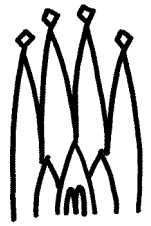
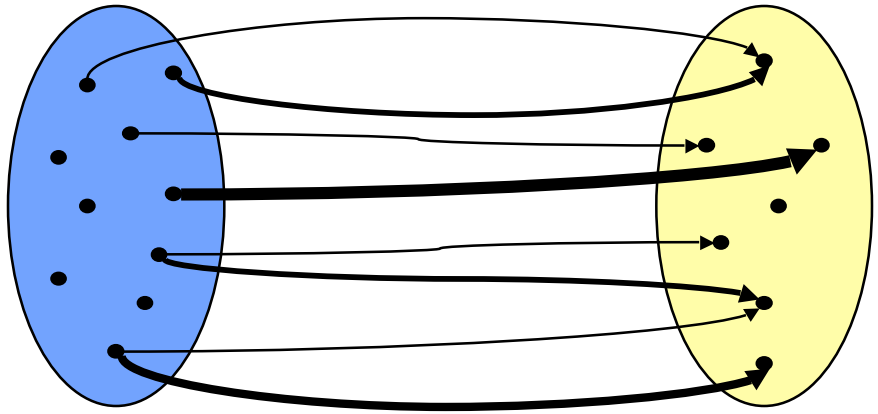
DaVinciAssociators

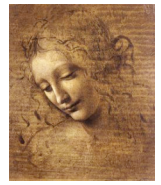
How to relate physics objects to MCParticles



Association between objects

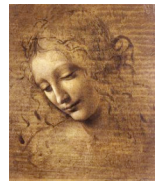
- **Which type of objects**
  - Physics objects: `ProtoParticles` & `Particles` to `MCParticles`
- **Which type of association?**
  - Established usually from the physics object to the MC truth
  - An association can as well contain a weight (e.g. a double)
  - But often useful to retrieve the set of `From` objects associated to a `To` object





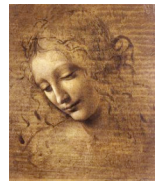
A bit of history

- DaVinciAssociators were initially based on the Relation tables and tools
- Several tools to create and access Relations
  - by Links, Chi2, WithChi2, Composite associator etc…
  - `ProtoParticles` to `MCParticles` associators (charged and neutrals)
- One TES: one Relations table per relations tool
- RelationsMaker algorithms associated to each associator
- Some RelationsMaker algs use other associators
- Associators common properties
  - `InputData`: list of input containers
  - `OutputTable`: TES path for the Relations table
- Caveat: auxiliary associators have to be dedicated, no reuse possible, as Input containers are unknown for a given table



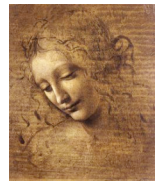
The Linkers

- See the Event Model session (Thursday afternoon)
- Creates and retrieves links between objects in KeyedContainers
  - Doesn't use SmartDataPtr, but uses the container id and the object key
- Not based on tools, no automatic generation by an algorithm
- but simple use, through helper classes (strawman's tools!) both for creation and access
  - LinkerByKey
  - LinkedFrom
  - LinkedTo



