Update on status of GNA fitter from Dubna

Anya Fatkina, Dmitry, Kostya, Maxim

JINR, DLNP

2018-04-21 IHEP

Outline

Introduction

Why new fitter?

dybOscar

- 90% DYB-oriented code and 10% generic.
- Strong coupling between modules.
- Difficult to extend:
  - May substitute modules, but difficult to change the structure.
  - Little room to embrace new computational approaches.
- No documentation and unit testing.
- Difficult to use by others.

GNA

- 10% DYB-oriented code and 90% generic.
- Encapsulation.
- Room for extension:
  - Flexibility.
  - Easy to include new computational approaches.
- Extensive documentation, tutorials, unit tests.
- Some groups in JUNO are already using it.

Goals

- **G**lobal **N**eutrino **A**nalysis.
- Implement framework for statistical analysis of comprehensive models with large number of parameters. Efficient, but flexible.
- Try to split physics and programming:
  - Not all physicists like to dig into programming/implementation…

    …while they usually have to.
  - May involve computer scientists.
  - Maximally reuse existing code base.

dybOscar calculation scheme

$$N^{\nu}_{dk} = B^{dk} + \sum_{kj} C^{k}_{j} \sum_{t} \varepsilon^{d}_{t}\, T^{d}_{t}\, M^{d}_{t} \times$$

$$\times \int_{-1}^{+1} d\cos\theta \int_{E^{\mathrm{vis}}_{j}}^{E^{\mathrm{vis}}_{j+1}} dE_{\mathrm{vis}} \frac{d\sigma(E_\nu, \cos\theta)}{d\cos\theta} \frac{dE_\nu(E_{\mathrm{vis}}, \cos\theta)}{dE_{\mathrm{vis}}} \times$$

$$\times \sum_{r} \frac{1}{4\pi \left(L^{d}_{r}\right)^{2}} \sum_{c} \omega_c P_c(E_\nu, L^{d}_{r}) \times$$

$$\times \left[\frac{W_{rt}}{\sum_{i'} f_{i'rt} \left\langle e \right\rangle_{i'}} \sum_{i} f_{irt}\, S_i(E_\nu)\, C_i(E_\nu) + F_r(E_\nu) \right]$$

Daya Bay: lots of objects

$$N^{\nu}_{dk} = B^{dk} + \sum_{kj} C^{k}_{j} \sum_{t} \varepsilon^{d}_{t} T^{d}_{t} M^{d}_{t} \times$$

$$\times \int_{-1}^{+1} d\cos\theta \int_{E^{\mathrm{vis}}_{j}}^{E^{\mathrm{vis}}_{j+1}} dE_{\mathrm{vis}} \frac{d\sigma(E_\nu, \cos\theta)}{d\cos\theta} \frac{dE_\nu(E_{\mathrm{vis}}, \cos\theta)}{dE_{\mathrm{vis}}} \times$$

$$\times \sum_{r} \frac{1}{4\pi \left(L^{d}_{r}\right)^{2}} \sum_{c} \omega_c P_c(E_\nu, L^{d}_{r}) \times$$

$$\times \left[\frac{W_{rt}}{\sum_{i'} f_{i'rt} \langle e \rangle_{i'}} \sum_{i} f_{irt} S_i(E_\nu) C_i(E_\nu) + F_r(E_\nu) \right]$$

- Thermal power.
- Energy per fission.
- Fissile isotopes.
- Fission fractions.
- $\overline{\nu}_e$ spectrum.
- Off-equilibrium correction.
- SNF contribution.

Daya Bay: lots of objects

$$N^{\nu}_{dk} = B^{dk} + \sum_{kj} C^{k}_{j} \sum_{t} \varepsilon^{d}_{t} T^{d}_{t} M^{d}_{t} \times$$

$$\times \int_{-1}^{+1} d\cos\theta \int_{E^{\mathrm{vis}}_{j}}^{E^{\mathrm{vis}}_{j+1}} dE_{\mathrm{vis}} \frac{d\sigma(E_\nu, \cos\theta)}{d\cos\theta} \frac{dE_\nu(E_{\mathrm{vis}}, \cos\theta)}{dE_{\mathrm{vis}}} \times$$

$$\times \sum_{r} \frac{1}{4\pi \left(L^{d}_{r}\right)^2} \sum_{c} \omega_c P_c(E_\nu, L^{d}_{r}) \times$$

$$\times \left[\frac{W_{rt}}{\sum_{i'} f_{i'rt} \langle e \rangle_{i'}} \sum_{i} f_{irt} S_i(E_\nu) C_i(E_\nu) + F_r(E_\nu) \right]$$

- Sum over reactors.
- Baseline.
- Oscillation probability (split).

Daya Bay: lots of objects

- Integral over positron direction (GH).
- $E_{\text{vis}} \to E_\nu$ Jacobian.
- IBD cross-section.

$$N^\nu_{dk} = B^{dk} + \sum_{kj} C^k_j \sum_t \varepsilon^d_t \, T^d_t \, M^d_t \times$$

$$\times \int_{-1}^{+1} d\cos\theta \int_{E^{\text{vis}}_j}^{E^{\text{vis}}_{j+1}} dE_{\text{vis}} \, \frac{d\sigma(E_\nu, \cos\theta)}{d\cos\theta} \, \frac{dE_\nu(E_{\text{vis}}, \cos\theta)}{dE_{\text{vis}}} \times$$

$$\times \sum_r \frac{1}{4\pi \left(L^d_r\right)^2} \sum_c \omega_c P_c(E_\nu, L^d_r) \times$$

$$\times \left[\frac{W_{rt}}{\sum_{i'} f_{i'rt} \, \langle e \rangle_{i'}} \sum_i f_{irt} \, S_i(E_\nu) \, C_i(E_\nu) + F_r(E_\nu) \right]$$

- Energy integration E_{vis} (Gauss-Legendre, predefined precision).

Daya Bay: lots of objects

- Detector effects, final binning.
- Background.
- Sum over days.
- AD efficiency.
- DAQ time.
- Target mass.

$$N^{\nu}_{dk} = B^{dk} + \sum_{kj} C^{k}_{j} \sum_{t} \varepsilon^{d}_{t}\, T^{d}_{t}\, M^{d}_{t} \times$$

$$\times \int_{-1}^{+1} d\cos\theta \int_{E^{\mathrm{vis}}_{j}}^{E^{\mathrm{vis}}_{j+1}} dE_{\mathrm{vis}} \frac{d\sigma(E_\nu, \cos\theta)}{d\cos\theta} \frac{dE_\nu(E_{\mathrm{vis}}, \cos\theta)}{dE_{\mathrm{vis}}} \times$$

$$\times \sum_{r} \frac{1}{4\pi \left(L^{d}_{r}\right)^{2}} \sum_{c} \omega_c P_c(E_\nu, L^{d}_{r}) \times$$

$$\times \left[\frac{W_{rt}}{\sum_{i'} f_{i'rt} \langle e \rangle_{i'}} \sum_{i} f_{irt}\, S_i(E_\nu)\, C_i(E_\nu) + F_r(E_\nu) \right]$$

Daya Bay: lots of objects

$$N^{\nu}_{dk} = B^{dk} + \sum_{kj} C^{k}_{j} \sum_{t} \varepsilon^{d}_{t} T^{d}_{t} M^{d}_{t} \times$$

$$\times \int_{-1}^{+1} d\cos\theta \int_{E^{\mathrm{vis}}_{j}}^{E^{\mathrm{vis}}_{j+1}} dE_{\mathrm{vis}} \, \frac{d\sigma(E_\nu, \cos\theta)}{d\cos\theta} \, \frac{dE_\nu(E_{\mathrm{vis}}, \cos\theta)}{dE_{\mathrm{vis}}} \times$$

$$\times \sum_{r} \frac{1}{4\pi \left(L^{d}_{r}\right)^{2}} \sum_{c} \omega_c P_c(E_\nu, L^{d}_{r}) \times$$

$$\times \left[\frac{W_{rt}}{\sum_{i'} f_{i'rt} \langle e \rangle_{i'}} \sum_{i} f_{irt} \, S_i(E_\nu) \, C_i(E_\nu) + F_r(E_\nu) \right]$$

Daya Bay: lots of objects

- Oscillation amplitudes.

$$N^{\nu}_{dk} = B^{dk} + \sum_c \omega_c(\theta_{12}, \theta_{13}) \sum_{kj} C^k_j \sum_r \frac{1}{4\pi \left(L^d_r\right)^2} \sum_i \times$$

$$\times \int_{-1}^{+1} d\cos\theta \int_{E^{\text{vis}}_j}^{E^{\text{vis}}_{j+1}} dE_{\text{vis}} \frac{d\sigma(E_\nu, \cos\theta)}{d\cos\theta} \frac{dE_\nu(E_{\text{vis}}, \cos\theta)}{dE_{\text{vis}}} \widetilde{S}_i(E_\nu) P_c(E_\nu, L^d_r, \Delta m^2_c) \times$$

$$\times \sum_t \varepsilon^d_t T^d_t M^d_t \frac{f_{irt} W_{rt}}{\sum_{i'} f_{i'rt} \langle e \rangle_{i'}}$$

- Spectra, kinematics, mass splittings.
- Rate: contribution of each isotope chain to each detector.

