Software & Simulation for TLEP a few ideas

TLEP Workshop, Fermilab

Colin Bernet, Gerardo Ganis, Benedikt Hegner
(CERN)

Outline

- Goal:
  - Get started with TLEP physics analyses
  - Use these analyses to guide detector design
  - Collaborate on software & computing

- Outline
  - Existing event processing frameworks
  - Event data model
  - Particle flow simulation (PFSim)

Event Processing Framework

- Structure to contain code
  - configuration and compilation tools
  - sharing of compiled code
  - Insertion of local, private code
  - Use mainstream languages: C++, python
  - Use free, well established tools
    - fastjet, pythia, HepPDT, etc.
  - Managed by Git
  - Minimize dependencies
- Scheduler
  - declare event processing modules
  - organize them in a processing sequence
  - programmable (python)
  - concurrent processing (multicore hardware)
  - functional
    - e.g. stop in an event to reprocess it with different conditions
- Event data model
  - Definition of data objects
    - particles, jets, etc
  - Reuse existing event formats
    - HepMC, LHE, LCIO
  - keep it simple
    - small size (physics at the Z pole…)
    - no object C object C object C object
- Persistency system
  - Event I/O system
  - Database system for non-event info
  - Metadata
  - Stay flexible (should be able to change later)
  - Start with ROOT

Thanks to Lucie Linssen and Mark Thomson for discussions

Marlin + LCIO (Linear Collider)

- Structure to contain code
  - configuration and compilation tools
  - sharing of compiled code
  - Insertion of local, private code
- Scheduler
  - declare event processing modules
  - organize them in a processing sequence
- Event data model
  - Definition of data objects
    - particles, jets, etc
- Persistency system
  - Event I/O system
  - Database system for non-event info
  - Metadata

  - Use mainstream languages: C++, python
  - Use free, well established tools
    - fastjet, pythia, HepPDT, etc.
  - Managed by Git
  - Minimize dependencies

  - **programmable (python)**
  - **concurrent processing (multicore hardware)**
  - **functional**
    - **e.g. stop in an event to reprocess it with different conditions**

  - Reuse existing event formats
    - **HepMC**, LHE, LCIO
  - keep it simple
    - small size (physics at the Z pole…)
    - no object C object C object C object

  - **Stay flexible (should be able to change later)**
  - Start with ROOT

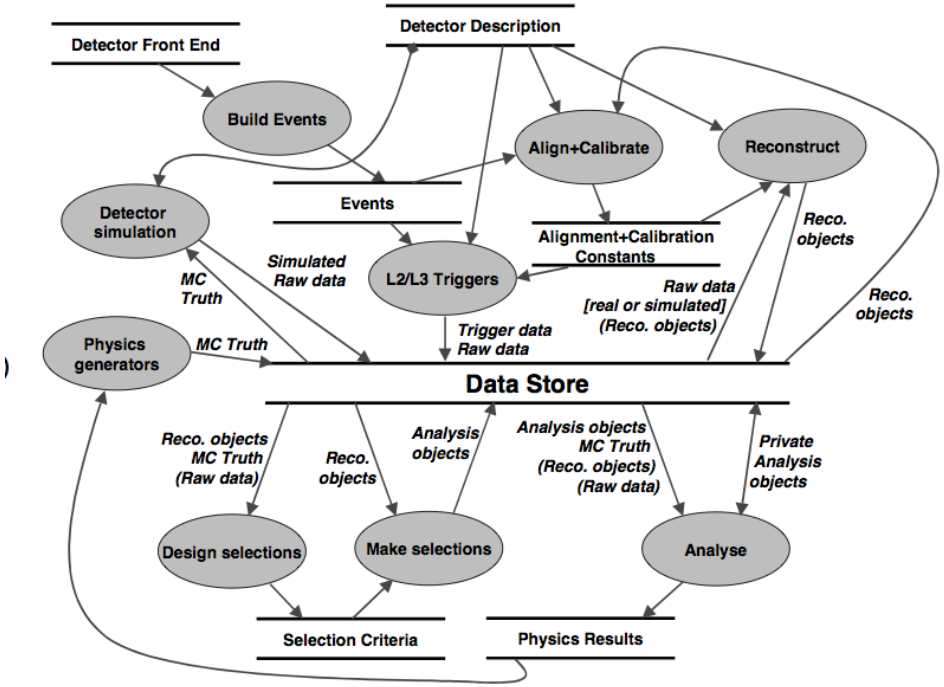
CMSSW (CMS)

