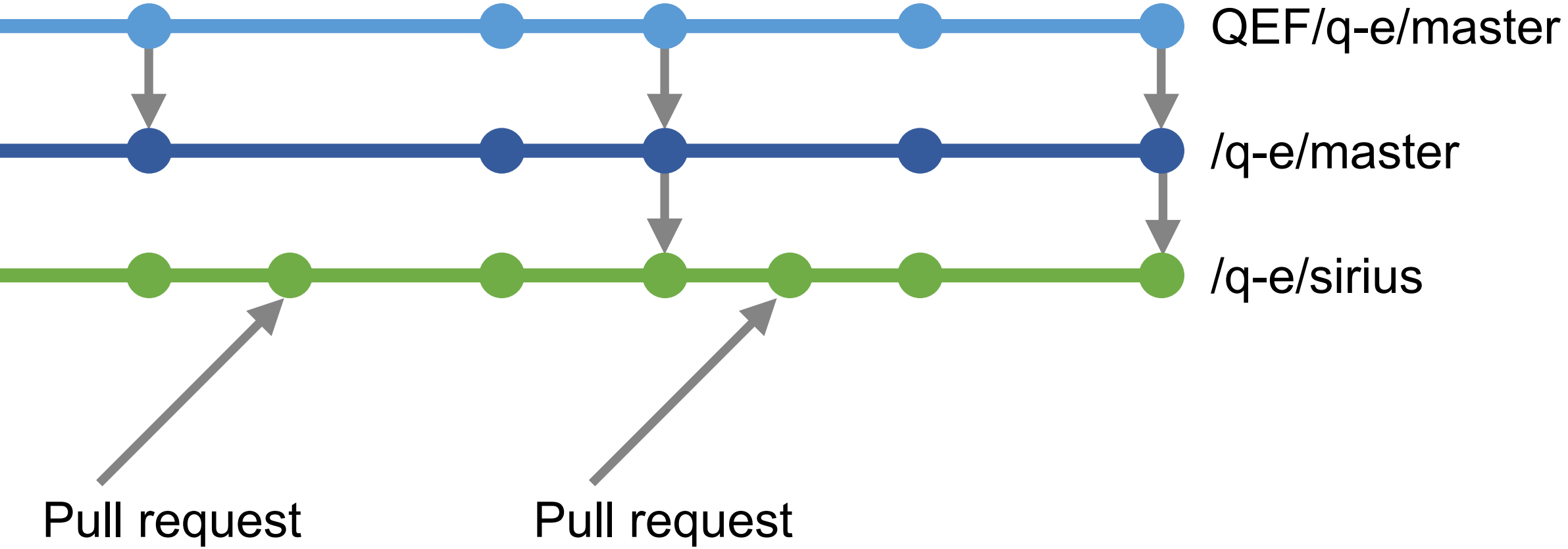
CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich

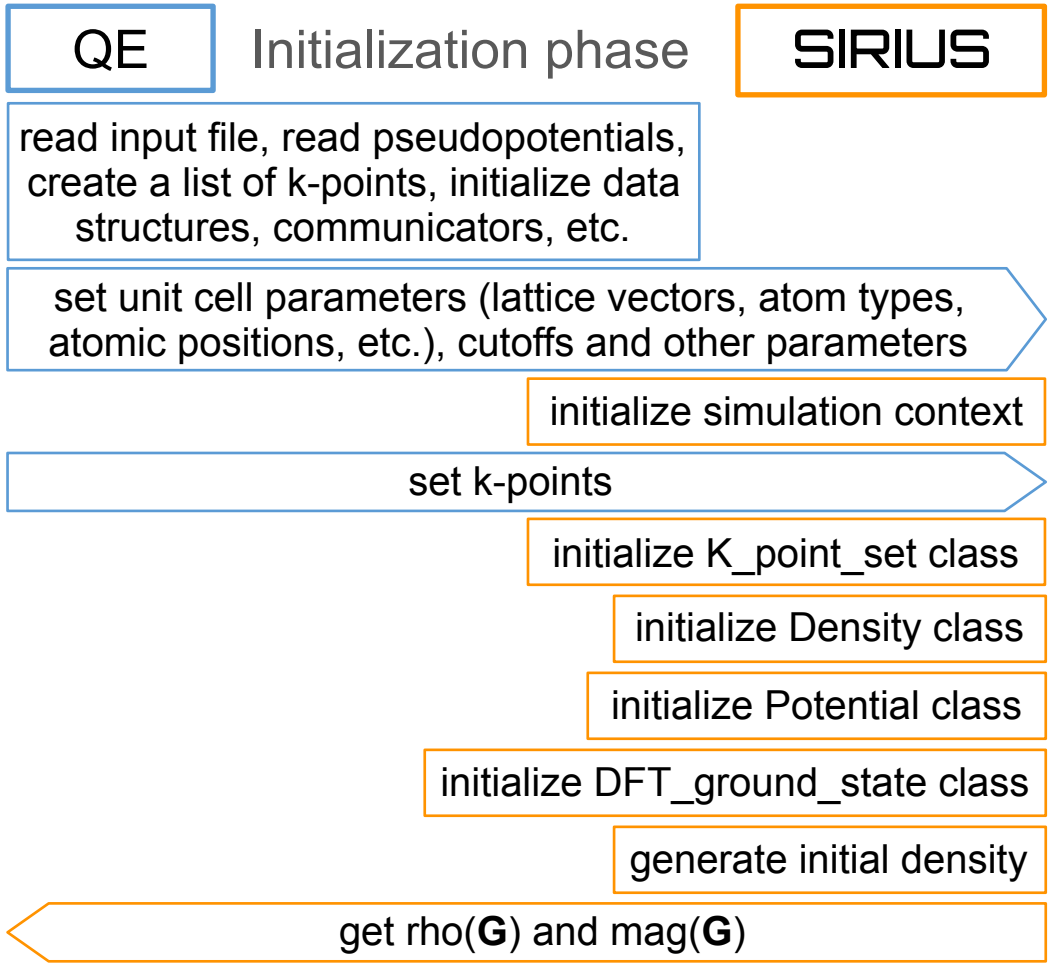
SIRIUS-enabled Quantum ESPRESSO

Development cycle

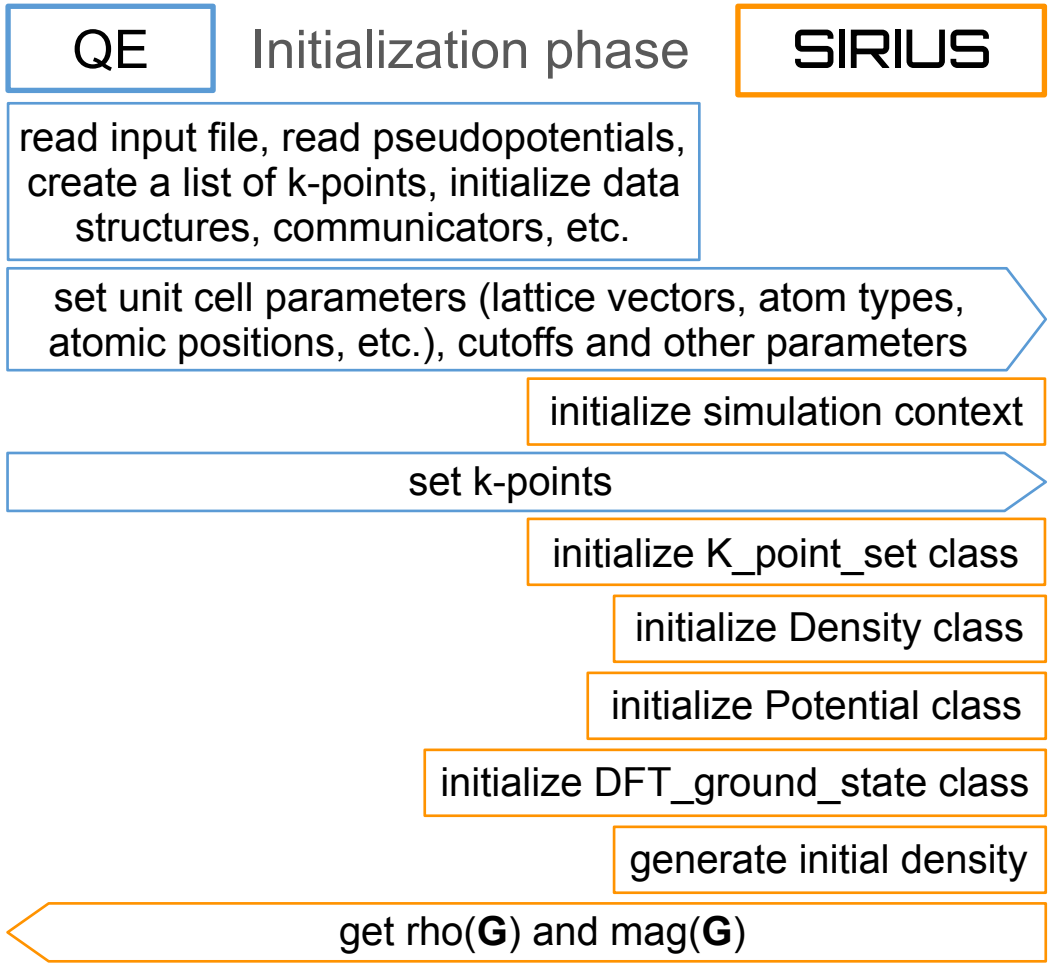
https://github.com/electronic-structure/q-e

