Experiment (non)Specifics

Luke Pickering

MaCh3 Workshop

2026/11/5

UKRI Science and Technology Facilities Council

Supported by URF\R1\211661

The Royal Society

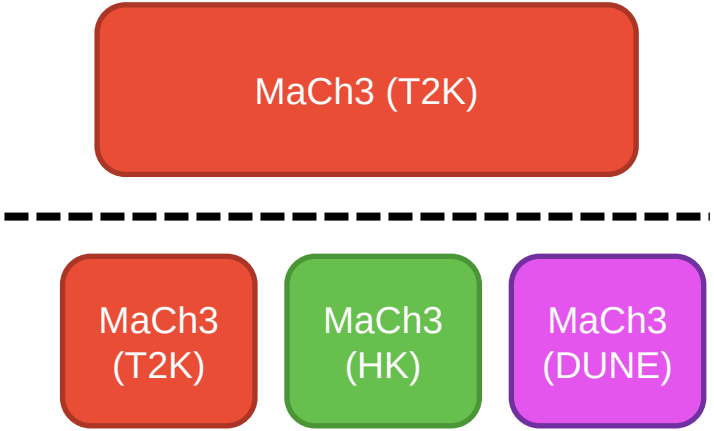
The Split

- In the beginning there was only MaCh3:
  - T2K-only



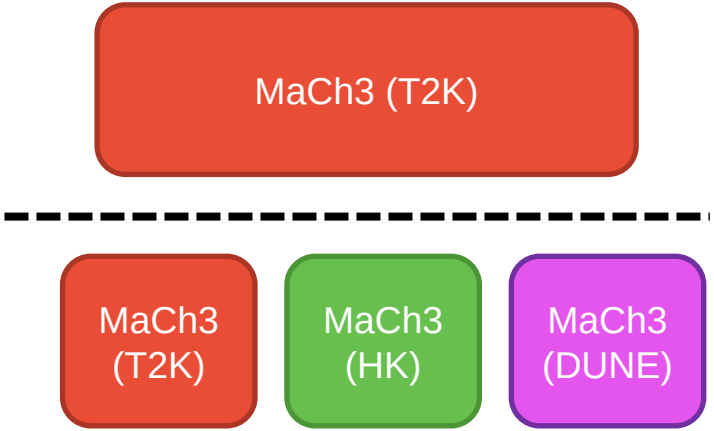
The Split

- In the beginning there was only MaCh3:
  - T2K-only
- Then we had copies of the code for different experiments:
  - T2K, HK, DUNE
- New functionality was driven by different experiments/users/components (ND v. FD)
  - Maintainence and keeping new developments in sync was a manual nightmare.



