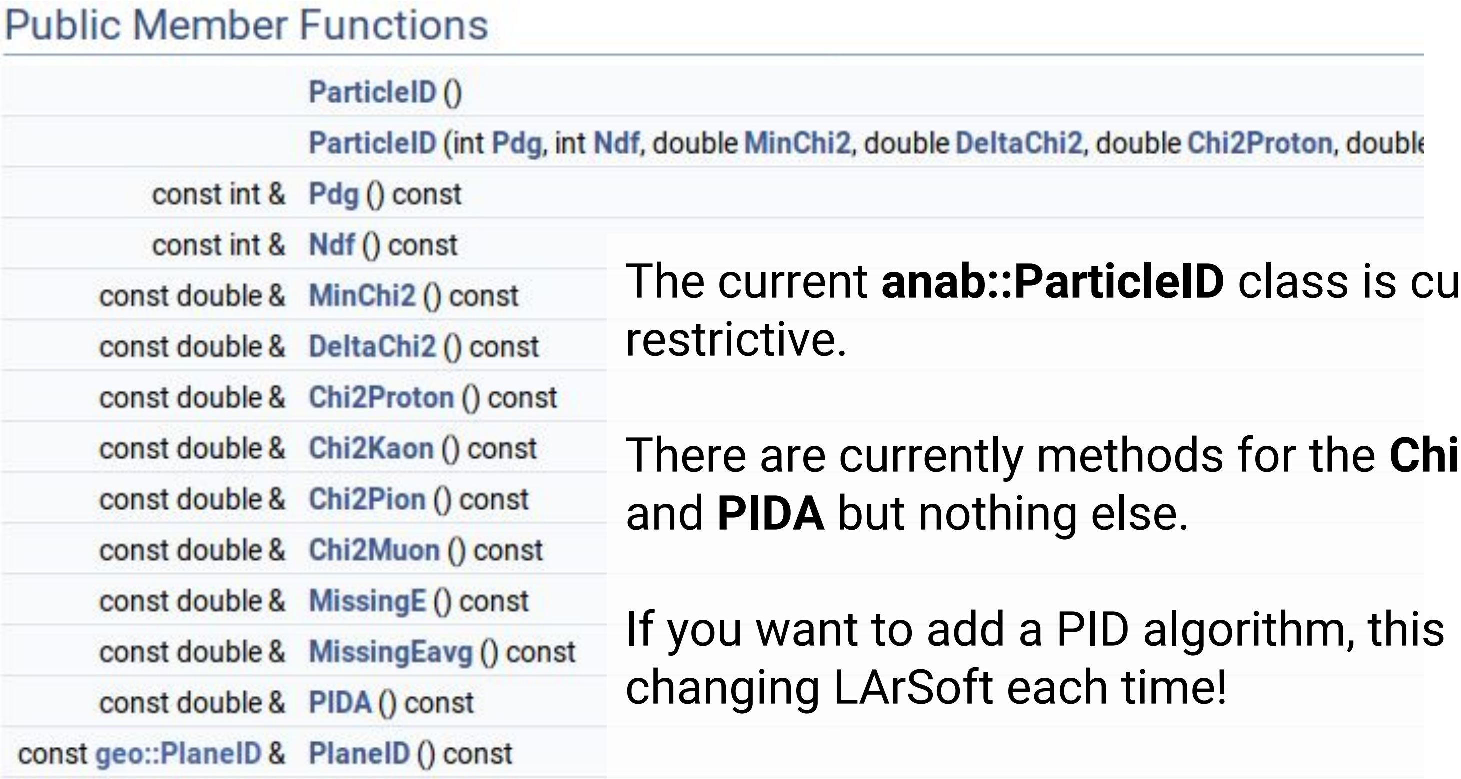
Restructuring anab::ParticleID

Kirsty Duffy and **Adam Lister**

Introduction



Public Member Functions

	ParticleID ()
	ParticleID (int Pdg, int Ndf, double MinChi2, double DeltaChi2, double Chi2Proton, doubl
const int &	Pdg () const
const int &	Ndf () const
const double &	MinChi2 () const
const double &	DeltaChi2 () const
const double &	Chi2Proton () const
const double &	Chi2Kaon () const
const double &	Chi2Pion () const
const double &	Chi2Muon () const
const double &	MissingE () const
const double &	MissingEavg () const
const double &	PIDA () const
const geo::PlaneID &	PlaneID () const

The current **anab::ParticleID** class is currently **very** restrictive.

