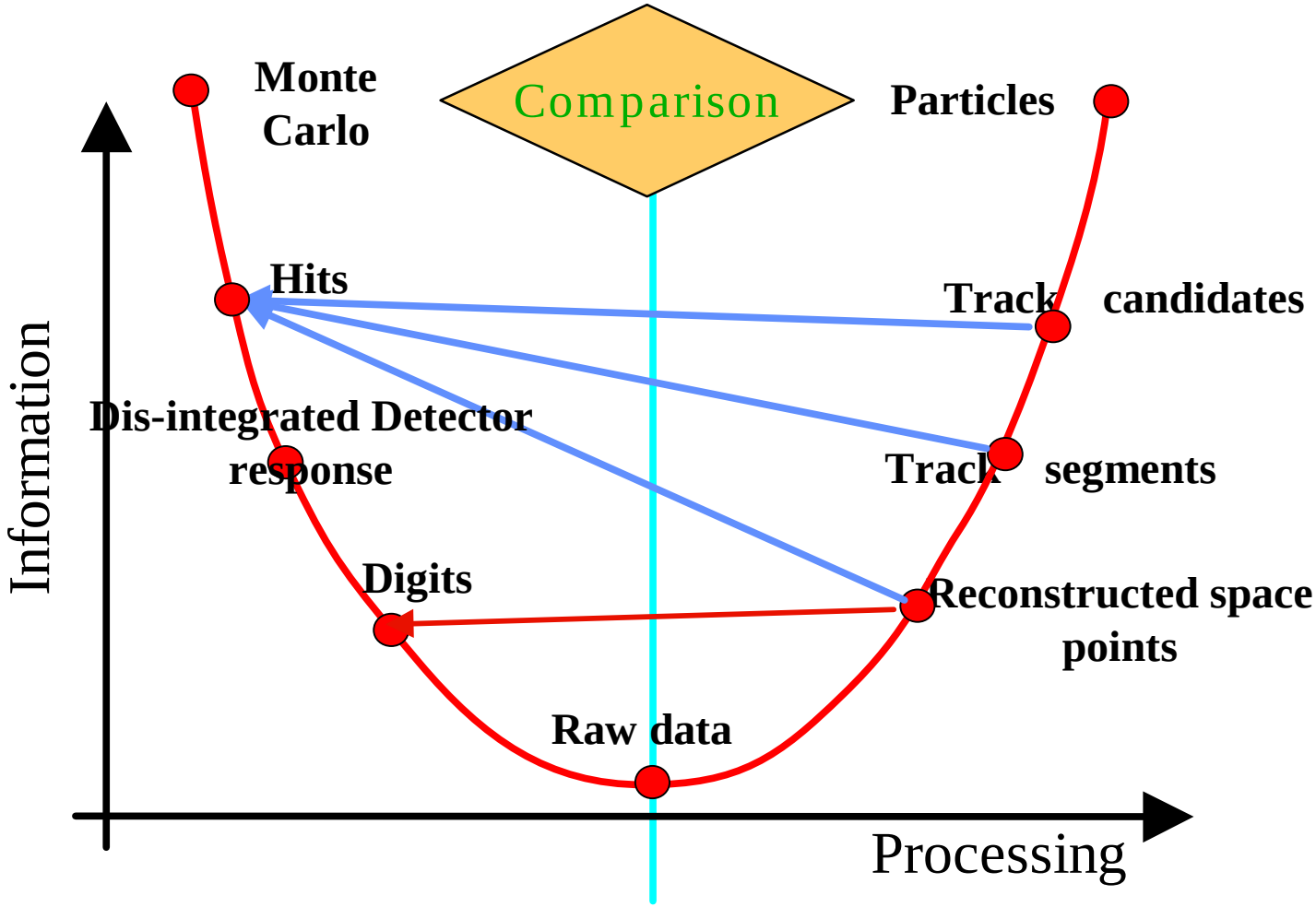
Design and example implementation of Gaudi Associator tools



