

# Lines of communication in scientific software

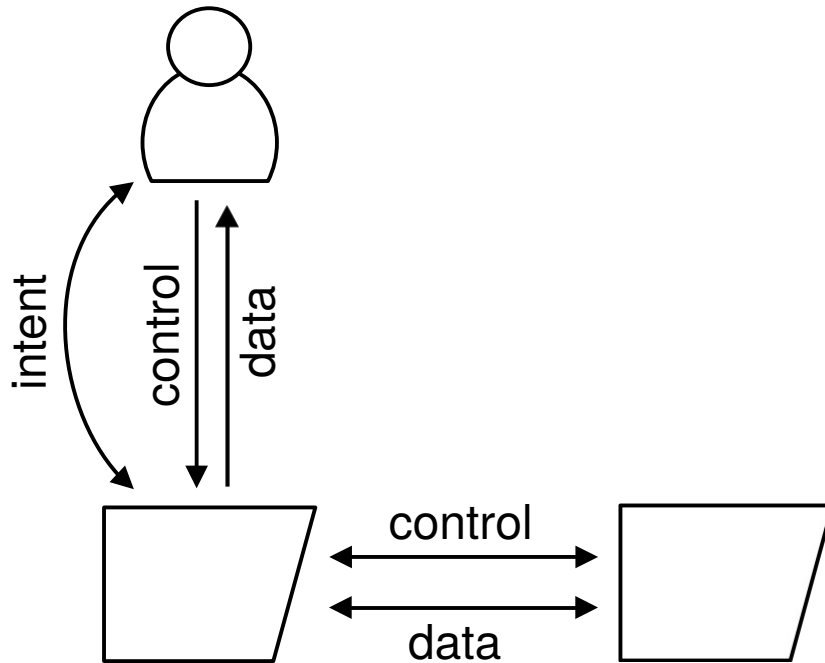
**Markus Wallerberger\***

\* [markus.wallerberger@tuwien.ac.at](mailto:markus.wallerberger@tuwien.ac.at)



