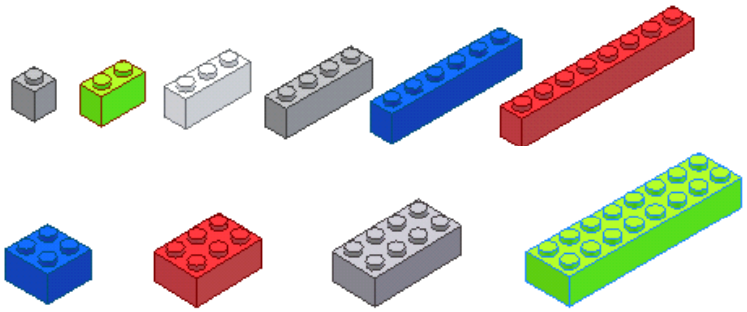
CMS ‘AOD’ Model

Luca Lista

INFN

Outline

- CMS Event Data Model
- Data Tiers in CMS
  - FEVT, RECO, AOD, …
- User data



CMS Event Data Model (EDM)

- Different data access patterns:
  - Bare-root interactive mode:
    - Access object data members only
  - Root + data formats libraries
    - Interactive access full object methods
    - Compiled analysis code based on Root
  - Full Framework access
    - Batch processing with:
      - Access to external resources (DB, Geometry, …)
      - Interfaced to data management system

Persistency technology

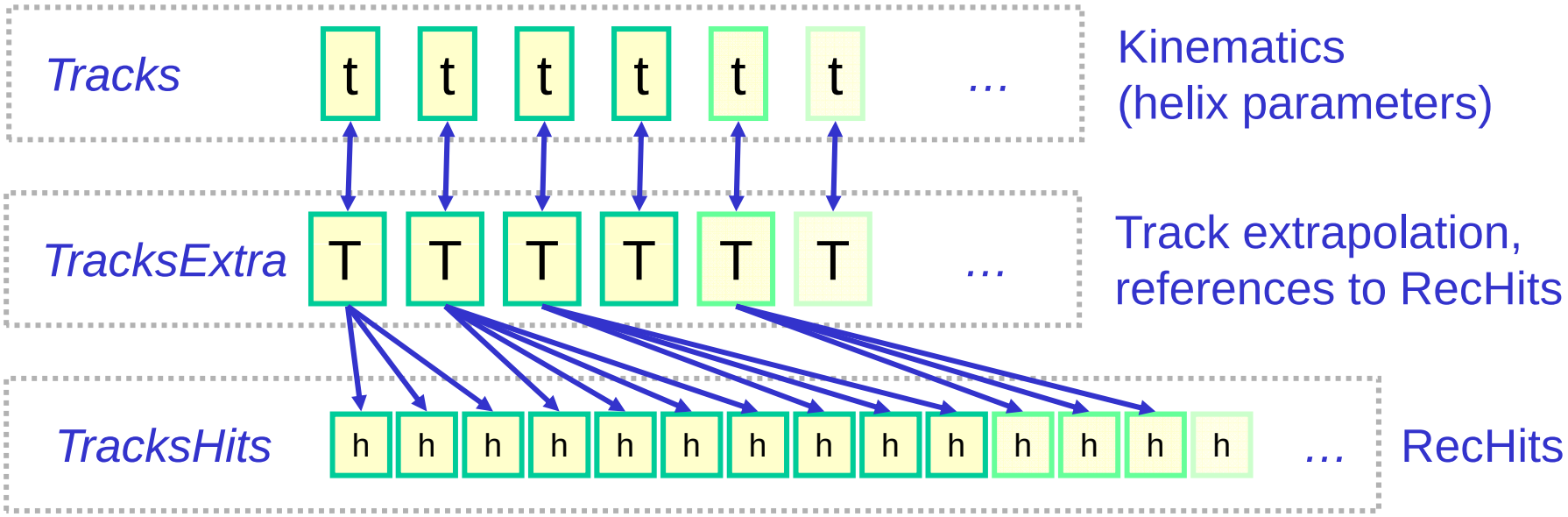
- Events are written using POOL with ROOT as underlying technology
- Persistent class dictionaries are generated with REFLEX
- Automatic loading of data formats shared library supported via SEAL plugin mechanism
- Selective output of event branches is supported to allow configuring different data ‘tiers’
- Adding new data types to the Event is a simple task
  - The Event content is flexible and can be extended for user needs

