High-throughput DFPT

Guido Petretto, Matteo Giantomassi, David Waroquiers, Xavier Gonze, Geoffroy Hautier, Gian-Marco Rignanese

NAPS, Institute of Condensed Matter and Nanosciences, Université catholique de Louvain, 1348 Louvain-la-neuve, Belgium

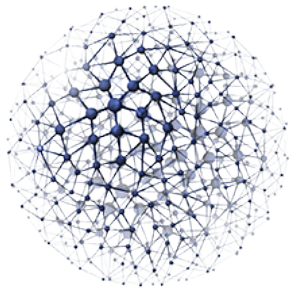
09/05/2017

Outline

Outline

High-Throughput databases

Diffusion of large databases based on DFT calculations



AFLOW

Automatic - FLOW for Materials Discovery

High-throughput frameworks

- Many high-throughput frameworks
  - Mostly focused on ground state calculations
  - No native support for abinit ☹
- What do we need for high-throughput (DFPT)?
  - Input generation
    - Generic input parameters for DFPT
  - Managing complicated workflows
    - Beyond relaxations or band-structures
  - Error handling
  - Insertion in the database
  - Analyze data

Outline

Input generation

pymatgen

Handling and generating abinit inputs based on few input parameters:

- Input objects (`import abipy.abio.inputs`):
  - `class AbinitInput`
  - `class AnaddbInput`
- Factory functions (`import abipy.abio.factories`)

Relax inputs:

```
ion_ioncell_relax_input(structure, pseudos, accuracy="normal",
                        kppa=1000, shift_mode='Monkhorst-pack',
                        smearing="fermi_dirac:0.1 eV",
                        spin_mode="polarized")
```

Input generation

Phonon perturbations inputs, parallelized over the perturbations:

```
phonons_from_gsinput(gs_inp, ph_ngqpt=[6,6,6],
                     with_ddk=True, with_dde=True)
```

- List of `AbinitInput` objects
  - phonon perturbations
  - DDK
  - DDE
  - NSCF to get WFQ if needed
- Calls abinit to determine the list of irreducible perturbations.
- Tags to easily sort out the type of input

Managing workflows

- Many frameworks available
- Scales are important
  - $\Rightarrow$ How many systems? (Tens? Hundreds? Thousands?)

Managing workflows

- Many frameworks available
- Scales are important
  - $\Rightarrow$ How many systems? (Tens? Hundreds? Thousands?)

Managing workflows

- Many frameworks available
- Scales are important
  - ⇒ How many systems? (Tens? Hundreds? Thousands?)

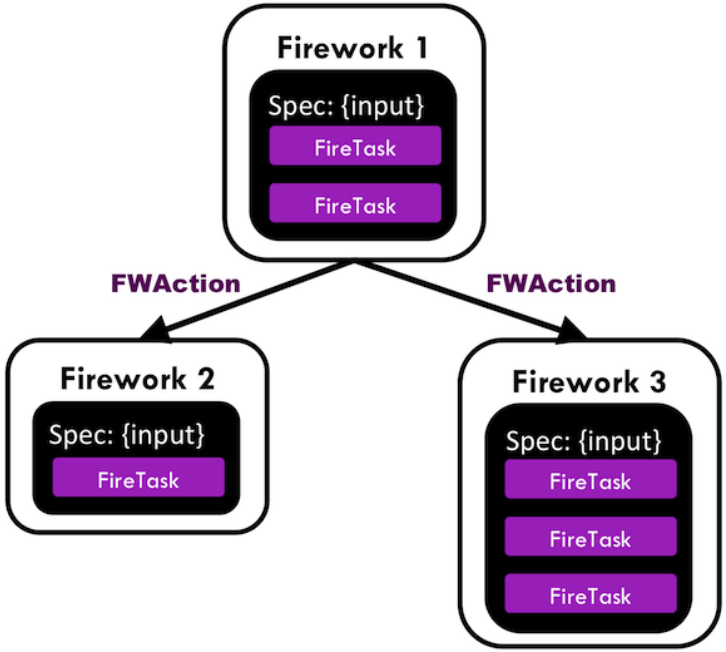
- General purpose workflow manager
- Database backend (MongoDB)
- Support for dynamic workflows
- Support for several queueing systems such as PBS, SLURM...
- Web gui monitor

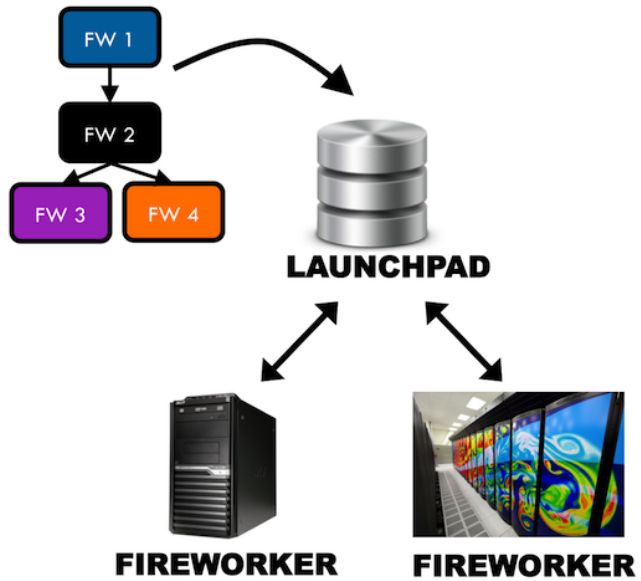
Managing workflows - Fireworks

- Firetask: The basic object defining an action

```
class HelloWorldTask(FiretaskBase)
    def run_task(self, fw_spec):
        print("Hello world!")
```

- Firework: A group of sequential tasks. $1 \leftrightarrow 1$ with a job on a cluster
- Workflow: A collection of fireworks with dependencies