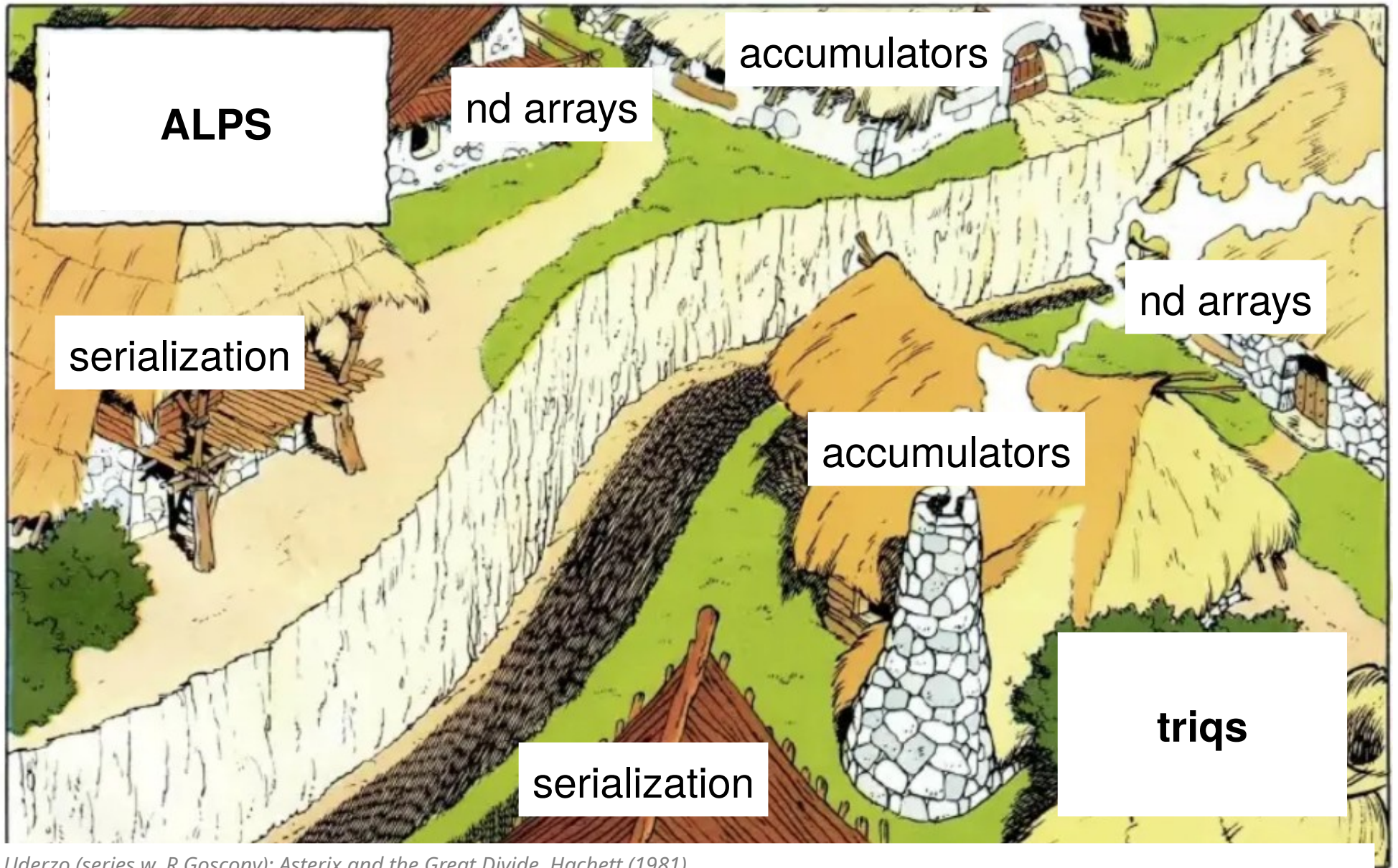
FEEL FREE TO SHARE AND REMIX THESE  
SLIDES UNDER THE CC-BY-SA 4.0 LICENSE

# Lines of communication in scientific software



Research question: How to design & maintain LoCs that are

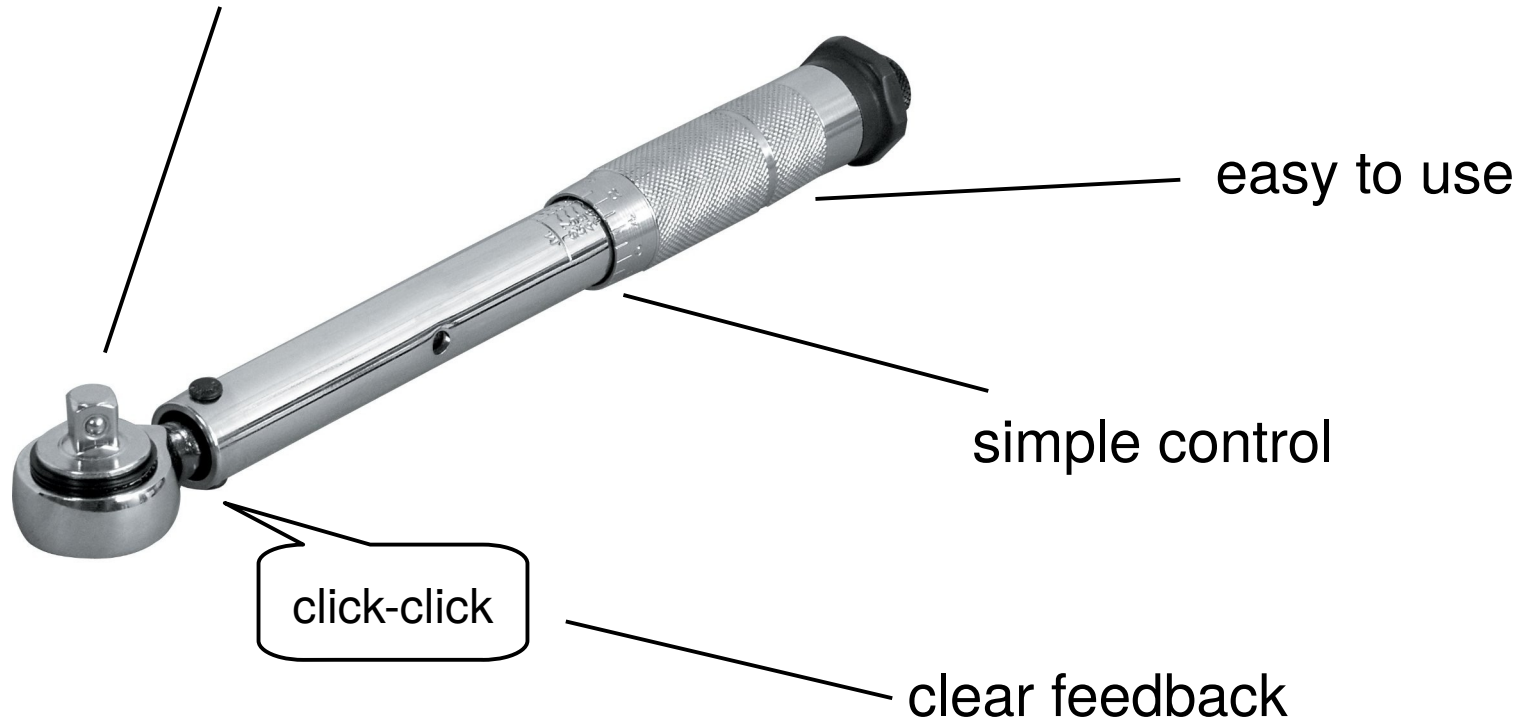
- **Clear:** easy to understand and work with for humans
- **Robust:** stable and unambiguous for computers
- **Efficient:** enough in terms of space and time



[1] A Uderzo (series w. R Goscony): Asterix and the Great Divide, Hachett (1981).

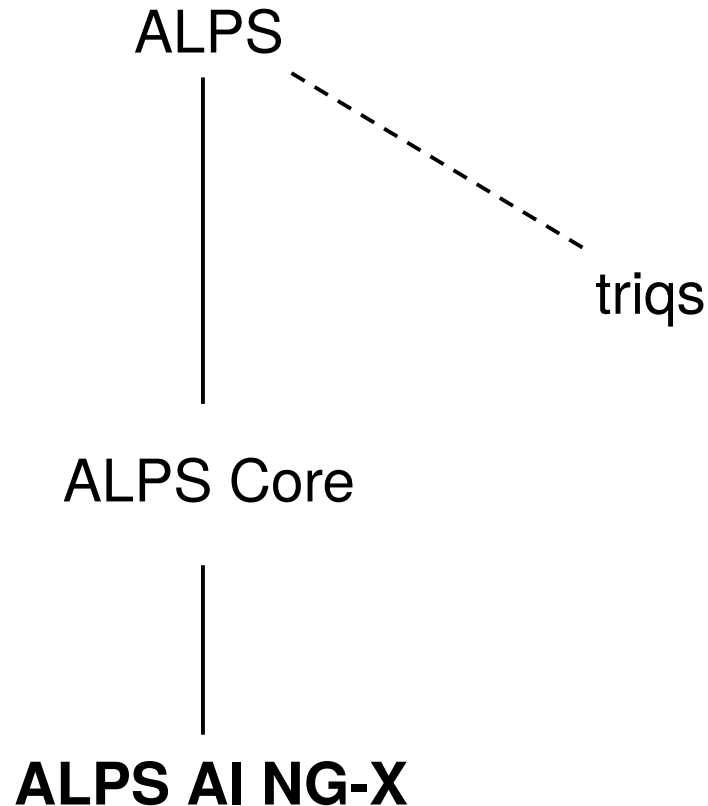
# The torque wrench

robust interface to other tools



That also means: **competition!**

# Why does this matter?



- How do we increase and hold our user base?
- How do we get to a sustainable development & maintenance model?
- How do we stop packages from collapsing under their own weight?

# Case series

- Three codes:
  - ALPS Core: ALEA, HDF5, CMAKE
  - w2dynamics: f2py, log files
  - Sparse-ir: language proliferation, py extension
- Valuable: in depth-analysis
- Dangerous: anecdotal; vulnerable to researcher bias<sup>[1]</sup>

[1] *RETRACTED: AJ Wakefield et al., Lancet 351, 637 (1998); cf. T. S. Sathyanarayana Rao & C. Andrade, Ind. J Psych. 53, 95 (2011).*

# Who am I?

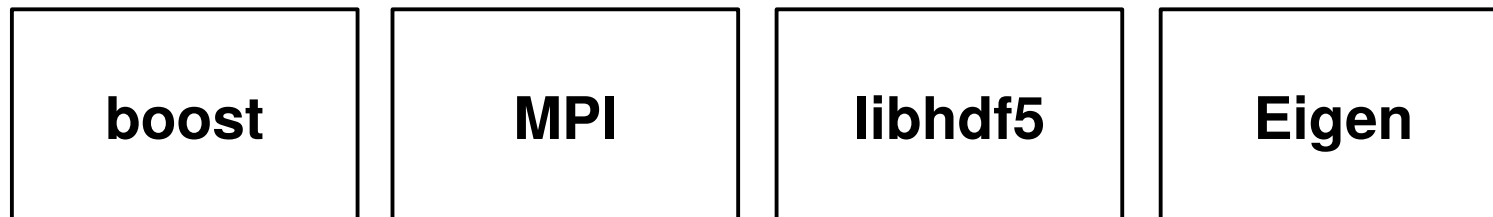
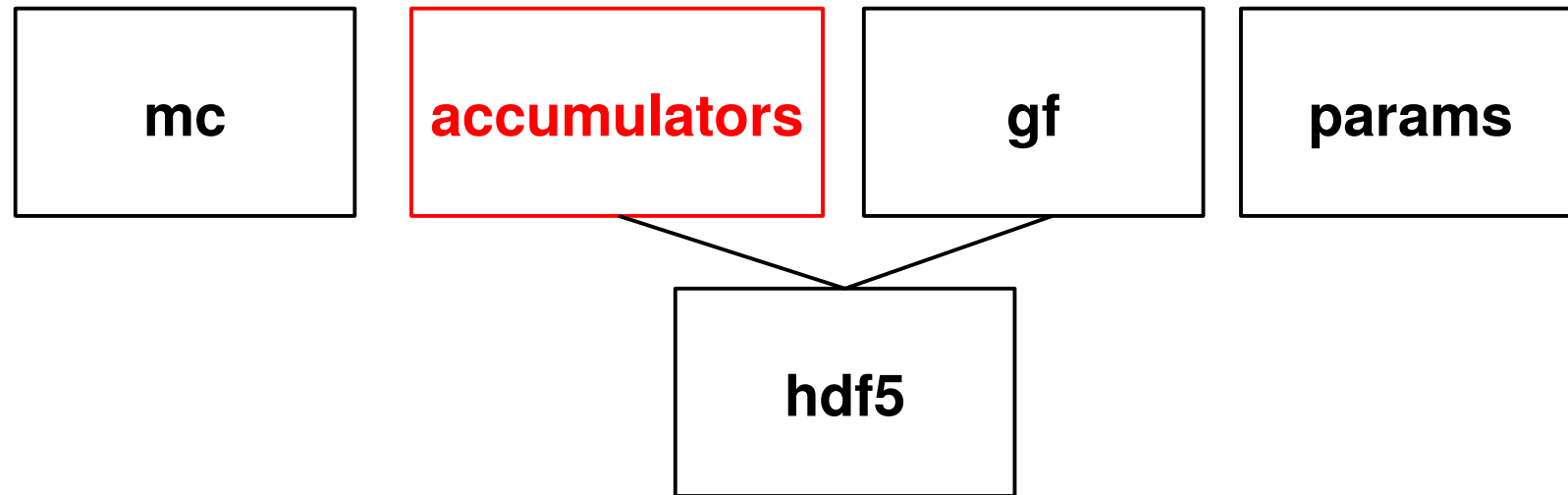
- Senior scientist in computational materials science group
  - ~ crystal grower: infrastructure for or my students and colleagues
- Software
  - **w2dynamics**: primary author, maintainer
  - **ALPS Core**: ALEA, infrastructure work, some maintenance
  - **Sparse-ir**: primary author python, co-maintainer
- Teaching
  - **C++** intro to physics undergrads for 10+ years