CMS Framework Data Access

- CMS Framework allows modular event processing via different module types:
  - Analyzer: doesn't change the event content
    - E.g.: histogrammer
  - Producer: creates new collections and store in the event:
    - Each collection is stored in a separate branch in the events tree
    - E.g.: track producer
  - Filter: selects events for further processing. Adding new collections to the event is supported:
    - E.g: HLT filters
  - Event Source: provides events for subsequent processing
    - E.g.: Pool event source: read a POOL/ROOT file
    - E.g.: Event generator module
  - Output module: saves events to output file
    - can be configured to selectively write or skip root branches
    - POOL Output module is used as CMS default

Modular Event Products

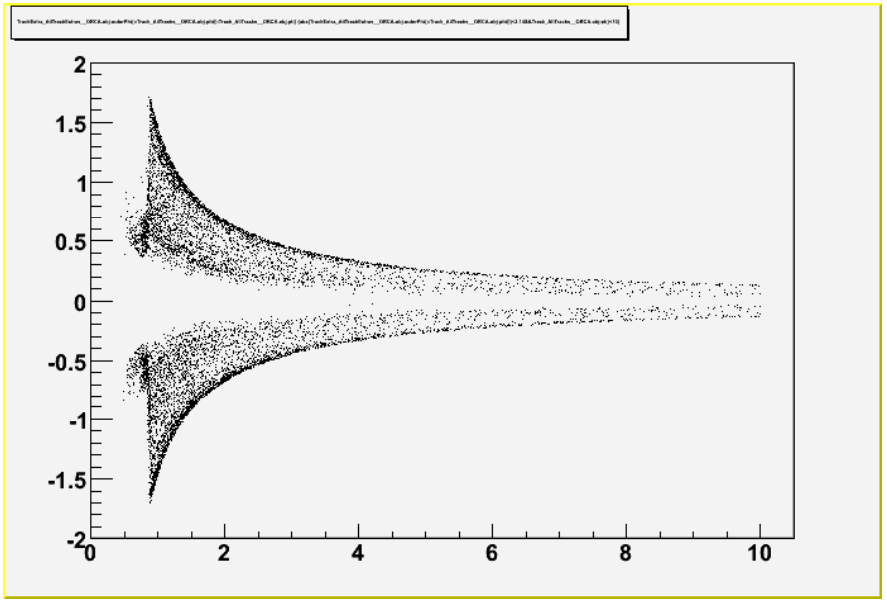
- Different data layers ('tiers') can be configured
- The required lavels of detail can be used for different applications
- Different branches are loaded (on-demand) and can be dropped if not needed



ROOT Interactivity

- Straightforward file access via ROOT

```
> gSystem->Load("libFWCoreFWLite")
> AutoLibraryLoader::enable()
> TFile f("reco.root")
> Events.Draw("tracks.phi() - trackExtras.outerPhi():
    tracks.pt()",
    "tracks.pt()<10")
```



Access without library loading (bare-root) is also possible, but limited to data member inspection

Object Cross References

- Object cross-references are implemented with a CMS specific type: **`edm::Ref<Collection>`**
  - No explicit ROOT dependency
  - Can “autoload” data if not already loaded
  - Contain:
    - A product identifier (unsigned int)
    - An object identifier in a collection (typically, but not necessarily, an unsigned int)
  - ROOT interactive use is facilitated by the index availability
    - also possible in ‘bare’ mode:

```
Events.Draw("electrons.track().get().pt()");
```

`track()` returns an:
`edm::Ref<TrackCollection>`

`get()` performs dereferencing
Can’t use ‘*’ with Draw(“…”)

Generic Containers

- Most object collections are stored as `std::vector<T>`.
- Polymorphic collections are stored as `std::vector<T*>`
  - plus a policy for automatic 'owned' object deletion
- Associative maps are implemented using `edm::Ref<…>` . Interactive access can be done using the indices.
  - Different flavors exist (one-to-one, one-to-many, etc.)