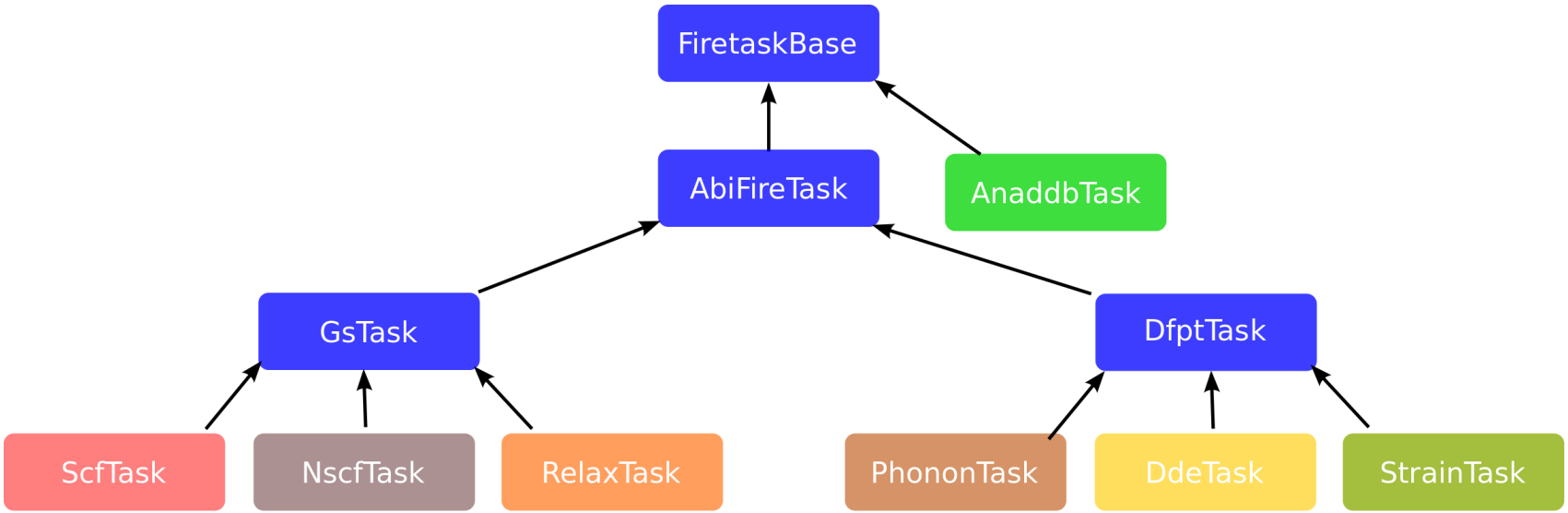


Managing workflows - Abiflows

Abinit + FireWorks = Abiflows

Implementation of `AbinitFireTask` to handle several types of calculations.

Some of the available classes:

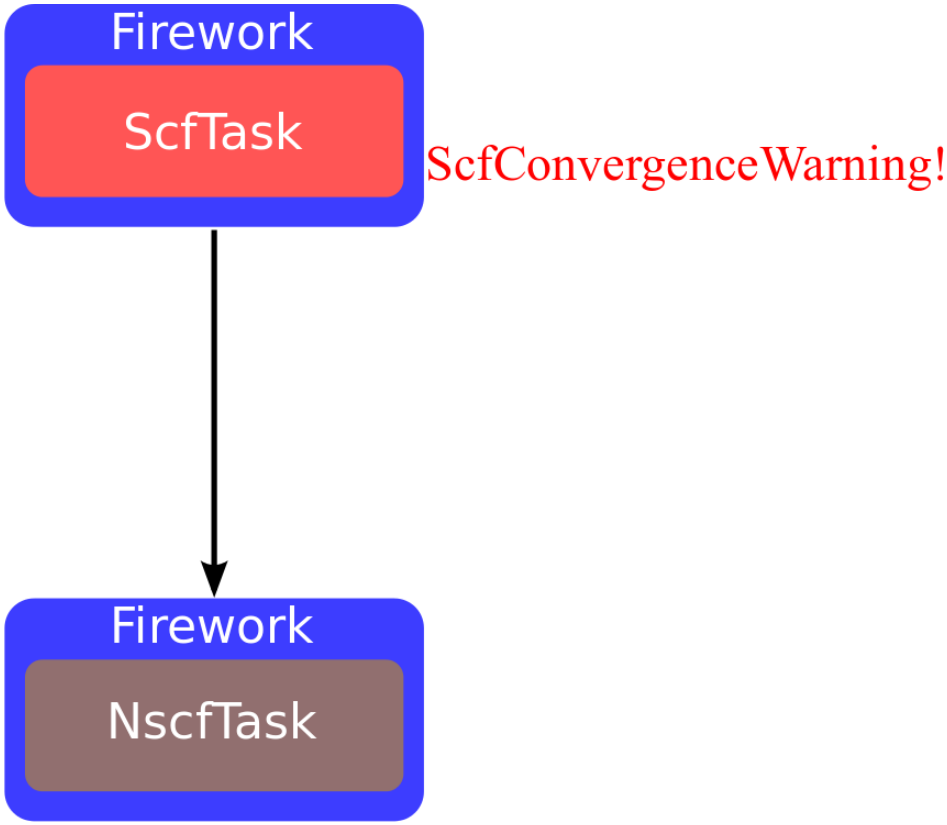


Managing workflows - Abiflows

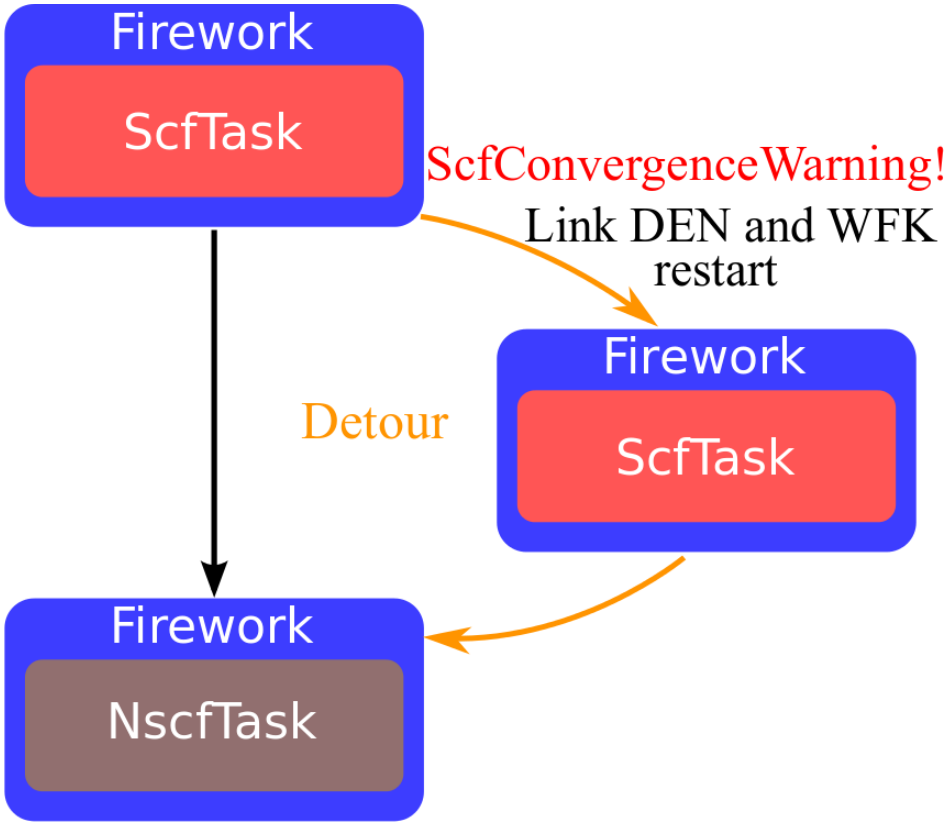
Abiflows relies on Abipy, Abinit and pymatgen providing

- Tasks object to handle different types of calculations
- Automatized choice of number of cores (autoparal)
- Handling of the dependencies (wavefunction, densities, DDB,. . .)
- Workflows generators for common cases
  - Relaxation
  - Band structures
  - Phonons
  - . . .
- Error handling
- Templates for database insertion