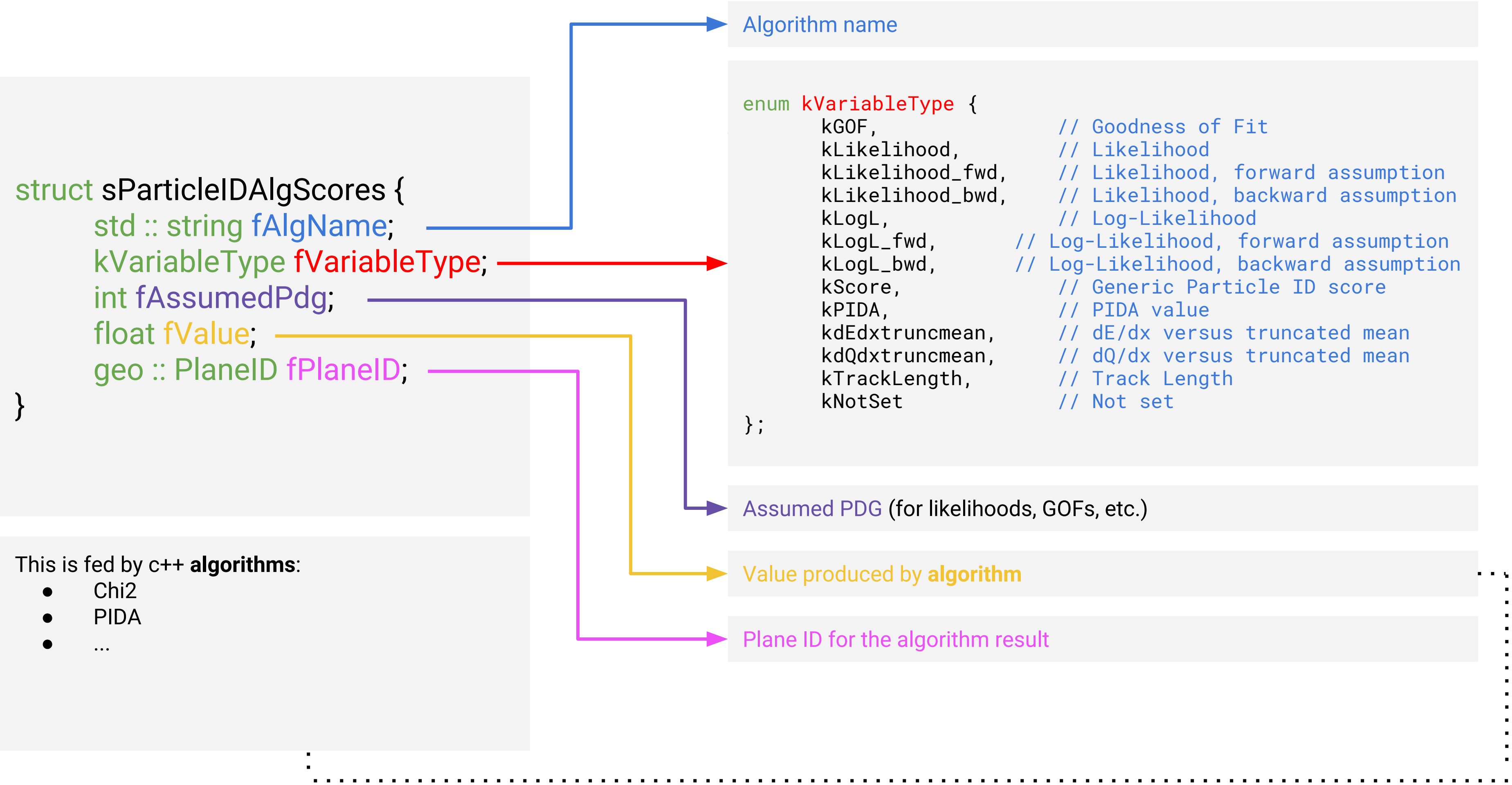
There are currently methods for the **Chi2** algorithm and **PIDA** but nothing else.

If you want to add a PID algorithm, this requires changing LArSoft each time!

We've recently been doing some PID work on MicroBooNE. In the process, we have developed a new organisation of the anab::ParticleID class which is **easily extendable**, and should be able to hold results for any potential algorithm we could think of.