GNA: requirements

- Handle lots of parameters.
- Do not (re)calculate parts, that are not required to be (re)calculated:

  lazy evaluation, caching.
- Never ever create static types for detectors, physics or whatever:

  structure should be flexible.
- The code created once should be valid for a long period of time:

  self consistency and backwards compatibility.
- Operate on arrays, avoid explicit loops:

  clean readable code.
- Use existing libraries as much as possible:

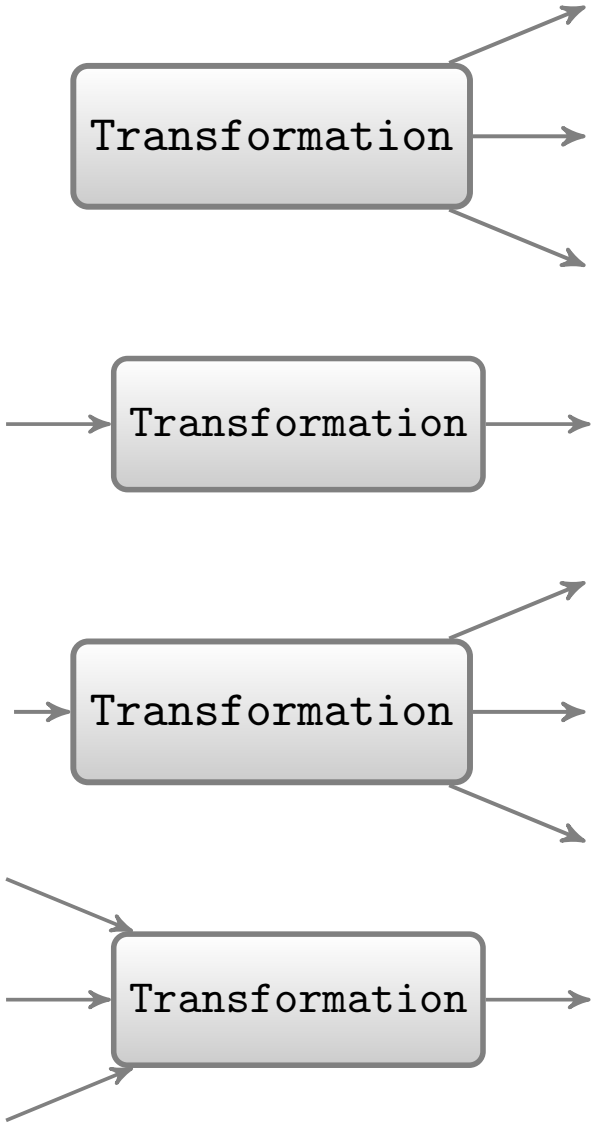
  use Eigen matrix library,

  minimize usage of ROOT.

Transformations: function abstraction layer

Transformation

- A wrapper for C++ function:
  - return values (array): output
  - arguments (array): inputs
  - arguments (double): variables



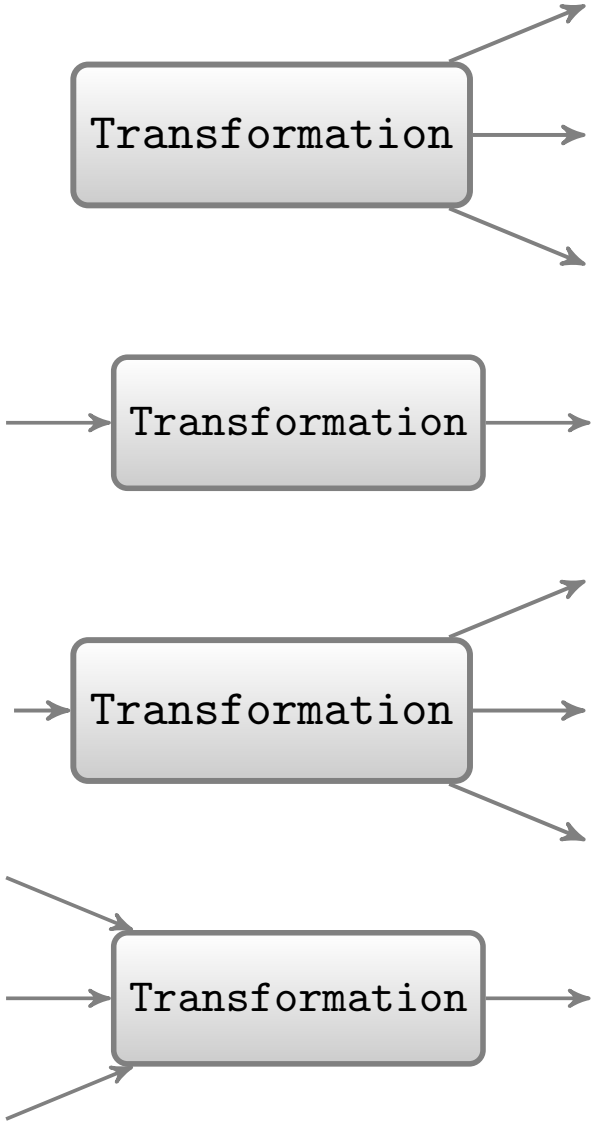
Transformations: function abstraction layer

Definition

- Type functions:
  - Check input types.
  - Derive output types.
- Function:
  - Perform actual calculation.

Framework

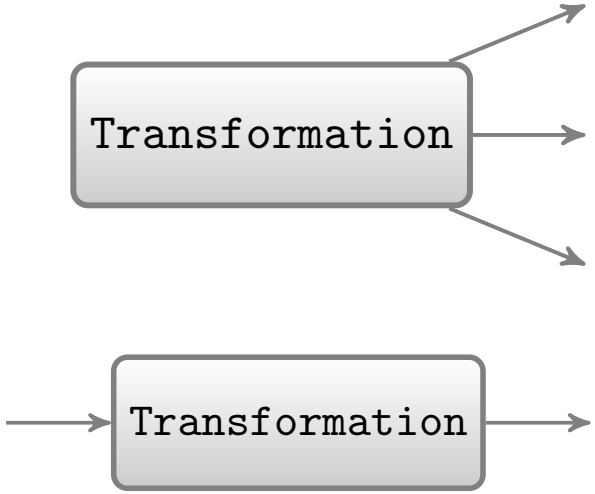
- Evaluates types (once).
- Allocates memory (once).
- Performs calculation (lazy!).
- Stores results when calculated.
- Propagates the `taintflag`:
  call function/return stored