- Direct interactive access, e.g.: MC truth match map :

```
Event.Draw(
  "truth.keys.pt() : truth.values.pt()" );
```

- Also available in bare-root mode using indices in collections:

```
Event.Draw(
 "reco[ truth.map_.first  ].pt_:
  gen [ truth.map_.second ].pt_" );
```

Explicit collection access

Assuming pt_ is a data member…

Data format definition

- FEVT
  - Full event content
- RECO
  - Complete reconstruction output
  - Main client: detector studies, complex analyses

- AOD (Analysis Object Data)
  - Subset of the RECO information
  - Main clients: a large fraction of analyses

- The actual of the different tier is just a conventional choice

User-Defined Data

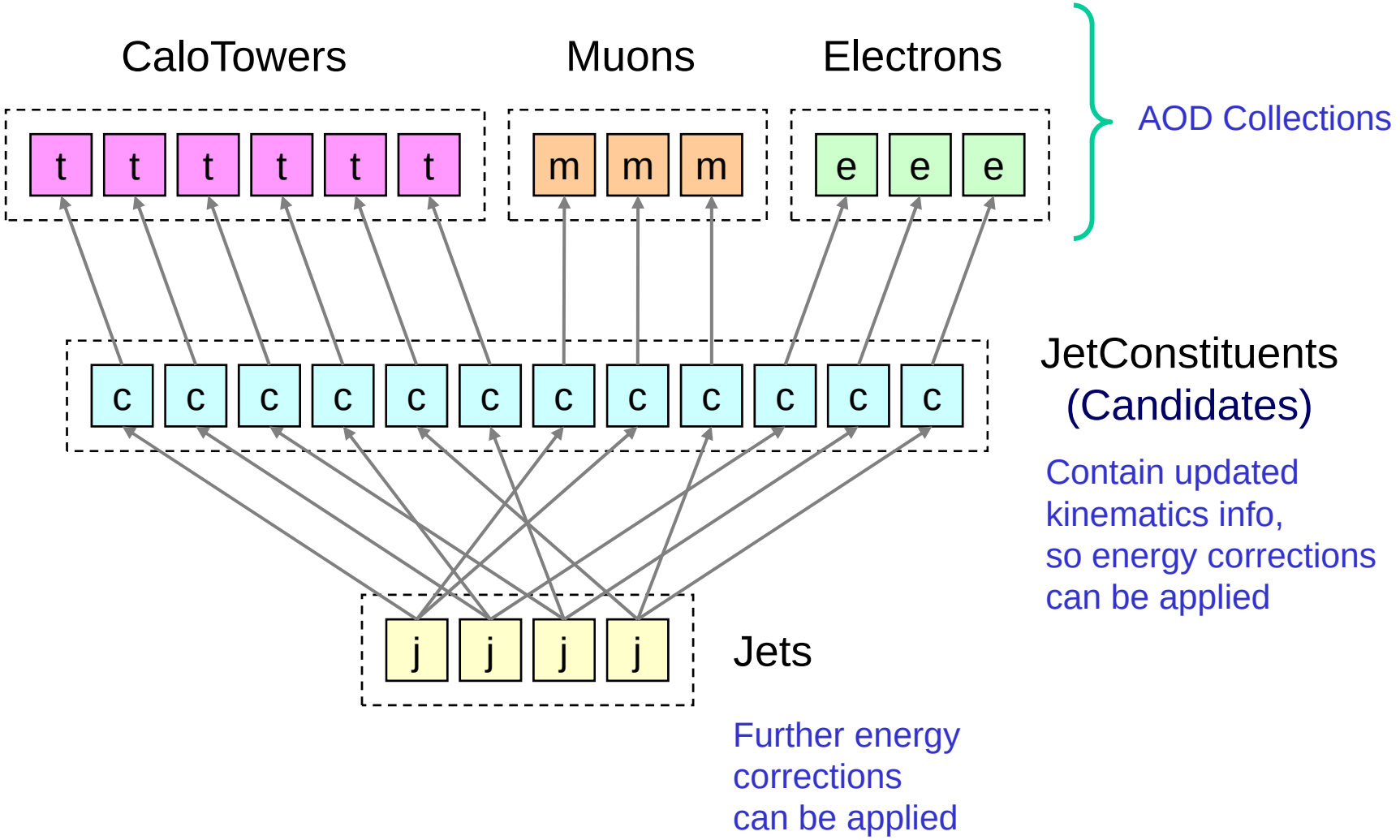
- At some stage users (i.e.: Analysis Groups) can add custom quantities to the event
  - Straightforward in CMS EDM
- User Data can be associated to objects in existing collections extending the available info
- Save processing time
- Store information derived from another data tier (RECO)
  - E.g.: energy in a specific cone
- What is "User Data" and "CMS Data" can change during the experiment lifetime

The EDM can replace custom tree dumps!