New Struct

The change comes down to addition of a new vector of **sPArticleIDAlgScores** structs to the class.



```
struct sParticleIDAlgScores {
        std :: string fAlgName;
        kVariableType fVariableType;
        int fAssumedPdg;
        float fValue;
        geo :: PlaneID fPlaneID;
}
```

Algorithm name

```
enum kVariableType {
        kGOF,                  // Goodness of Fit
        kLikelihood,           // Likelihood
        kLikelihood_fwd,       // Likelihood, forward assumption
        kLikelihood_bwd,       // Likelihood, backward assumption
        kLogL,                 // Log-Likelihood
        kLogL_fwd,         // Log-Likelihood, forward assumption
        kLogL_bwd,         // Log-Likelihood, backward assumption
        kScore,                // Generic Particle ID score
        kPIDA,                 // PIDA value
        kdEdxtruncmean,        // dE/dx versus truncated mean
        kdQdxtruncmean,        // dQ/dx versus truncated mean
        kTrackLength,          // Track Length
        kNotSet                // Not set
};
```

Assumed PDG (for likelihoods, GOFs, etc.)

Value produced by **algorithm**

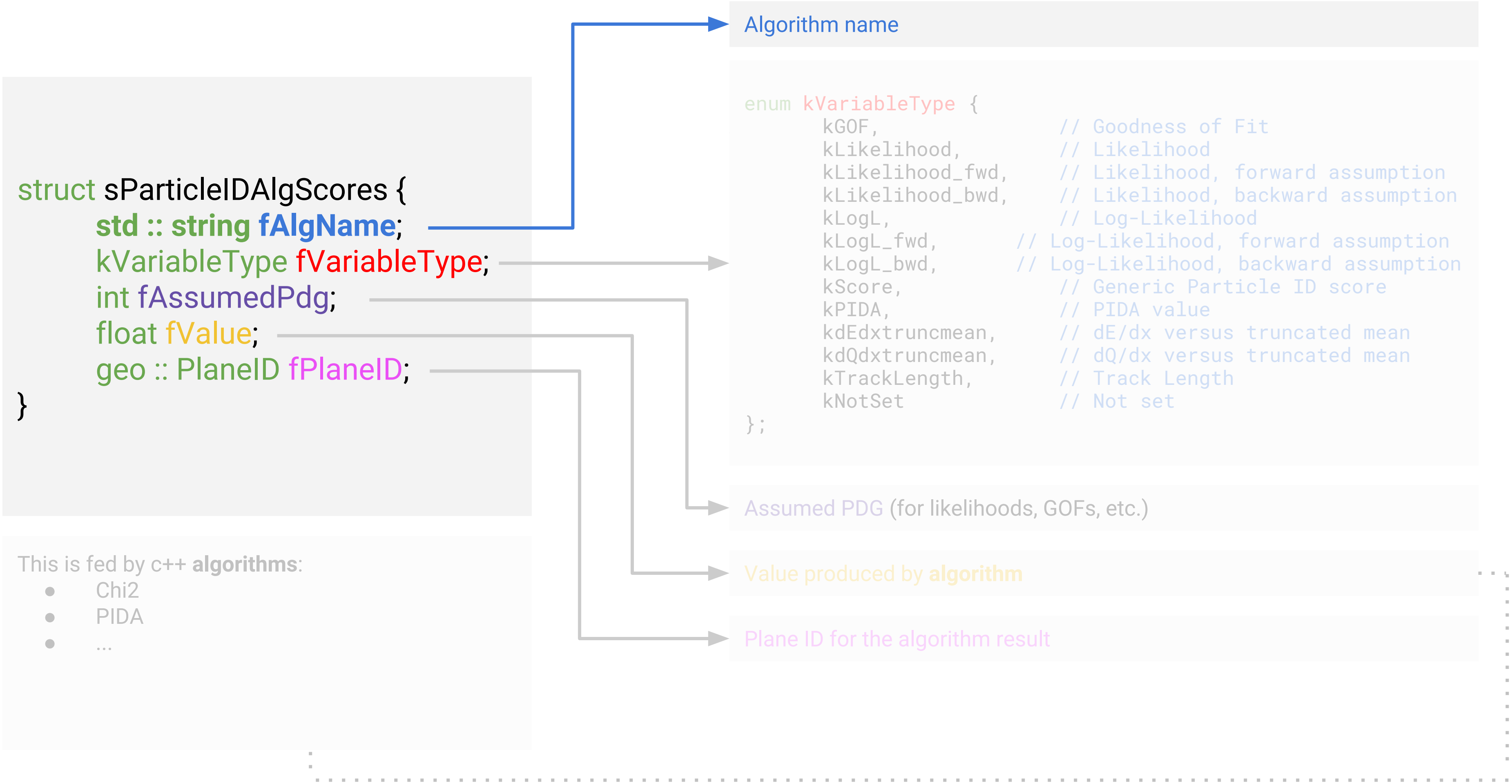
Plane ID for the algorithm result

This is fed by c++ **algorithms**:

- Chi2
- PIDA
- ...