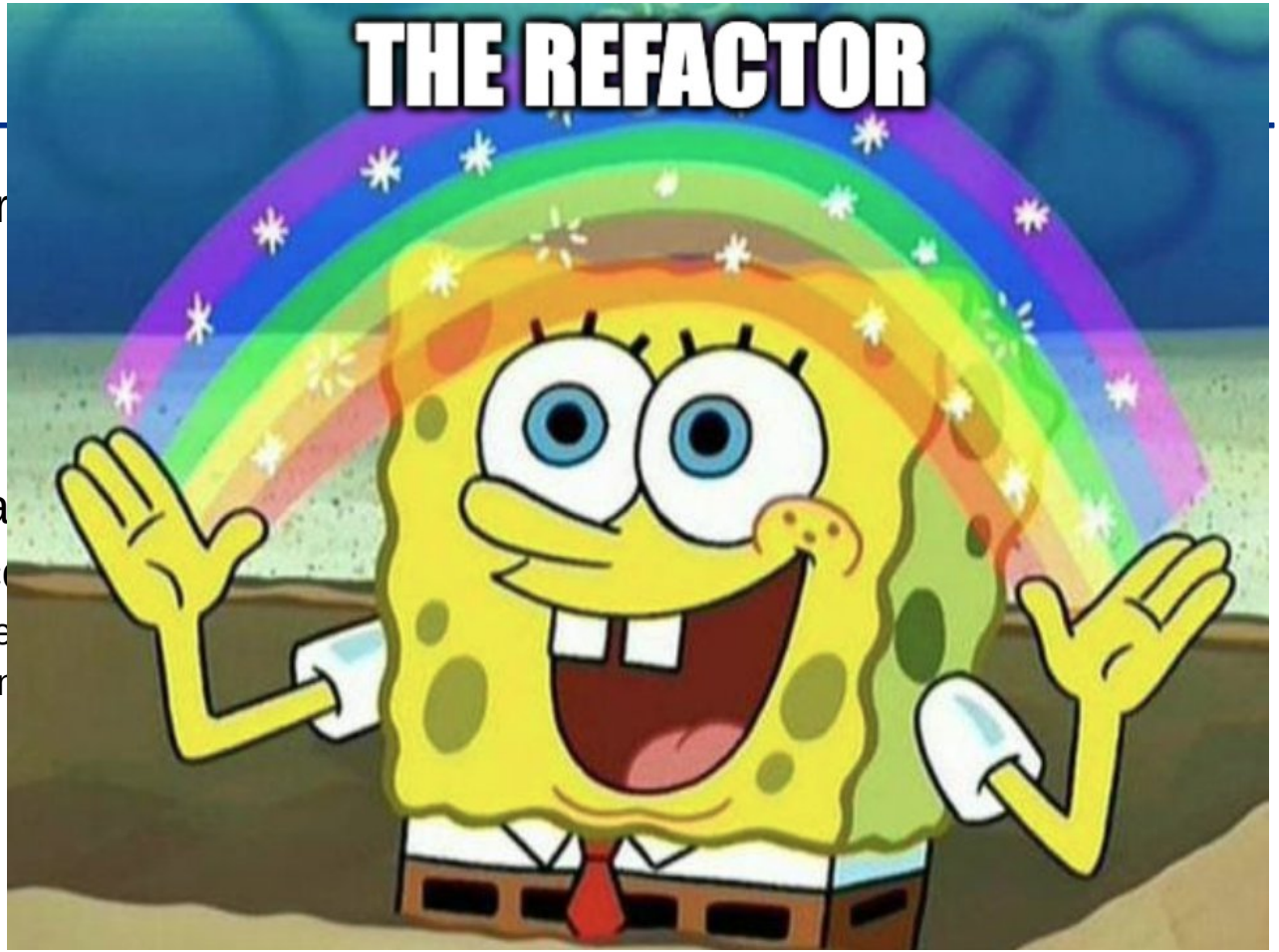
The Split

- In the beginning there was only MaCh3:
  - T2K-only
- Then we had copies of the code for different experiments:
  - T2K, HK, DUNE
- New functionality was driven by different experiments/users/components (ND v. FD)
  - Maintainence and keeping new developments in sync was a manual nightmare.
- Then there was...