# Collaborators / Disclaimer

- **ALPS:** Emanuel Gull, Sergei Iskakov, Igor Krivenko, Alexander Gaenko, ...
- **sparse-ir:** Hiroshi Shinaoka, Samuel Badr, Satoshi Terasaki, ...
- **w2dynamics:** Giorgio Sangiovanni, Karsten Held, Andreas Hausoel, ...
- **However:** The opinions in this talk are mine.



# Case 1: ALPS Core

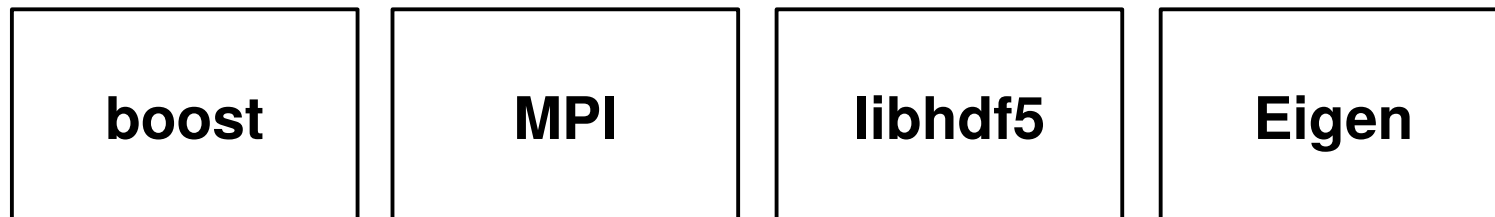
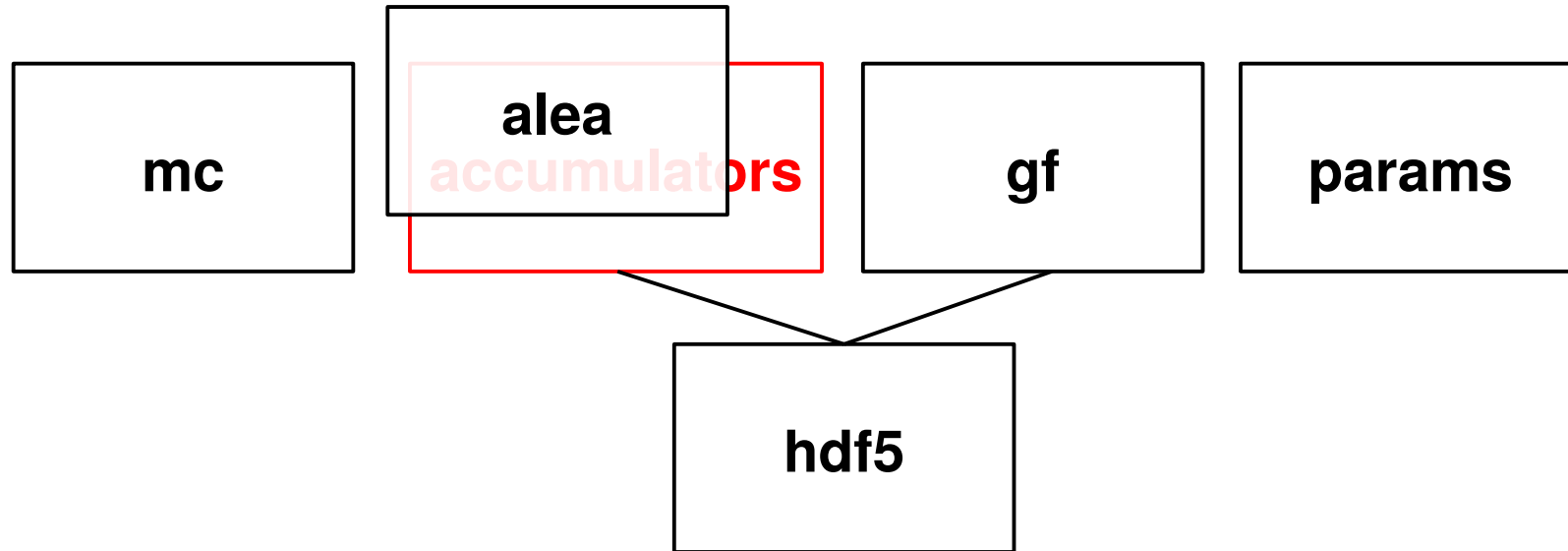


# Statistical analysis of time series

- Autocorrelation estimation:  $\Delta \hat{X}^2 = \frac{\text{Var}(X)(2\tau_X + 1)}{N}$
- Nonlinear error propagation:  $Y = f(X); \quad \Delta \hat{Y} = ?$
- Hypothesis testing:  $H_0 : \langle \hat{X} \rangle = \langle \hat{Y} \rangle$
- Needs to be performed on-line
- Old (alps::accumulators): slow to compile, inefficient, incomplete
- Related (boost::accumulators): slow to compile, incomplete

[1] A Sokal: Monte Carlo method in statistical physics, 1980.