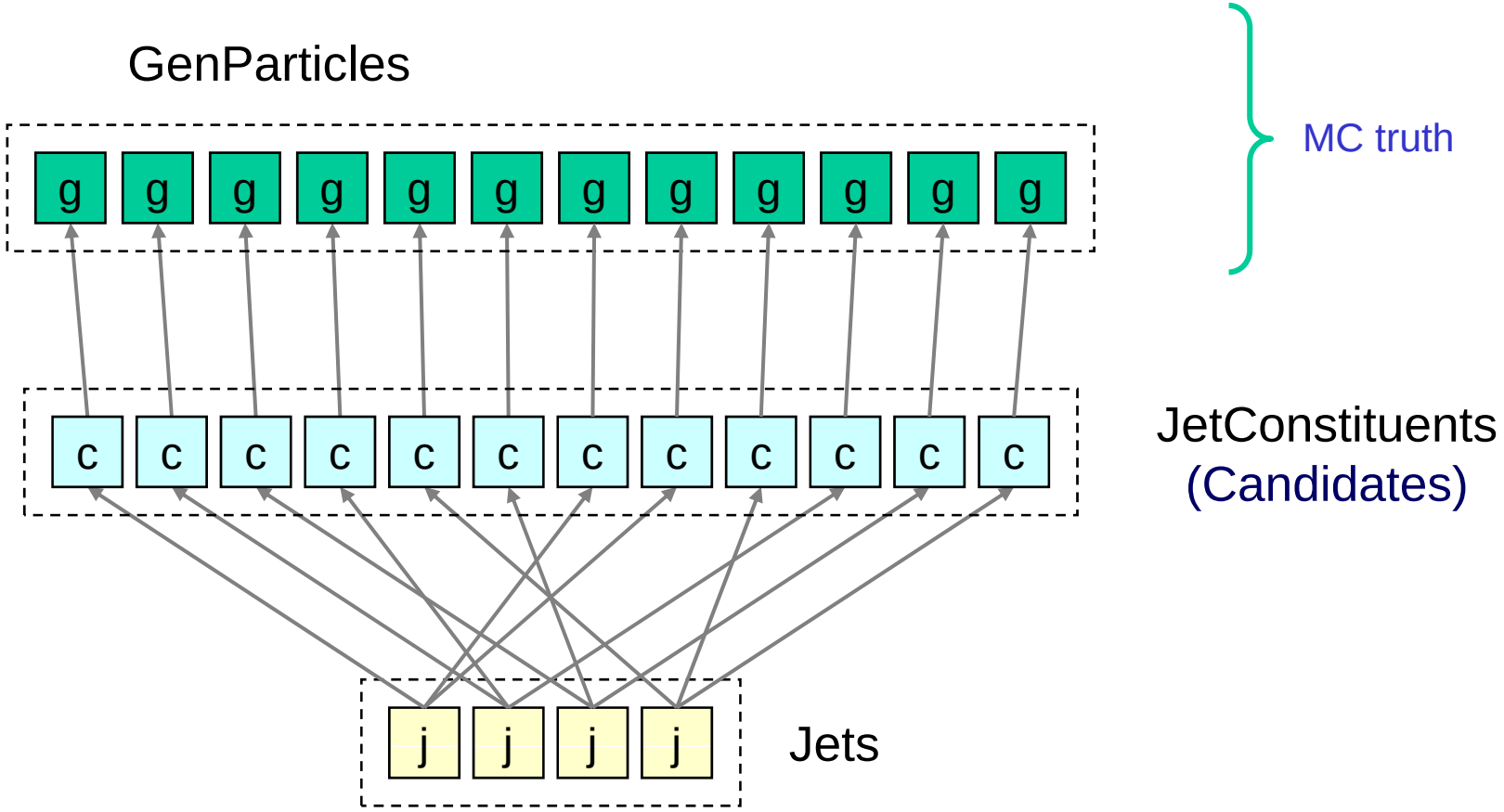
Particle Candidates

- Establish a common “language” for analysis
- Provide a common interface to many Physics tools
  - Constrained fits, Combiners, Jet clustering,…
- Speed up the learning curve for newcomers
  - Learning by examples, web pages, …
- It has been a successful approach in BaBar
  - Beta toolkit