- Structure to contain code
  - configuration and compilation tools
  - sharing of compiled code
  - Insertion of local, private code
  - Use mainstream languages: C++, python
  - Use free, well established tools
    - fastjet, pythia, HepPDT, etc.
  - Managed by Git
  - Minimize dependencies
- Scheduler
  - declare event processing modules
  - organize them in a processing sequence
  - programmable (python)
  - **concurrent processing (multicore hardware)**
  - **functional**
    - **e.g. stop in an event to reprocess it with different conditions**
- Event data model
  - Definition of data objects
    - particles, jets, etc
  - Reuse existing event formats
    - HepMC, LHE, LCIO
  - keep it simple
    - small size (physics at the Z pole…)
    - no object C object C object C object
- Persistency system
  - Event I/O system
  - Database system for non-event info
  - Metadata
  - **Stay flexible (should be able to change later)**
  - Start with ROOT

Gaudi (LHCb, ATLAS, …)

- Structure to contain code
  - configuration and compilation tools
  - sharing of compiled code
  - Insertion of local, private code
  - Use mainstream languages: C++, python
  - Use free, well established tools
    - fastjet, pythia, HepPDT, etc.
  - Managed by Git
  - **Minimize dependencies (not like ATLAS)**
- Scheduler
  - declare event processing modules
  - organize them in a processing sequence
  - programmable (python)
  - concurrent processing (multicore hardware)
  - functional
    - e.g. stop in an event to reprocess it with different conditions
- Event data model
  - Definition of data objects
    - particles, jets, etc
  - Reuse existing event formats
    - HepMC, LHE, LCIO
  - keep it simple
    - small size (physics at the Z pole…)
    - no object C object C object C object
- Persistency system
  - Event I/O system
  - Database system for non-event info
  - Metadata
  - Stay flexible (should be able to change later)
  - Start with ROOT

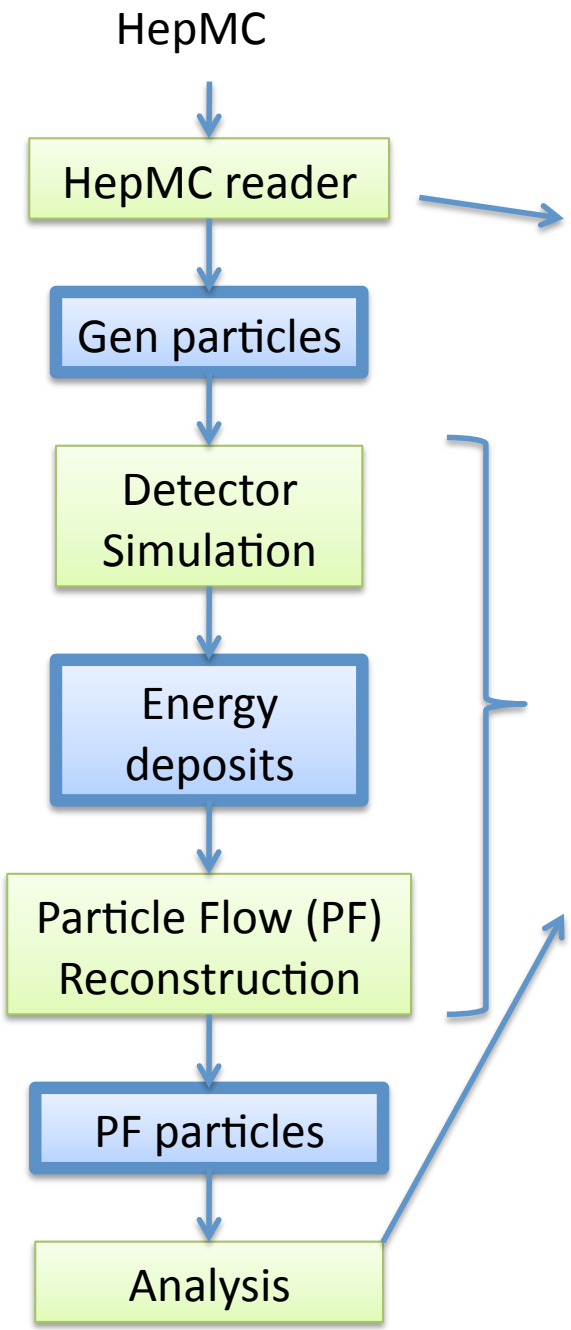
Gaudi is powerful
(So it's possible to mess things up ☺)