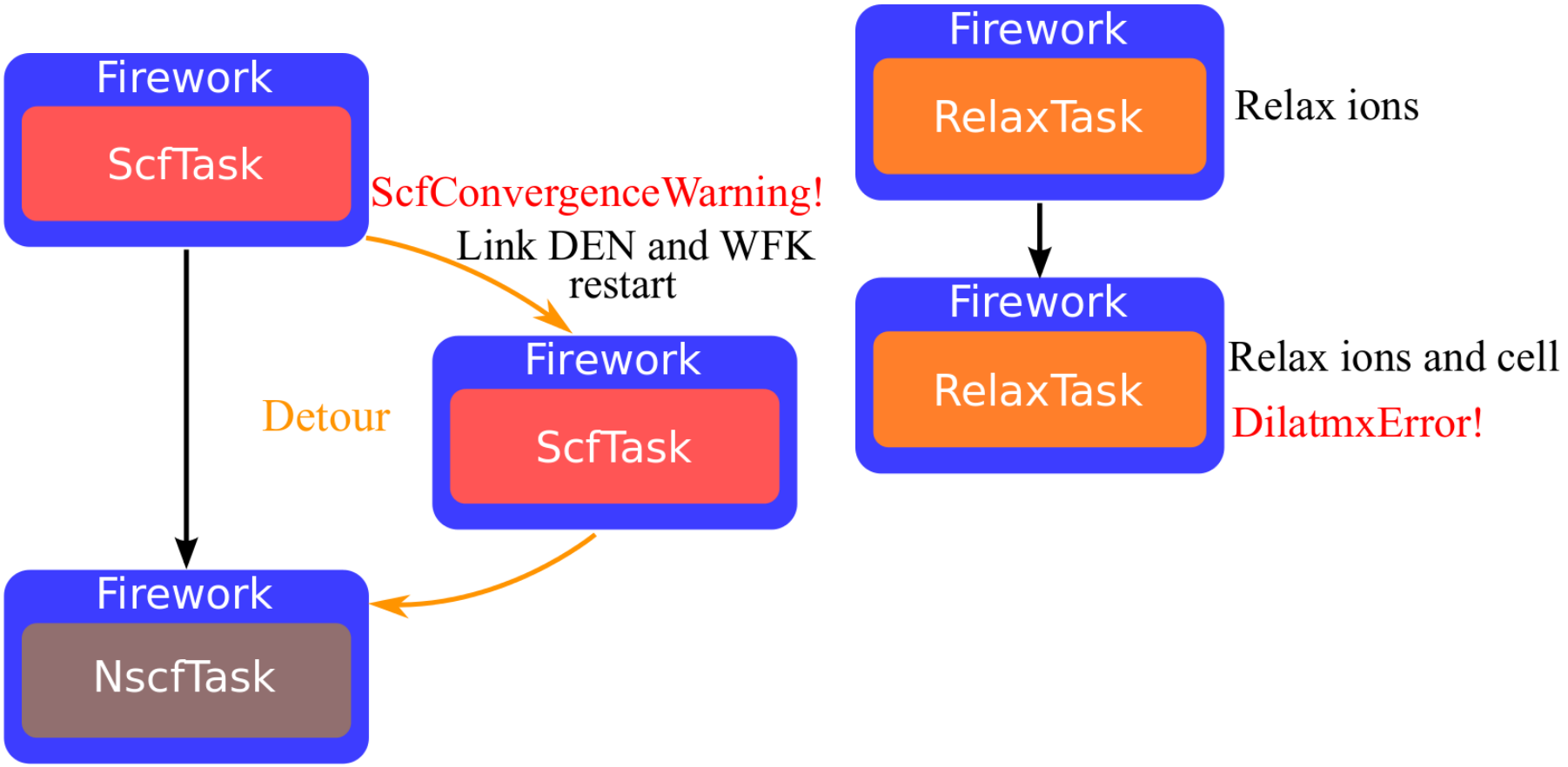
Error handling



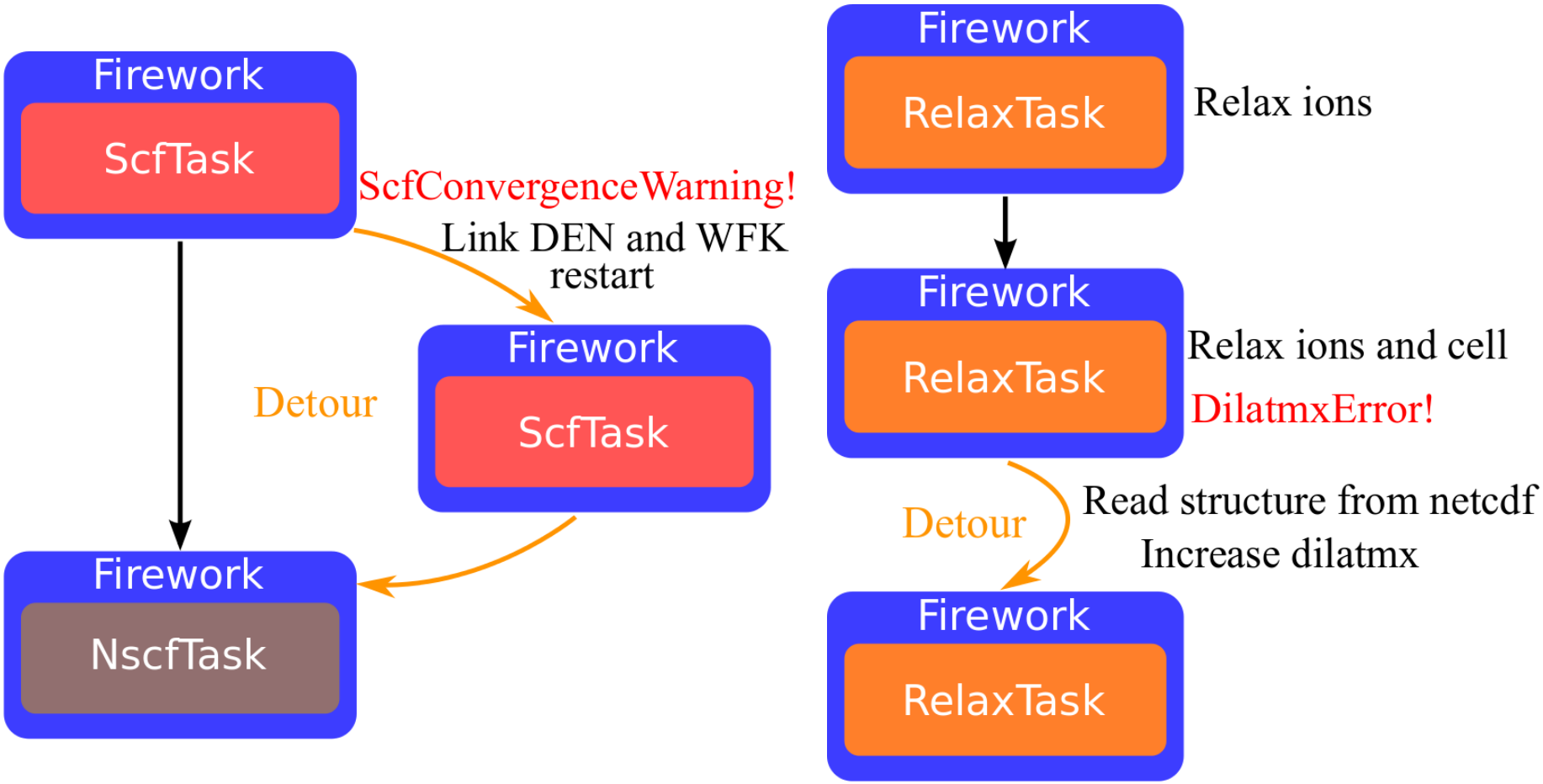
Error handling



Error handling



Error handling



Database insertion

mongoengine

Document-Object Mapper (like ORM, but for document databases)

- Mixins for standard quantities
- Document associated with workflows

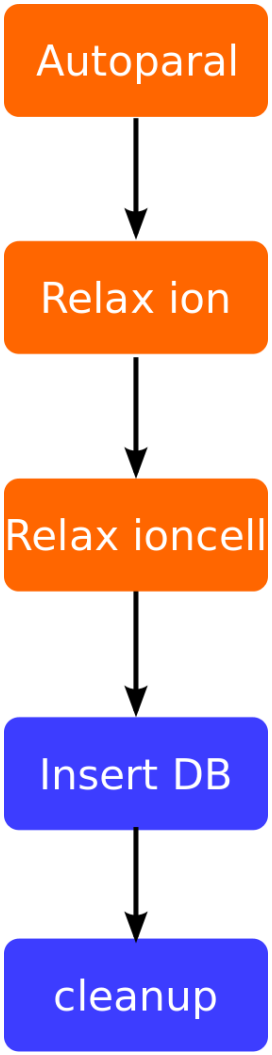
$\Rightarrow$ Standardized output

`abiflows.database.mongoengine` modules:

```
class MaterialMixin(object):
    """
    fields describing the material examined in the calculation
    """
class AbinitPseudoData(EmbeddedDocument):
    """
    fields and function to save abinit pseudopotential data
    """
class PhononResult(MaterialMixin, DateMixin, DirectoryMixin, Document):
    """
    results for a phonon workflow
    """
```

