Typical features of analysis workflows in CMS

Marco Peruzzi, Giovanni Petrucciani (CERN)

HSF analysis working group
Analysis requirements jamboree
CERN, January 23rd, 2019

Introduction

- Physics analysis workflows in CMS are quite **diverse** in terms of:
  - input datasets,
  - data processing and reduction techniques,
  - analysis procedures,
  - final number of selected events,
  - physical interpretation of results.
- We try to present here our personal view on some typical features common to a substantial fraction of them

- The **input datasets** typically consist of the **MINIAOD** data tier (~30 kB per event):
  - includes all reconstructed particle candidates, with sufficient information to recalibrate them and recluster jets
  - centrally produced, typically a few times per year
  - information is stored in classes that are read via ROOT dictionaries
- Input files are distributed across several geographical sites

Tree production

- Production of analysis trees, running on central datasets:
  - **implement object "recipes"** that change often, or become available only after the central processing has taken place
  - using the **latest version of calibrations** (e.g. jet energy corrections)

- This is done using a **privately-maintained framework** based on CMSSW:
  - either directly in CMSSW analyzer modules written in C++, or
  - in a python looper through the FWLite classes
    (allow reading the content of MINIAOD files, with some limitations)

- Computing resources:
  - GRID queues, often running at sites hosting the input data
  - direct submission on large batches (T2/T3 sites, LXBATCH),
    with either local or remote access to data (via XROOTD)

Tree production

- General feature: find the best compromise between **skimming the set of events** and **slimming the event content**. This is very analysis dependent!
- Typical size is **few kB per event, O (10 Hz)** processing speed

- Typical turnaround time is **~2 weeks**, the most efficient groups do it in about 1 week
- Having a quick turnaround time is a strong competitive factor under deadline pressure
- Analyzers often tend to store much more than what they really need:
  - to avoid relying on prompt availability and performance of remote resources
  - to handle requests of additional studies and cross-checks that can come extremely late in the internal analysis review process

- As maintaining a framework is very expensive in terms of workload, "**macro-frameworks**" are being maintained by several groups in a collaborative way
- Used for several analyses, also across quite different realms (Higgs, searches, …)
- **Recently commissioned a new central flat tree format: NANOAOD**

Event content

- The typical structure of an analysis ntuple is a ~flat tree containing **event singleton variables, objects, and collections of objects of varying size**
- Data types are heterogeneous, as they refer to physics quantities with different meaning: the data format should be able to store different types efficiently

	Singleton	Splitted object		Splitted collection of objects	
ev. #1	$nVtx_1$	MET_Px_1	MET_Py_1	$[Muon\_Px^1_1, Muon\_Px^2_1, \ldots]$	$[Muon\_Py^1_1, Muon\_Py^2_1, \ldots]$
ev. #2	$nVtx_2$	MET_Px_2	MET_Py_2	$[Muon\_Px^1_2, Muon\_Px^2_2, \ldots]$	$[Muon\_Py^1_2, Muon\_Py^2_2, \ldots]$
ev. #3	$nVtx_3$	MET_Px_3	MET_Py_3	$[Muon\_Px^1_3, Muon\_Px^2_3, \ldots]$	$[Muon\_Py^1_3, Muon\_Py^2_3, \ldots]$

Event and object weights

- We associate **various weights to each event**
  - either based on object quantities and calculated at analysis time, or inherent to the event since production (e.g. generator weights)

	Muon_Px	Muon_Py	Muon_SF	Event_SF
ev. #N	[Muon_Px^1_N, Muon_Px^2_N, …]	[Muon_Py^1_N, Muon_Py^2_N, …]	[Muon_SF^1_N, Muon_SF^2_N, …]	Π_I Muon_SF^I_N

- Keep **flexibility of using a simulated event more than once, with different weights**:
  - important to track detector evolution across data taking periods, via **time-dependent data/MC corrections**, or for techniques like "**fake rate**"
  - requires great care in downstream statistical treatment; an alternative is pseudo-random association of each MC event to a defined era or category