Architecture Design Document, Fig 2.1

Need to apply policies like in CMS, e.g.:
- Dependencies between event processing modules forbidden
- Common algorithms well defined and organized

Basic Workflow



- Keep generation out for now
  - just read HepMC events
    - from any modern generator
  - Generator interfaces can be provided later
    - e.g. pythia interface reading LHE to produce HepMC
- Simulation & Reconstruction
- Analysis
  - Start from the list of PF particles
    - stored in a ROOT file
  - Just do what you need in the way you prefer (jets, id, iso)
  - Tools can be provided
    - e.g. fastjet interface, event shapes, etc.

Event Data Model: Generic Particle Class

- Just a starting point
  - Bare minimum to get started with physics now
  - will evolve with time
- For PF and Gen Particles, and for jets
  - 4 vector 4 floats
    - vector library: ROOT::MathCore
  - pdg ID 1 int
    - HepPDT to get more info (mass, charge, etc)
  - links vector<int>
    - indices to other objects in the event, e.g. :
      - PF jets : indices of PF particles
      - Gen jets : indices of Gen particles
      - Gen Particle : indices of daughters
      - Hemisphere : indices to PF particles in the hemisphere

Event Data Model: ZH

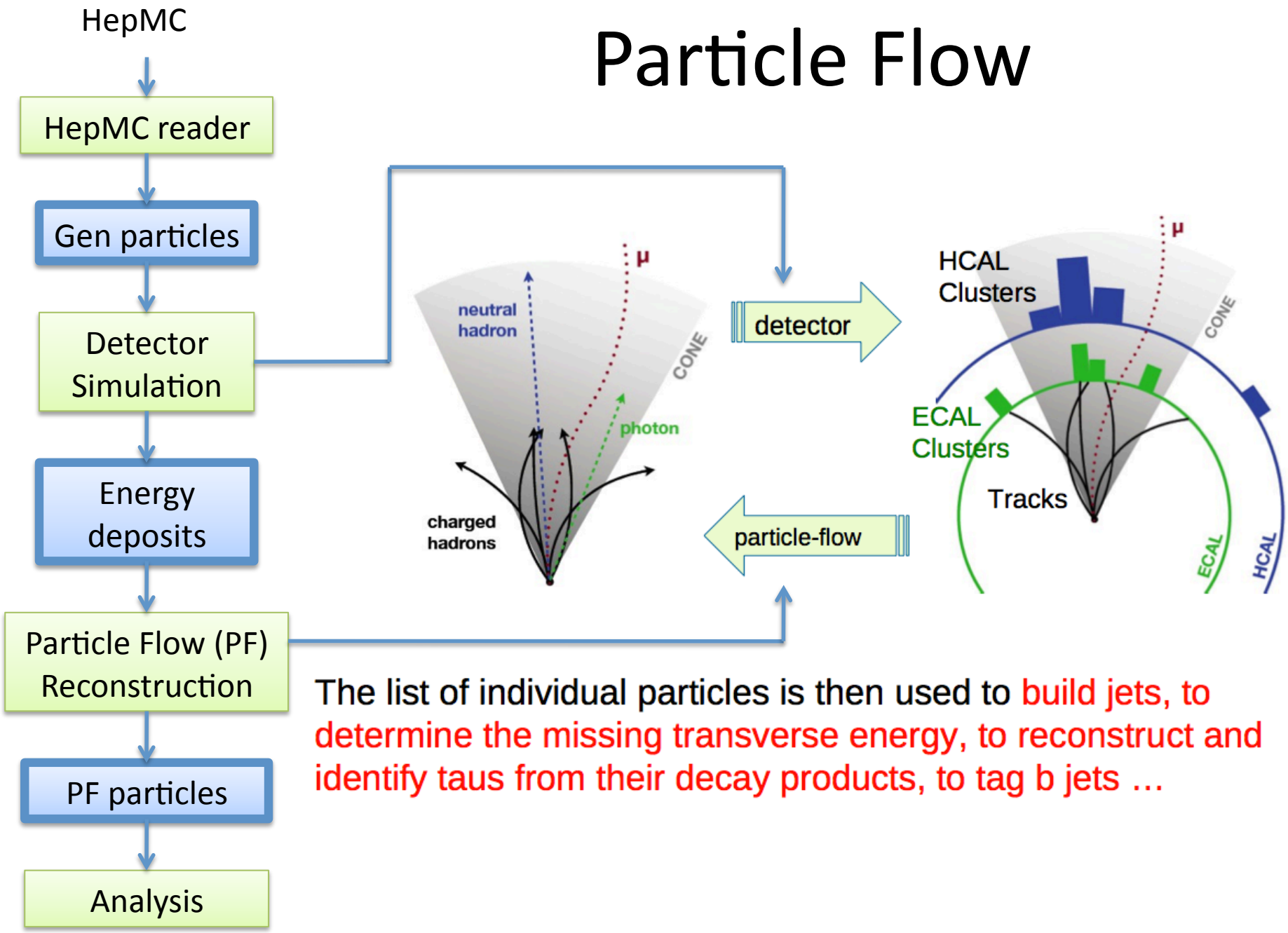
- DST: Assume we keep Gen and PF particles
- Disk space needed
  - Number of particles for 5 years @ 240 GeV = 2.6 10^9
    - 10^7 ZH-like events (~4 jets / event)
    - 200 Gen particles / event
    - 60 PF particles / event
  - Minimum size of a particle < 10 floats = 40 bytes
    - 4 vector
    - pdg id
    - daughters for Gen particles ~ 2 floats
  - Total size needed < 100 GB
    - can store everything on a notebook

Event Data Model: Z pole

- DST: Assume we keep Gen and PF particles
- Disk space needed
  - Number of particles for 1 year @ 91 GeV = 1.3 10^{14}
    - 10^{12} Z events (~2 jets / event)
    - 100 Gen particles / event
    - 30 PF particles / event
  - Minimum size of a particle < 10 floats = 40 bytes
    - 4 vector
    - pdg id
    - daughters for Gen particles ~ 2 floats
  - Total size needed < 5 PB
    - won't be a problem in 2030
    - for now, need micro DST format
      - e.g. keep jets, leptons, filtered gen particles