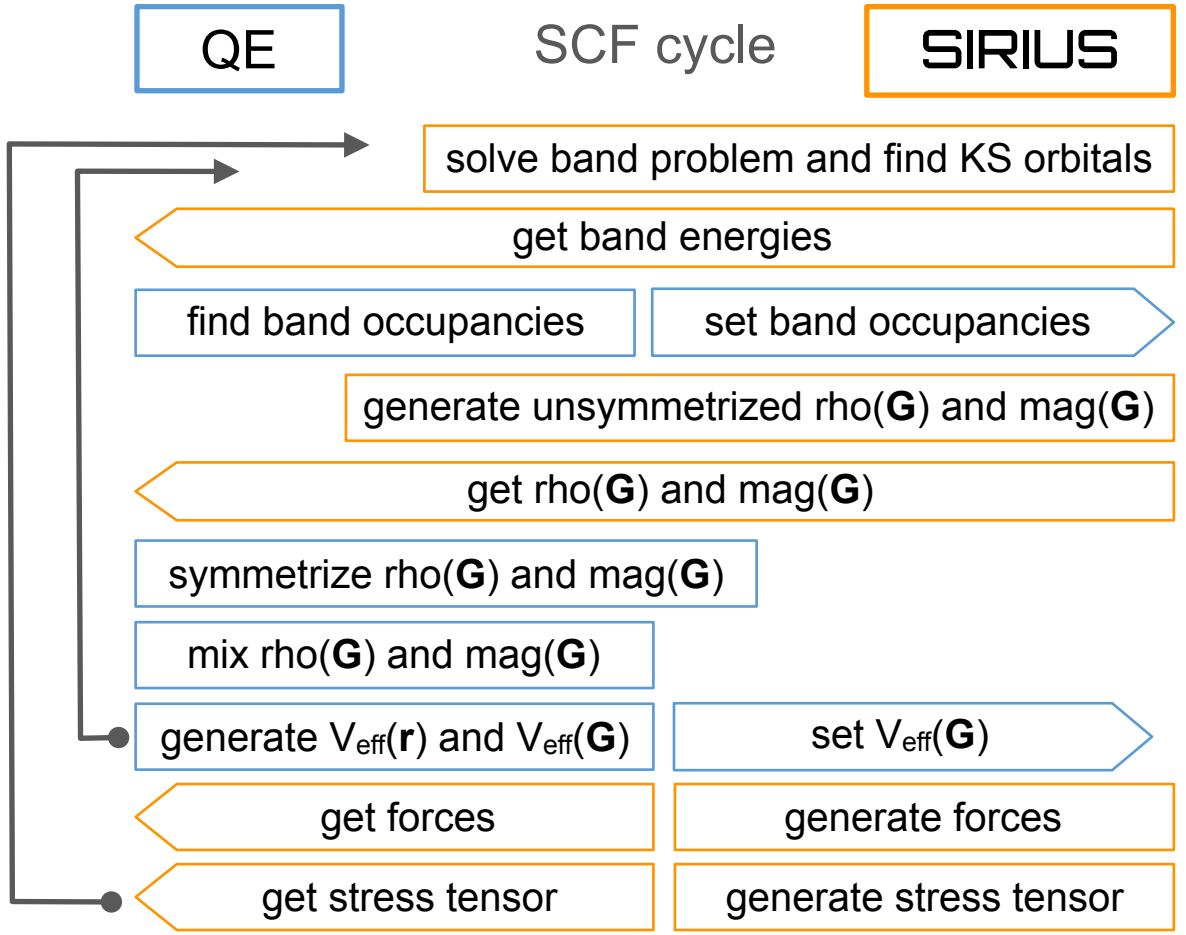


Example of QE/SIRIUS interoperability



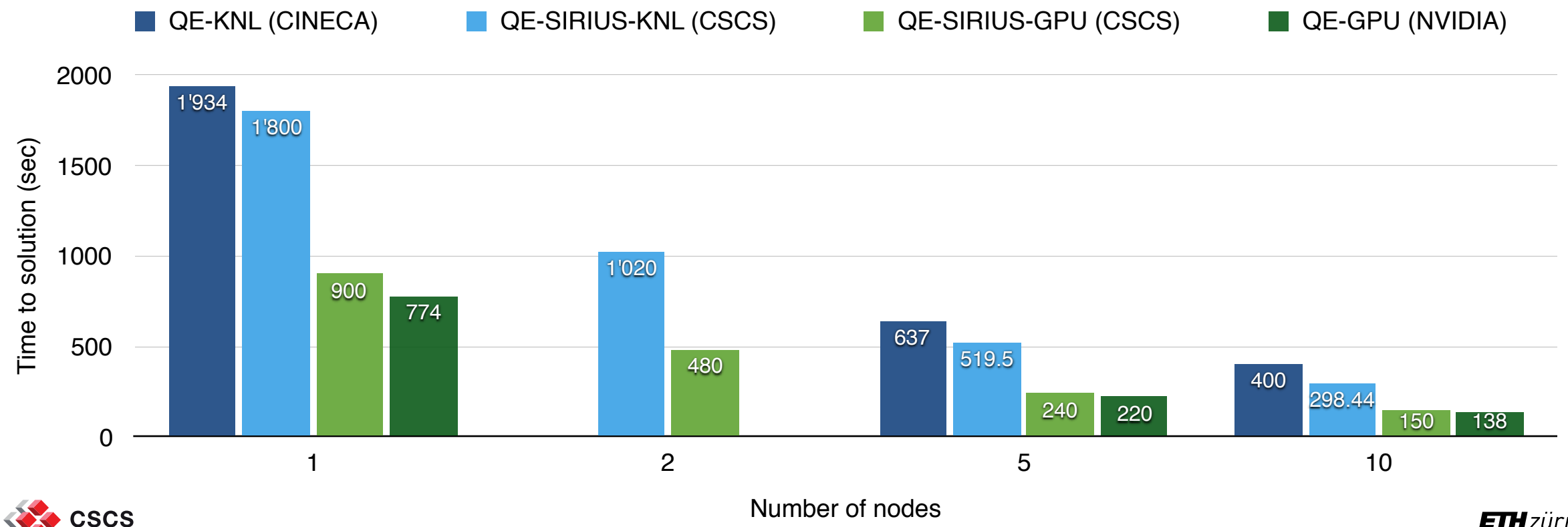
Example of QE/SIRIUS interoperability





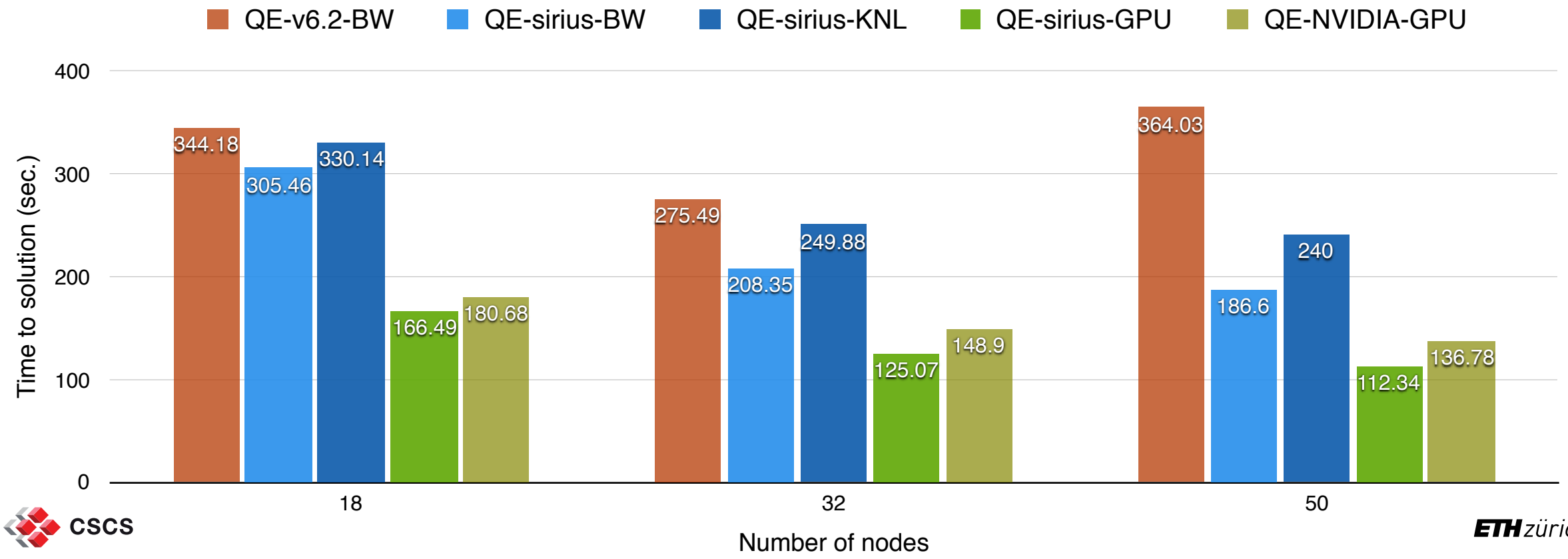
Variable cell relaxation of $Si_{63}Ge$

Performance benchmark of the QE, Cuda Fortran version of QE and SIRIUS-enabled QE codes for the 64-atom unit cell of $Si_{1-x}Ge_x$ The runs we performed on