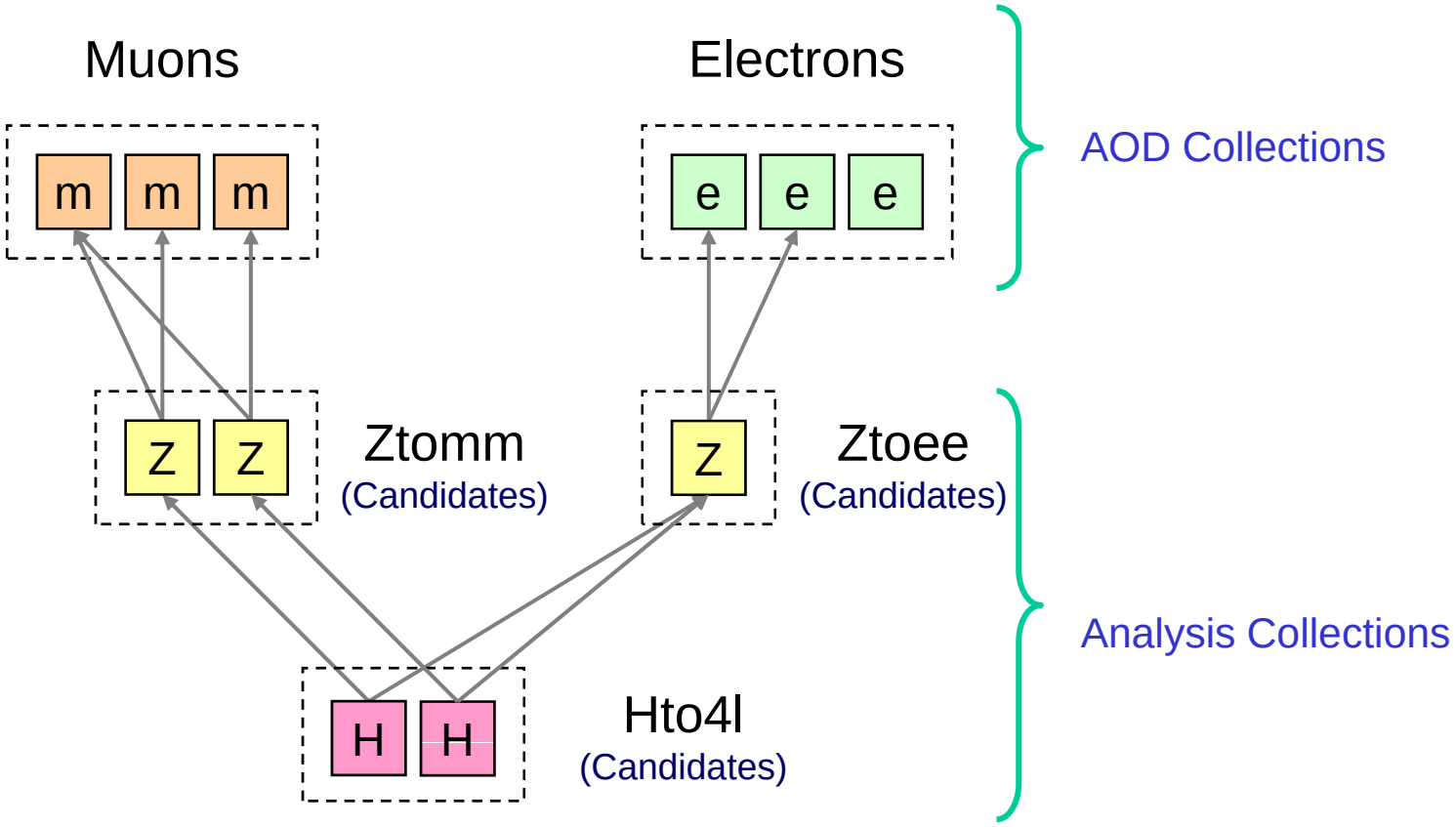
Jet from Heterogeneous Sources



Jets from Generator Particles



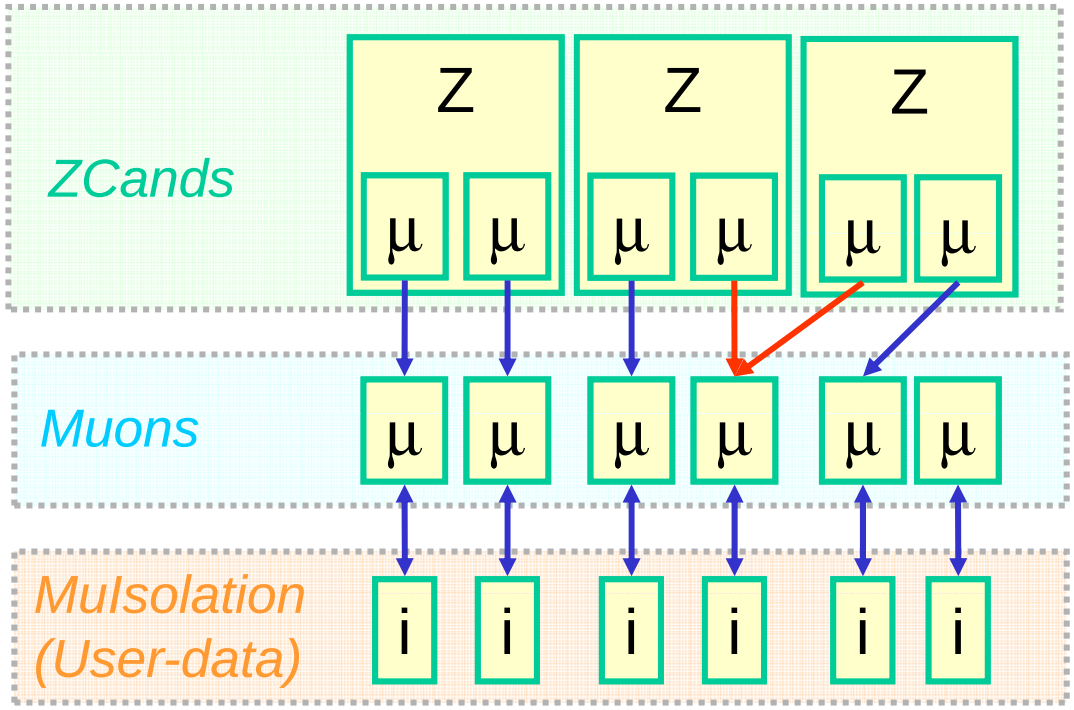
$H \rightarrow e^+e^-\mu^+\mu^-$ with Candidates



Particle Candidates and User Data

- Physics users can drop product and add new quantities to the Event at any processing stage
- The output Event can be analyzed interactively with ROOT
- The Event can be used as "n-tuple" for final analysis