Transformations: function abstraction layer

Definition

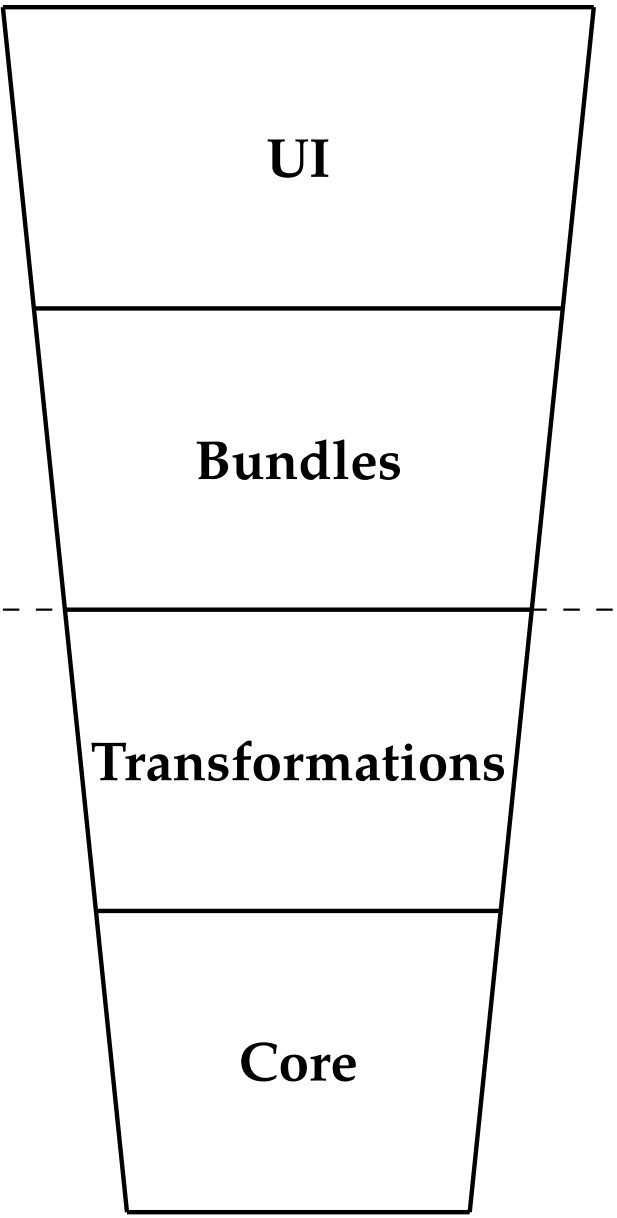
- Type functions:
  - Check input types.
  - Derive output types.
- Function:
  - Perform actual calculation.



Framework

- Evaluates types (once).
- Allocates memory (once).
- Performs calculation (lazy!).
- Stores results when calculated.
- Propagates the `taintflag`:

call function/return stored



- Comprehensive command line chain.
- Computational graphs.
- Analysis.

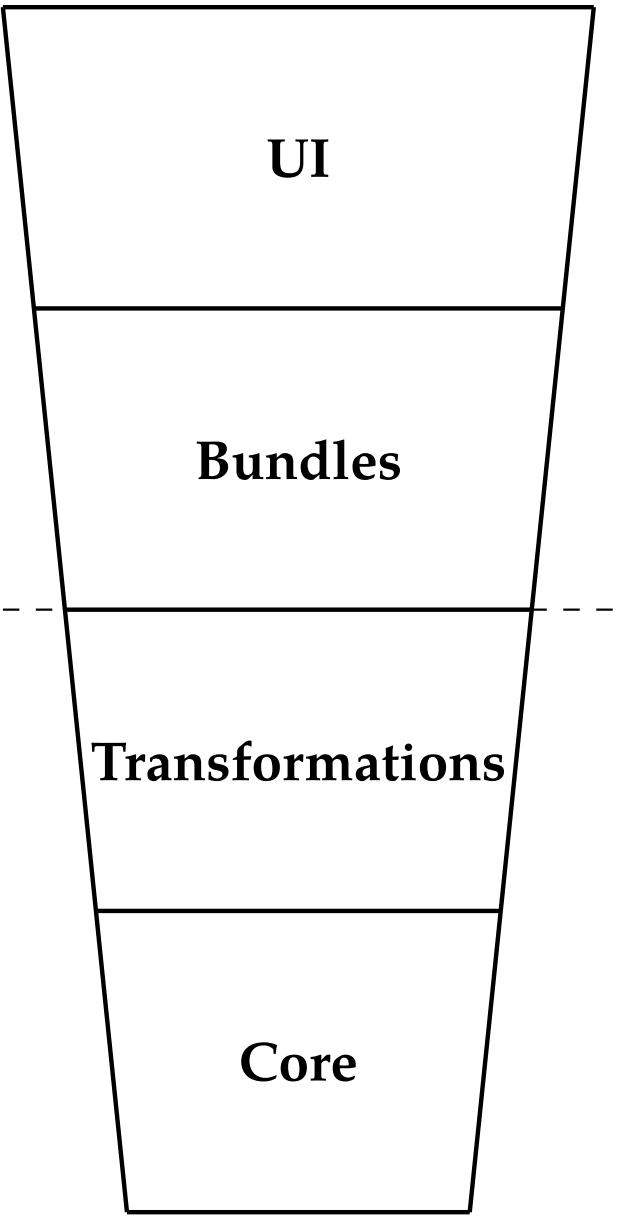
- Read configuration.
- Variables.
- Small computational chain.

Python
(flexibility)

C++
(efficiency)

- Linear algebra
- Integration
- Statistics
- Physics

- Data
- Variable
- Transformation



- Comprehensive command line chain.
- Computational graphs.
- Analysis.

- Read configuration.
- Variables.
- Small computational chain.

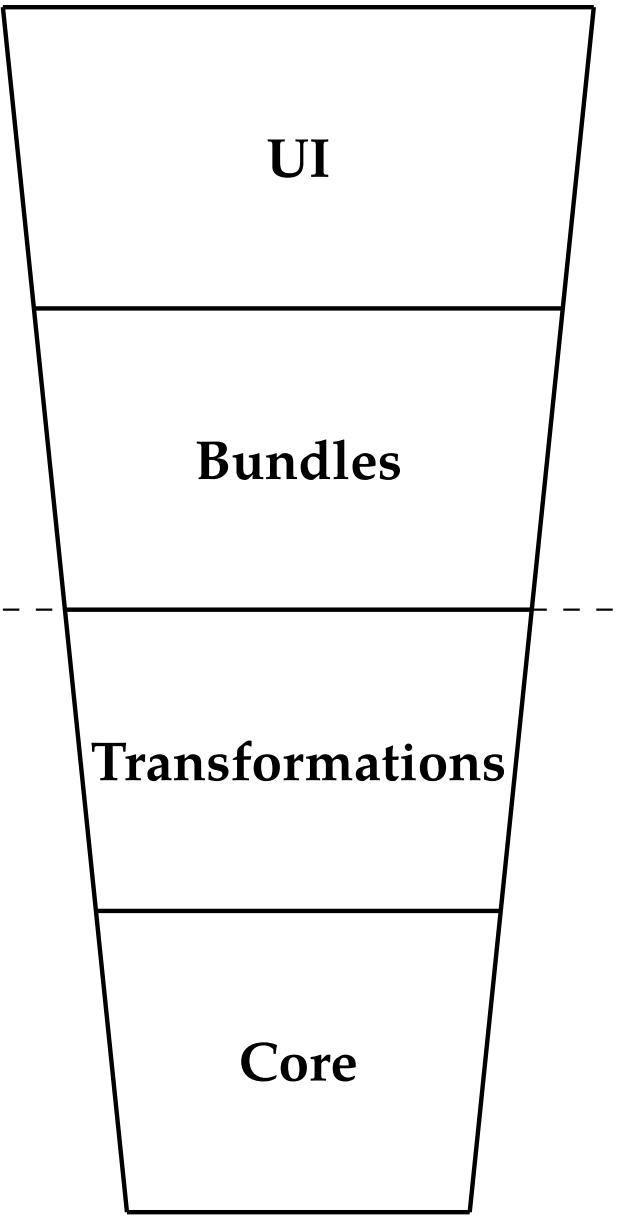
Python
(flexibility)

C++
(efficiency)

- Linear algebra
- Integration

- Statistics
- Physics

- Data
- Variable
- Transformation



- Comprehensive command line chain.
- Computational graphs.
- Analysis.

- Read configuration.
- Variables.
- Small computational chain.