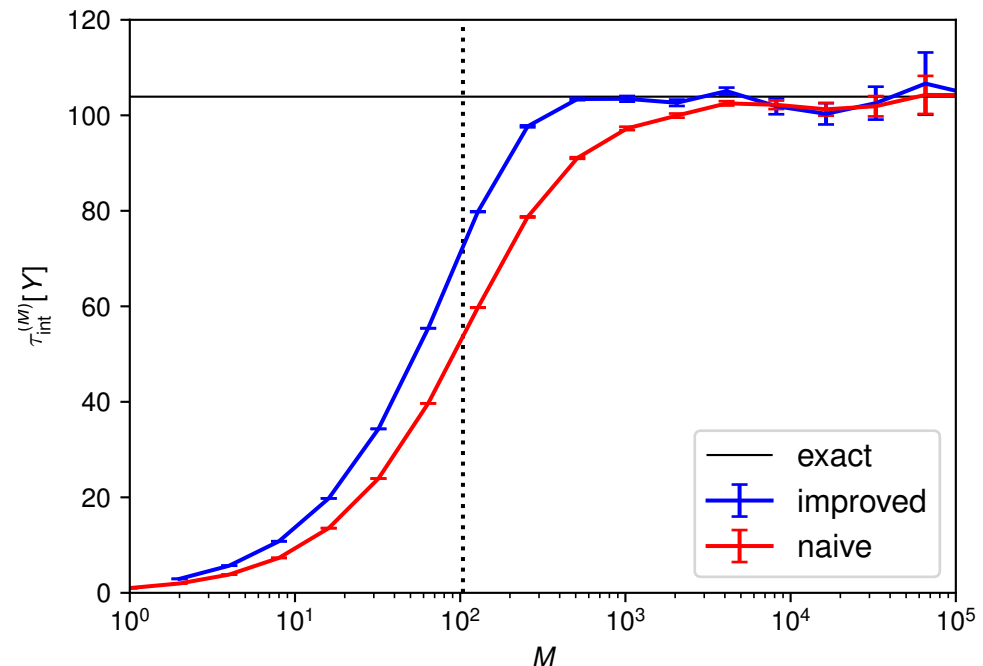
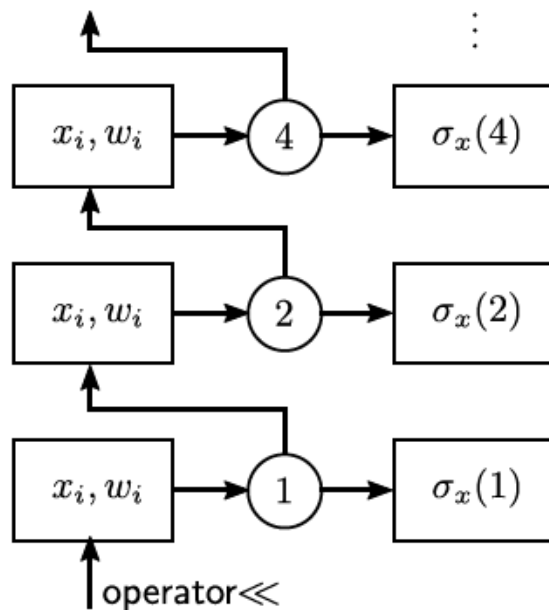
# New statistics library



# Autocorrelation estimation

$$\Delta \hat{X}^2 = \frac{\text{Var}(X)(2\tau_X + 1)}{N}$$

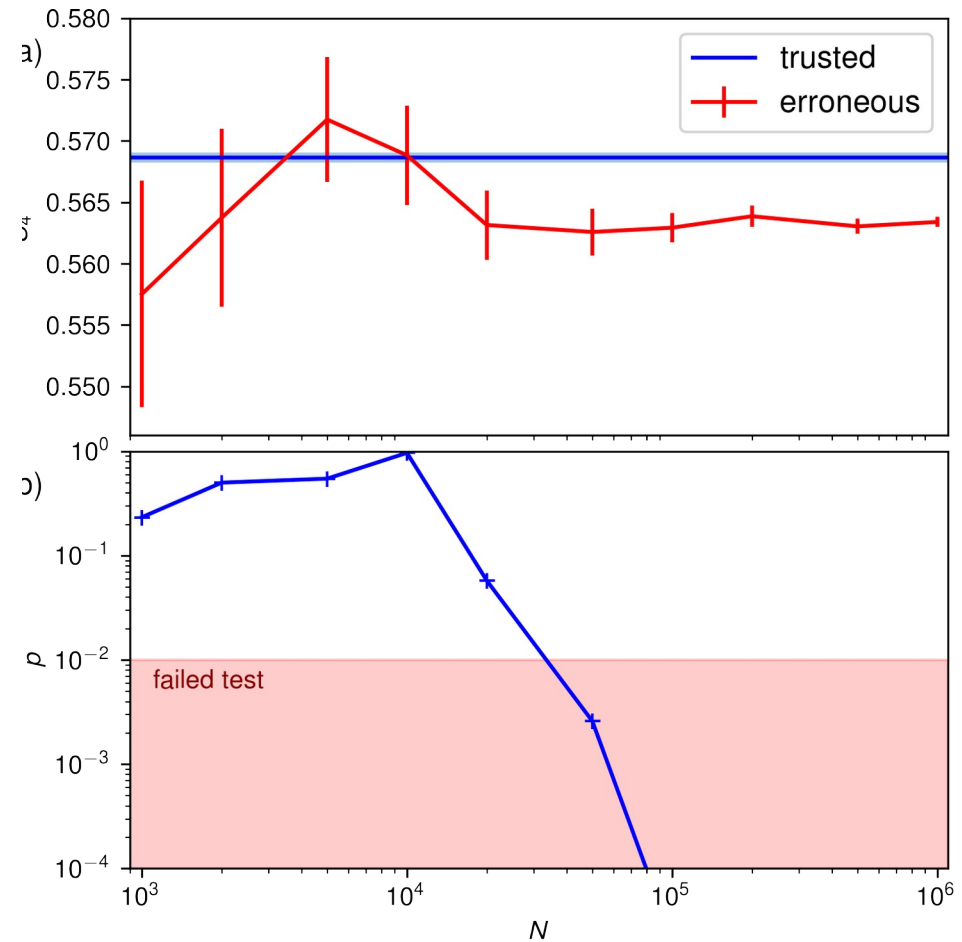
	memory	runtime	bias
alps::accumulators	$k$	$k N \log(N)$	$1 / N$
alps::alea	$k$	<b><math>k N</math></b>	<b><math>\exp(-cN)</math></b>



[1] M.W.: Efficient estimation of autocorrelation spectra, arXiv:1810.05079 (2018)