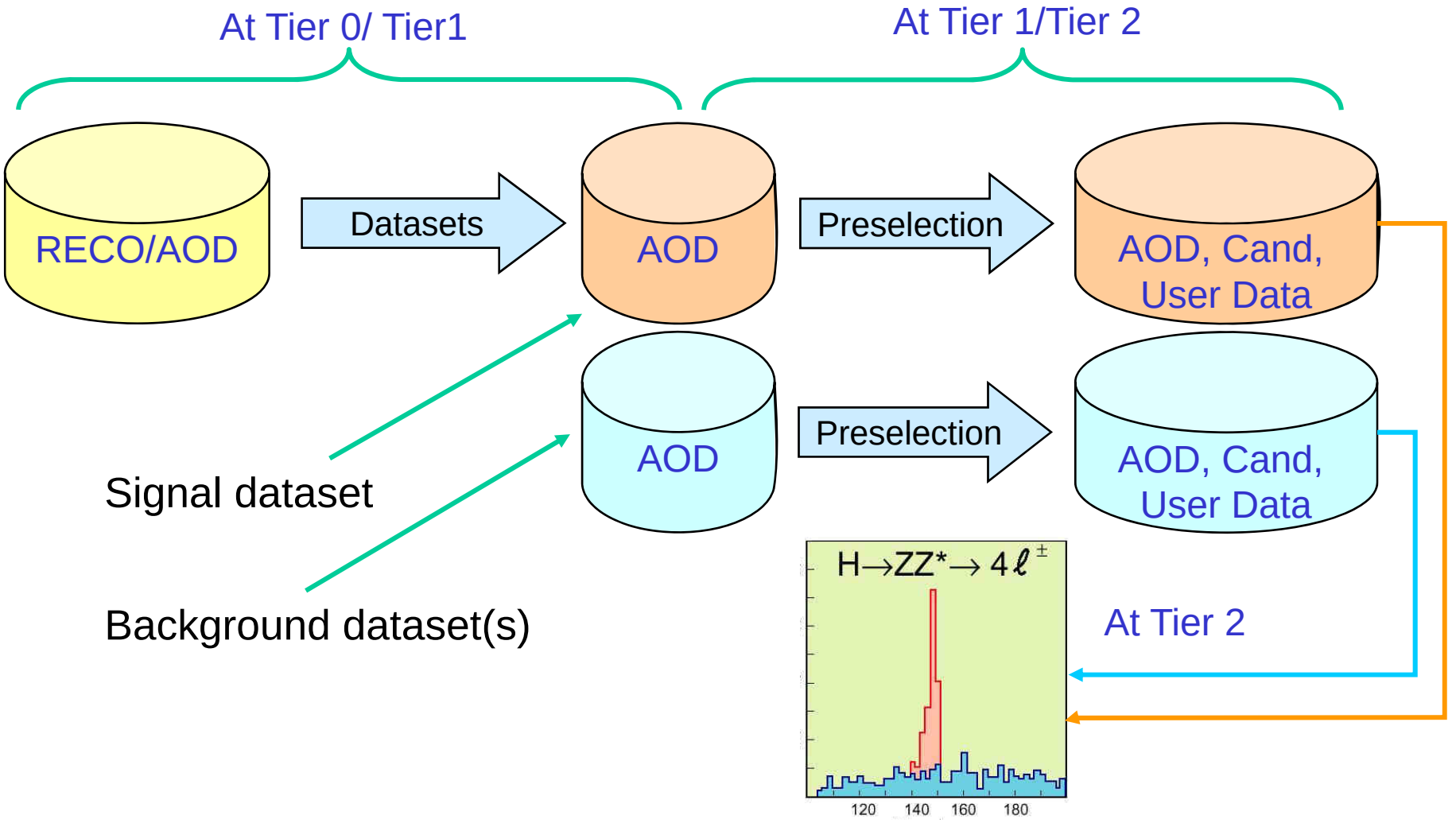
Es. $Z \rightarrow \mu\mu$, with added mu isolation