New Struct: fAlgName

fAlgName: this is just a string which can be used to identify an algorithm in the absence of anything else (“Chi2”, “PIDA_mean”, etc.).



```
struct sParticleIDAlgScores {
        std :: string fAlgName;
        kVariableType fVariableType;
        int fAssumedPdg;
        float fValue;
        geo :: PlaneID fPlaneID;
}
```

Algorithm name

```
enum kVariableType {
       kGOF,                   // Goodness of Fit
       kLikelihood,            // Likelihood
       kLikelihood_fwd,        // Likelihood, forward assumption
       kLikelihood_bwd,        // Likelihood, backward assumption
       kLogL,                  // Log-Likelihood
       kLogL_fwd,          // Log-Likelihood, forward assumption
       kLogL_bwd,          // Log-Likelihood, backward assumption
       kScore,                 // Generic Particle ID score
       kPIDA,                  // PIDA value
       kdEdxtruncmean,         // dE/dx versus truncated mean
       kdQdxtruncmean,         // dQ/dx versus truncated mean
       kTrackLength,           // Track Length
       kNotSet                 // Not set
};
```

Assumed PDG (for likelihoods, GOFs, etc.)

Value produced by **algorithm**

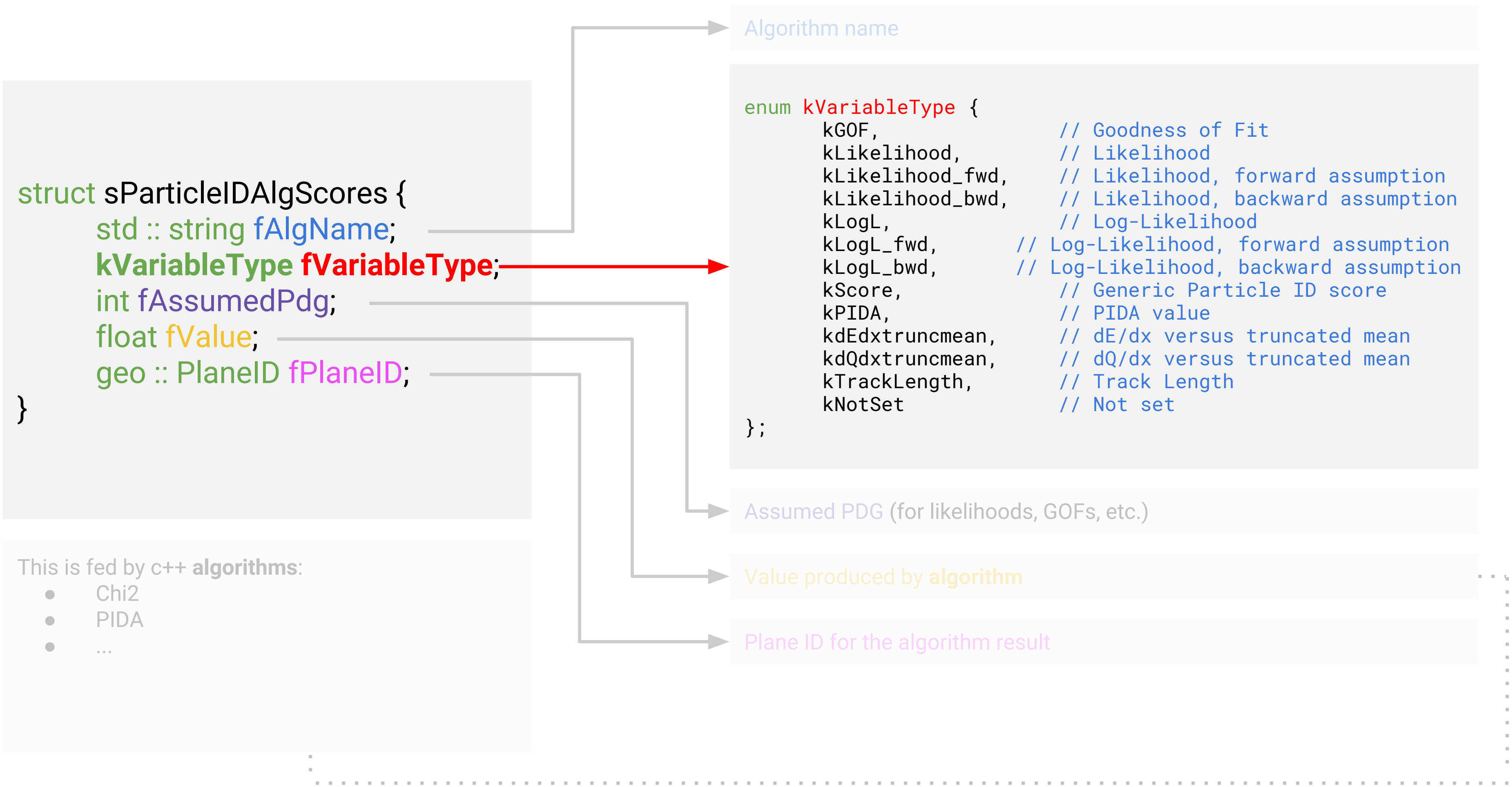
Plane ID for the algorithm result

This is fed by c++ **algorithms**:

- Chi2
- PIDA
- ...