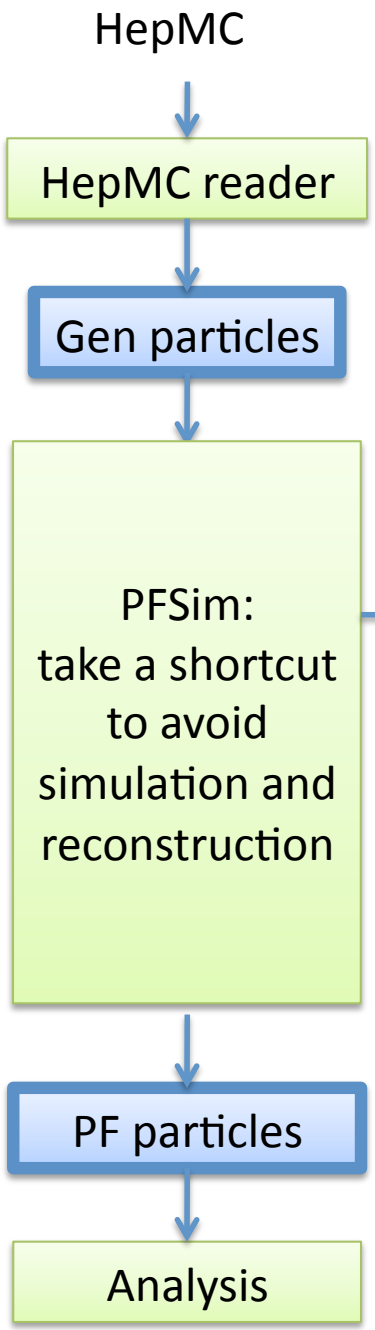
200 GHz
5 PB RAM
$19.99

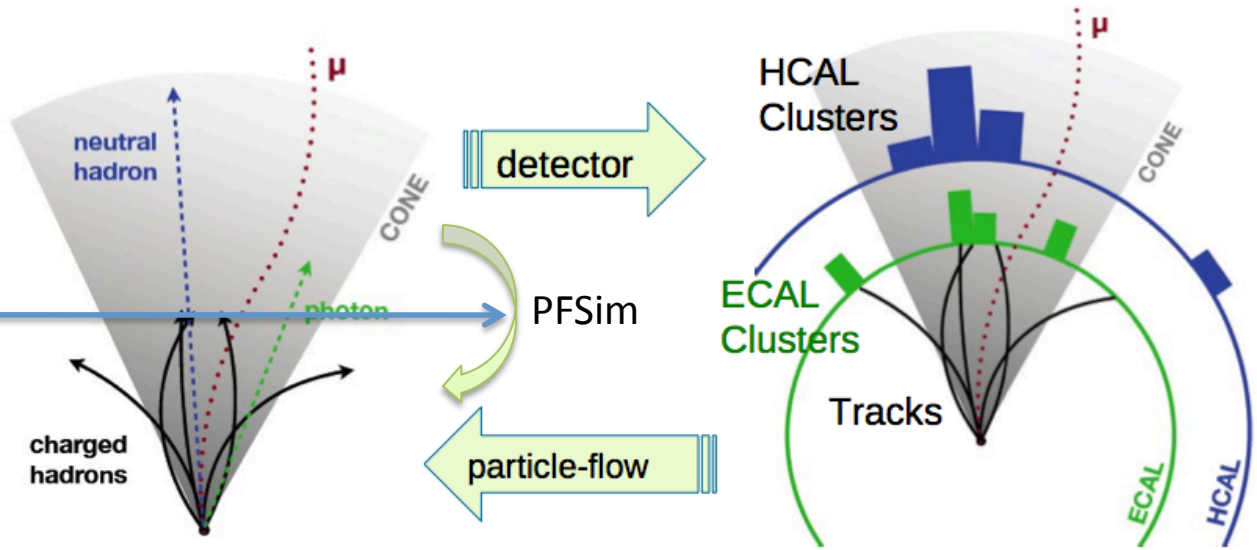
Particle Flow



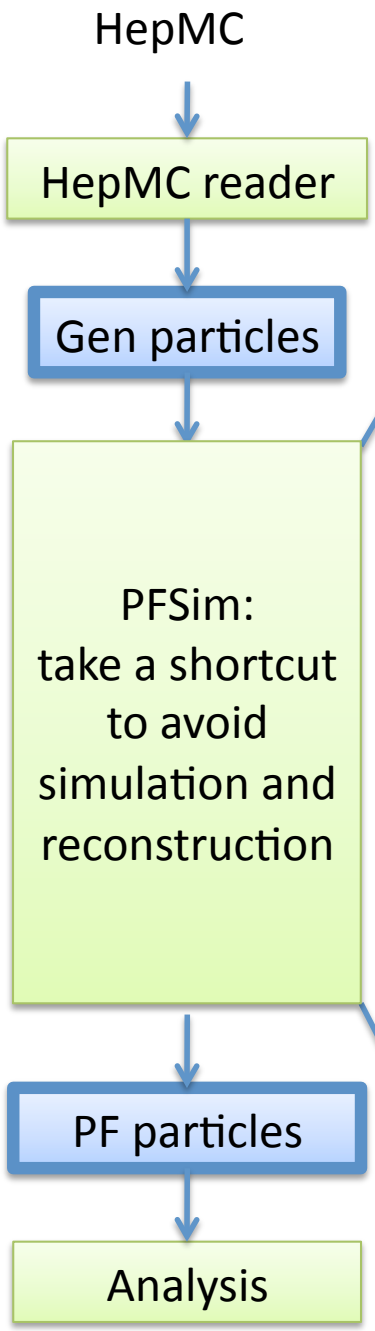
The list of individual particles is then used to build jets, to determine the missing transverse energy, to reconstruct and identify taus from their decay products, to tag b jets …