Outline

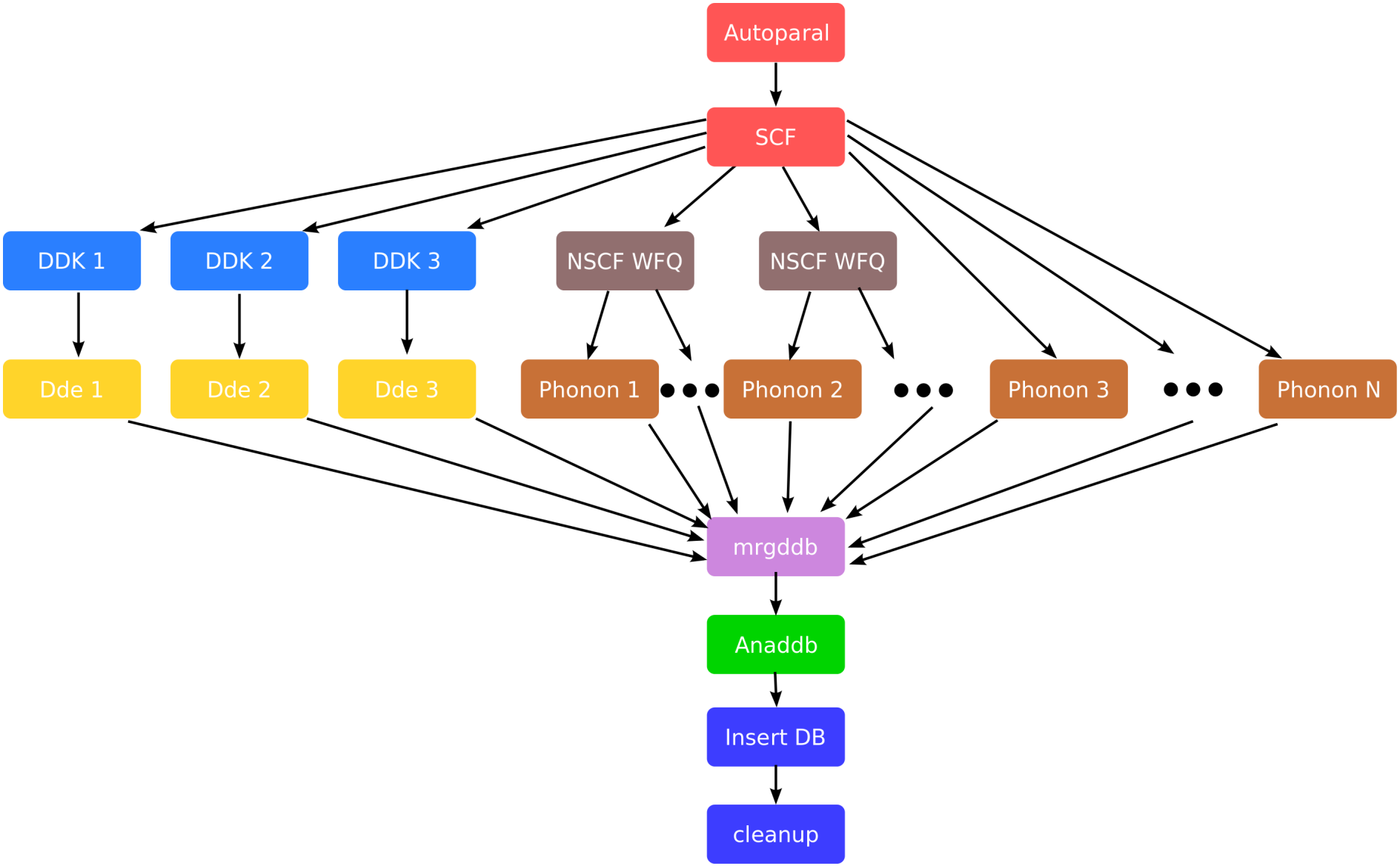
Relax workflow



Split the set of calculations as much as possibile. First relax, then phonons.