New Struct: fVariableType

kVariableType: an enum which can be used to easily get at the type of variable you want.



```
struct sParticleIDAlgScores {
        std :: string fAlgName;
        kVariableType fVariableType;
        int fAssumedPdg;
        float fValue;
        geo :: PlaneID fPlaneID;
}
```

This is fed by c++ **algorithms**:

- Chi2
- PIDA
- ...

Algorithm name

```
enum kVariableType {
        kGOF,                   // Goodness of Fit
        kLikelihood,            // Likelihood
        kLikelihood_fwd,        // Likelihood, forward assumption
        kLikelihood_bwd,        // Likelihood, backward assumption
        kLogL,                  // Log-Likelihood
        kLogL_fwd,          // Log-Likelihood, forward assumption
        kLogL_bwd,          // Log-Likelihood, backward assumption
        kScore,                 // Generic Particle ID score
        kPIDA,                  // PIDA value
        kdEdxtruncmean,         // dE/dx versus truncated mean
        kdQdxtruncmean,         // dQ/dx versus truncated mean
        kTrackLength,           // Track Length
        kNotSet                 // Not set
};
```

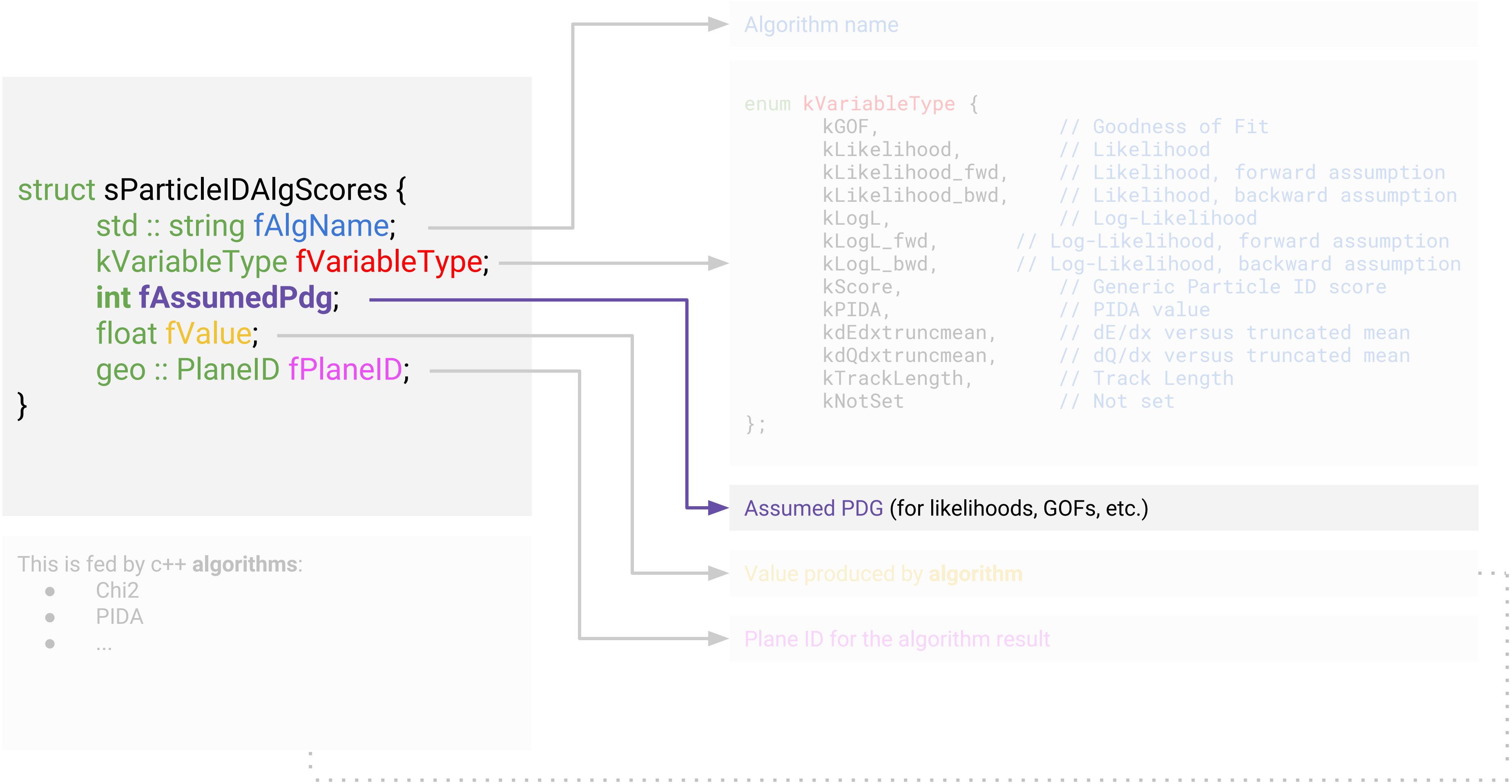
Assumed PDG (for likelihoods, GOFs, etc.)