Particle Flow





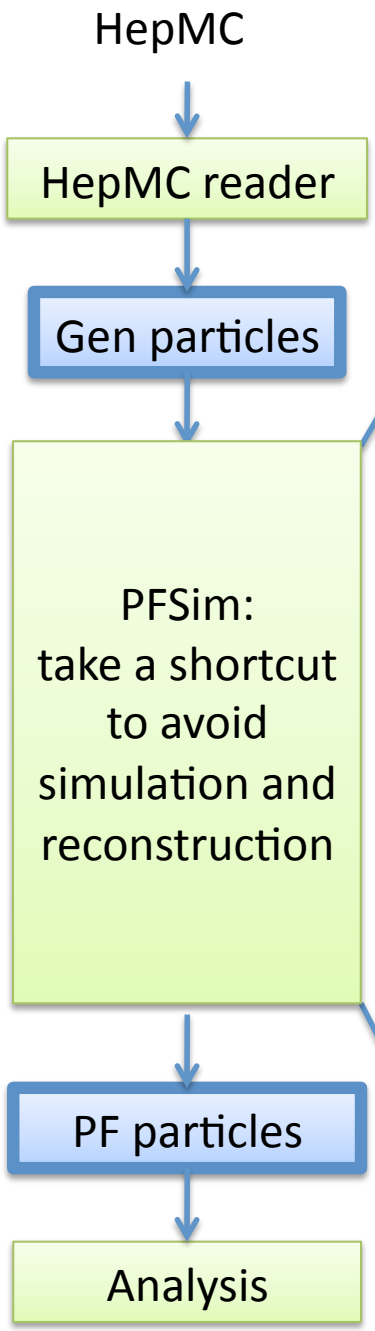
The list of individual particles is then used to build jets, to determine the missing transverse energy, to reconstruct and identify taus from their decay products, to tag b jets …



PFSim

- Super fast parametric simulation
- Provides a list of PF particles
- Goal:
  - Develop subsequent steps (analysis)
  - Guide GEANT design
    - e.g. study the influence of HCAL resolution on all TLEP physics results
- Same thing was done in ALEPH
  - QUFSIM
- In place of PFSim, can also plug Delphes, or a fastsim a la CMS

PFSim



- Filter GenParticles
  - stable, visible, sort by type
- Pass through a detector model
  - smear energy
  - smear direction
  - apply efficiency map
- Simulate particle flow algorithm