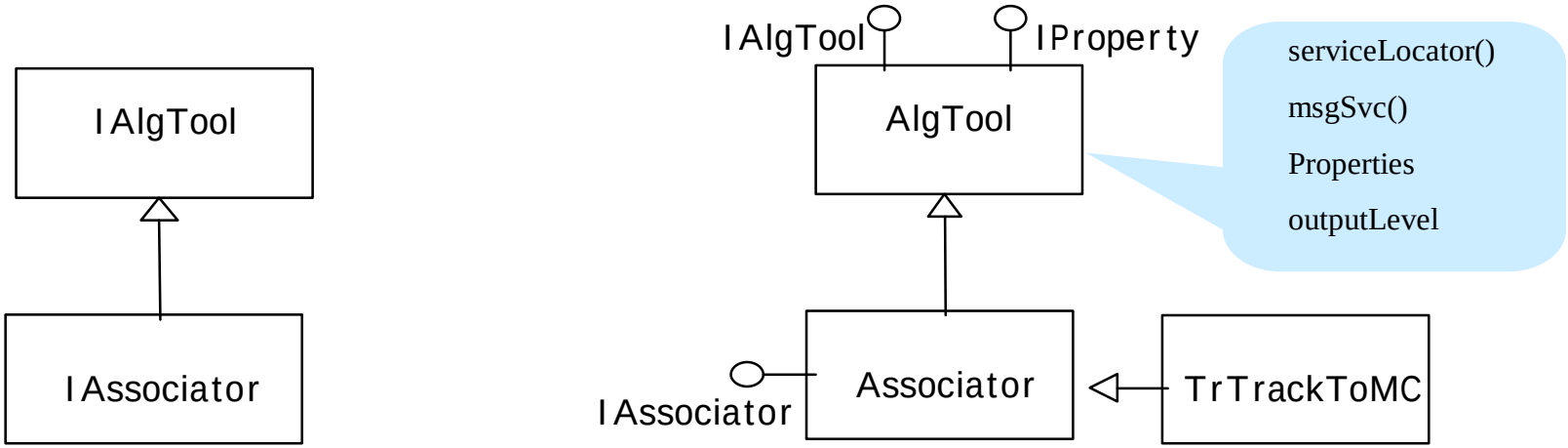
Design and example implementation of Gaudi Associator tools

Three different entities:

- Associator: provides the user with information that is "already" there
  - because of an inheritance mechanism or because it is possible to follow links
  - It will do algorithmic operations when necessary or read short-cuts from the data
  - Concrete associators have local copies of specific reverse connections (time consuming operation)
  - Concrete associators can have local copies of direct connections when those results from long link following or imply algorithmic choices.
  - It can be called at any time during the processing of an event but it should be done only in very well defined monitoring algorithms.

- AssociationMaker: creates the AssociationTables (or in Marco's early schema the MC corresponding class) and puts them in the store.
  - This should be done ONLY at the end of the reconstruction once all of the tracks for example are there.
  - This entity doesn't know anything about the algorithmic operation involved and uses the information returned by a concrete Associators.
- AssociationTable: hold the short-cuts info for high level entities.
  - This is a data object (SmartRefTable ?)
  - It is created and stored by the AssociationMaker but an Associator is able to read it if stored on a file

Associator and IAssociator



- The Associator is a type of AlgTool, it is retrieved via the ToolSvc that takes care of locating the appropriate factory, creates it, and manages it

```
IAssociator* pAsct;
std::string m_asctCalo = "CaloDigitMCSumDepAsct";
 StatusCode sa = m_toolSvc->retrieveTool( m_asctCalo , pAsct );
```

- Properties of the concrete associators can be set via jobOptions

```
ToolSvc.CaloDigit2MCAsct.DataLocation = "/Event/Raw/Ecal/Digits_0";
```

IAssociator interface

Five methods in the IAssociator interface must be implemented by the concrete associators

- flushCache(), to reset status of Associator as it is when it is created
  - it will be called automatically when the Event Data change

protected:

- i_retrieveDirect(ContainedObject* objFrom, ContainedObject*& objTo, const CLID idFrom, const CLID idTo)
  - one-to-one relation
- i_retrieveDirect(ContainedObject* objFrom, std::vector<ContainedObject*>& objTo, const CLID idFrom, const CLID idTo)
  - one-to-many relation
- i_retrieveInverse(….)
  - two corresponding methods for relation reverse to processing
  - could be incorporated in above methods BUT this way aware it is a time consuming operation

IAssociator interface (2)

Templated methods corresponding to the four retrieve methods for the client wanting to use the Associator

```
SmartDataPtr<CaloDigitVector>Digs(eventSvc(),"/Event/Raw/Ecal/Digits_0");
 for( CaloDigitVector::iterator it=Digs->begin(); Digs->end()!=it; ++it ){
    MCCaloSummedDeposit* pMC = 0;
    StatusCode sas = pAsct->retrieveDirect( *it, pMC );
 }
```

Associator Base Class

Additional methods in the base class:

- The standard event data service. Every associator will access the data.

```
IDataProviderSvc*  eventSvc();
```

- Return flag declaring if the associator follows links or looks into stored shortcuts

```
bool followLinks();
```
Property ("FollowLinks")

- Location of data where the associator will look for info

```
std::string whichTable();
```
Property ("DataLocation")

