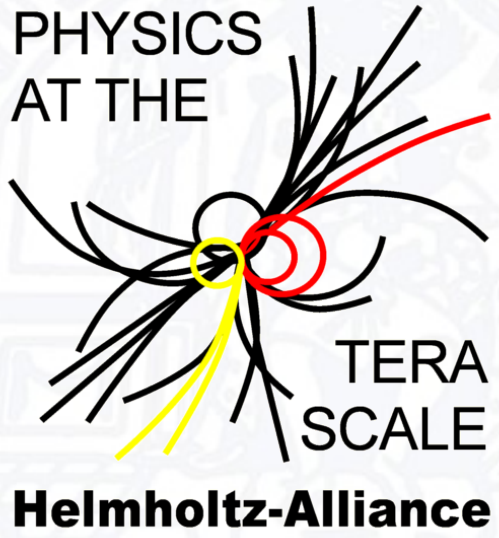
Monte Carlo Generators in Atlas software

CHEP Conference , Prague
24.03.2009

Cano Ay on behalf of the ATLAS Collaboration
aycano@cern.ch



Contents

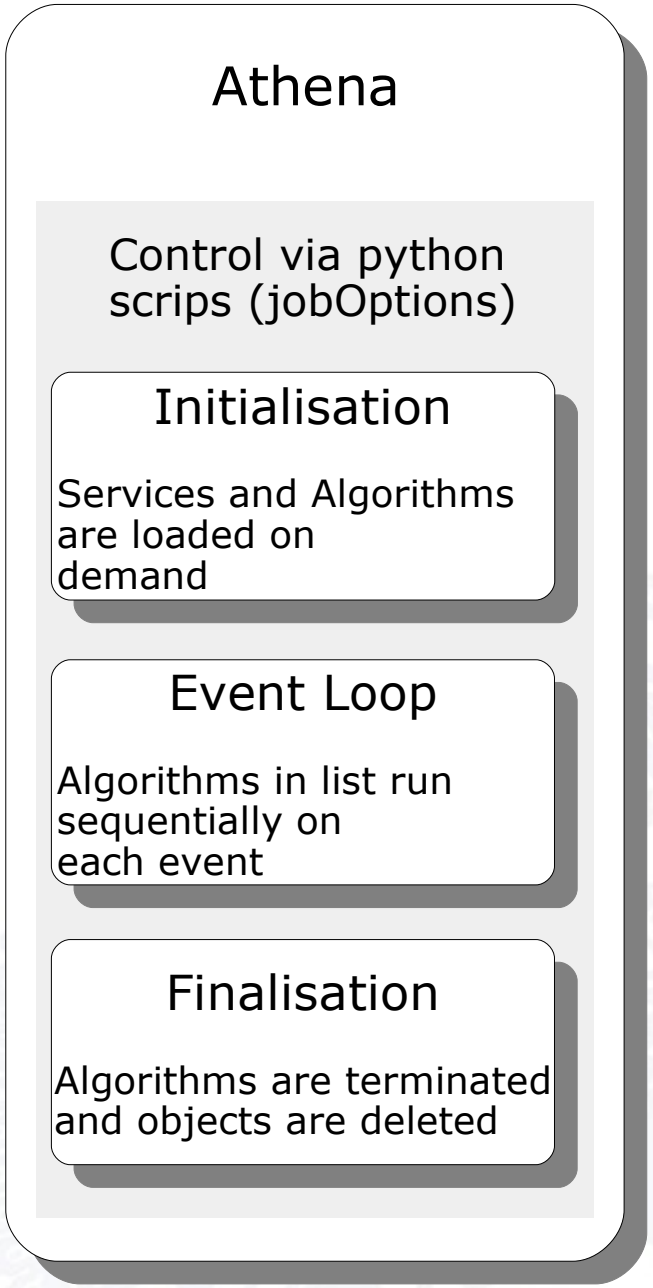
- Introduction
- The Atlas software
- Generators in Atlas
- Les Houches Event File
- JobTransform
  - MC Production
- Summary

Introduction

- LHC Experiments have rich physics programme:
  - Standard Model: Top, Electroweak, Higgs, ...
  - New Physics: Exotics, Susy, ...
- Huge background
  - From QCD processes
  - Multiple interactions and underlying events
  - One's Signal is background for other
- Integrate External MC Generators in Atlas software
  - Different Generators developed for different purposes
  - Physics: new Energy scale
    - Tune MC to match 'soon coming' data
    - Interface with steering possibility for LHC Energies
  - Design Interface for 'each' external Generator

