Physics Analysis Tools for the CMS experiment at LHC

Luca Lista, *INFN Napoli*
Francesco Fabozzi, *INFN Napoli*
Benedikt Hegner, *DESY*
Christopher D. Jones, *Cornell*

Outline

- Data Tiers in CMS EDM
- Analysis Tools
- Analysis Workflow

Main Features of CMS EDM

- CMS Event Data Model (EDM) is the uniform format for all CMS event data
  - An Event is a container of many "products" of any possible (C++) type
    - Most of the products are collections of objects such as tracks, clusters, particles, …
  - The EDM allows no "C" pointers allowed, and provides custom persistent references
    - Product ID and indices in a collection identify referred objects

- Persistent and transient data representations are identical (based on ROOT I/O)
- All EDM data are accessible with ROOT interactively
  - See → Chris Jones' talk, *Event processing* session
- Reflex dictionaries must be provided for all products

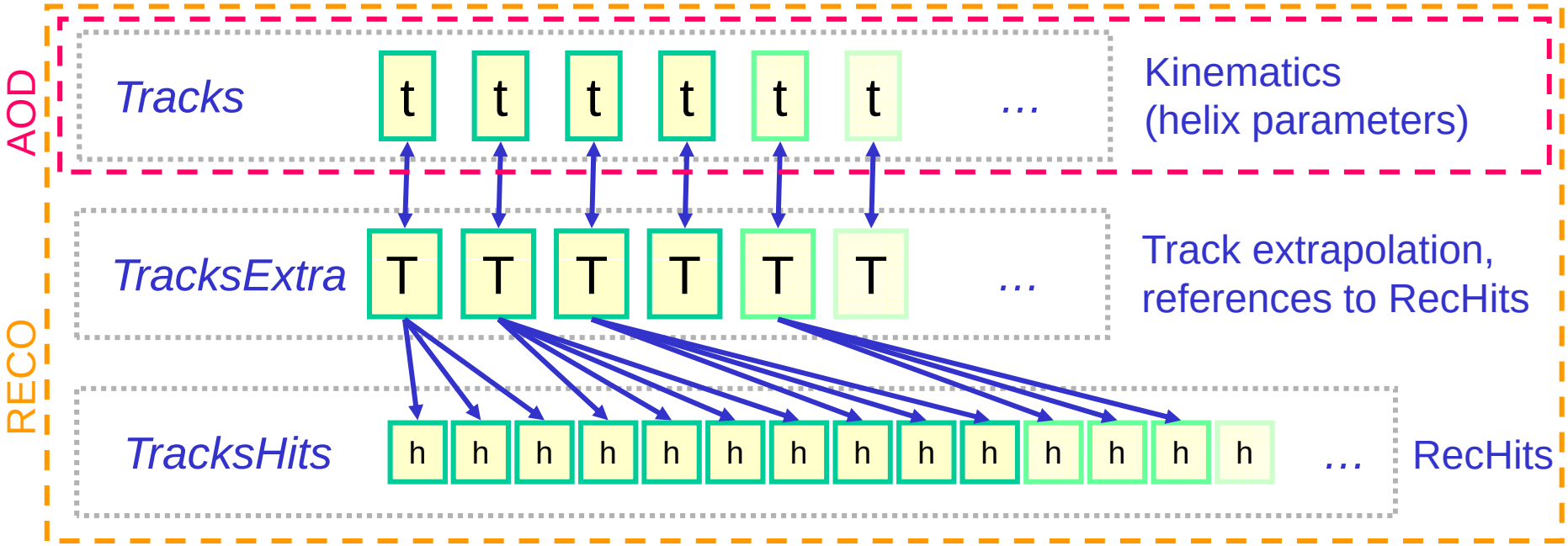
Data Tiers and Analysis Object Data

- CMS defines different data tiers containing different levels of details of an event
  - FEVT: full event output, containing (almost…) the complete output of all intermediate reconstruction steps
  - RECO: detailed reconstruction output allowing to apply new calibrations and alignments, and reprocess many of the products
  - AOD: a proper subset of RECO chosen to satisfy the needs of a large fraction of analysis studies