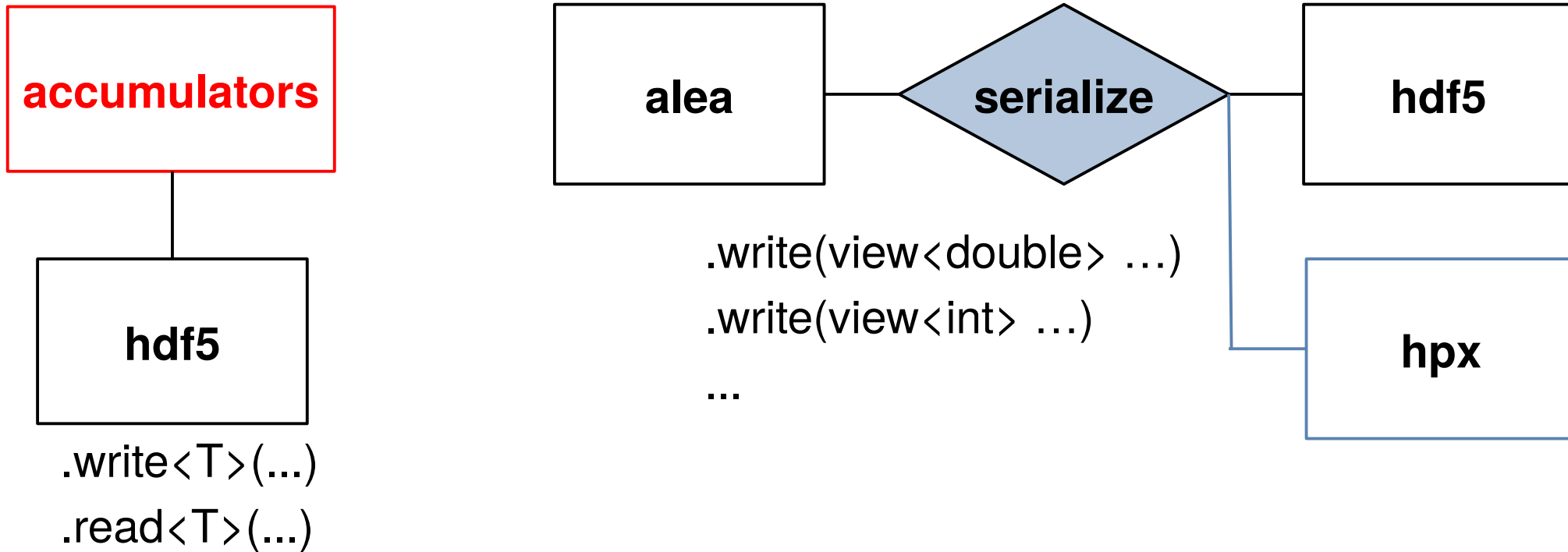
# Hypothesis testing

- Compare statistical results
  - stochastic unit tests
  - convergence criterion
- Hypotheses  $H_0 : \langle \hat{X} \rangle = \langle \hat{Y} \rangle$
- Hotelling  $T^2$  test  $\rightarrow p$  score



[1] M.W and E. Gull: Phys. Rev. E 96, 053303 (2017)

# [1A] Fixing compile times



# Virtual serialization interface

- Allows compilation of HDF5 shim
  - `.write<T>` is anyway overgeneralized: T must be HDF5 type
  - Virtual function call cost << filesystem write cost
- Directly enabled further implementations
- Design wart: struct ndview collecting pointer w/o clear ownership
  - `write(ndview<double> data);`
  - `write(const double *data, const size_t *dims, int ndim);`

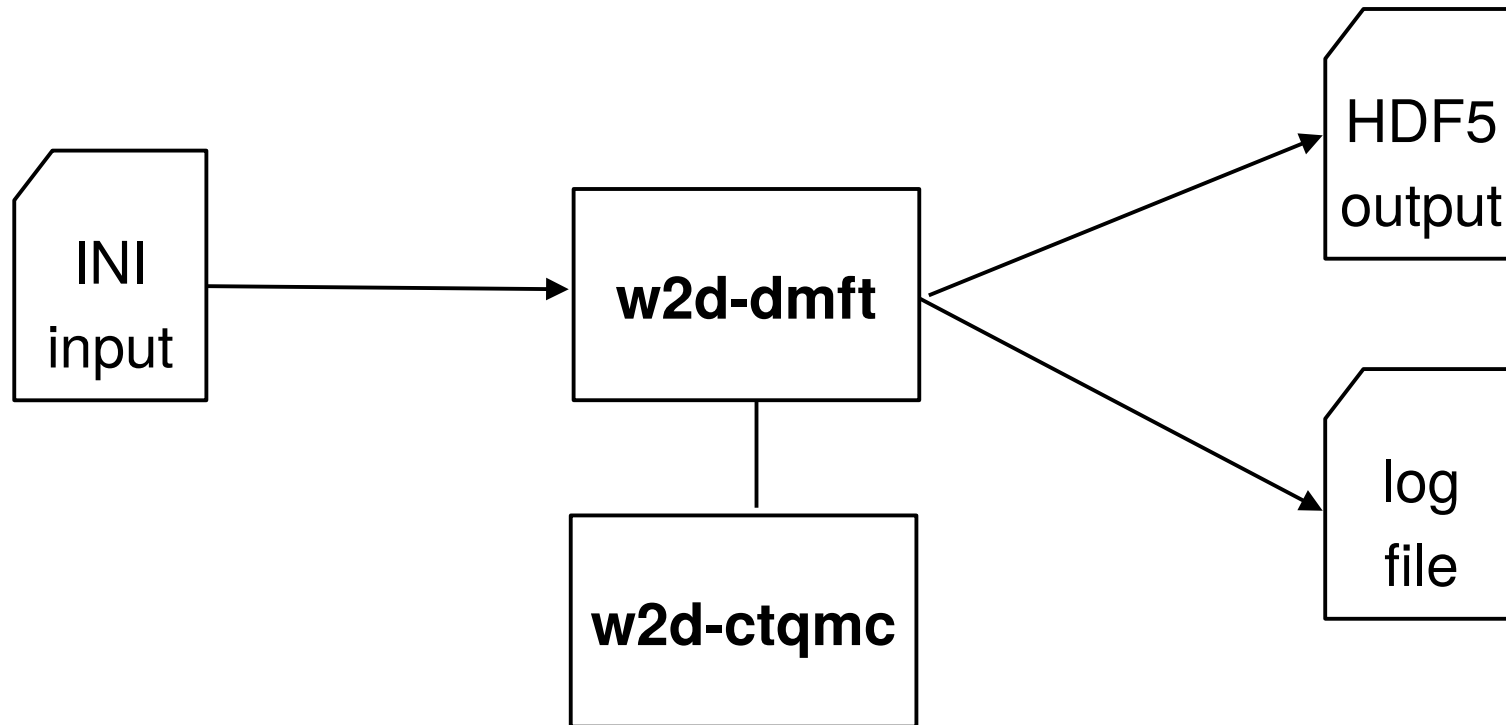
# [1B] Potential user: w2dynamics

- Diagrammatic Monte Carlo in dire need of statistics
- Written in Fortran 2003 → call C functions through `iso_c_binding`
- No inline functions; **no C++ since no stable ABI**
- Shim layer? Brittle deployment: ALPScore + shim layer + w2dyn
- Rewrite Monte Carlo in C++? ~100'000 LOC
- → Reimplemented relevant parts in Fortran

Even though I wrote the library  
and wrote a code that needed the functionality the  
library provided  
I did not – could not – use the library.



## Case 2: w2dynamics



# Diagrammatic solver for strong el. correlation

- Green's function = 1-electron propagator

$$G_{rr'}^R(t - t') = \frac{i}{\hbar} \langle [c_r(t), c_{r'}^\dagger(t')]_{\mp} \rangle \Theta(t - t')$$