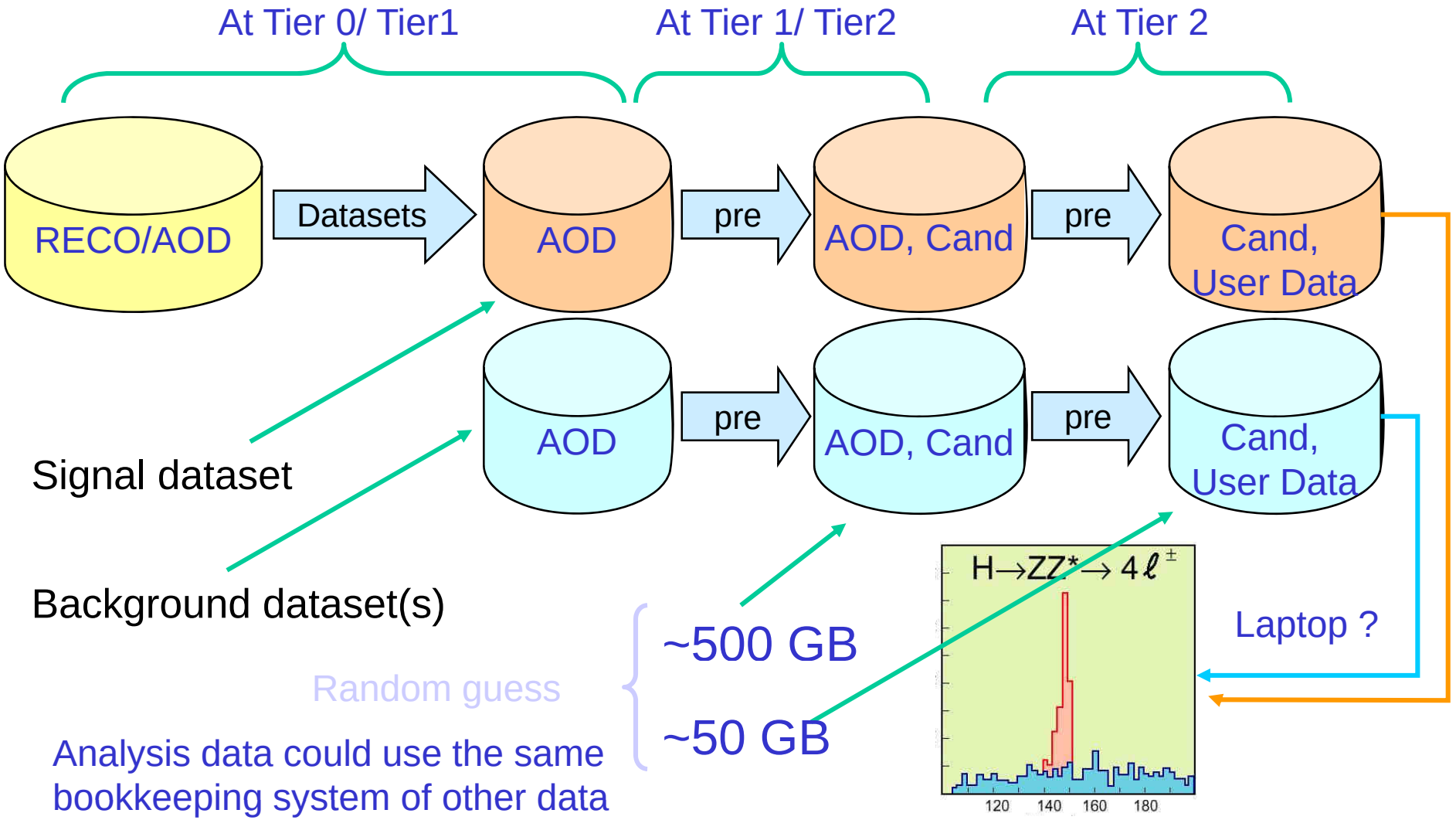
Event Skimming

- Combining event selection with configurable event output allows very flexible event skimming
- Skimming AOD out of RECO or FEVT can be done without running any conversion module
  - AOD $\subset$ RECO $\subset$ FEVT
- Skimming jobs specific for analysis can write a subset of AOD (or RECO) plus quantities specific for that analysis
  - Collection of particle candidates, e.g.: $Z \rightarrow l^+l^-$
  - Sets of variables relevant for specific studies
    - E.g.: lepton isolation, tagging information, …
- The output of the skim is an event tree that can be used as format for final analysis
  - Suitable as well for further batch processing via framework!
- Event skims can also be used as 'event lists'
  - Further re-skimming can be done going back to the 'main' data store

Foreseen Analysis Pattern



Multiple Step Example



Parallel Processing

- We explored TSelector as technique for parallel processing via PROOF
- The main issue is to ensure that the code developed in interactive environment using TSelector is easily portable to batch processing
- Solution: Customized TSelector subclass for CMS
  - User develop an algorithm class using the same API's used for batch processing implementing a simple common interface
  - Both a CMS-TSelector and a Framework Analyzer module can be implemented from the same algorithm class
- Sending shared libraries to PROOF worker nodes is under test at the moment
  - Trying to use the same tools used for GRID analysis job submission (CRAB)

Schema evolution

- CMS release cycle is determined by data backward compatibility issues at the moment
  - Scheduled breaking of data backward compatibility is planned
- CMS schema evolution is only relying on schema changes supported by ROOT at the moment
  - Can't address all possible schema changes we have in reality
- Managing within CMS software generic schema evolution will allow us to have release *N+1* able to read data of release *N*
  - We need to be able to manage conversion from one data version to another with CMS-specific converters

Schema evolution (cont.)

- CMS requirements for non-trivial schema changes:
  - Conversion from class version N to class version N+1 should be managed within CMS code by experts of the subsystems that are not expert of ROOT core code
    - E.g.: no explicit management of streamers, etc.
  - Class names used for analysis should not be affected by version information
    - e.g.: always use 'Track'
  - 'Core' code can manage name changes
    - e.g.. 'Track_001'
  - A solution we would like: configure the dictionary generation to read obsolete class versions as objects with a different class name
    - In release *N+1* read versoin *N* of class 'Track' as 'Track_00N'
    - CMS code provide a conversion function from 'Track_00N' to 'Track', which is used in analysis
- We look forward for next ROOT team ideas on generic schema evolution to address this problem

Conclusions

- CMS Event Data Model is a flexible tool for event storage in any format
  - RECO/AOD, but also analysis-oriented
- Interactive capabilities make the EDM a suitable tool for both batch and interactive analysis via ROOT
- 2007 “challenge” will exercise the EDM/AOD capabilities in concrete analysis exercises