- Adding or dropping object collections to/from AOD/RECO/FEVT is just a matter of changing a job's configuration
  - The actual AOD content (and disk size…) is till under definition, it will likely evolve also with data taking

Modular Event Products

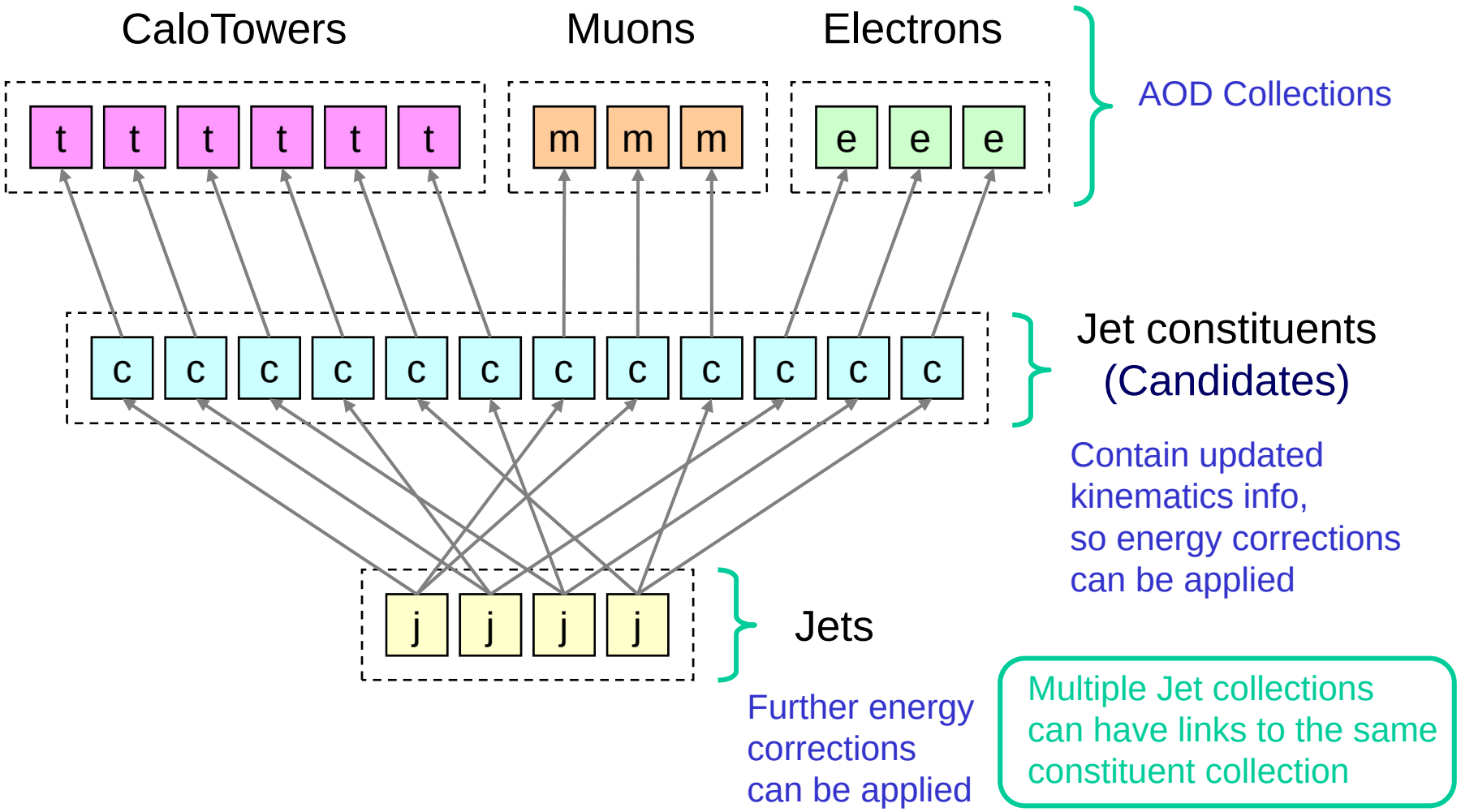
- Object collections can be split into different products
- This allows us to define different levels of details avoiding to store redundant information