Event selection

- After producing the analysis trees from MINIAOD files, flexibility is much enhanced
- A **series of object and event selection steps** takes place, often leading to the production of **various slimmed versions of the analysis trees** (“mini-trees”)

- This is typically done via **event loops** implemented in ROOT macros or scripts
- Their level of efficiency and flexibility varies greatly across different analysis groups
- The code becomes **very analysis dependent** at this stage; some efforts to modularize and share routines for typical tasks (e.g. application of object scale factors)

- Directly accessible resources (like batches) are used, or even just done interactively
- Analyzers often **trade efficiency for flexibility and ease-of-use**:
  - duplication of information (sometimes use **friend trees** to extend event content)
  - limiting #steps means running expensive routines on more events than necessary (a framework with **lazy evaluation** of expensive steps would improve this)

Further processing

- General: these steps include operations that are performed at **very different speeds**:
  - adding identification bits to objects is usually very **cheap**
  - calculating matrix elements, or testing all combinations of objects in each event (e.g. jet triples to reconstruct resolved top decays) can be **very expensive**, from seconds to minutes per event
- **External software packages** often have to be invoked (e.g. multivariate inference)

- The selection framework has to be flexible enough to perform **optimization studies**: define a figure of merit, change the selection requirements to obtain the best result
- **Lack of automation**: optimizations are sometimes done quasi-interactively to satisfy constraints that are difficult to code (avoid empty sidebands, focus on “interesting” regions for a specific class of signal, “do not make too many signal regions” etc.)
- A **proxy** for the real figure of merit can be used (e.g. S/B without uncertainties), to avoid re-running the analysis until the end multiple times: not really optimal

Abstract data operations

	Singleton	Splitted object		Splitted collection of objects	
ev. #1	$nVtx_1$	MET_Px_1	MET_Py_1	$[Muon\_Px^1{}_1, Muon\_Px^2{}_1, \ldots]$	$[Muon\_Py^1{}_1, Muon\_Py^2{}_1, \ldots]$
ev. #2	$nVtx_2$	MET_Px_2	MET_Py_2	$[Muon\_Px^1{}_2, Muon\_Px^2{}_2, \ldots]$	$[Muon\_Py^1{}_2, Muon\_Py^2{}_2, \ldots]$

- **Count** number of objects passing a certain selection, e.g. F ($Muon\_Px^I{}_N$, $Muon\_Py^I{}_N$) and retain only those who pass this requirement
- **Create new object variables** from existing ones,
  e.g. $Muon\_Pt^I{}_N = F\ (Muon\_Px^I{}_N\ ,\ Muon\_Py^I{}_N)$
- **Create new event variables** from quantities of different objects,
  e.g $HT_N = Sum_I\ \{\ Jet\_Pt^I{}_N\ \}$
- **Cross-cleaning collections** i.e. removing overlap according to some metric:
  e.g. evaluate F ($Muon\_Px^I{}_N$, $Muon\_Py^I{}_N$, $Electron\_Px^J{}_N$, $Electron\_Py^J{}_N$)
  to select whether to retain and consider $Muon^I$ or $Electron_J$

Across event boundaries

- In some cases, we need to **use more than one event at a time**
- For instance, build a global distribution of a certain variable and use it **to build event weights** (e.g. pileup reweighting)

	nVtx	PU_weight	Embed_Muon_Px
ev. #N	$nVtx_N$	F ($nVtx_1$, $nVtx_2$, $nVtx_3$, …)	[$Muon\_Px^1{}_M$, $Muon\_Px^2{}_M$, …]