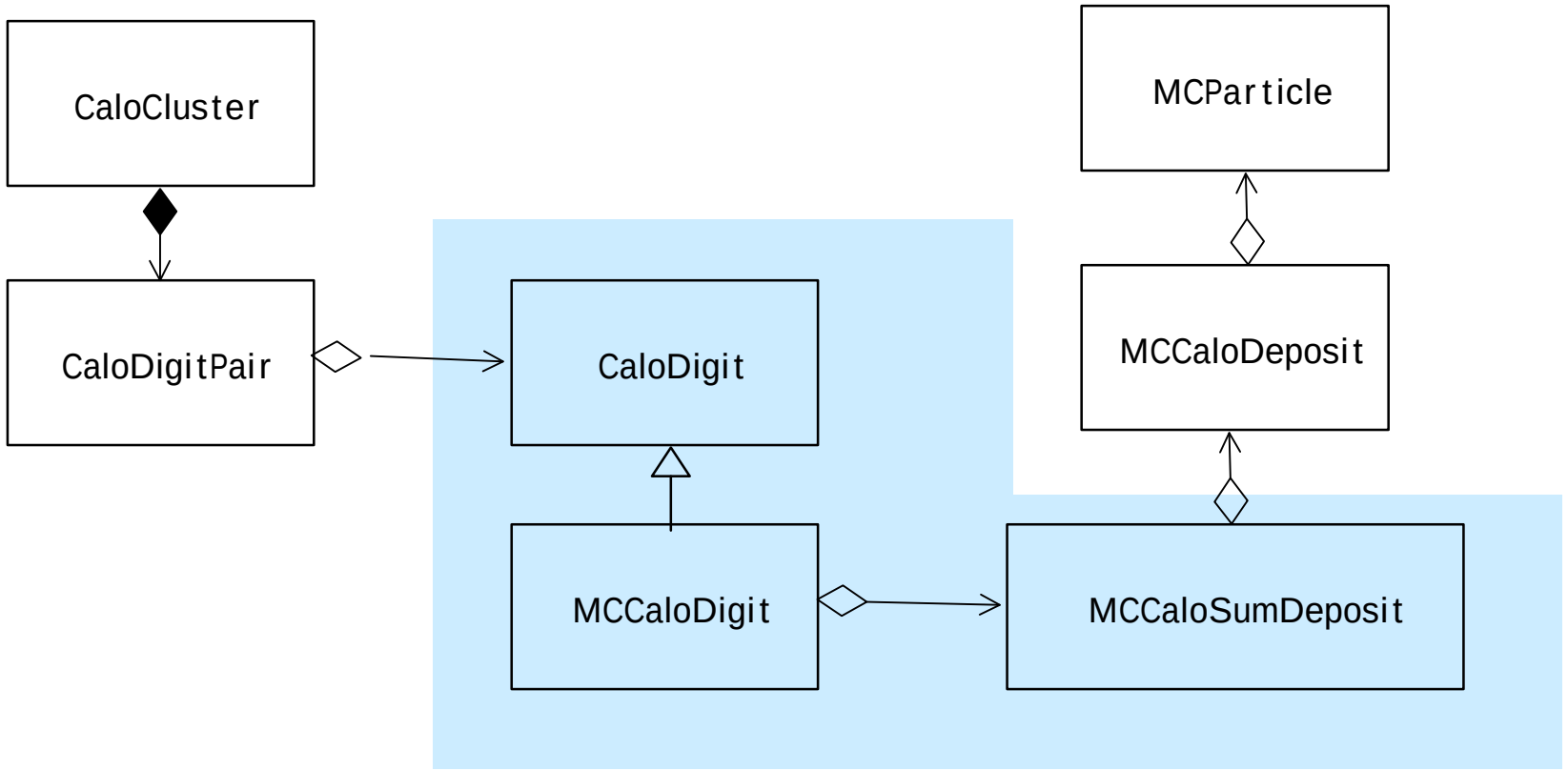
- For inverse association

```
bool inverseExist();
```
flag if the inverse is locally kept

```
void setInverseFlag( bool value ) { m_inverse = value; }
```
protected

```
virtual StatusCode buildReverse() {return StatusCode::SUCCESS;}
```
to be overriden by concrete Asct

Concrete example: CaloDigitMCSumDepAsct



CaloDigitMCSumDepAsct

Need to declare the tool factory

```
static ToolFactory<CaloDigitMCSumDepAsct> s_factory;
const IToolFactory& CaloDigitMCSumDepAsctFactory = s_factory;
```

Need to inherit from Associator

```
CaloDigitMCSumDepAsct::CaloDigitMCSumDepAsct( const std::string& type,
          const std::string& name, const IInterface* parent )      :
Associator ( type, name, parent ), m_inverseTable() { }
```

Implements the interfaces

- i_retrieveDirect (...), i_retrieveInverse(...)
- between CaloDigit* and MCCaloSummedDeposit*
- if CLID are not of the right type return StatusCode::FAILURE and null pointers
- one-to-many in this case return StatusCode::FAILURE and empty vector

Locally keeps the inverse table, filled at first request

The example works on Linux, both for direct and reverse relation

- tried out few CaloDigit and MCCaloSumDeposit for few events and got the same pointers as dynamic_cast
- both the example and the IAssociator and the Associator base class need polishing