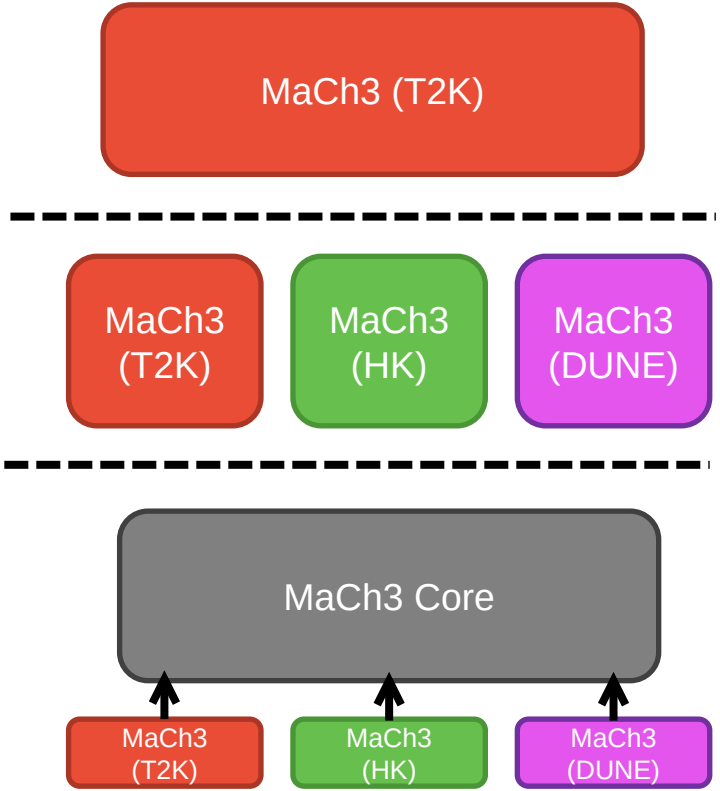
The Split

- In the beginning ther
  - T2K-only
- Then we had copies
  experiments:
  - T2K, HK, DUNE
- New functionality wa
  experiments/users/c
  - Maintainence and ke
    sync was a manual r
- Then there was...



The Split

- In the beginning there was only MaCh3:
  - T2K-only
- Then we had copies of the code for different experiments:
  - T2K, HK, DUNE
- New functionality was driven by different experiments/users/components (ND v. FD)
  - Maintainence and keeping new developments in sync was a manual nightmare.
- Then there was...
  - Core was born!
- A versioned home for experiment agnostic code!



What's Does Core Know About?

- Mainly concerned with SampleHandler/LLH
- Some Event-level info:
  - https://github.com/mach3-software/MaCh3/blob/v2.7.1/Samples/EventInfo.h
    - Neutrino species (before/after osc)
    - Neutrino energy
    - Neutrino approach angle (atmospheric)
    - CC/NC interaction

```
/// Pointers to weights like oscillation spline, normalisation etc
std::vector<const M3::float_t*> total_weight_pointers;

/// The x_var and y_vars and beyond that you're binning in
std::vector<const double*> KinVar;
/// starting bins for each dimensions allowing to perform quick lookup
std::vector<int> NomBin;

/// Nominal sample to which event is associated
int NominalSample = M3::_BAD_INT_;

/// PDG of neutrino after oscillation
int nupdg  = M3::_BAD_INT_;
/// PDG of neutrino before oscillation
int nupdgUnosc = M3::_BAD_INT_;
/// Is event NC or not
bool isNC = false;

/// Pointer to true Neutrino Energy
double enu_true = M3::_BAD_DOUBLE_;
/// Pointer to true cosine zenith
double coszenith_true = M3::_BAD_DOUBLE_;
```

What's Does Core Know About?

- Mainly concerned with SampleHandler/LLH
- Some Event-level info:
  - https://github.com/mach3-software/MaCh3/blob/v2.7.1/Samples/EventInfo.h
    - Neutrino species (before/after osc)
    - Neutrino energy
    - Neutrino approach angle (atmospheric)
    - CC/NC interaction
- Splines:
  - Binned: In-file naming convention is parsed by experiment, but the rest of the handling (reading/storing/applying) is in core.
  - Ev-by-ev: Handled at the experiment level, but common format splines passed back to core.

```
/// Pointers to weights like oscillation spline, normalisation etc
std::vector<const M3::float_t*> total_weight_pointers;

/// The x_var and y_vars and beyond that you're binning in
std::vector<const double*> KinVar;
/// starting bins for each dimensions allowing to perform quick lookup
std::vector<int> NomBin;

/// Nominal sample to which event is associated
int NominalSample = M3::_BAD_INT_;

/// PDG of neutrino after oscillation
int nupdg  = M3::_BAD_INT_;
/// PDG of neutrino before oscillation
int nupdgUnosc = M3::_BAD_INT_;
/// Is event NC or not
bool isNC = false;

/// Pointer to true Neutrino Energy
double enu_true = M3::_BAD_DOUBLE_;
/// Pointer to true cosine zenith
double coszenith_true = M3::_BAD_DOUBLE_;
```

What's Does Core Know About?