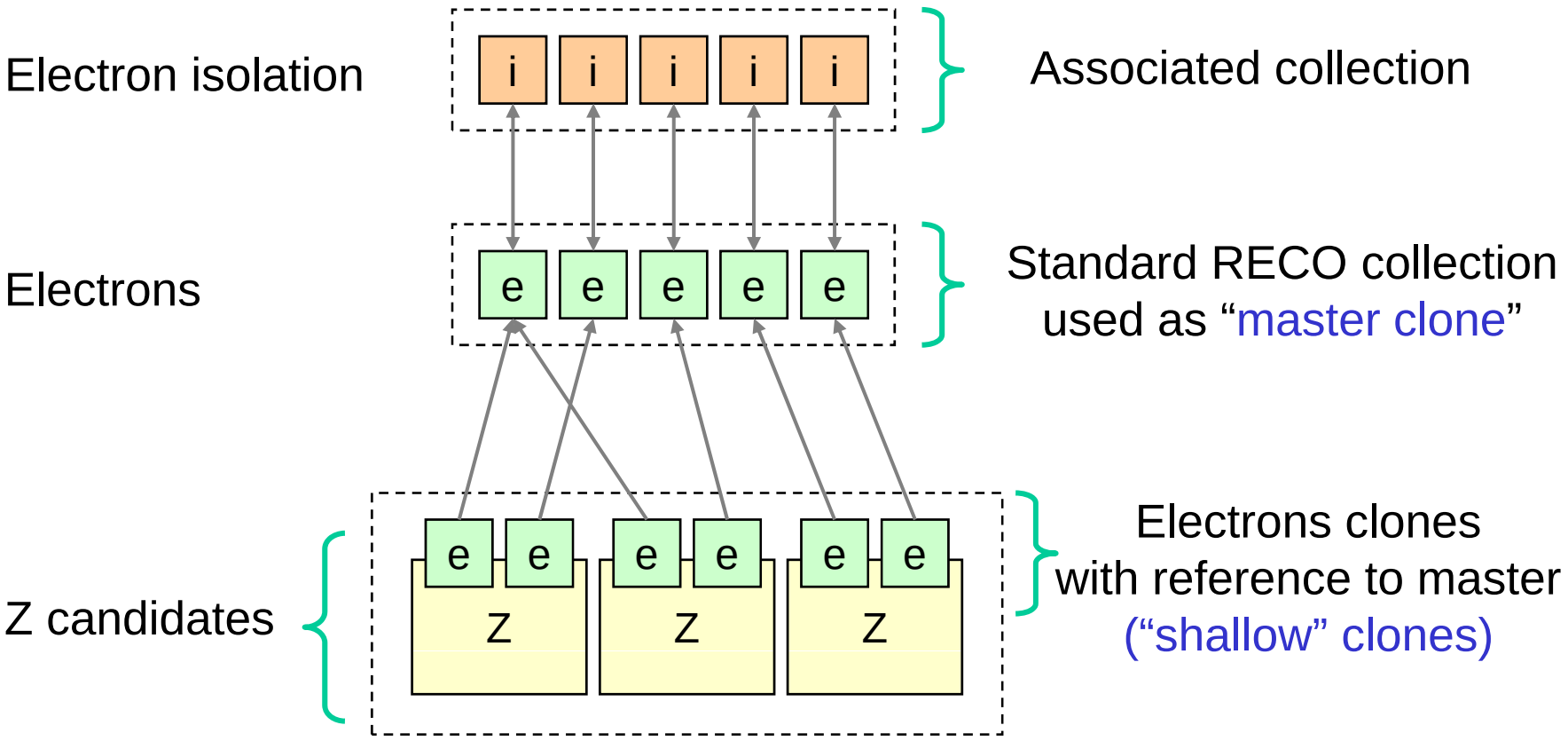
Particle Candidates

- **`Candidate`** is a common base class for all high-level physics objects
  - Muons, electrons, photons, jets, missing E_T, … inherit from *Candidate*
  - Can contain references to AOD components, like tracks, clusters, calorimeter towers, …
  - Supports mother(s)↔daughter(s) navigation in specialized sub-classes
- Composite particle reconstruction from multi-body decay chains uses specialized *Candidates*
  - E.g.: $Z \to \mu\mu$, $H \to ZZ \to \mu\mu ee$, $B_s \to J/\psi\phi \to \mu\mu KK$, …
- Event generator tree in AOD is stored using *Candidates* with mother/daughter references

Jet from Heterogeneous Sources



Candidates and Associated Data



Framework modules

- Reconstruction and analysis code is organized as independent modules steered by the framework
- A job configuration script defines the modules to be loaded (as plugins), their parameters and their execution order
  - Modules execution sequences are organized into "paths"
- Each module can get data from the Event and can add new products to the Event
- Product provenance tracking including module parameters is saved as part of the Event output file
- Once a product is added to the Event it can't be changed by another module
- Modules can act as event filters, stopping the processing *path* if a condition is not fulfilled
  - E.g.: High Level Trigger paths

Available Common Tools

- Layered approach to common tools:
  - AOD (and RECO…): basic "primitive" objects for analysis
    - Tracks, super-clusters, calo-towers, μ, e, γ, jets, ME_T
    - Mainly data container, no "fancy" C++ structures
  - Generic common tools (for AOD and more)
    - Selectors, filters, lepton isolation, matching tools
  - Particle Candidates
    - Generic class hierarchy to manage particles for analysis
    - Base class for high level objects: μ, e, γ, jets, Met, gen-particles, composite decays (Z, J/ψ, B_s, Higgs, …)
  - Particle Candidates common tools
    - Combiners, selectors, filters, overlap removal
    - MC truth matching tools
    - Generic isolation algorithms
    - Constrained fitters (initial integration examples)

Event collections

Algorithms and modules