Python
(flexibility)

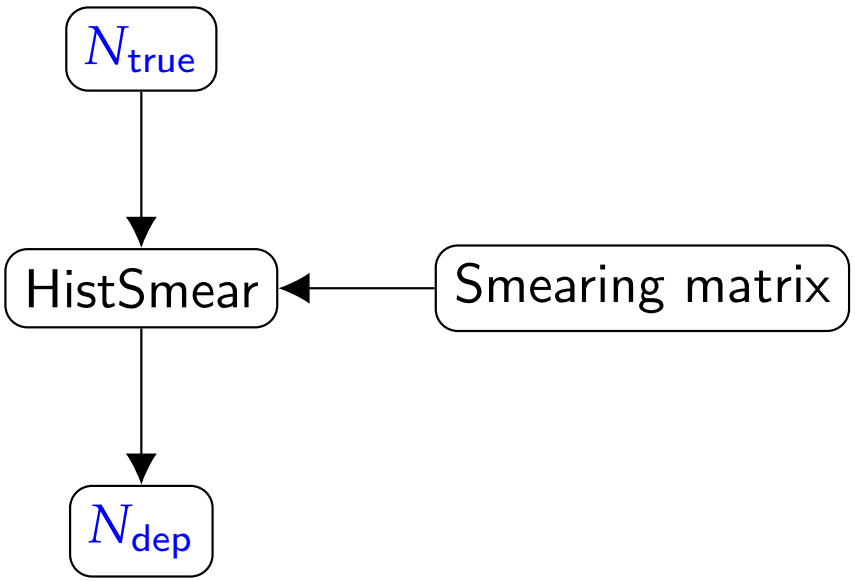
C++
(efficiency)

- Linear algebra
- Integration

- Statistics
- Physics

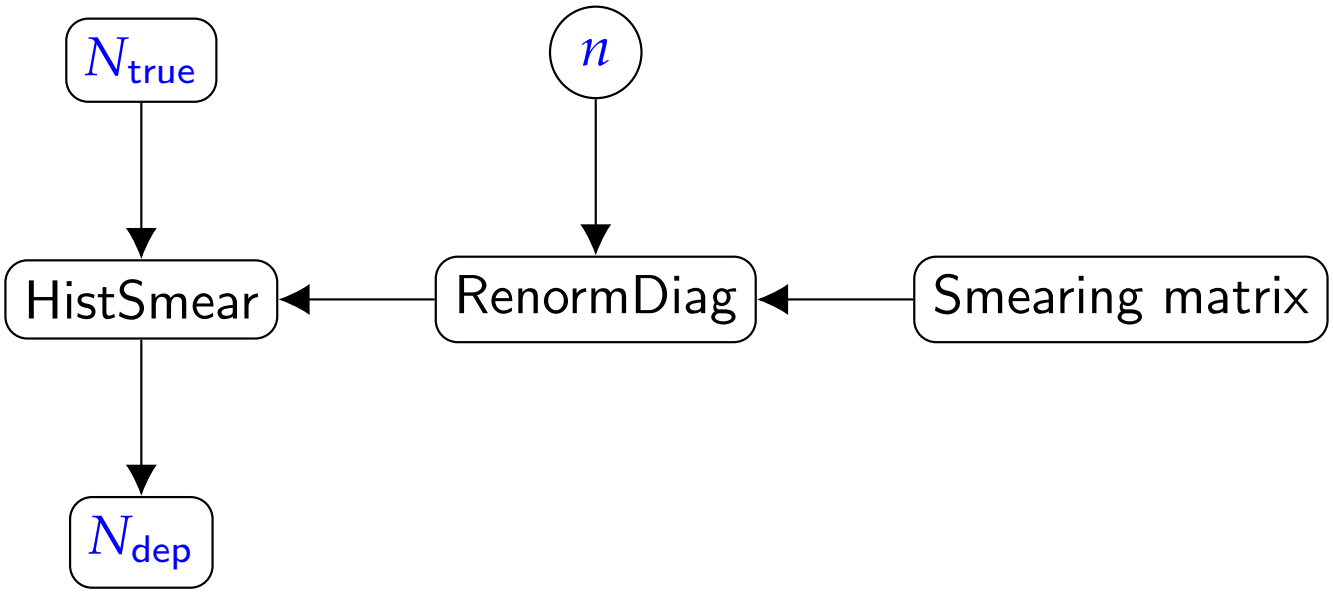
- Data
- Variable
- Transformation

IAV correction: idea



- $N_{\text{dep}} = C_{\text{IAV}} N_{\text{true}}$
- Two inputs: matrix and spectrum
- IAV uncertainty: additional transformation, renormalizes off-diagonal elements
- Variable n may be passed to the minimizer

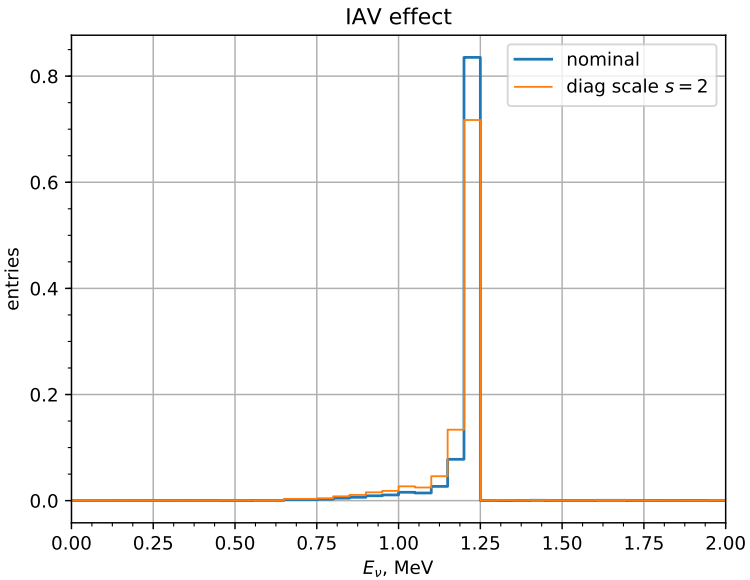
IAV correction: idea

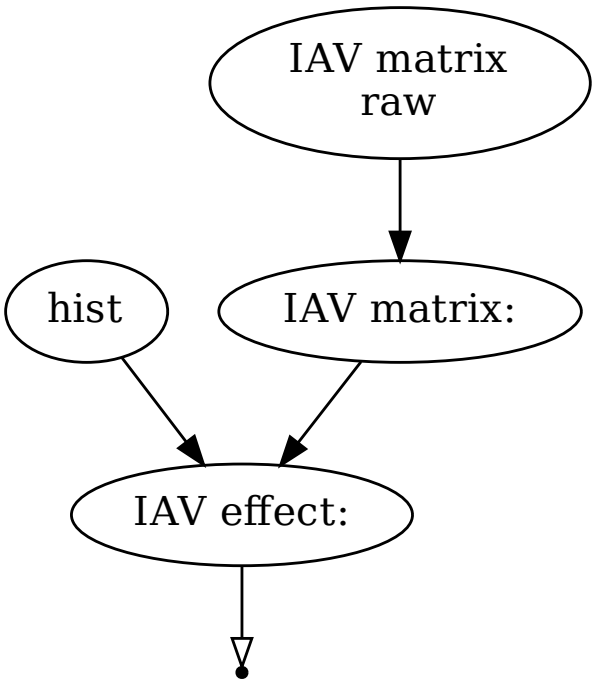


- $N_{\text{dep}} = C_{\text{IAV}}(n) N_{\text{true}}$
- Two inputs: matrix and spectrum
- IAV uncertainty: additional transformation, renormalizes off-diagonal elements
- Variable n may be passed to the minimizer

IAV correction: implementation

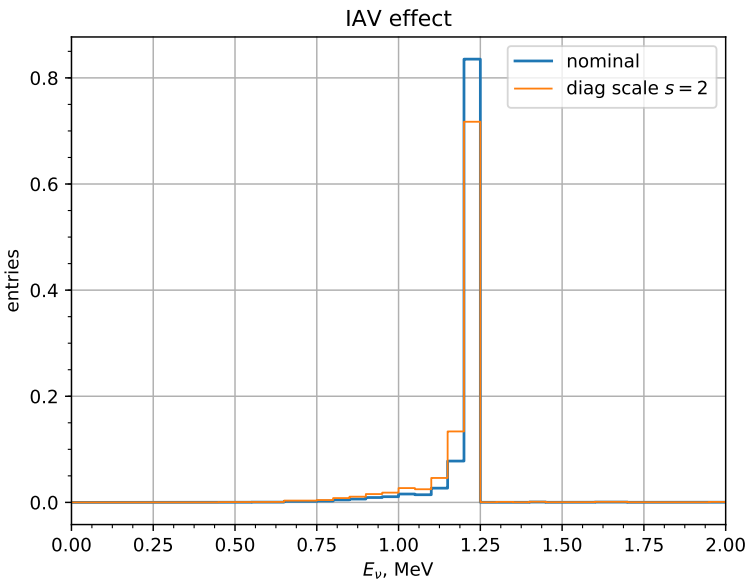
- The graph is autogenerated and represents the actual structure.