Value produced by **algorithm**

Plane ID for the algorithm result

New Struct: fAssumedPdg

fAssumedPdg: This is used for algorithms where an assumption about the particle species is made (e.g. Chi2 with respect to the Muon hypothesis).



```
struct sParticleIDAlgScores {
        std :: string fAlgName;
        kVariableType fVariableType;
        int fAssumedPdg;
        float fValue;
        geo :: PlaneID fPlaneID;
}
```

This is fed by c++ **algorithms**:

- Chi2
- PIDA
- ...

Algorithm name

```
enum kVariableType {
        kGOF,                   // Goodness of Fit
        kLikelihood,            // Likelihood
        kLikelihood_fwd,        // Likelihood, forward assumption
        kLikelihood_bwd,        // Likelihood, backward assumption
        kLogL,                  // Log-Likelihood
        kLogL_fwd,          // Log-Likelihood, forward assumption
        kLogL_bwd,          // Log-Likelihood, backward assumption
        kScore,                 // Generic Particle ID score
        kPIDA,                  // PIDA value
        kdEdxtruncmean,         // dE/dx versus truncated mean
        kdQdxtruncmean,         // dQ/dx versus truncated mean
        kTrackLength,           // Track Length
        kNotSet                 // Not set
};
```

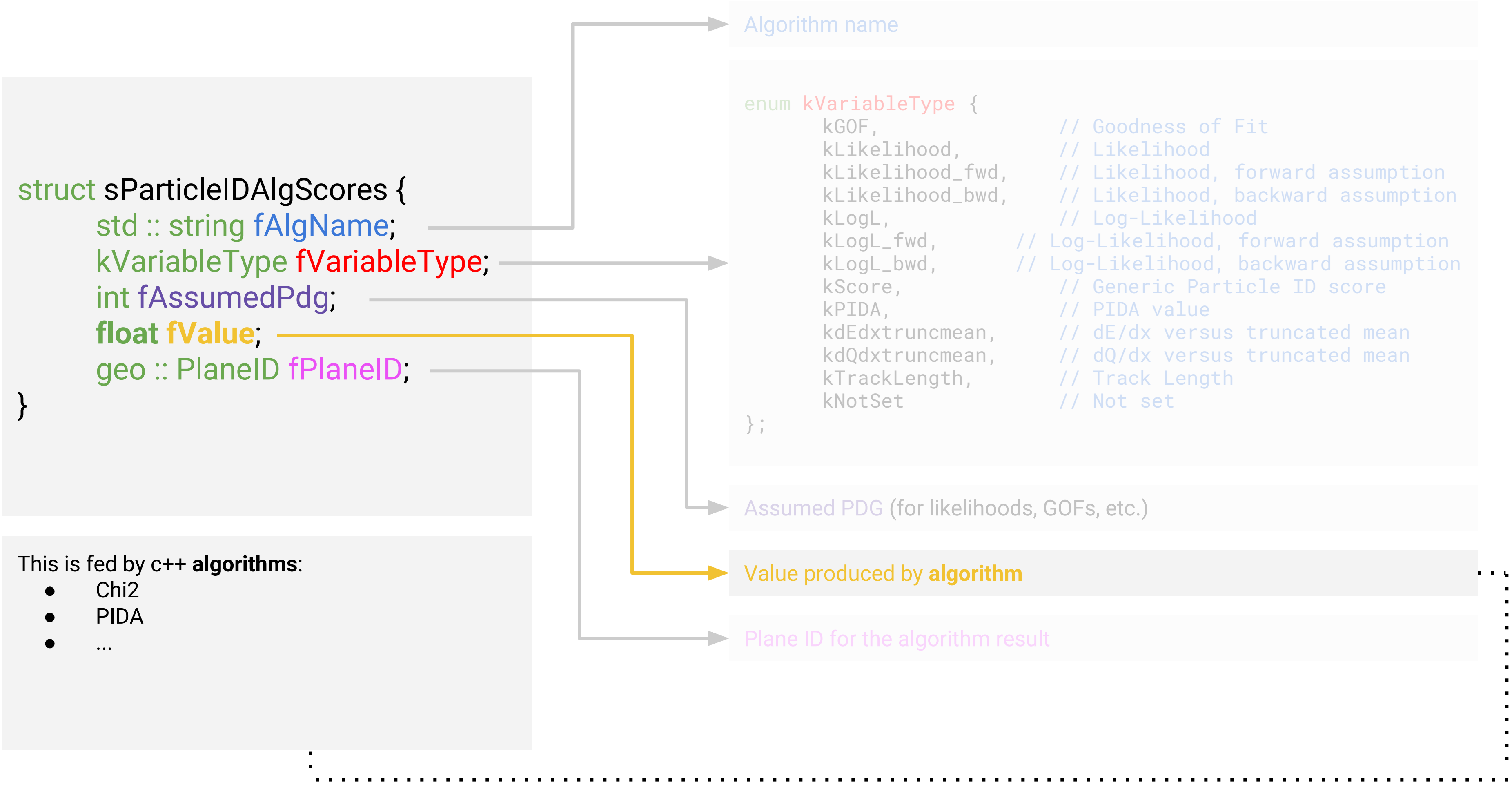
Assumed PDG (for likelihoods, GOFs, etc.)