Generic AOD Framework Modules

- Uniform interface is enforced throughout AOD classes
  - Everywhere `pt()`, `eta()`, `phi()`, etc.
- Generic programming is used to write algorithms applicable to different object types
- A suite of generic selector and filter modules is provided as part of the common Physics Tools
- More high level algorithms are being written using generic programming
  - Isolation algorithms can run on muons, electrons, tracks, …

Generic Object Selectors

- A selection criteria can generate specialized selectors performing specific actions:
  - Save clones of the selected objects
  - Save references to the selected objects (i.e.: “indices”)
  - Clone the selected objects and all the underlying constituents
    - e.g.: clone selected electrons with clones of tracks and clusters
- Internal implementation specializations use template traits on the basis of the input and output collection types

- The simplest object selections can be written as a simple function object (returning a Boolean result)
- A string-configurable selector functor is provided to parse a configurable string-based cut:

```
string cut =
  "(pt>10 & abs(eta)<2.5) & normalizedChi2<10"
```

  - Variable names are mapped to objects methods via Reflex dictionary

Generic Selector Examples

```
struct PtMinSelector {
  PtMinSelector(double ptMin) : ptMin_(ptMin) { }
  template<typename T>
  bool operator()(const T& t) const { return t.pt()>=ptMin; }
  private:
    double ptMin_;
};
```

```
typedef SingleObjectSelector<
  reco::TrackCollection,
  StringCutObjectSelector<reco::Track> >
  TrackSelector;
```

```
typedef SingleObjectSelector<
  reco::MuonCollection,
  PtMinSelector>
  PtMinMuonSelector;
```

```
typedef SingleObjectSelector<
  reco::TrackCollection,
  StringCutObjectSelector<reco::Track>,
  reco::TrackRefVector>
  TrackRefSelector;
```

Selector configuration

```
module highPtMuons = PtMinMuonSelector {
  InputTag src = allMuons
  double ptMin = 10
}

module bestTracks = TrackSelector {
  InputTag src = allTracks
  string cut = "pt > 10 & normalizedChi2 < 20"
}

module bestTrackReferences = TrackRefSelector {
  InputTag src = allTracks
  string cut = "pt > 10 & normalizedChi2 < 20"
}
```