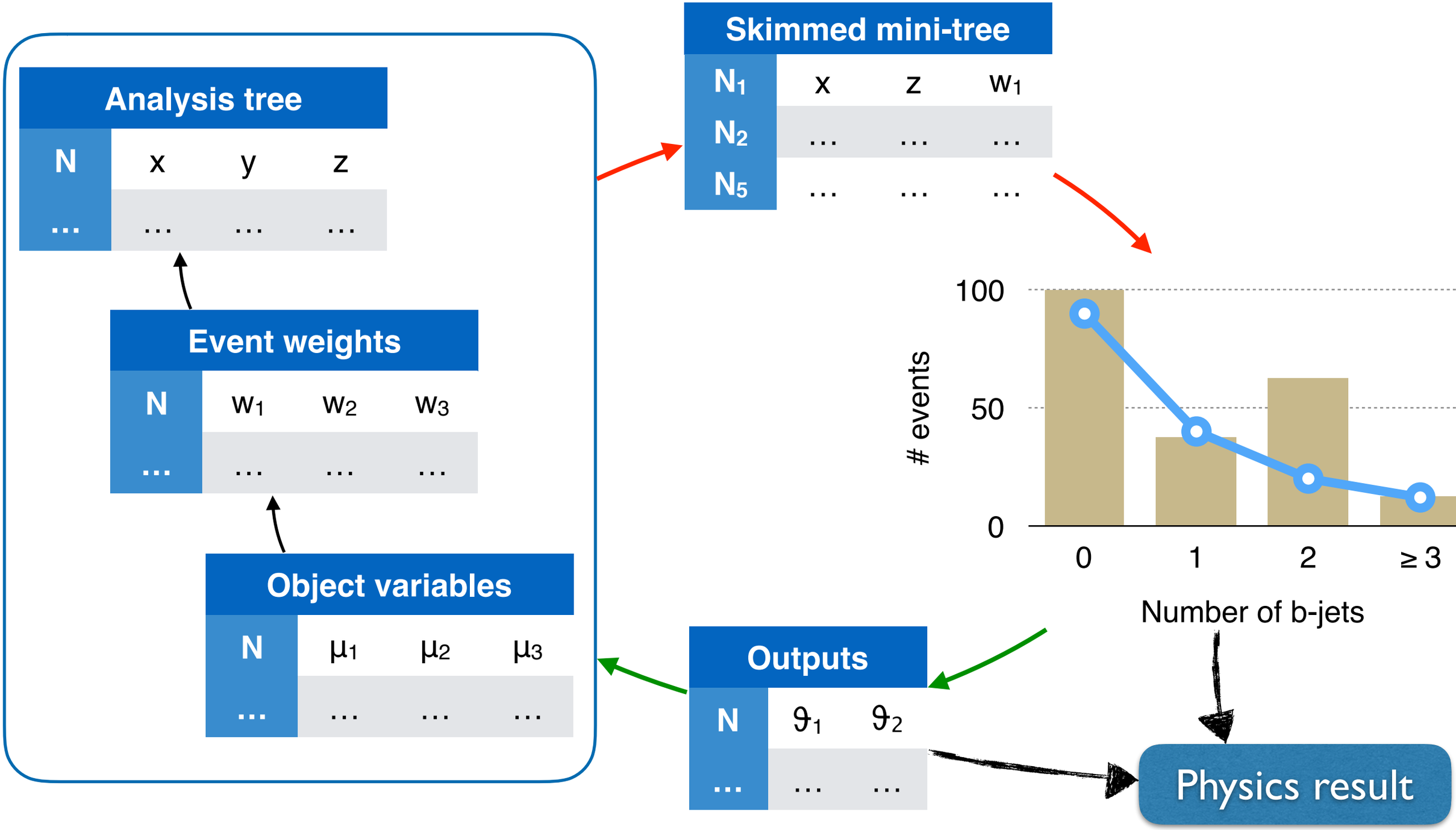
- Event **embedding techniques**, where objects taken from one event are overlaid with other objects from a different event
- **Training discriminator algorithms** on a set of events is also an example
- And, of course, we aggregate events when producing categories and histograms for the final interpretation leading to physics conclusions