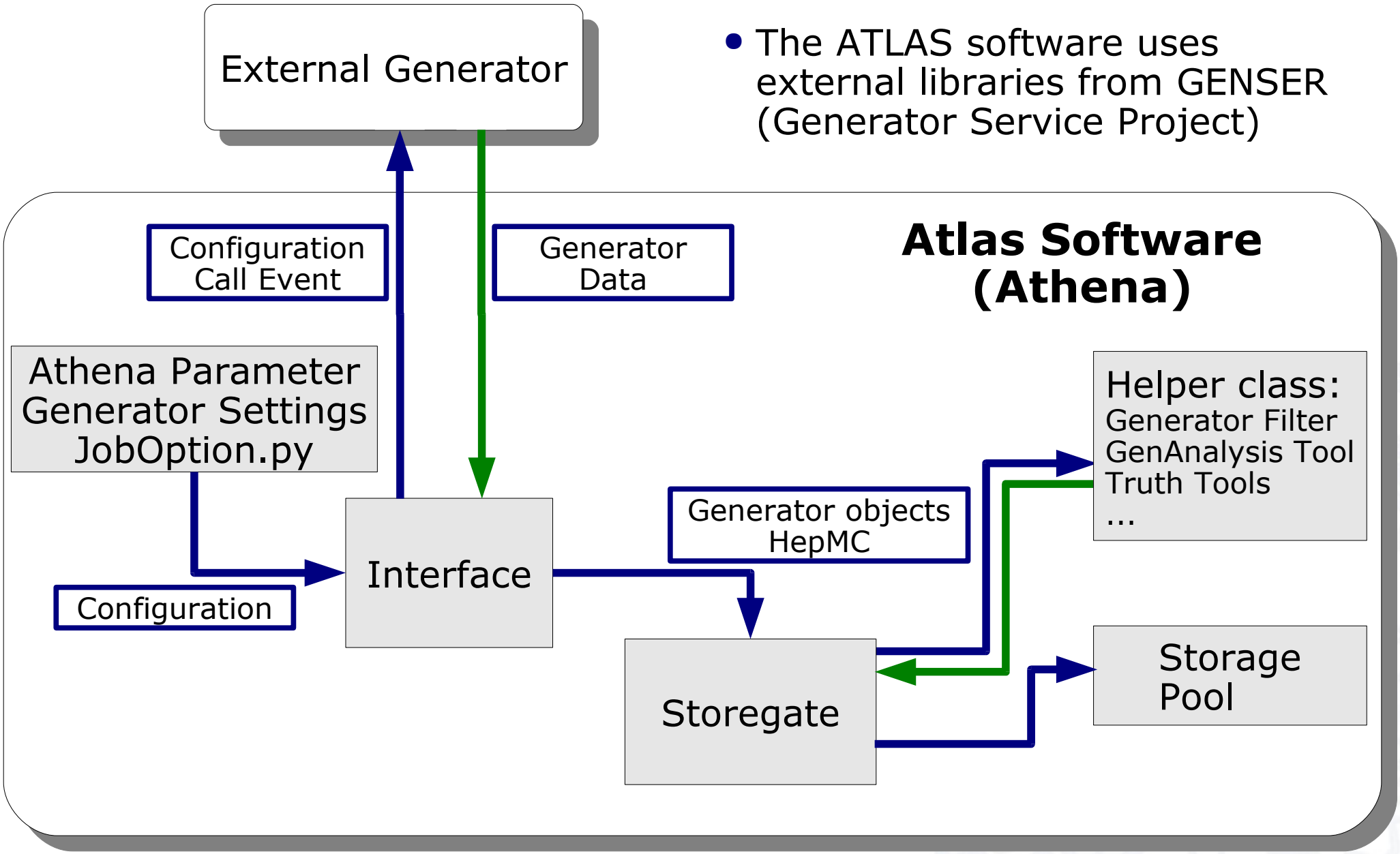
The Atlas Software

- The Atlas software
  - Control framework and based on Gaudi
  - Three steps
  - Python driven
    - configure and load C++ algorithms and objects for more flexibility
    - No need to recompile
- Control packages via CMT (Configuration Management Tool) and cvs
  - Packages have common directory structure
    - cmt, src, <PackageName>, run, share, ...
  - requirements file
    - Control of Dependencies
    - Creates Makefiles and setup scripts automatically
- Nightly build for Code consistency checks
- Tag Collector to control packages in release