- Mainly concerned with SampleHandler/LLH
- Some Event-level info:
  - https://github.com/mach3-software/MaCh3/blob/v2.7.1/Samples/EventInfo.h
    - Neutrino species (before/after osc)
    - Neutrino energy
    - Neutrino approach angle (atmospheric)
    - CC/NC interaction
- Splines:
  - Binned: In-file naming convention is parsed by experiment, but the rest of the handling (reading/storing/applying) is in core.
  - Ev-by-ev: Handled at the experiment level, but common format splines passed back to core.

```
/// Pointers to weights like oscillation spline, normalisation etc
std::vector<const M3::float_t*> total_weight_pointers;

/// The x_var and y_vars and beyond that you're binning in
std::vector<const double*> KinVar;
/// starting bins for each dimensions allowing to perform quick lookup
std::vector<int> NomBin;
```

Allows oscillation weights to be fully handled in Core

```
/// Pointer to true Neutrino Energy
double enu_true = M3::_BAD_DOUBLE_;
/// Pointer to true cosine zenith
double coszenith_true = M3::_BAD_DOUBLE_;
```

What's Does Core Know About?

- Mainly concerned with SampleHandler/LLH
- Some Event-level info:
  - https://github.com/mach3-software/MaCh3/blob/v2.7.1/Samples/EventInfo.h
    - Neutrino species (before/after osc)
    - Neutrino energy
    - Neutrino approach angle (atmospheric)
    - CC/NC interaction
- Splines:
  - Binned: In-file naming convention is parsed by experiment, but the rest of the handling (reading/storing/applying) is in core.
  - Ev-by-ev: Handled at the experiment level, but common format splines passed back to core.

```
/// Pointers to weights like oscillation spline, normalisation etc
std::vector<const M3::float_t*> total_weight_pointers;

/// The x_var and y_vars and beyond that you're binning in
std::vector<const double*> KinVar;
/// starting bins for each dimensions allowing to perform quick lookup
std::vector<int> NomBin;
```

Allows oscillation weights to be fully handled in Core

Binning is handled in core, so if a spline knows which bin it applies to, no more info required,

What's Does Core Know About?

- Mainly concerned with SampleHandler/LLH
- Some Event-level info:
  - https://github.com/mach3-software/MaCh3/blob/v2.7.1/Samples/EventInfo.h
    - Neutrino species (before/after osc)
    - Neutrino energy
    - Neutrino approach angle (atmospheric)
    - CC/NC interaction
- Splines:
  - Binned: In-file naming convention is parsed by experiment, but the rest of the handling (reading/storing/applying) is in core.
  - Ev-by-ev: Handled at the experiment level, but common format splines passed back to core.

```
/// Pointers to weights like oscillation spline, normalisation etc
std::vector<const M3::float_t*> total_weight_pointers;

/// The x_var and y_vars and beyond that you're binning in
std::vector<const double*> KinVar;
/// starting bins for each dimensions allowing to perform quick lookup
std::vector<int> NomBin;
```

Allows oscillation weights to be fully handled in Core

Binning is handled in core, so if a spline knows which bin it applies to, no more info required.

Event looping and histogramming driven in core.

A Tale of Two (Three) Philosophies

- **Event Info:** Fixed structure in core for required information, experiment tasked with setting it per event.

A Tale of Two (Three) Philosophies

- **Event Info:** Fixed structure in core for required information, experiment tasked with setting it per event.
- **Binned Splines:** All functionality is in core except spline name ordering
  - Could alternatively mandate a name ordering (no components are optional anyway) and then purely a core functionality.