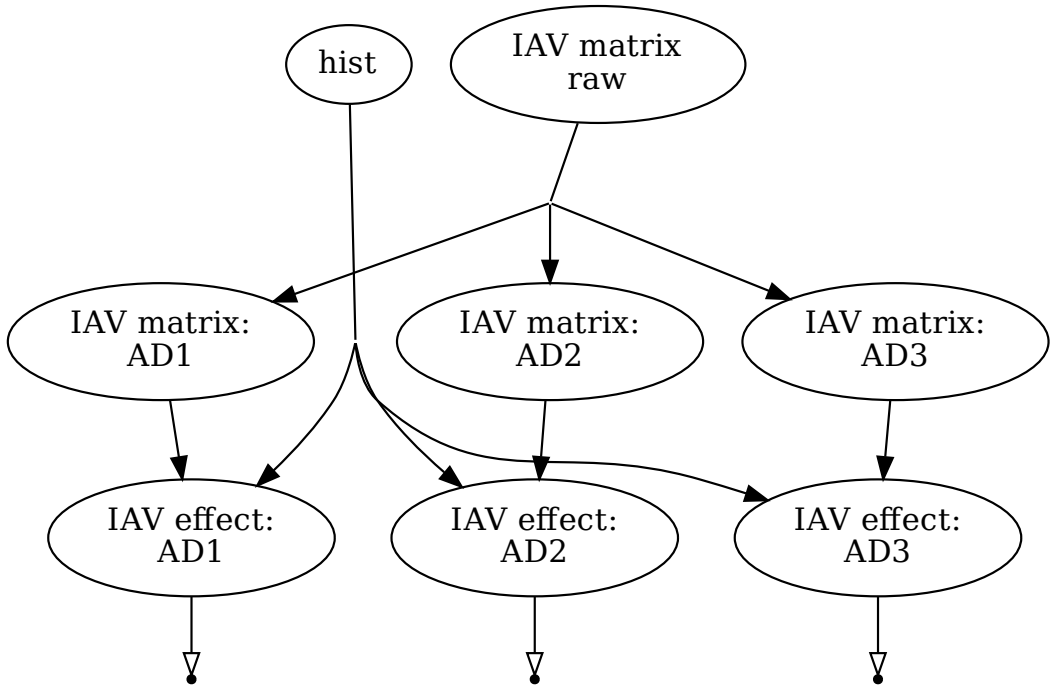




IAV correction: implementation

- The graph is autogenerated and represents the actual structure.





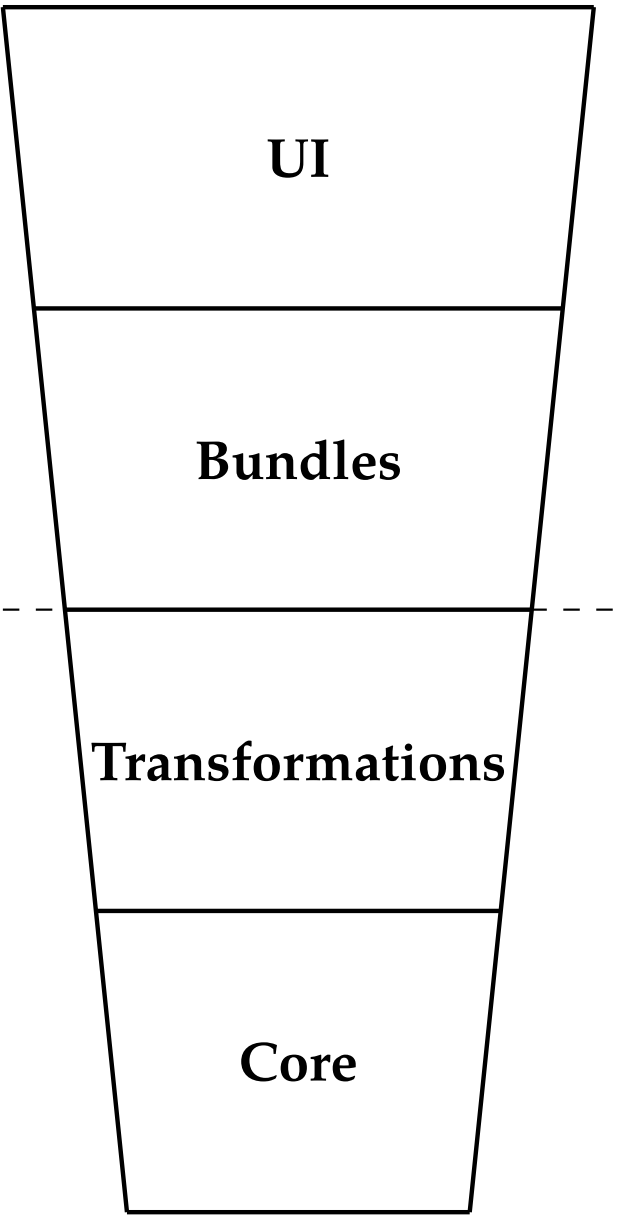

```
Variables in namespace 'OffdiagScale'
  AD1 = 1 | 1+- 0.04 [ 4%] | IAV offdiagonal contribution scale for AD1
  AD2 = 1 | 1+- 0.04 [ 4%] | IAV offdiagonal contribution scale for AD2
  AD3 = 1 | 1+- 0.04 [ 4%] | IAV offdiagonal contribution scale for AD3
```

Bundles

- Bundle — a set of transformations:
  - Reads configuration.
  - Creates transformations.
  - Connects inputs/outputs.
  - Backwards compatible.

IAV configuration

```
cfg = NestedDict(
    # Bundle name
    bundle = 'detector_iav_db_root_v01',
    # Parameter name
    parname = 'OffdiagScale',
    # Parameter uncertainty and its type (absolute or relative)
    scale   = uncertain(1.0, 4, 'percent'),
    # Number of diagonals to treat as diagonal.
    # All other elements are considered as off-diagonal.
    ndiag = 1,
    # File name to read
    filename = 'data/dayabay/tmp/detector_iavMatrix_P14A_LS.root',
    # Matrix name
    matrixname = 'iav_matrix'
    )
```



- Comprehensive command line chain.
- Computational graphs.
- Analysis.

- Read configuration.
- Variables.
- Small computational chain.

Python
(flexibility)

C++
(efficiency)

- Linear algebra
- Integration

- Statistics
- Physics

- Data
- Variable
- Transformation

Computational chain

- Transformations (bundles) → computational chain.

Daya Bay chain

- Detector part ready and assembled.
- All other bundles ready… still needs assembly.
- See the external file.

JUNO chain

- Full chain.
- Much simpler than Daya Bay.
- See the external file.

Computational chain

- Transformations (bundles) → computational chain.

Daya Bay chain

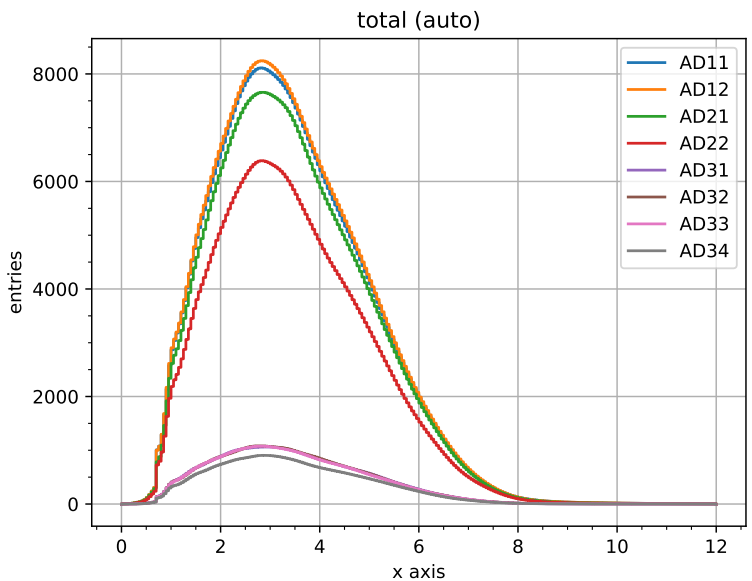
- Detector part ready and assembled.
- All other bundles ready… still needs assembly.
- See the external file.