Common Physics Tools

- Combinatorial analysis
- Overlap checking
- Monte Carlo matching tools
  - Implement navigation to parent to find matching to a composite particle
- Constrained fitter
  - Examples of integration with external fitting packages exist
  - Covariance matrices (5x5) are fetched from AOD object for vertex fits using tracks
  - Specialized candidate containing error matrices are being developed for the cases where errors are not stored in AOD objects
    - E.g.: jet or photon mass-constrained fits require Ecal and Hcal energy resolutions, retrieved from specialized framework services

Example of Combinatorial Search

```
module JPsiCandidates = CandCombiner {
  string decay = "muonCandidates@+ muonCandidates@-"
  string cut = "2.8 < mass < 3.4"
}

module PhiCandidates = CandCombiner {
  string decay = "trackCandidates@+ trackCandidates@-"
  string cut = "0.9 < mass < 1.1"
}

module BsCandidates = CandCombiner {
  string decay = "JPsiCandidates PhiCandidates"
  string cut = "5.3 < mass < 5.6"
}
```

Analysis Custom Data Types

- Analysis Groups can easily define new data types to be added to the Event for analysis
  - The output of a Analysis jobs is fully configurable
  - Needs not always be standard RECO or AOD
- Analysis "skim" productions run centrally
  - Event pre-selection is performed in central skims
  - New analysis collection can be added to standard AOD (or any other data format) for the events selected by each particular analysis skim
    - Analysis collections can contain either standard or any user-defined type
    - Particle *Candidate* collections can be added to the Event as analysis output