- Less complicated workflows
- Easier error recovery
- Save intermediate results

Phonon workflow



Application - Convergence study

Find optimal K-points and Q-points sampling for high-throughput

$\Downarrow$

Set of 48 semiconductors. Calculations for several K and Q grids

Various sizes, crystal symmetries, gaps

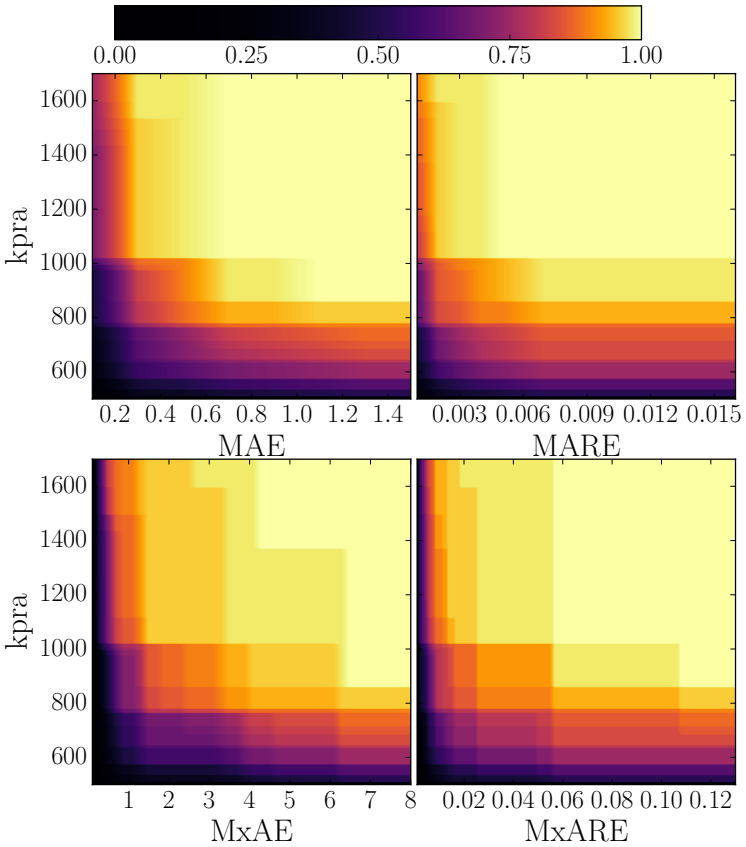
$\Downarrow$

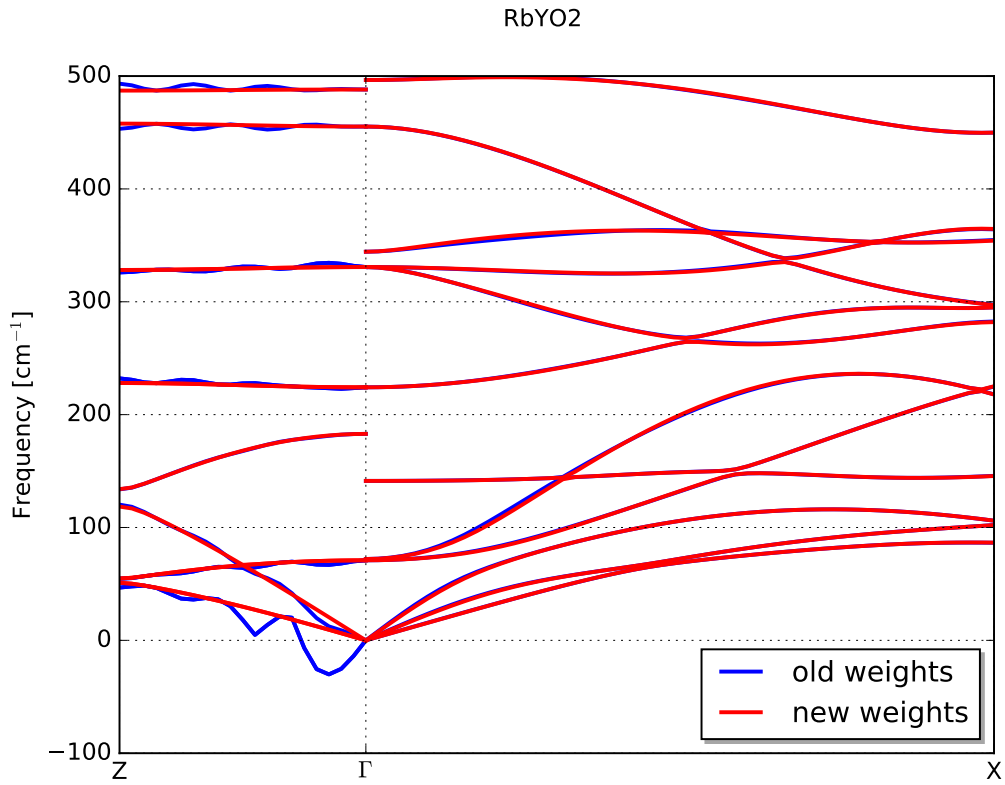
$\backsim$800 workflows run (relax + phonons)

NaLi2Sb	Ca(CdP)2	CdS	SrLiP	InS	GaN	RbYO2
SiO2	BP	AlSb	LiZnP	MgCO3	ScF3	ZnGeN2
LiMgAs	P2Ir	Si	Li3Sb	K2O	Ga3Os	Be3P2
ZnSe	MgO	AgCl	SiC	YWN3	SrO	PbF2
MgSiP2	SiO2	GaP	Be2C	SnHgF6	MgMoN2	ZnO
ZrSiO4	Ba(MgP)2	Ba(MgAs)2	Ca(MgAs)2	C	RbI	FeS2

Convergence study - results

- 1500 points per reciprocal atom $\Rightarrow$ N_{kpt}*N_{atoms} $\backsimeq$ 1500
- Better using a Q-grid commensurable with K-grid
- Improve weights for Fourier interpolation (set `new_wght=1` in m_ifc.F90)





Applications

Using the whole framework in production

- Screening for thermoelectric materials
  - $\Rightarrow$ 270 phonon band structures
- Set of small semiconducting systems
  - Running at NERSC computing center in the framework of the Materials Project
  - $\Rightarrow$ $\backsim$ 600 phonon band structures (and growing)