JUNO chain

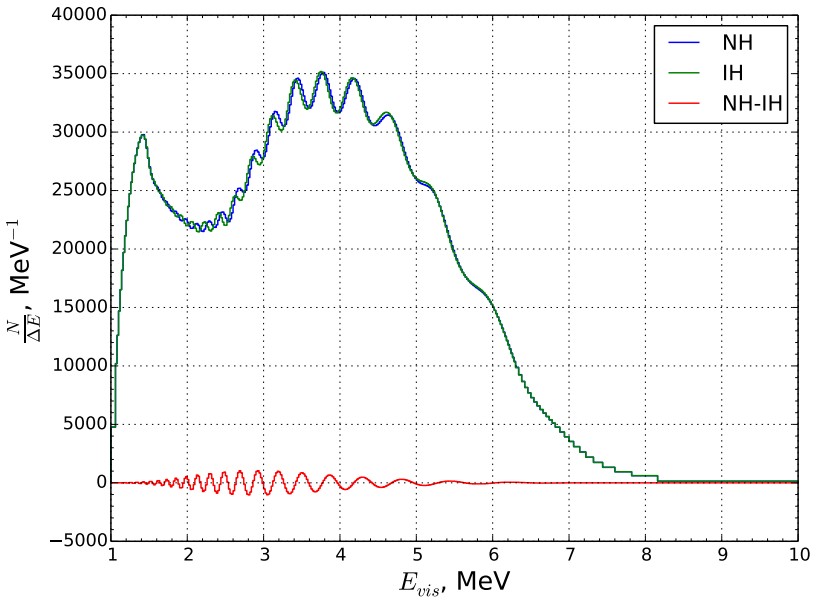
- Full chain.
- Much simpler than Daya Bay.
- See the external file.

Example output

Daya Bay



JUNO



Variables

- A subset of Daya Bay parameters dumped to the terminal:

```
Variables in namespace 'dayabay'
  weight_nominal=       1 |[fixed]                  | Nominal nonlinearity curve weight
  weight_pull0  =       0 |       0+-        1      | Correction nonlinearity weight for pu
  weight_pull1  =       0 |       0+-        1      | Correction nonlinearity weight for pu
  weight_pull2  =       0 |       0+-        1      | Correction nonlinearity weight for pu
  weight_pull3  =       0 |       0+-        1      | Correction nonlinearity weight for pu
  Eres_a        =0.014764 |0.014764+-0.0044292 [30%]| Energy resolution (spatial/temporal r
  Eres_b        =  0.0869 |  0.0869+-  0.02607 [30%]| Energy resolution (photon statistics)
  Eres_c        =  0.0271 |  0.0271+-  0.00813 [30%]| Energy resolution (dark noise)
  frac_li       =    0.95 |    0.95+-   0.0475 [ 5%]| li fraction
  frac_he       =    0.05 |                         | he fraction: 1-frac_li
Variables in namespace 'dayabay.AD11'
  livetime      =    1000 |[fixed]                  | AD11 DAQ livetime
  acc_rate      =  8.4636 |[fixed]                  | AD11.acc_rate for AD11
  acc_num       =  8463.6 |                         | Norm of acc for AD11: acc_norm.AD11*A
  lihe_num      =    2460 |                         | Norm of lihe for AD11: lihe_rate.EH1*
  fastn_num     =     792 |                         | Norm of fastn for AD11: fastn_rate.EH
  amc_num       =     180 |                         | Norm of amc for AD11: amc_rate.AD11*A
  alphan_num    =      80 |                         | Norm of alphan for AD11: alphan_rate
...
```

Daya Bay progress

Progress since previous collaboration meeting

- Implemented transformations, bundles required for Daya Bay:
  - All the detector effects.
  - Reactor antineutrino spectra (model with free parameters).
  - SNF and NEC corrections.

Still working on

- Wiring…

Daya Bay progress

Progress since previous collaboration meeting

- Implemented transformations, bundles required for Daya Bay:
  - All the detector effects.
  - Reactor antineutrino spectra (model with free parameters).
  - SNF and NEC corrections.

Still working on

- Wiring…

Transformation binding via expression evaluation

Why?

- JUNO chain initialization/connection code: 500 lines.
- Mostly loops over reactors and oscillation components.
- May be generalized.

The idea

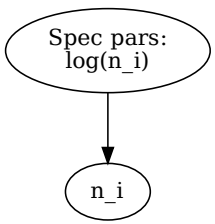
- Parse text mathematical expression.
- Each item is a bundle providing the outputs for each AD, reactor, whatever.
- Operations `+`, `-`, `*`, `function call` define the way to bind.

Transformation binding via expression evaluation

Expression: antineutrino spectra

```
expr = norm()
```

Chain

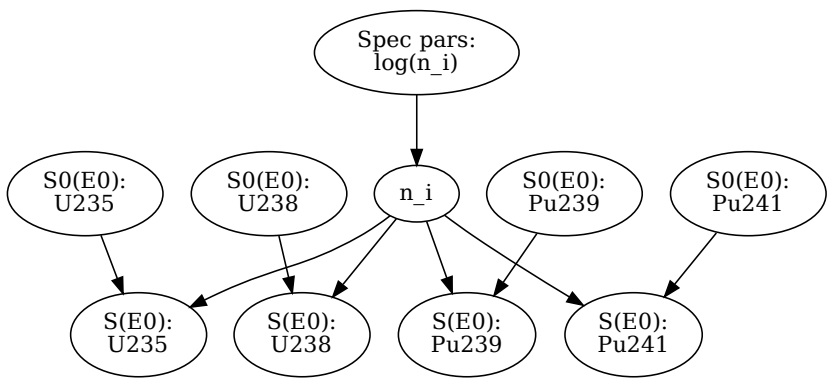


Transformation binding via expression evaluation

Expression: antineutrino spectra

```
expr = norm()*spec[i]()
```

Chain

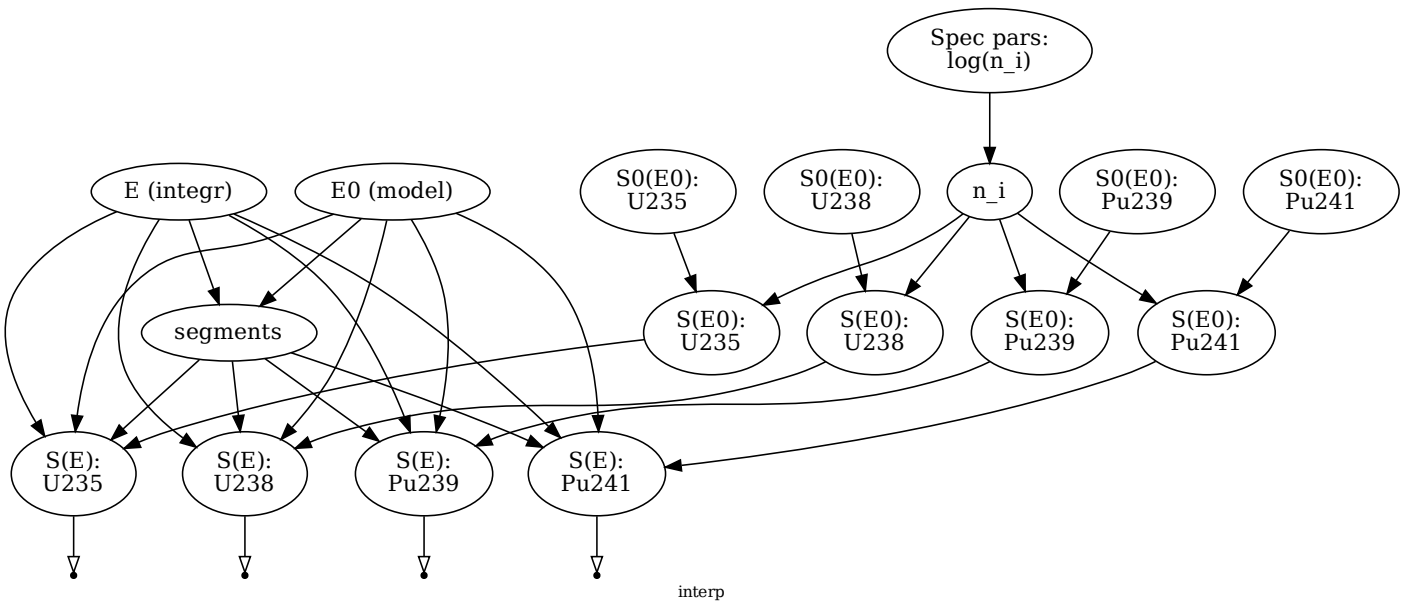


Transformation binding via expression evaluation

Expression: antineutrino spectra

```
expr = interp | norm()*spec[i](), enu()
```