CMS Analysis Work-Flow

First pass at Tier0/CAF

RAW → RECO → AOD

RECO, AOD shipped at Tier1

Central analysis skims at Tier1

AOD → Analysis algos → AOD + Analysis Data

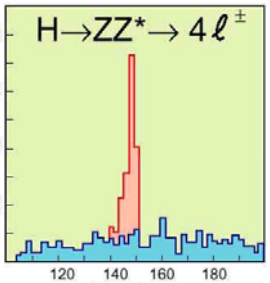
Analysis skim output shipped at Tier2

Final analysis pre-selection at Tier2

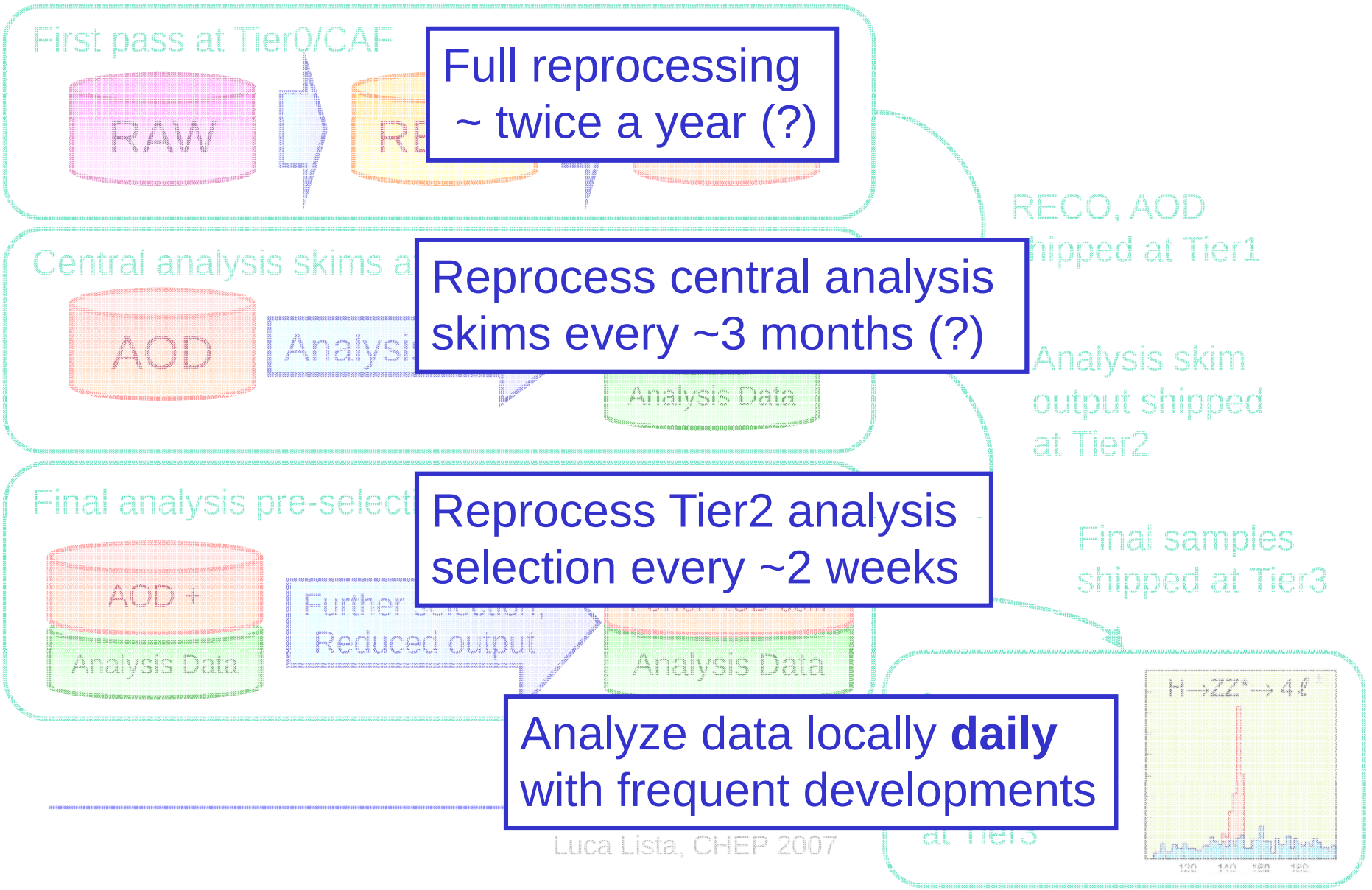
AOD + Analysis Data → Further selection, Reduced output → Fewer AOD coll. Analysis Data

Final samples shipped at Tier3

fast processing and FWLite at Tier3



CMS Analysis Work-Flow



Conclusions

- A flexible event content and a variety of common tools help implement the most commonly required tasks needed for CMS analysis.
- The organization of data formats and tools is designed to be integrated with CMS analysis workflow running on distributed computing as well as for the final stage of analysis.
- A realistic exercise of analysis skims using custom data formats containing analysis collections reconstructed with common analysis modules is being put in production
  - Will run in summer and autumn this year.

Backup slides

Polymorphism and “Views”

- Modules can retrieve event products in a type safe way specifying the collection type:
  - `Handle<MuonCollection> muons,`
  - `event.getByLabel(“muons”, muons);`

Product tag, typically part of the configuration

- Modules can also specify the base class of contained (or referred to) objects via collection “View”:
  - `Handle<View<Candidate> > leptons;`
  - `event.getByLabel(tag, leptons);`
- Both collections of objects and collections of references are supported

Generic Selectors Development

- The selection criteria definition is decoupled from the technical implementation details of selector module specializations
  - Specific selections are written for alignment and calibration samples by people with no necessary experience with “core” software
  - No explicit definition of cut configuration, reference and clone management is needed in most of the cases

- The most commonly used framework module are provided as part of the release, need not be explicitly instantiated by users
- If new modules are needed, most of the users request them centrally rather then instantiating them privately
  - The reuse of common module occurs very naturally

Utility Classes *vs* Modules

- Many common utilities are provided as framework modules
  - Plugging modules into sequences is easy to do, and module reuse is very simple
  - EDM Provenance mechanism is useful to tack the analysis process
- A number of tools are also provided as utility class that can be included in “private” modules
  - Framework overhead is reduced