Final event aggregation

- The last step of data processing is **event aggregation in histograms or categories**
- Some kind of **interpretation** takes place at this stage:
  - running **fit routines** to extract parameters from histograms
  - exporting the data to **advanced statistical analysis** tools (datacards)

- The code for this part is often kept separate from the selection code
  - very heterogeneous software environment
  - typically consists of scripts and interactive commands
- In some cases, **bookkeeping of non-event data is not trivial:**
  - scan hundreds of points in a multi-dimensional new physics parameter phase space, each with different cross section and choice of analysis categories
  - keep consistent naming for sources of systematic uncertainties in view of a future combination with different analyses
- Not to be forgotten: standardization of graphical elements (**plotting style**)

Abstract analysis block



Typical features of analysis workflows in CMS

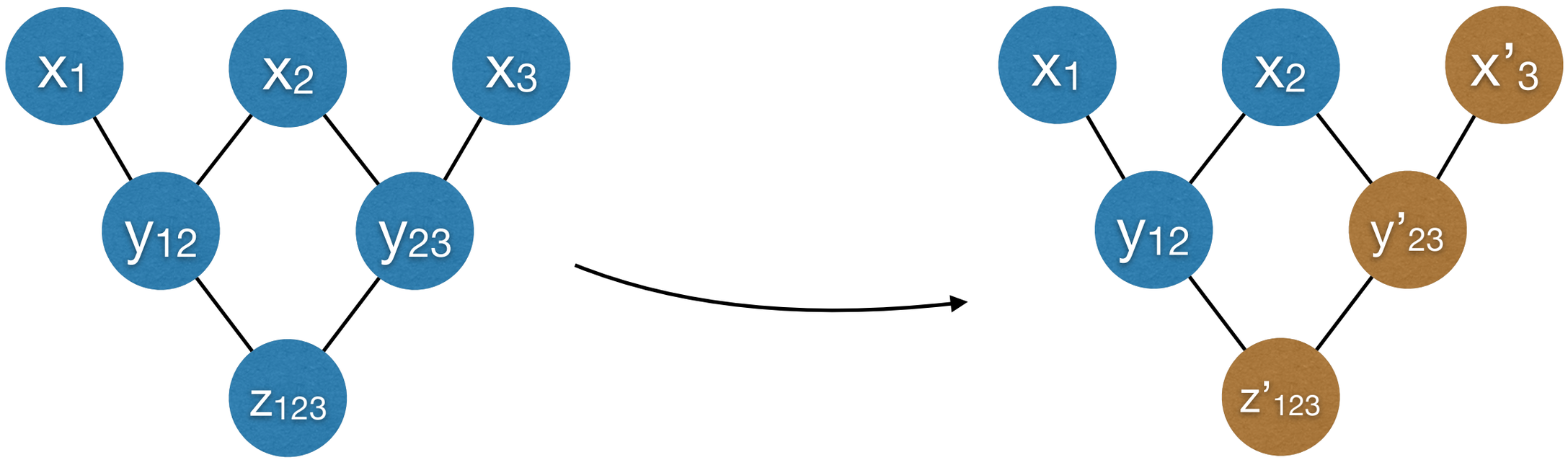
Implementing uncertainties

- Uncertainties are a crucial part of each analysis, and often complicated to treat