Chain

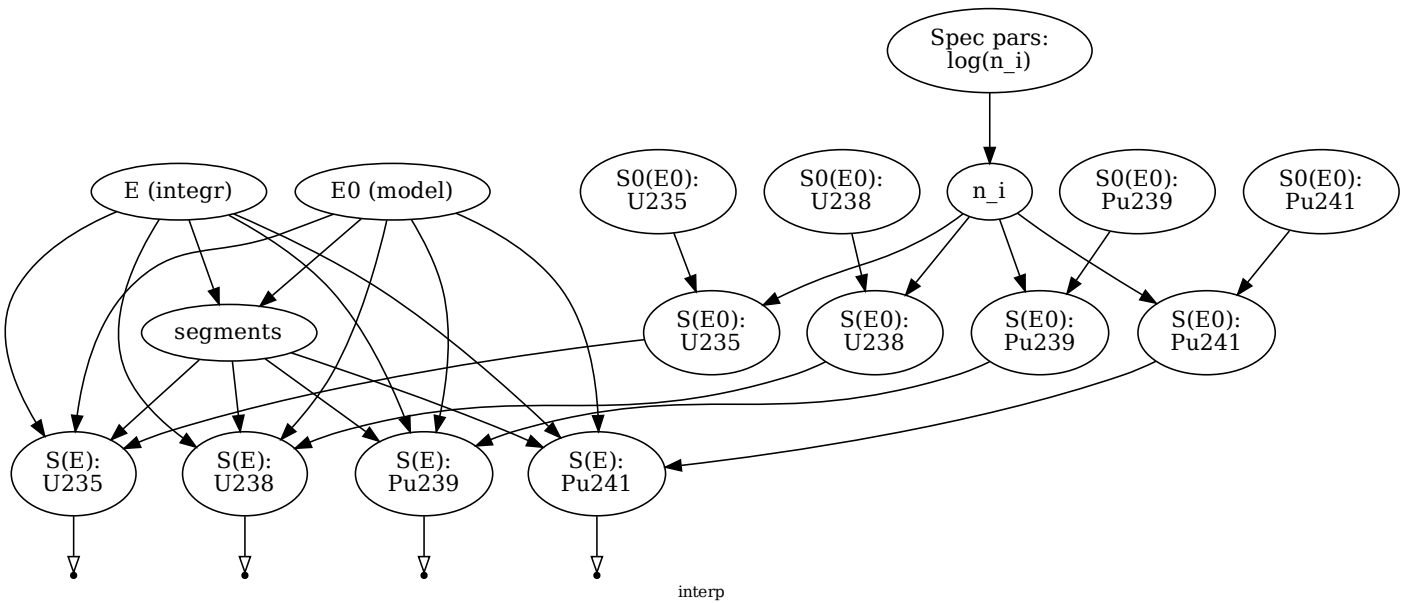


Transformation binding via expression evaluation

Expression: antineutrino spectra

```
expr = interp | norm()*spec[i](), enu()
expr = interp(norm()*spec[i](), enu())        (regular syntax)
```

Chain



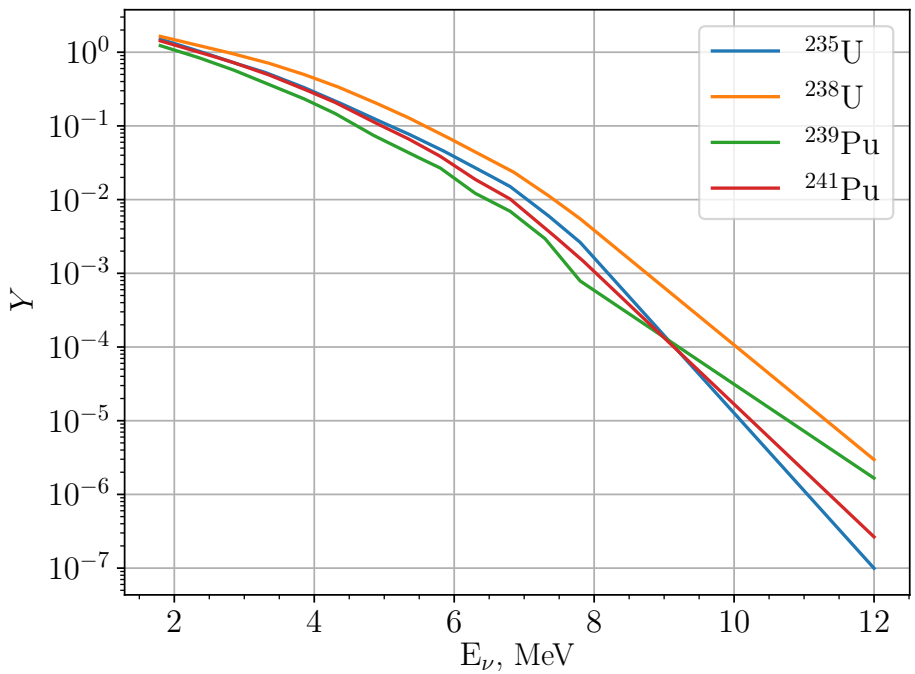
Transformation binding via expression evaluation

Expression: antineutrino spectra

```
expr = interp | norm()*spec[i](), enu()
```

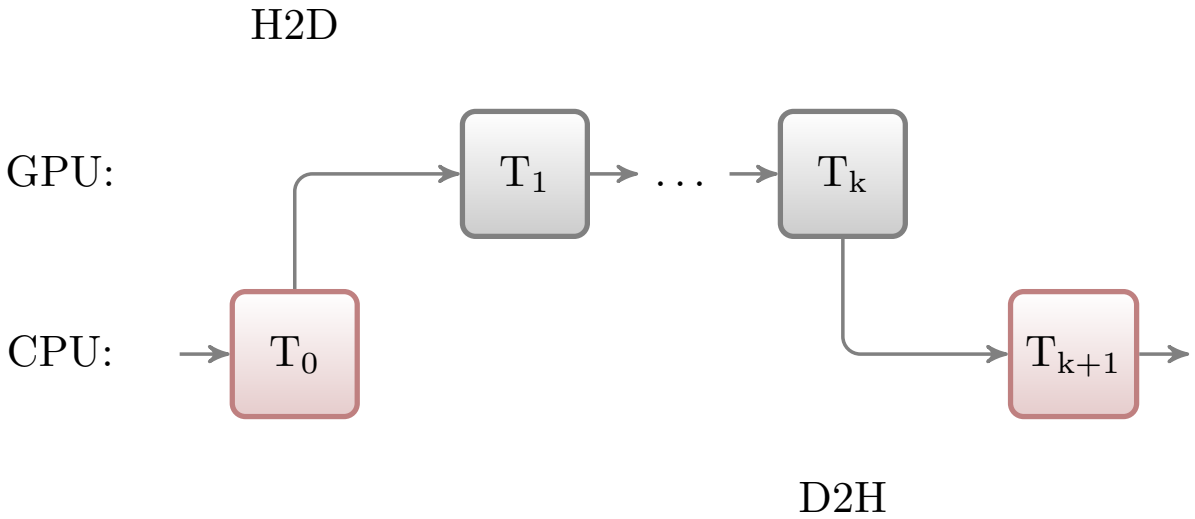
It is already a thing!

Result



Transparent GPGPU support

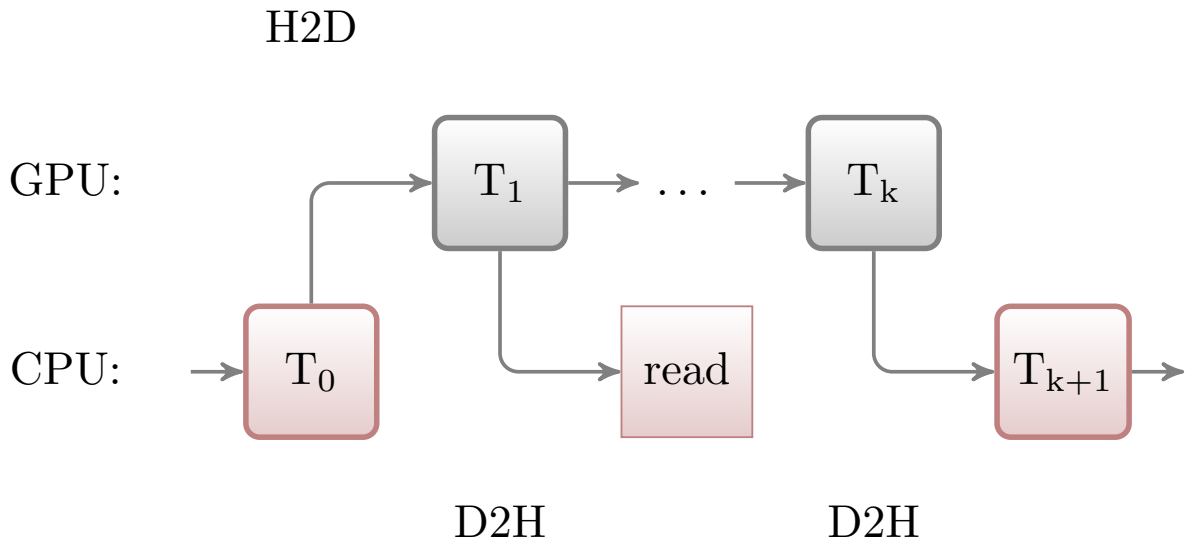
- The user builds the chain without thinking how and where the transformations will be executed.
- Transformation may be defined by several functions: CPU, GPU, ...
- Framework does memory allocation, data transfer, dispatching, etc.