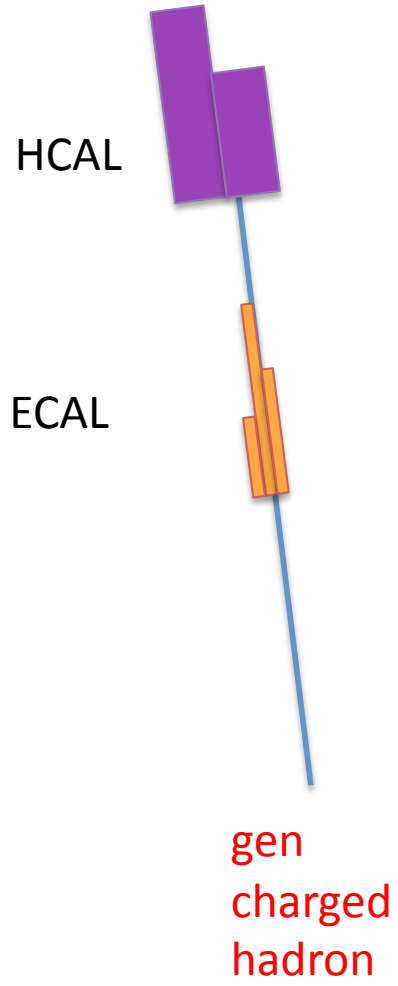
PFSim: Detector Model

- e.g. photons
  - similar functions for muons, electrons, charged hadrons, neutral hadrons.

```
float PFSim::CMS::photonEfficiency(const HepMC::FourVector& mom) const {
  float energy = mom.e();
  float eta = mom.eta();
  float effvalue = 0.;
  if (energy>0.25 && fabs(eta)<2.95)
    effvalue = 1.0;
  return effvalue;
}
```

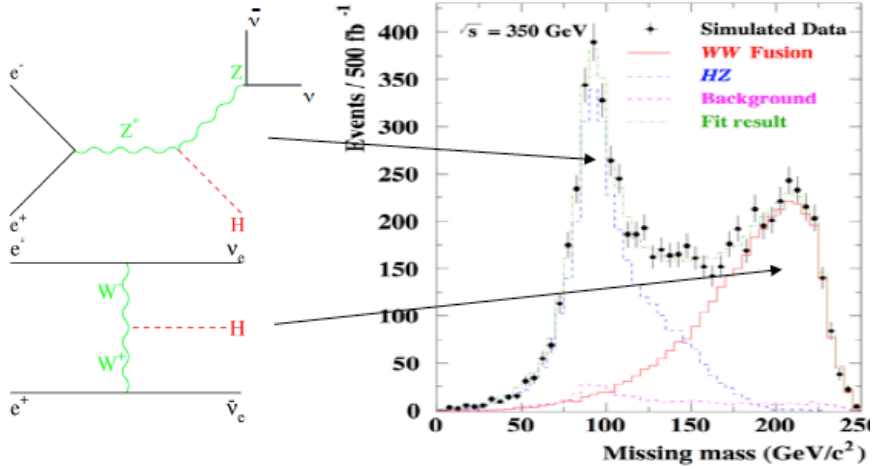
```
float PFSim::CMS::photonResolution(const HepMC::FourVector& mom) const {
  float energy = mom.e();
  return 0.03 / sqrt(energy);
}
```