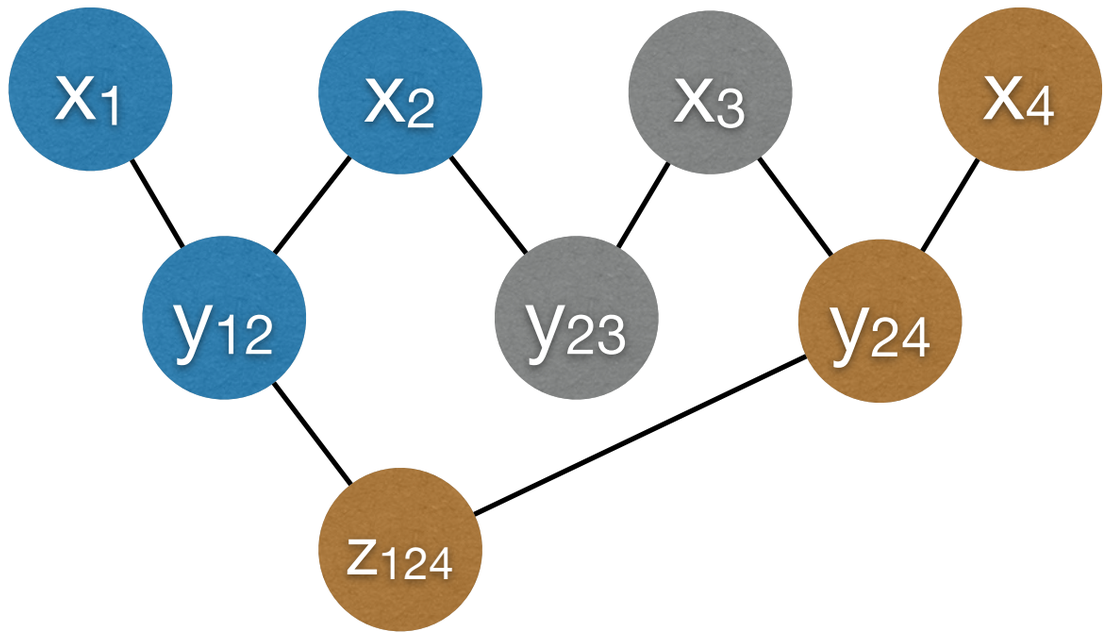
- Different ways to define their effect are used:
  - **alternative object and event weights** (e.g. for object selection efficiency)
  - **different selection or variable** (e.g. vary the object energy calibration)
  - **alternative dataset** (e.g. use same process from another generator)
  - **statistical nature** (e.g. number of events in the MC dataset, needs special care e.g. in case of negative event weights, or low number of events)

- Sources of uncertainty can be **correlated, uncorrelated, or just partially correlated**
- This is typically implemented at the very end of the analysis workflow:
  - keep all needed variations of the inputs in the event content
  - repeat the event aggregation and fitting steps with alternate definitions

Implementing uncertainties



- How can this be implemented efficiently?



Typical features of analysis workflows in CMS

Uncertainty propagation

- Handling the different cases of uncertainty propagation is typically done manually

- Note: **variations can interact with skimming requirements**
  (e.g. require at least one muon above 20 GeV, then change muon calibration)
  - sometimes it is possible to code looser requirements allowing for variations, sometimes multiple versions of the mini-tree content are produced
  - complicated file handling, bookkeeping and **propagation of dependencies**
    (e.g. if one changes jet calibration also MET, HT etc. have to be redone)

- **Issue with computationally intensive steps**:
  - rerunning them for varied inputs is sometimes difficult to imagine
    (e.g. many jet energy calibration variations propagated through matrix element)
  - how to simplify the problem in a controlled way?
  - implement criteria for dropping and/or grouping sources of uncertainties?
    Would open the way to a more central and automated treatment.

Uncertainty propagation

- Efforts done to integrate the concept of variations at least in final histogram classes
  - possible esp. if variations can be done “on-the-fly” with a standalone function
  - also simplifies plots with post-fit values of the nuisance parameters

- *Embedding* variations in the analysis workflow graph,
  and closely *linked to the data structure*,
  would greatly simplify these procedures
- For instance, one could imagine all logical nodes in the workflow endowed with:
  - lazy fetching of event content and output evaluation
  - persistent output caching
  - rules for input and output content variation driven by global parameters (nuisances)
  - calls to external programs producing output (e.g. fits, discriminators)