- Dynamical mean field theory = local dynamics given by auxiliary problem

$$G_{rr}(t) = G_{imp}(t)$$

- Continuous-time quantum Monte Carlo for auxiliary problem

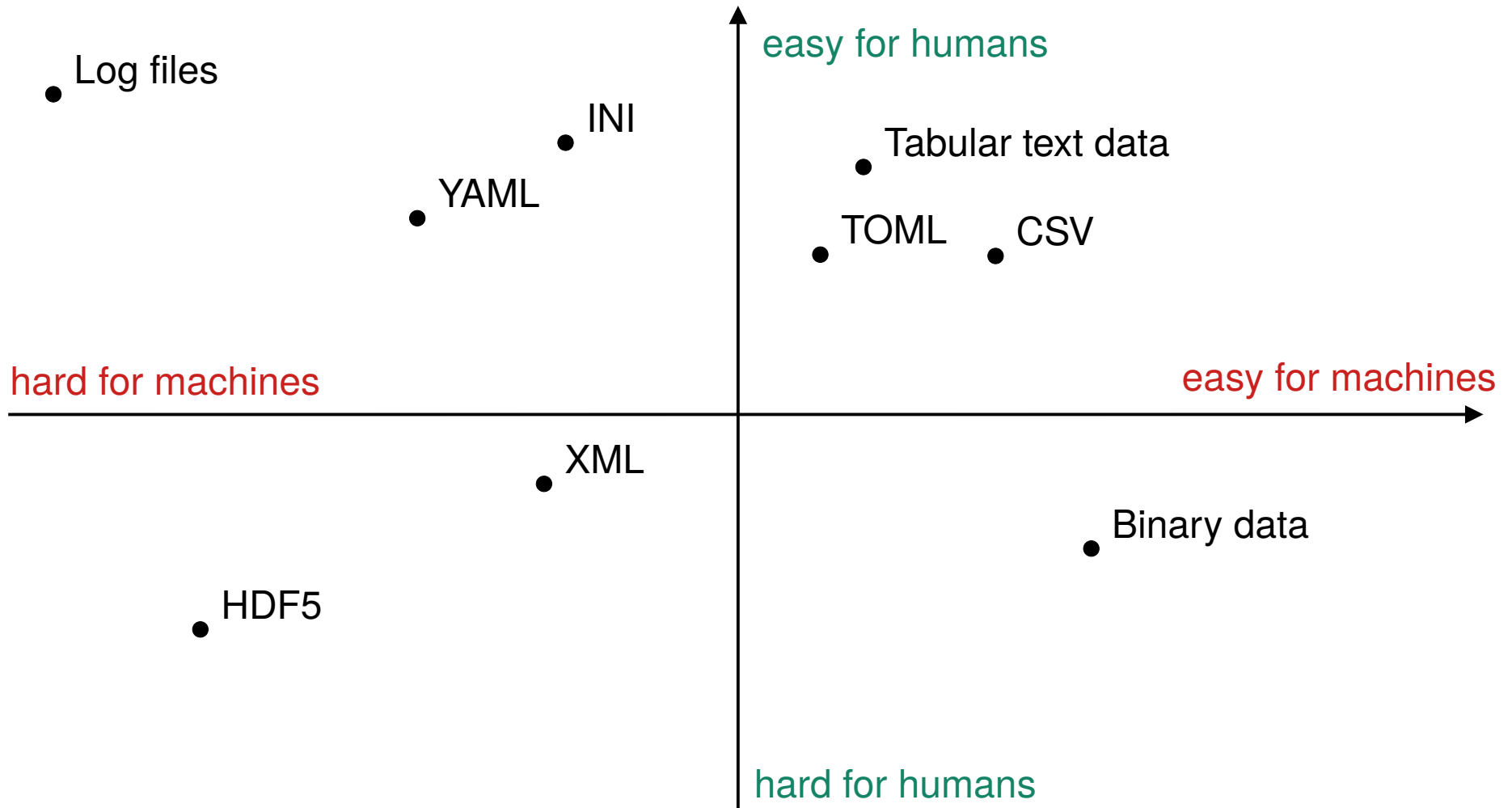
$$Z = \sum_{MC} \{ \text{all strong-coupling diagrams} \}$$
$$G_{imp}(t) = \frac{1}{Z} \frac{\partial Z}{\partial \Delta(-t)}$$

[1] M.W., et al.: *Comput. Phys. Commun.* 235, 388 (2019)

## [2A] Obscure outputs

- Huge spec → libhdf5 only complete implementation (of the spec?)
- Archives: Not for human consumption
  - H5tools, scripts, or notebooks for extracting data (→ hgrep)
  - Hard or impossible to alter (deleting datasets?)
  - Cluster admins like it: fewer small files
- Work space: performance problems
  - Memmapping: crucial for large dataset, only supported recently
  - Parallel access: brittle, partial support, slow

# Parsing



# Problematic computation – Logfile

Constructing diagram ...

Initialize sampling ...

Simulation ...

\*\*\*\*\*

```
=====
```

MOVE	PROPOSALS	ACCEPTED	ILLEGAL
Add pairs 1 .....	35.624 %	25.11 %	0.00 %
Add pairs 2 .....	8.892 %	5.60 %	0.00 %
Remove pairs 1 .....	35.596 %	25.15 %	66.39 %
Remove pairs 2 .....	8.910 %	0.00 %	94.25 %
Shift tau .....	7.920 %	100.00 %	0.00 %
Switch outer in empty ....	2.049 %	100.00 %	0.00 %
Permute flavours .....	1.009 %	71.62 %	21.00 %

```
=====
```

<n> = 1.001136 (exact = 1.000000)

<(n - <n>)<sup>2</sup>> = 0.165397 (exact = 0.166724)

sign = [[ -1.000E+00]]

rho = [

[ 1.648E-01, -0.000E+00, -0.000E+00, -0.000E+00],

[ -0.000E+00, 3.334E-01, -0.000E+00, -0.000E+00],

[ -0.000E+00, -0.000E+00, 3.358E-01, -0.000E+00],

[ -0.000E+00, -0.000E+00, -0.000E+00, 1.660E-01]]

## [2B] Extension module generators

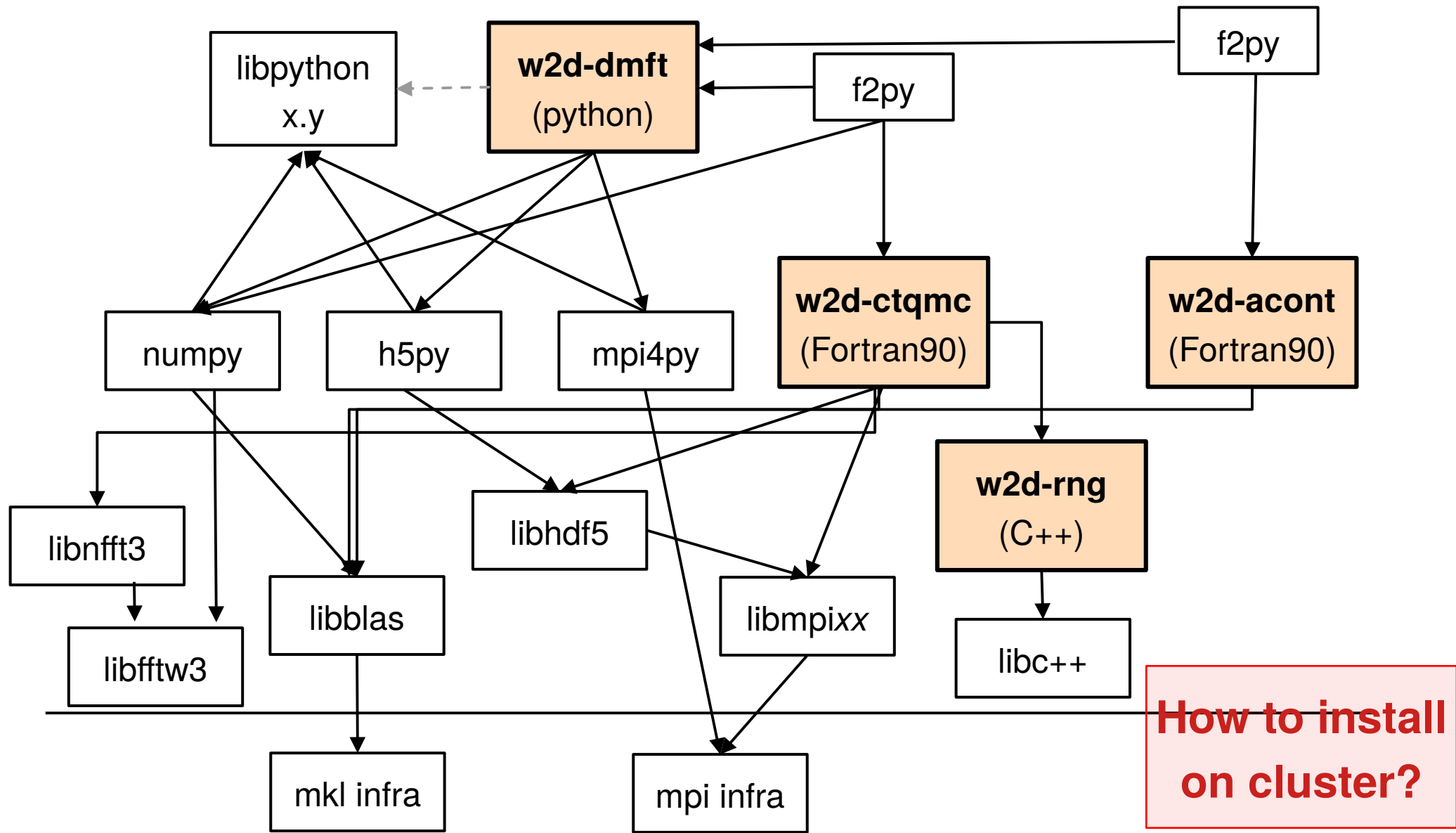
- Mostly conceptual challenges
- Varied dependencies & setups