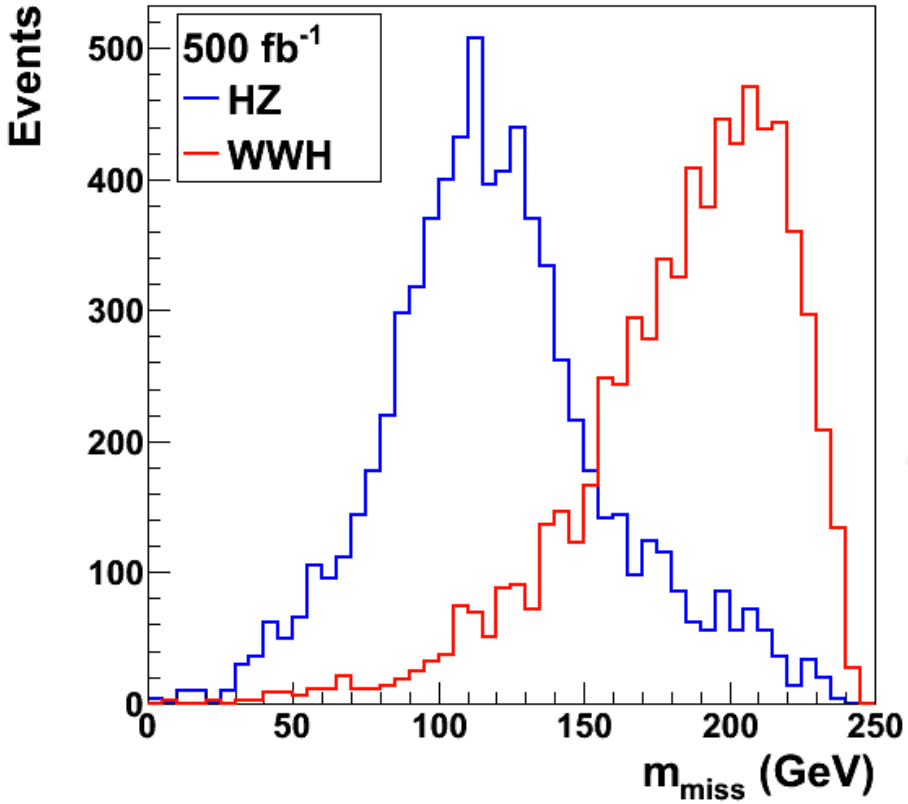
Particle Flow Simulation



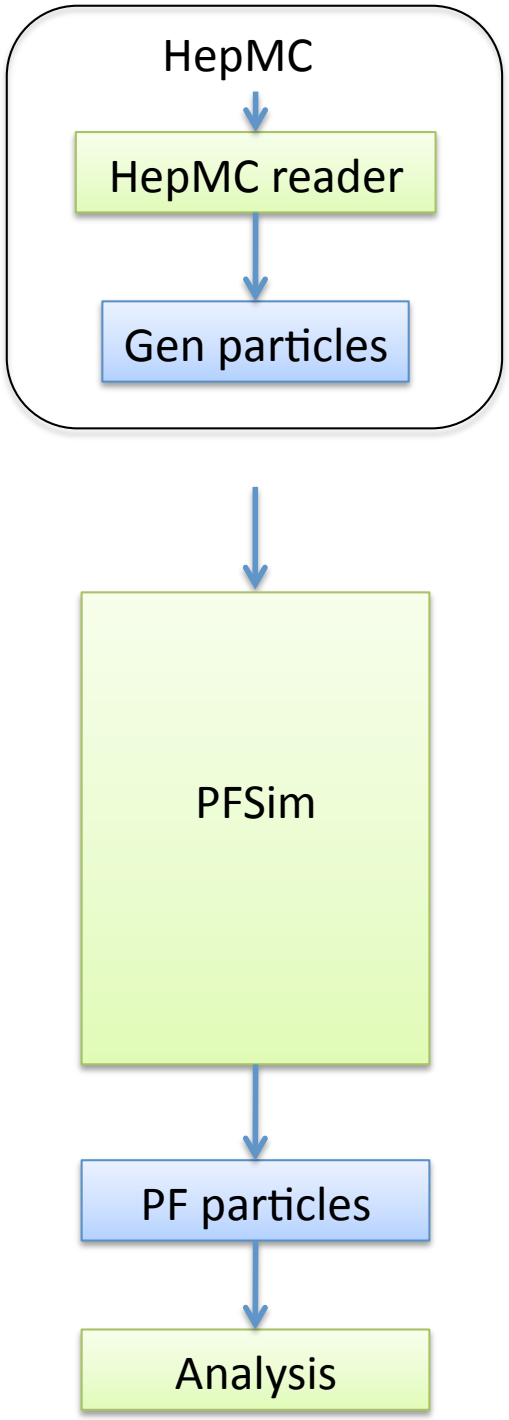
- Example:
  gen charged hadron seen by the tracker?
  - yes:
    - PF charged hadron
  - no:
    - neutral hadron energy smearing (120%/√E) and direction smearing
    - seen by the calorimeters?
      - yes : PF neutral hadron
      - no : nothing

Towards a Γ_H measurement





- CMS-like detector model, with slightly better PF
- visible mass:
  - m_{vis} = mass of all PF particles
  - m_H-20 < m_{vis} < m_H+10
- m_{miss} = missing mass



Immediate Plans

- First example:
  HepMC reader in Gaudi within a couple weeks
- PFSim ready within a couple months
  - developed within CMSSW for now
  - validating against CMS simulation & reconstruction
  - plug into example when PFSim ~ok
- /afs/cern.ch/project/tlep/
  - will start installing software soon
    - pythia, HZHA, fastjet, root, …
- github organization created
  - https://github.com/tlep
- Feedback & help very welcome