Data analysis - plotting results

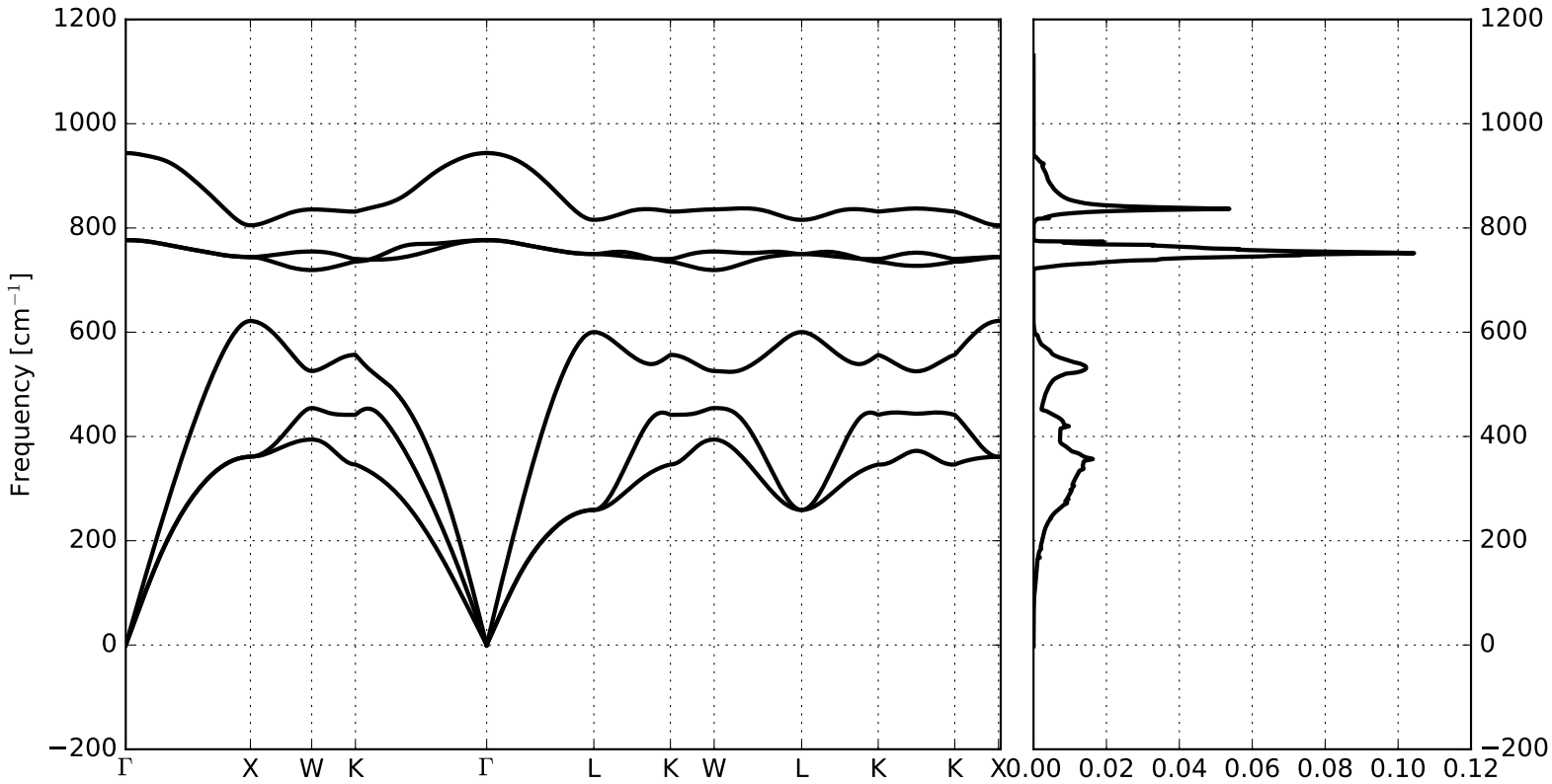
```
r = PhononResult.objects(mp_id='mp-8062')[0] # find result SiC
with r.abinit_output.ddb.abiopen() as ddb: # download ddb and open
    phbst, phdos = ddb.anaget_phbst_and_phdos_files(ngqpt=[8,8,5],
        lo_to_splitting=True, asr=1) # run anaddb
    phb = phbst.phbands # get PhononBands object

phb.plot_with_phdos(phdos.phdos, units='cm-1')
```

Data analysis - plotting results

```
r = PhononResult.objects(mp_id='mp-8062')[0] # find result SiC
with r.abinit_output.ddb.abiopen() as ddb: # download ddb and open
    phbst, phdos = ddb.anaget_phbst_and_phdos_files(ngqpt=[8,8,5],
        lo_to_splitting=True, asr=1) # run anaddb
    phb = phbst.phbands # get PhononBands object

phb.plot_with_phdos(phdos.phdos, units='cm-1')
```