A Tale of Two (Three) Philosophies

- **Event Info:** Fixed structure in core for required information, experiment tasked with setting it per event.
- **Binned Splines:** All functionality is in core except spline name ordering
  - Could alternatively mandate a name ordering (no components are optional anyway) and then purely a core functionality.
- **Event-splines:** Helper functions in core that support easy conventional usage by experiments. Interface is actually fully flexible

A Tale of Two (Three) Philosophies

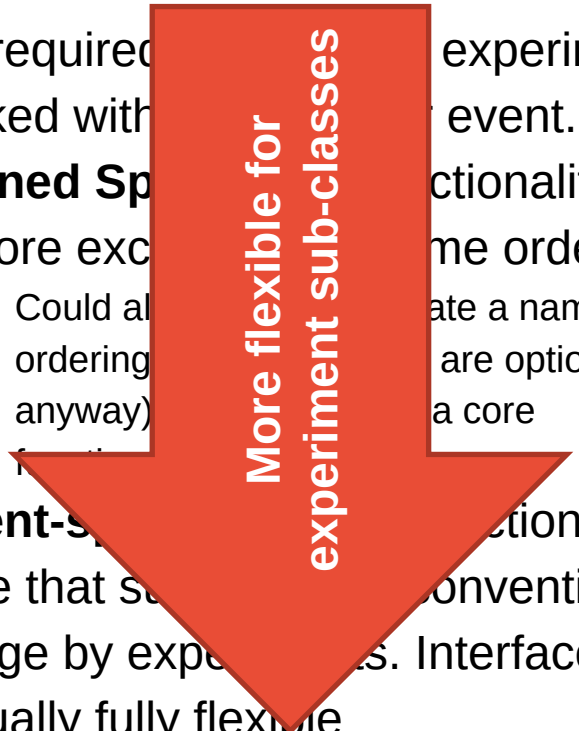
- **Event Info:** Fixed structure in core for required information, experiment tasked with setting it per event.
- **Binned Splines:** All functionality is in core except spline name ordering
  - Could alternatively mandate a name ordering (no components are optional anyway) and then purely a core functionality.
- **Event-splines:** Helper functions in core that support easy conventional usage by experiments. Interface is actually fully flexible



- **Binned Splines:** All functionality is in core except spline name ordering
  - Could alternatively mandate a name ordering (no components are optional anyway) and then purely a core functionality.
- **Event Info:** Fixed structure in core for required information, experiment tasked with setting it per event.
- **Event-splines:** Helper functions in core that support easy conventional usage by experiments. Interface is actually fully flexible

A Tale of Two (Three) Philosophies

- **Event Info:** Fixed structure in core for required information, experiment tasked with setting it per event.
- **Binned Splines:** All functionality is in core except spline name ordering
  - Could alternatively mandate a name ordering (no components are optional anyway) and then purely a core functionality.
- **Event-splines:** Helper functions in core that support easy conventional usage by experiments. Interface is actually fully flexible



Abstract Approaches For Sharing Information

- **No Experiment-specific Code:** Core initialisation routines pick up standard configuration keys.

```
44      InputFiles:
45        mtupleprefix: "Inputs/DUNE_CAF_files/FD_FHC_ger_"
46        mtuplesuffix: "_nueselec.root"
47        splineprefix: "Inputs/DUNE_spline_files/FD_FHC_ger_"
48        splinesuffix: "_nueselec_splines.root"
49
50
51      OscChannels:
52        - Name: "FHC_numu_x_numu"
53          LatexName: "FHC_numu_x_numu"
54          mtuplefile: ["numu_x_numu"]
55          splinefile: "numu_x_numu"
56          nutype: 14
57          oscnutype: 14
```