- Achieved ×20 speedup for osc. prob. (double precision)
- Another ×100 speedup may be achieved (single precision)
- arXiv:1804.07682.

Transparent GPGPU support

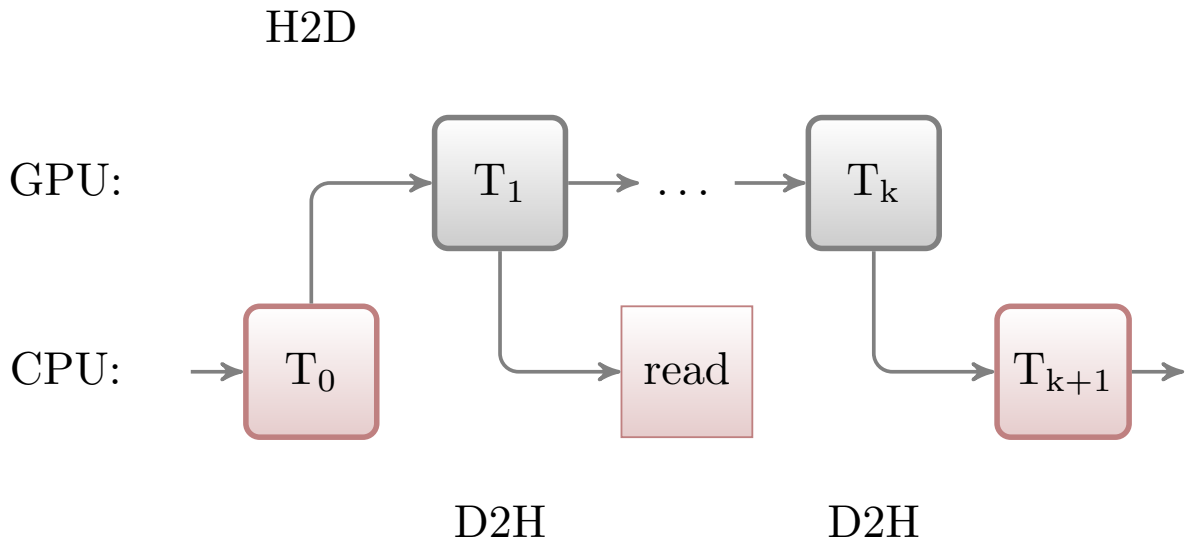
- The user builds the chain without thinking how and where the transformations will be executed.
- Transformation may be defined by several functions: CPU, GPU, ...
- Framework does memory allocation, data transfer, dispatching, etc.



- Achieved ×20 speedup for osc. prob. (double precision)
- Another ×100 speedup may be achieved (single precision)
- arXiv:1804.07682.

Transparent GPGPU support

- The user builds the chain without thinking how and where the transformations will be executed.
- Transformation may be defined by several functions: CPU, GPU, ...
- Framework does memory allocation, data transfer, dispatching, etc.

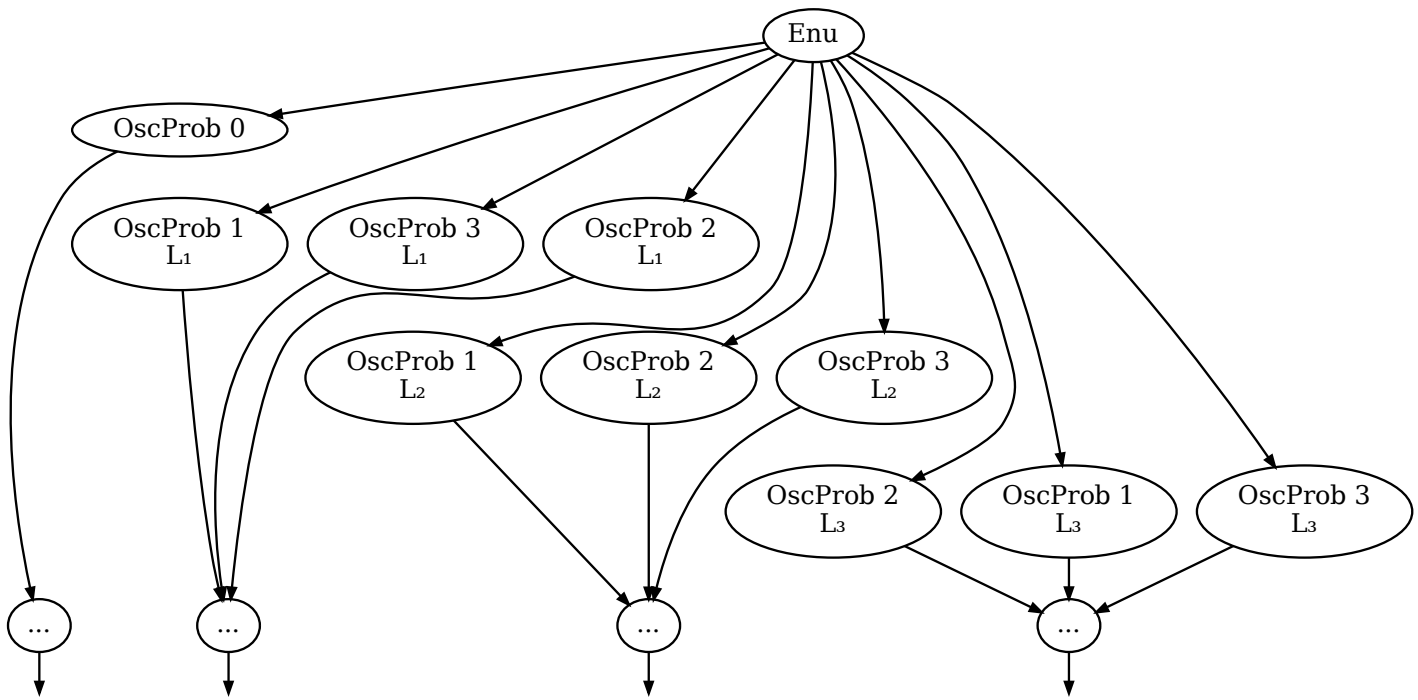


- Achieved $\times 20$ speedup for osc. prob. (double precision)
- Another $\times 100$ speedup may be achieved (single precision)
- arXiv:1804.07682.

Multi threading

Same idea

- Framework controls the execution transparently for the end user.
- Runs independent transformations in parallel threads.
- Dispatches the access to common inputs.



Introduction

dybOscar case

GNA structure

Forthcoming features

Conclusions

Conclusions

GNA

- Report on further developing of GNA.
- GNA evolves in a powerful set of tools for the fitting.

Development plan for 2018

- Finalize expressions and Daya Bay implementation.
- Multi threading.
- Documentation, documentation, documentation…
- Automatic unit tests.
- Expect to be released before 2019 on github.

Conclusions

GNA

- Report on further developing of GNA.
- GNA evolves in a powerful set of tools for the fitting.

Development plan for 2018

- Finalize expressions and Daya Bay implementation.
- Multi threading.
- Documentation, documentation, documentation…
- Automatic unit tests.
- Expect to be released before 2019 on github.

Conclusions

Collaboration

- Several groups in JUNO already using GNA.
- Expect GNA analysis by Dubna group of NOvA this year.
- New users and contributers are welcome!

Links

- Internal repository:
  `https://git.jinr.ru/gna/gna`
- Documentation:
  `http://gna.pages.jinr.ru/gna/`
- Reference guide: (regularly updated)
  `http://gna.pages.jinr.ru/gna/reference_guide.html`