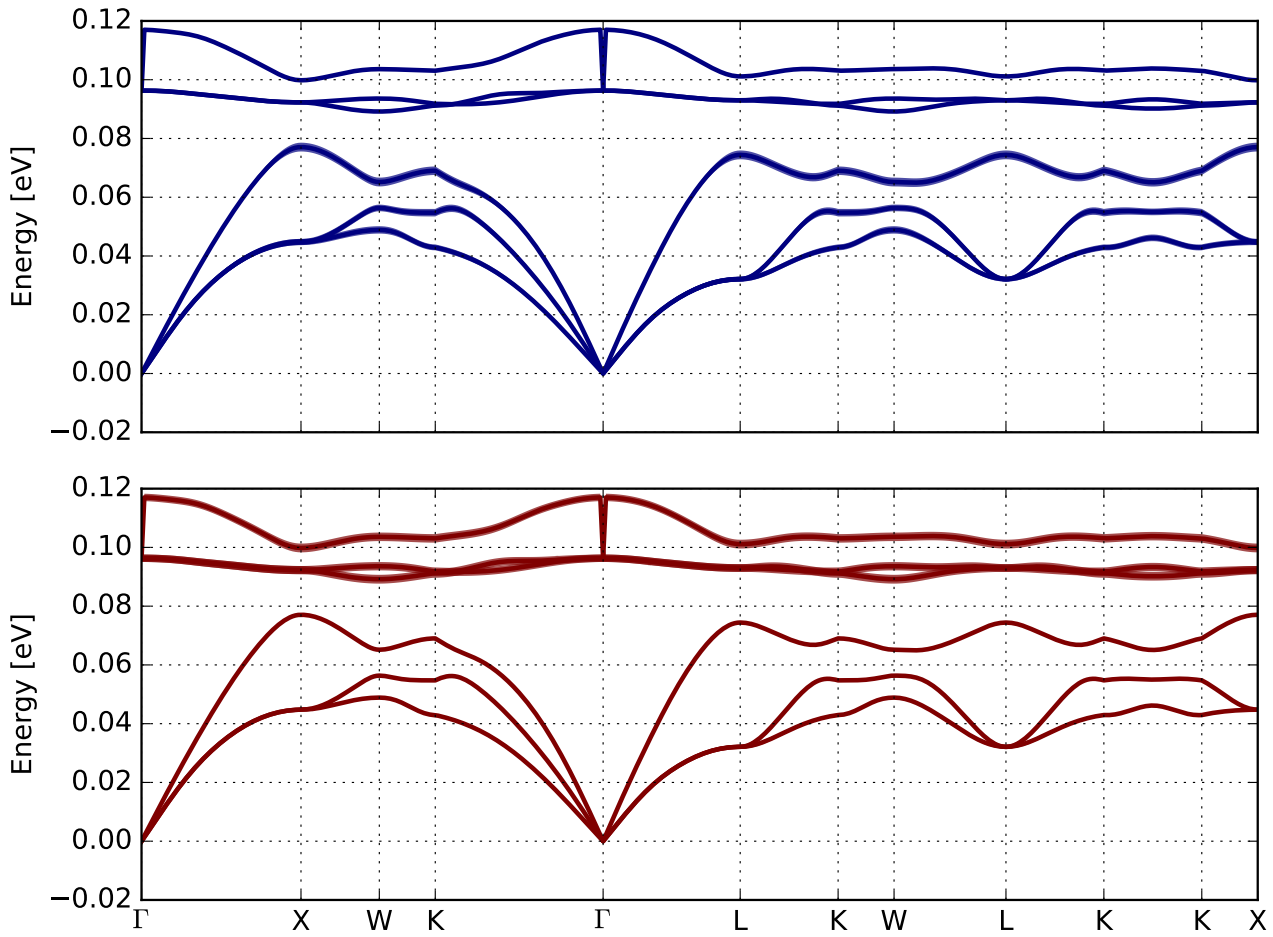


Data analysis - plotting results

```
phb.plot_fatbands(units='eV')
```

Data analysis - plotting results

```
phb.plot_fatbands(units='eV')
```

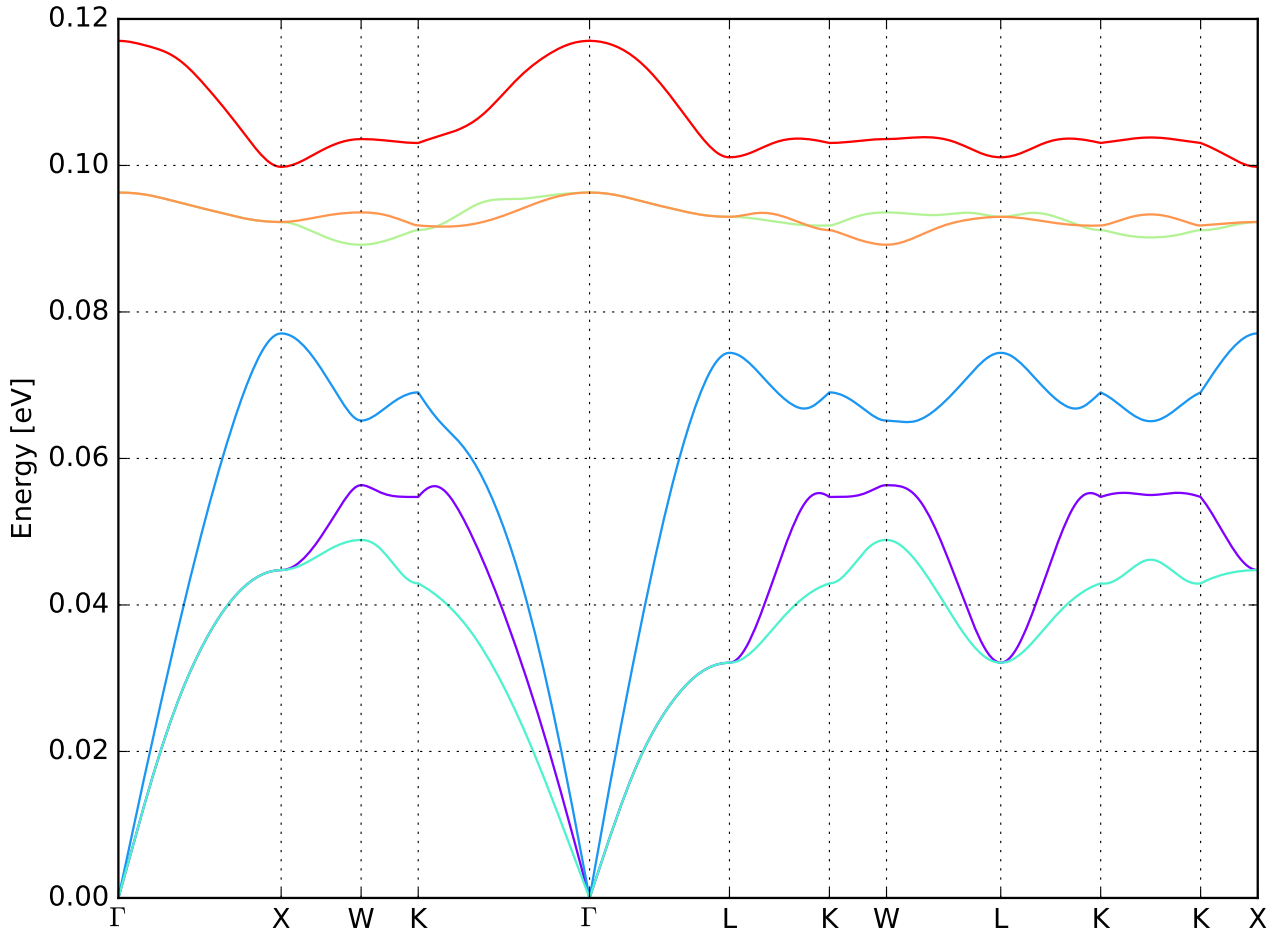


Data analysis - plotting results

```
phb.plot_colored_matched(units='eV')
```

Data analysis - plotting results

```
phb.plot_colored_matched(units='eV')
```



Outline

Conclusion

- Generic high-throughput framework for abinit
  - Automatic input generation
  - Fireworks based framework
  - Error handling
  - Database insertion templates
  - Post-processing tools
- Convergence study for phonons
- Production environment

Thank you for your attention