Abstract Approaches For Sharing Information

- **No Experiment-specific Code:** Core initialisation routines pick up standard configuration keys.
- **Experiment Directly Modifies Core-owned Data:** Relies on shared event index and public/protected inheritance.
  - This is how the required event-level data is set when reading experiment-specifc formats

```
void SampleHandlerBeamFD::SetupMC() {
  // dunemc_base *duneobj = &(dunemcSamples[iSample]);
  // FarDetectorCoreInfo *fdobj = &(MCEvents[iSample]);

  for (unsigned int iEvent = 0; iEvent < GetNEvents(); ++iEvent) {
    MCEvents[iEvent].enu_true = dunemcSamples[iEvent].enu_true;
    MCEvents[iEvent].isNC = !(dunemcSamples[iEvent].rw_isCC);
    MCEvents[iEvent].nupdg = dunemcSamples[iEvent].nupdg;
    MCEvents[iEvent].nupdgUnosc = dunemcSamples[iEvent].nupdgUnosc;
    MCEvents[iEvent].NominalSample = dunemcSamples[iEvent].SampleIndex;
  }

}
```

Abstract Approaches For Sharing Information

```
virtual const double* GetPointerToKinematicParameter(const int KinematicVariable, const int iEvent) const = 0;
```

```
const double* SampleHandlerBeamFD::GetPointerToKinematicParameter(const int KinPar, const int iEvent) const {
  switch(KinPar){
  case kTrueNeutrinoEnergy:
    return &(dunemcSamples[iEvent].enu_true);
  case kRecoNeutrinoEnergy:
    return &(dunemcSamples[iEvent].rw_erec_shifted);
    break;
  case kTrueXPos:
    return &(dunemcSamples[iEvent].rw_vtx_x);
  case kTrueYPos:
    return &(dunemcSamples[iEvent].rw_vtx_y);
```

- **Experiment Overrides Virtual Base Class Methods**: Often relies on shared event index:

- **Experiment Supplies Callback Function:**
  - Equivalent to the above, but with free functions/lambdas rather than virtual functions.

Abstract Approaches For Sharing Information

- ```
  RegisterIndividualFunctionalParameter(
      DUNEMCEvents, syst::GetFluxFocussingParamNames(),
      [](std::vector<double> const &par_vals, EventInfo &ev) {
        for (size_t i = 0; i < par_vals.size(); ++i) {
          ev.syst.flux.total_weight *=
              1 + (par_vals[i] * (ev.syst.flux.focussing_weights[i] - 1));
        }
      });
  ```
- **Experiment Overrides Virtual Base Class Methods**: Often relies on shared event index:
- **Experiment Supplies Callback Function:**
  - Equivalent to the above, but with free functions/lambdas rather than virtual functions.
  ```

CRTP?

- The event-level required information must currently be duplicated between the experiment and core:
  - Read in from experiment-specific data formats
  - Copied to core structures

Core

```
struct EventInfo {
  /// @brief Default constructor.
  EventInfo(){}
  /// @brief Copy constructor (deleted to prevent copying).
  EventInfo(EventInfo const &other) = delete;
  /// @brief Move constructor (defaulted to allow moving).
  EventInfo(EventInfo &&other) = default;
  /// @brief Copy assignment operator (deleted).
  EventInfo& operator=(EventInfo const &other) = delete;
  /// @brief Move assignment operator (deleted).
  EventInfo& operator=(EventInfo &&other) = delete;
  /// @brief default destructor
  ~EventInfo(){}

  /// Pointers to weights like oscillation spline, normalisation etc
  std::vector<const M3::float_t*> total_weight_pointers;

  /// The x_var and y_vars and beyond that you're binning in
  std::vector<const double*> KinVar;
  /// starting bins for each dimensions allowing to perform quick lookup
  std::vector<int> NomBin;

  /// Nominal sample to which event is associated
  int NominalSample = M3::_BAD_INT_;