**w2d-dmft**  
(python)

```
graph TD; A[w2d-dmft (python)] --- B[w2d-ctqmc (Fortran90)];
```

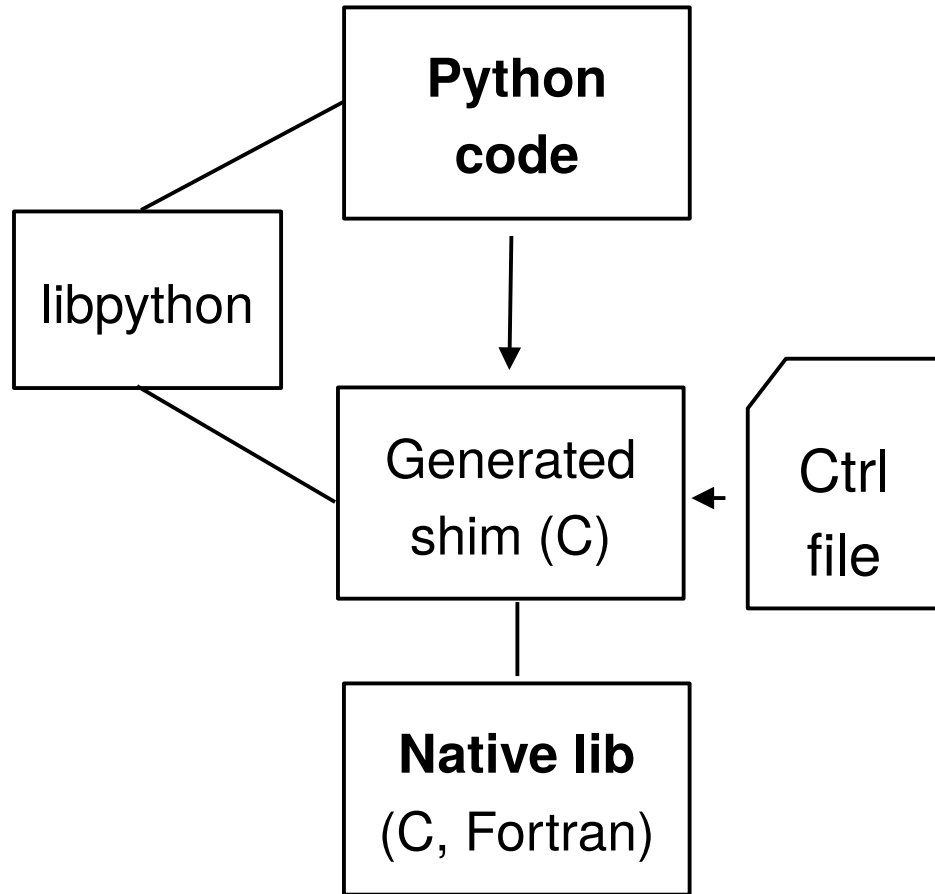
**w2d-ctqmc**  
(Fortran90)

- Mostly computational challenges
- Similar problems



**How to install on cluster?**

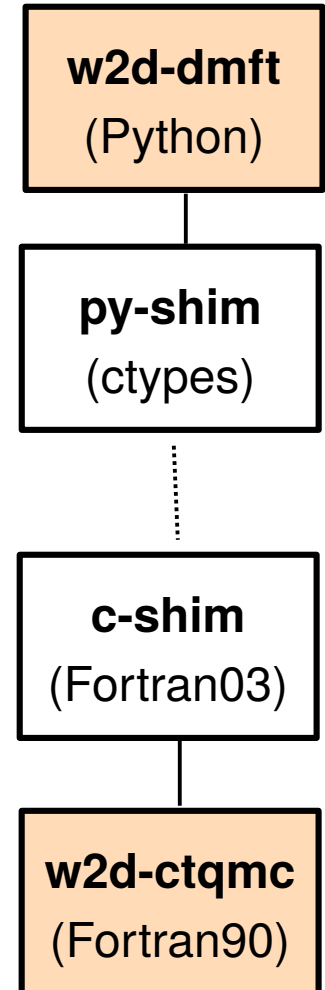
# Extension module generators



- Python – C/C++: SWIG; triqs/cpp2py
- Python – Fortran: f2py
- Links everything into one dependency ball through libpython
- Targets only subset of compiled language
- Lifetime model (often w/ manual intervention)

# Foreign Function interfaces

- Python can call C functions by default (ctypes, cffi)
- Fortran 2003 can expose functions with C ABI (iso\_c\_binding)
- Python FFI opens library, translates arguments, calls function → **no static dependency**
- Object lifetime management done in python

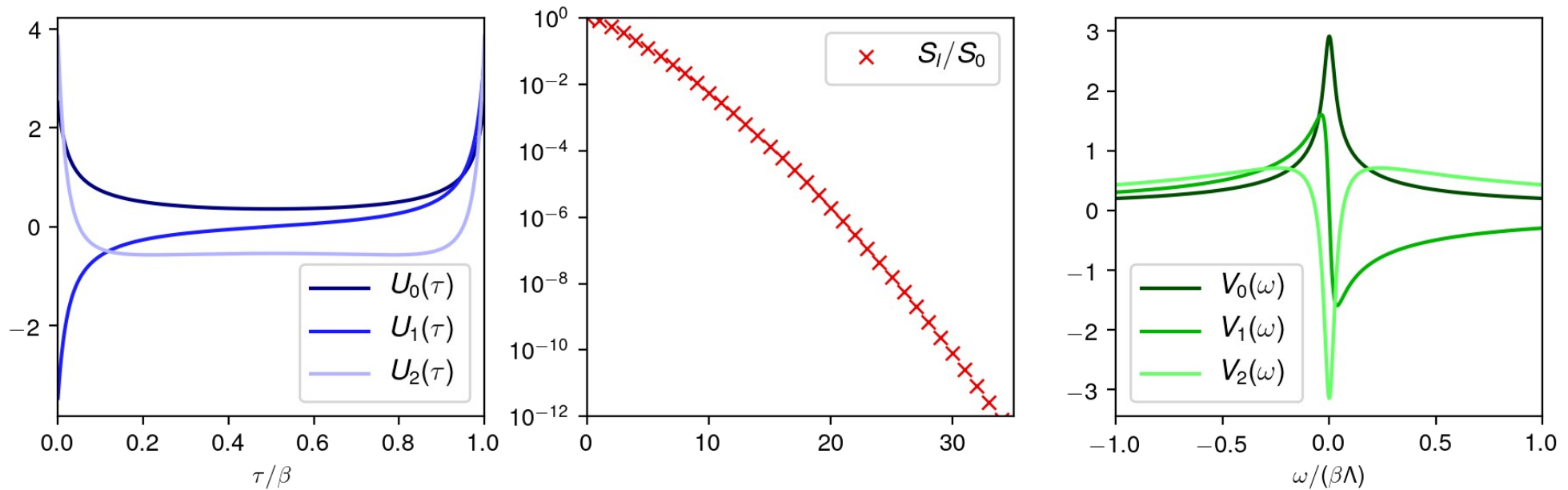


[1] L. Crippa., ..., M.W., et al., arXiv:2506.01363v1 (2025)