Value produced by **algorithm**

Plane ID for the algorithm result

New Struct: fValue

fValue: This contains the value or score from a list of algorithms which feed the ParticleID producer module. These algorithms can be general use or experiment specific!



```
struct sParticleIDAlgScores {
        std :: string fAlgName;
        kVariableType fVariableType;
        int fAssumedPdg;
        float fValue;
        geo :: PlaneID fPlaneID;
}
```

This is fed by c++ **algorithms**:

- Chi2
- PIDA
- ...

Algorithm name

```
enum kVariableType {
       kGOF,                 // Goodness of Fit
       kLikelihood,          // Likelihood
       kLikelihood_fwd,      // Likelihood, forward assumption
       kLikelihood_bwd,      // Likelihood, backward assumption
       kLogL,                // Log-Likelihood
       kLogL_fwd,        // Log-Likelihood, forward assumption
       kLogL_bwd,        // Log-Likelihood, backward assumption
       kScore,               // Generic Particle ID score
       kPIDA,                // PIDA value
       kdEdxtruncmean,       // dE/dx versus truncated mean
       kdQdxtruncmean,       // dQ/dx versus truncated mean
       kTrackLength,         // Track Length
       kNotSet               // Not set
};
```

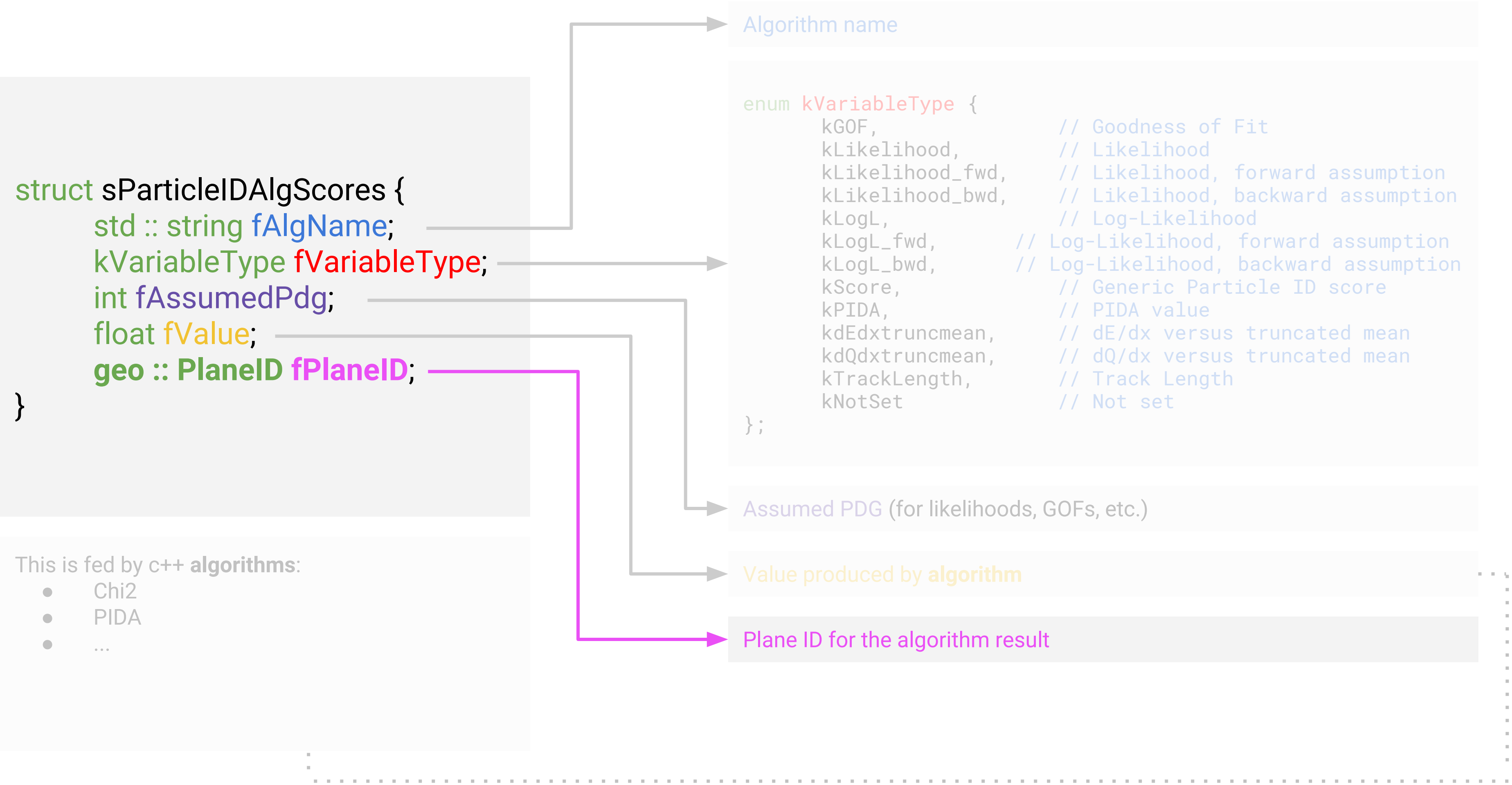
Assumed PDG (for likelihoods, GOFs, etc.)

Value produced by **algorithm**

Plane ID for the algorithm result

New Struct: fPlaneID

fPlaneID: Many algorithms make use of charge information from a single plane. This allows you to know which!



```
struct sParticleIDAlgScores {
        std :: string fAlgName;
        kVariableType fVariableType;
        int fAssumedPdg;
        float fValue;
        geo :: PlaneID fPlaneID;
}
```

Algorithm name

```
enum kVariableType {
        kGOF,                   // Goodness of Fit
        kLikelihood,            // Likelihood
        kLikelihood_fwd,        // Likelihood, forward assumption