  /// PDG of neutrino after oscillation
  int nupdg   = M3::_BAD_INT_;
  /// PDG of neutrino before oscillation
  int nupdgUnosc = M3::_BAD_INT_;
  /// Is event NC or not
  bool isNC = false;

  /// Pointer to true Neutrino Energy
  double enu_true = M3::_BAD_DOUBLE_;
  /// Pointer to true cosine zenith
  double coszenith_true = M3::_BAD_DOUBLE_;
};
```

Copy

Subclass

```
struct CAFEventInfo {
  struct Truth {

    int generator_mode;
    double tgt_a;
    double mach3_mode;
    double is_cc;

    struct Neutrino {
      int pdg, pdg_unosc;
      double e;
    } nu;

  } truth;
```

CRTP?

- The event-level required information must currently be duplicated between the experiment and core:
  - Read in from experiment-specific data formats
  - Copied to core structures
- CRTP – Curiously Recurring Template Pattern
  - Base class is a template class which is specialized over the subclass
  - Base class ‘knows’ about the subclass EventInfo structure and keeps a vector of those rather than it’s own type
- Not the most urgent or problematic, but just an interesting use-case for an interesting design pattern.
- Use ~this pattern for method templates in the functional argument interface.

CRTP?

- The event-level required information must currently be duplicated between
- than it's own type

```
RegisterIndividualFunctionalParameter(
    DUNEMCEvents, syst::GetFluxFocussingParamNames(),
    [](std::vector<double> const &par_vals, EventInfo &ev) {
      for (size_t i = 0; i < par_vals.size(); ++i) {
        ev.syst.flux.total_weight *=
            1 + (par_vals[i] * (ev.syst.flux.focussing_weights[i] - 1));
      }
    });
```

Subclass

- Not the most urgent or problematic, but just an interesting use-case for an interesting design pattern.
- Use ~this pattern for method templates in the functional argument interface.

CRTP?

- The event-level required information must currently be duplicated between
- ner than it's own type

```
RegisterIndividualFunctionalParameter(
    DUNEMCEvents, syst::GetFluxFocussingParamNames(),
    [](std::vector<double> const &par_vals, EventInfo &ev) {
      for (size_t i = 0; i < par_vals.size(); ++i) {
        ev.syst.flux.total_weight *=
            1 + (par_vals[i] * (ev.syst.flux.focussing_weights[i] - 1));
      }
    });
```

Subclass

- Not the most urgent or problematic, but just an interesting use-case for an interesting design pattern.
- Use ~this pattern for method templates in the functional argument interface.

CRTP?

- The event-level required information must currently be duplicated between
- Not the most urgent or problematic, but just an interesting use-case for an interesting d
- Use ~this pa
  - To me it n
  - (template

```
RegisterIndividualFunctionalParameter(
    DUNEMCEvents, syst::GetFluxFocussingParamNames(),
    [](std::vector<double> const &par_vals, EventInfo &ev) {
      for (size_t i = 0; i < par_vals.size(); ++i) {
        ev.syst.flux.total_weight *=
            1 + (par_vals[i] * (ev.syst.flux.focussing_weights[i] - 1));
      }
    });
```

Subclass

```
template <typename EventType, typename SFType>
void RegisterIndividualFunctionalParameter(
    std::vector<EventType> &ExptEvents,
    std::vector<std::string> const &par_names,
    SFType shift_func);
```

Core

Discussion Points

- Experiment/Core split is unambiguous benefit for many many reasons

- But exactly where the split lies and how information crosses has evolved organically:
  - Worth actively designing/writing down? (I think so)
  - Doesn't have to be one paradigm for all moving parts, some things can be fully offloaded to core and driven just by a fixed structure and configuration but some things need hooks/overideability.

- General push for all configuration all the time:
  - But IMHO sometimes things are easier to express in code.
  - Flexibility of hooks and helper functions is key.
  - If common patterns are implemented configuration only, thats great too!