on hybrid nodes with 12-core Intel Haswell @2.5GHz + NVIDIA Tesla P100 card (QE-GPU, QE-SIRIUS-GPU) and on nodes with 68-core Intel Xeon Phi processor @1.4 GHz (QE-KNL). Time for the full 'vc-relax' calculation is reported.



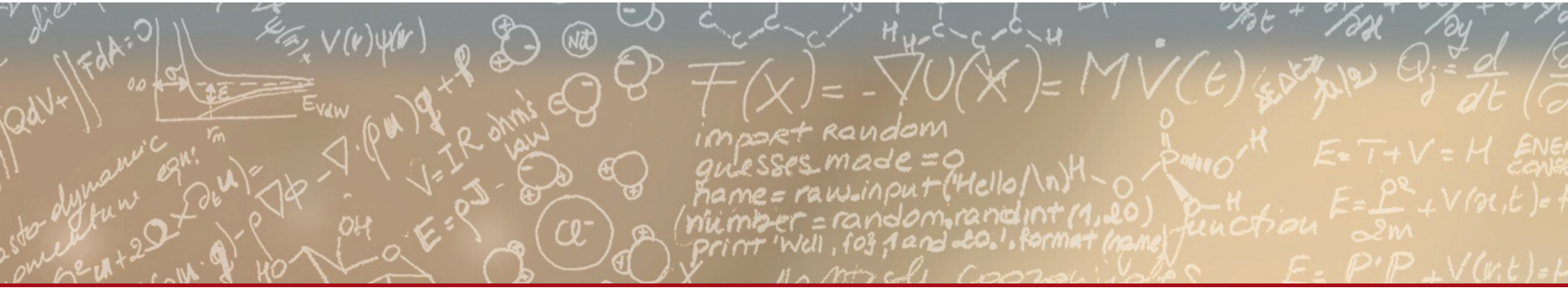
Ground state of Pt-cluster in water

Performance benchmark of the QE and SIRIUS-enabled QE codes for the 288-atom unit cell of Pt cluster embedded in the water. The runs we performed on dual socket 18-core Intel Broadwell @2.1GHz nodes (BW), on hybrid nodes with 12-core Intel Haswell @2.5GHz + NVIDIA Tesla P100 card (GPU) and on nodes with 64-core Intel Xeon Phi processor @1.3 GHz (KNL). ELPA eigen-value solver was used for CPU runs. Time for the SCF ground state calculation is reported.



CSCS

ETH zürich



Thank you for your attention.