        kLikelihood_bwd,        // Likelihood, backward assumption
        kLogL,                  // Log-Likelihood
        kLogL_fwd,          // Log-Likelihood, forward assumption
        kLogL_bwd,          // Log-Likelihood, backward assumption
        kScore,                 // Generic Particle ID score
        kPIDA,                  // PIDA value
        kdEdxtruncmean,         // dE/dx versus truncated mean
        kdQdxtruncmean,         // dQ/dx versus truncated mean
        kTrackLength,           // Track Length
        kNotSet                 // Not set
};
```

Assumed PDG (for likelihoods, GOFs, etc.)

Value produced by **algorithm**

Plane ID for the algorithm result

This is fed by c++ **algorithms**:

- Chi2
- PIDA
- ...

Concerns With Implementation

- Is there a better way to store these structs than a vector?
  - Each struct contains results from a single plane, for a single assumed particle species. This could easily get unwieldy.
  - Is a map of structs any better than this?
- Should we retire old methods?
  - Breaking changes are undesirable. Possible that we could have two accessors for the same variable in the mid-term but could result in code being written which isn't forward-compatible.
- How do we deal with algorithms with multiple planes?
  - Possible ideas would be a vector of geo::PlaneIDs, or a bitset, but neither of these seems very clean

Any input on these would be really appreciated.

Summary

We think that this reorganisation of the code is much more flexible: it allows for new PID algorithms and can be used for shower PID in addition to track PID.

The main downside is that this relies on the analyser knowing what’s in the struct, and so it requires **good experiment-specific documentation**.

There are also a number of concerns which we have about implementation, as noted on the previous slide.

We’re interested to hear any feedback you have on how this could be improved!