# Case 3: sparse-ir Singular Value Expansion<sup>[1,2]</sup>

$$G(\tau) = \int_{-W}^W d\omega K(\tau, \omega) \rho(\omega)$$

$$K(\tau, \omega) = \sum_{I=0}^{\infty} U_I(\tau) \times S_I \times V_I(\omega)$$



[1] J. Otsuki et al., *Phys. Rev. E* 95, 061302 (2017); H. Shinaoka et al., *Phys. Rev. B* 96, 035147 (2017)

- Insert SVD: 
$$G(i\nu) = \int_{-W}^W d\omega K(i\nu, \omega) \rho(\omega)$$

$$= \sum_{l=0}^{\infty} U_l(i\nu) S_l \int_{-W}^W d\omega V_l(\omega) \rho(\omega)$$

- Exponential decay: 
$$S_l \sim \exp\left(-\alpha \frac{l}{\log(\beta W)}\right)$$

- Saturation: 
$$\rho_l \sim \text{const}$$

- “a priori PCA”  
= IR Basis:

$$G(i\nu) = \sum_{l=0}^L G_l U_l(i\nu) + \epsilon_L \quad \text{with} \quad \epsilon_L \sim S_L$$

$$L \sim \log(\beta W) \log(\epsilon^{-1})$$

[1] J. Otsuki, H.S., et al., *Phys. Rev. E* 95, 061302 (2017)

[2] H.S., ..., *Phys. Rev. B* 96, 035147 (2017)

[3] Jia Li, M.W., ..., *Phys. Rev. B* 101, 035144 (2020)

[4] M.W., H.S., ..., *SoftwareX* (2023)

## [3a] Multi-language challenges

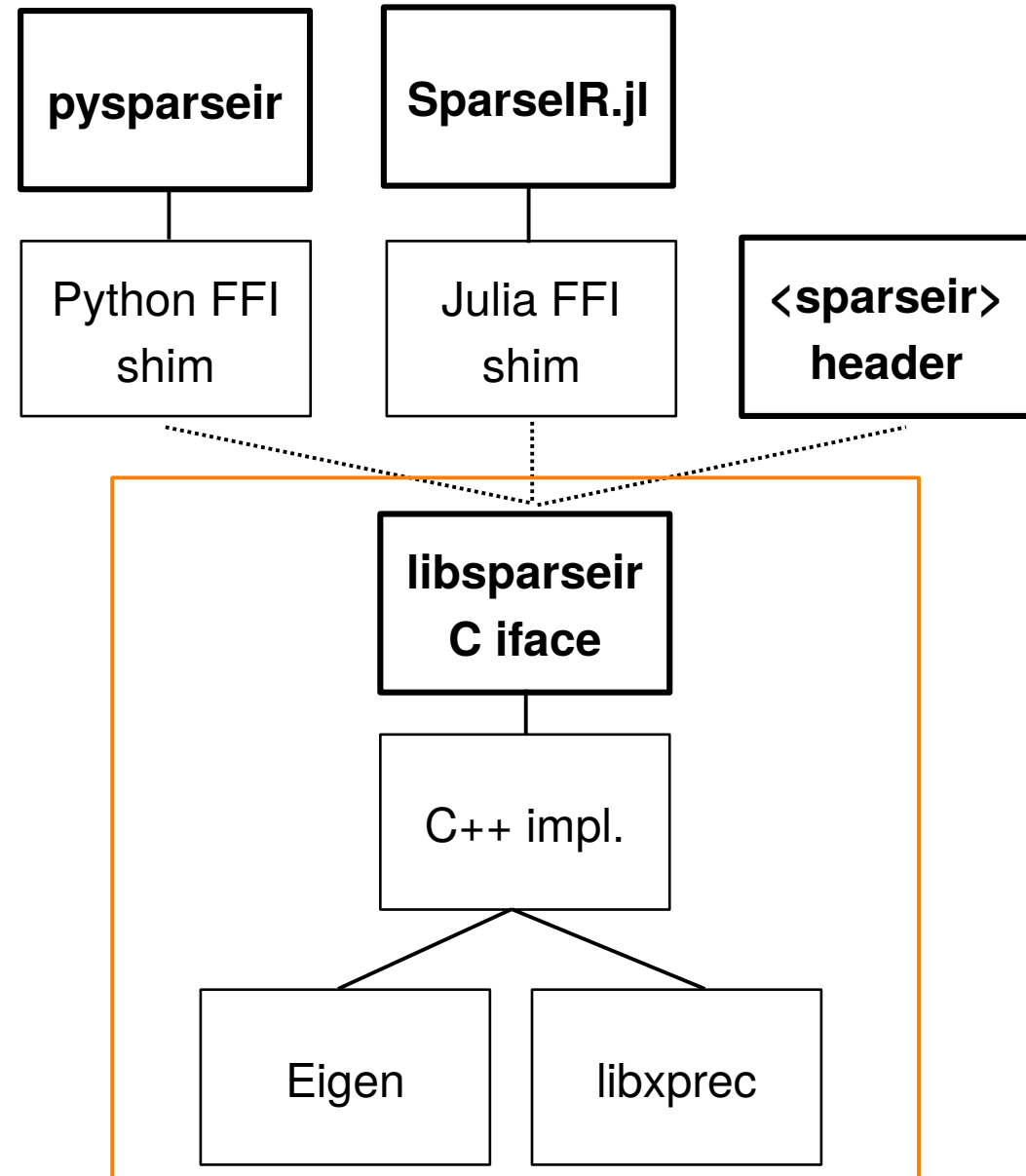
- Involved logic in translating SVE → SVD
- Need to perform SVD to **quad precision**.
- Careful evaluation of Matsubara U to avoid cancellation
- Support multitude of setups:
  - Ab-initio codes: Fortran, C
  - Model codes: Python, Julia, Rust?

# First try: sparse-ir + libxprec

- Wrap quad precision linalg into small numpy C extension → libxprec
- Broke with numpy 1.20 → 2.0
- Broke frequently with python versions
- Difficult to deploy with anaconda
- Port to Julia (SparselR.jl) could not reuse any code

# libsparseir

- C as a lingua franca
- Library exposes C interface
  - Implemented in C++
  - **Can be changed w/o ABI break!**
- Everyone can call C
  - Python, Julia FFI shim layer
  - C++ convenience header



## [3B] CMAKE

- Deployment is hard, particularly on supercomputers
- Julia, Rust: lock down dependencies
- CMake: try working with the system given
  - Nice if it works, woe is you if it does not
  - Syntax: “JavaScript on cocaine” → lots of bad code in the wild
- Impetus: reduce # of deps., bundles → ALPScore
- Smaller libraries help, use FetchContent.

# Tentative conclusions

- Task: finding a common language
- Everything speaks C
- Everyone speaks text files
- Screwdrivers work with lots of screws