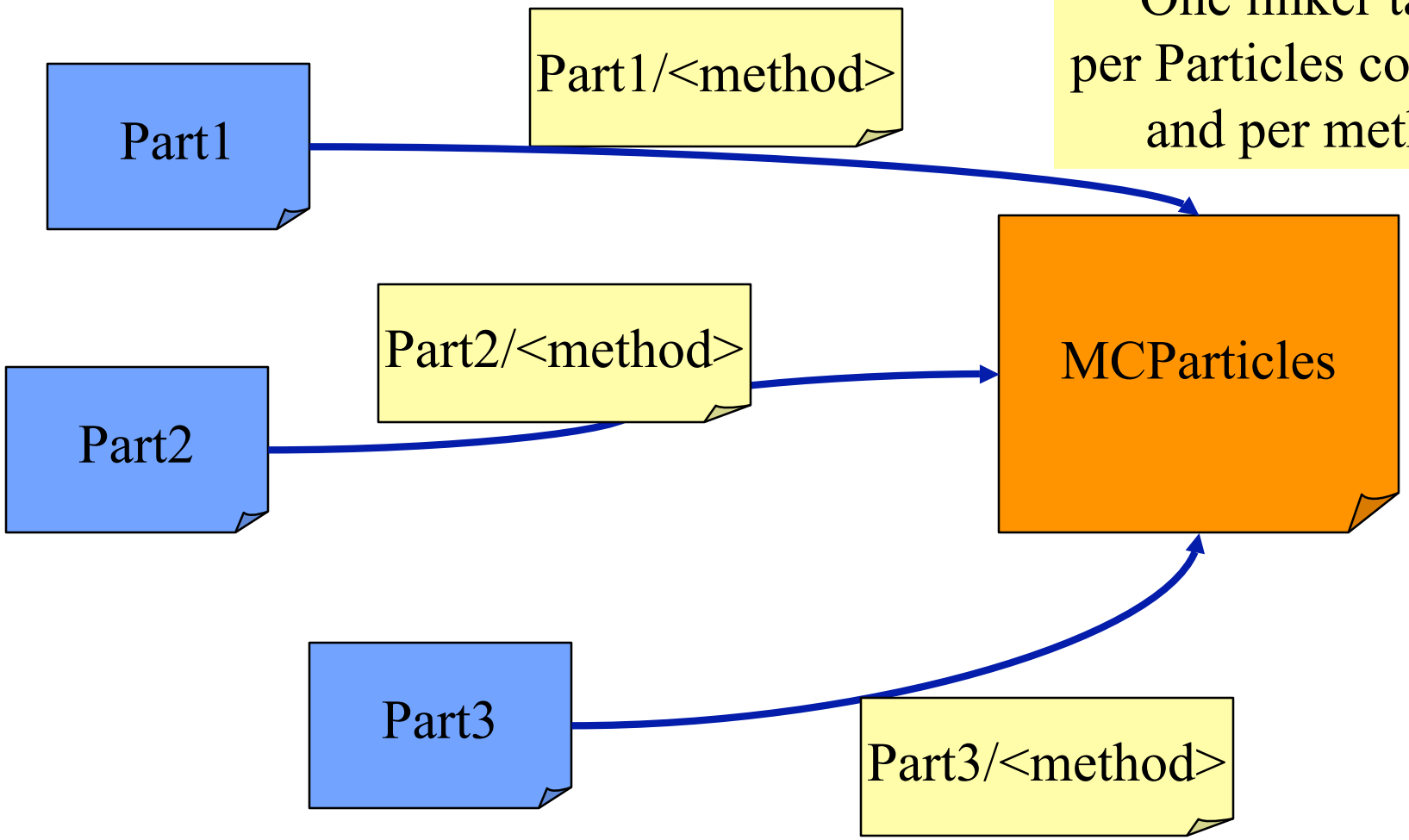
The new DaVinciAssociators/linkers

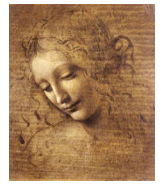
- The old access is still available!
  - See old DaVinci tutorial
  - Tool triggering an algorithm, using Relations table
- In parallel the algorithms create Linker tables
- DVAsct helper classes provide added value to the Linker ones
  - Multiple containers as input (any set of KeyedObjects)
  - Target is always `MCParticle`
  - Automatic invocation of the creation algorithm
  - One Linker table per input container
  - Automatic selection of the correct table from the container of the given object



The linker tables for DVAsct

One linker table per Particles container and per method





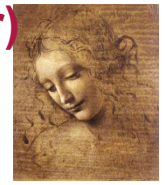
The `Object2MCLink` class

- Created in the `initialize()` method of your algorithm
- Can be used with any type of `KeyedContainer`
- Supports only *direct* relations (from an object to `MCParticles`)
- Constructor
  - `Object2MCLink(`

    `[Gaudi]Algorithm* this,`

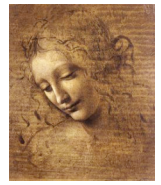
    `int Particle2MCLinkMethod::<method>,`

    `[std::vector<]std::string[>] container[s])`
    - ***`this`*** : used to get the eventSvc, msgSvc
      - If GaudiAlgorithm, instrumented for debug messages
      - ***`<method>`*** : any of: `Links / Chi2 / WithChi2 / Composite`
    - ***`container[s]`*** : list of containers used as input (or single container)



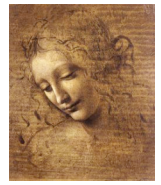
The association methods

- **Links method: `Particle2MCLinkMethod::Links`**
  - Follows links saved between tracks/clusters and MCParticles
  - First creates association from ProtoParticles, then from Particles
  - Weight: fraction of hits in common
- **WithChi2 method : `Particle2MCLinkMethod::WithChi2`**
  - Associates each Particle to the MCParticle that has the minimum chi2 with it (in space and momentum)
  - Weight: value of the chi2
- **Chi2 method : `Particle2MCLinkMethod::Chi2`**
  - Only associates Particles to MCParticles if the chi2 is below a threshold
  - Uses the WithChi2 association and sets a threshold
- **Composite method : `Particle2MCLinkMethod::Composite`**
  - Used for composite particles, checks all daughters are associated



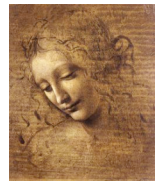
`Object2MCLink` helper class

- Member functions of `Object2MCLink`
  - `MCParticle* first[MCP]( KeyedObject* obj)`
  - `MCParticle* next[MCP] ()`
  - `double weight[MCP]()`
    - Same functionality as for `LinkedTo` helper class
  - `bool isAssociated[MCP] ( KeyedObject* obj)`
    - returns `true` if there is at least one `MCParticle` associated
    - Note: MCP is optional, but strongly suggested in order to avoid confusion



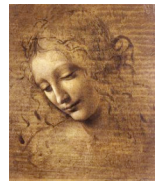
What happens behind the scene

- It gets the container where *obj* is: *<container>*
- It looks for a Linker table called *<container>/<method>*
- If found, it delegates to the `LinkedTo` helper…
- If not, it gets hold of an instance of the maker algorithm corresponding to the declared *<method>*
  - `Particle2MCLinks, Particle2MCChi2, CompositeParticle2MCLinks…`
  - The name of the instance of this algorithm is *<myAlg>.<makerAlgType>*
  - It sets the *InputData* property of the algorithm to the list of containers (and the *OutputTable* property to "", thus not creating the `Relations`)
- It looks if *<container>* is in the list provided in its constructor
  - If not, it adds it, and updates the *InputData* property of the algorithm
- It invokes the *execute()* method of the maker algorithm
  - Only non existing Linker tables are built
- It gets hold of the Linker table and delegates to a `LinkedTo` helper class



The `Particle2MCLink` class

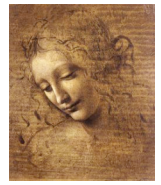
- In fact, it is a `typedef` for an *`Object2FromMCLink<Particle>`*
  - Hence other helper objects can be built using the same templated class (`ProtoParticle2MCLink` exists also)
- Inherits from `Object2MCLink` for direct links
  - Same constructors
- Only adds the possibility to access reverse links
  - Hence the need to specify the Source type, e.g. `Particle`
- Member functions (*`Particle`* can be any other class of course) - Note the `"P"`
  - `Particle* firstP( MCParticle* mcPart)`
  - `Particle* nextP ()`
  - `double weightP()`
    - Same functionality as for `LinkedFrom` helper class
  - `bool isAssociatedP ( MCParticle* mcPart)`
    - just true if there is at least one `Particle` associated



What happens behind the scene

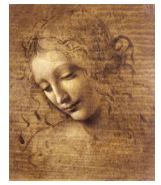
- One checks if a Linker table exists from the Particle containers specified in the list
  - If not, the previous procedure is applied to create them
- A vector of `LinkedFrom` helper classes is built
- One looks for an associated *Particle* successively in all Linker tables…

- Beware
  - Unlike for the direct relations, there is no way to guess in which container a Particle associated to a given `MCParticle` is
  - Hence it looks ONLY in the list provided at instantiation time…



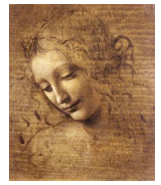
Example of use

```
#include DaVinciAssociators/Particle2MCLink.h
myAlg::initialize()
{
   m_part2MCLink = new Particle2MCLink( this,
        Particle2MCLinkMethod::Links, m_ParticleContainer);
}
myAlg::execute()
{  ……
   Particle* part = …;
   MCParticle* mcPart = m_part2MCLink->firstMCP( part );
   while( NULL != mcPart ) {
        …
        mcPart = m_part2MCLink->next();
   }
   …
   part = …
   if( m_part2MCLink->isAssociated( part ) {……
   }
}
myAlg::finalize()
{  if( NULL != m_part2MCLink ) delete m_part2MCLink; }
```



Advanced features

- **More constructors**
  - `Object2MCLink( this, std::string algName, std::string extension, std::vector<std::string> containers);`
    - Specify a maker algorithm
      - It must have a property *`InputData`* that is set to *`containers`*
      - If it has a property *`Extension`*, it is set to *`<extension>`*
      - It should create linker tables called *`<container>/<extension>`*
  - Same for `Particle2MCLink`



Conclusions

- The DaVinciAssociators have been revisited
  - Fully backward compatible if needed, but…
  - New way to access associations using the Linker tables
- Same predefined types of associations as before
  - Easily configurable when creating the helper class
- Able to discover the containers to associate
  - Only works for direct relations…
  - Once a container is added to the list, it remains there
- Caveat: cannot associate objects not in a container…