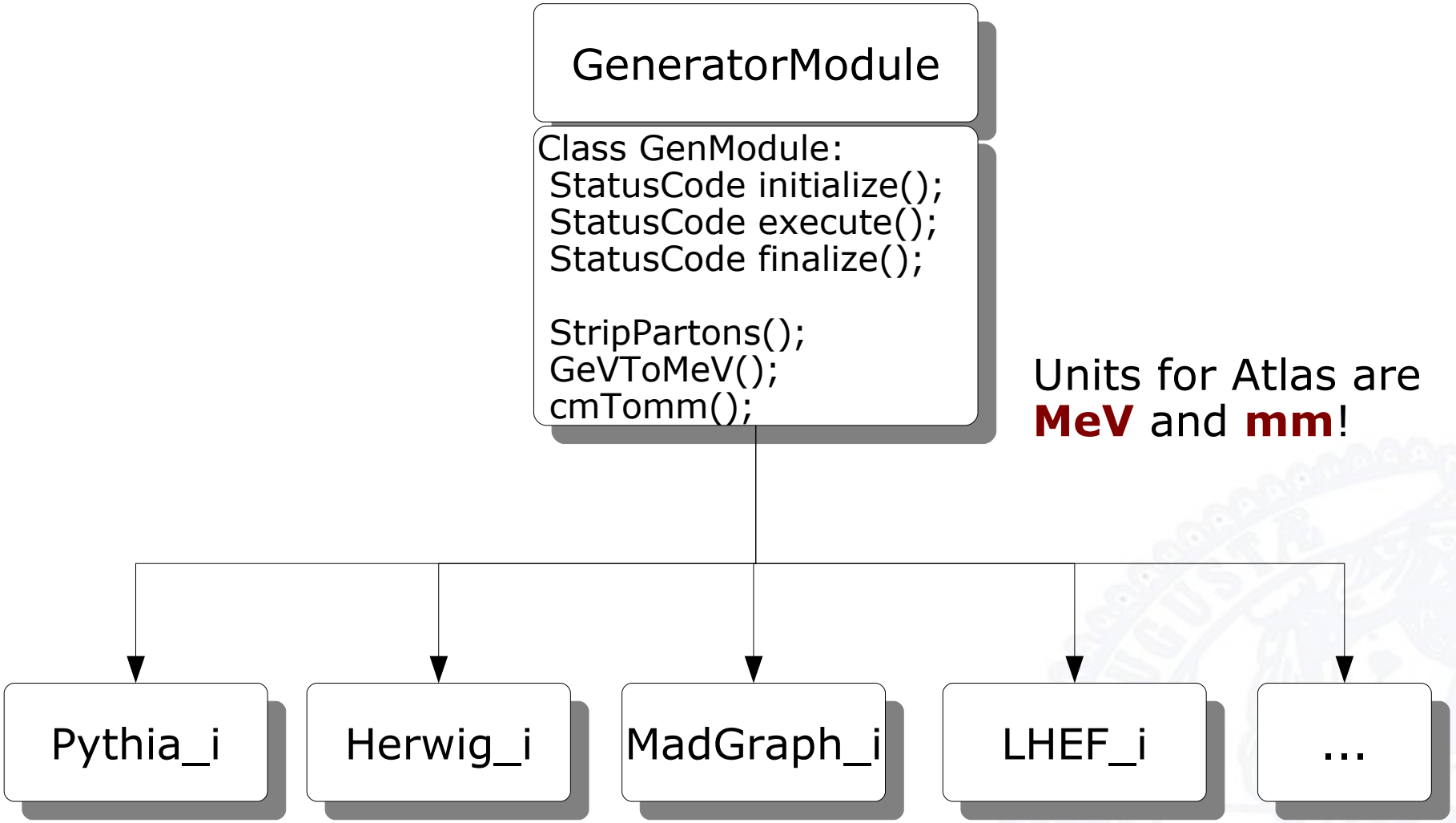


Default Workflow

- The ATLAS software uses external libraries from GENSER (Generator Service Project)



Interface packages



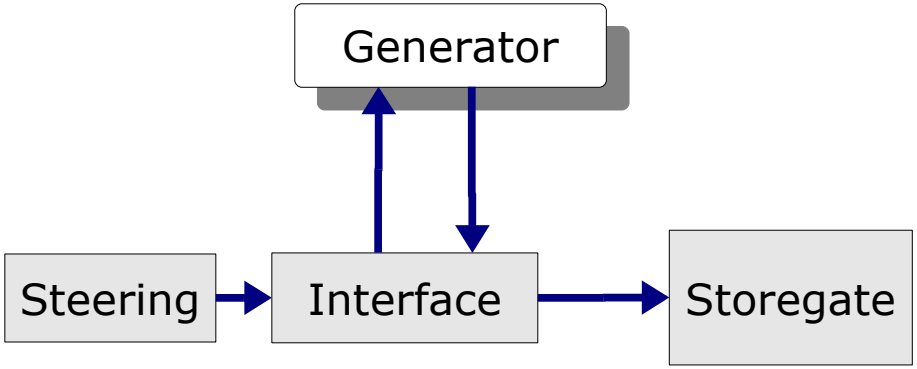
Units for Atlas are **MeV** and **mm**!

+ Configuration via jobOptions
  - Handle specific parameter of Generator
  - Valid options from documentation of generator

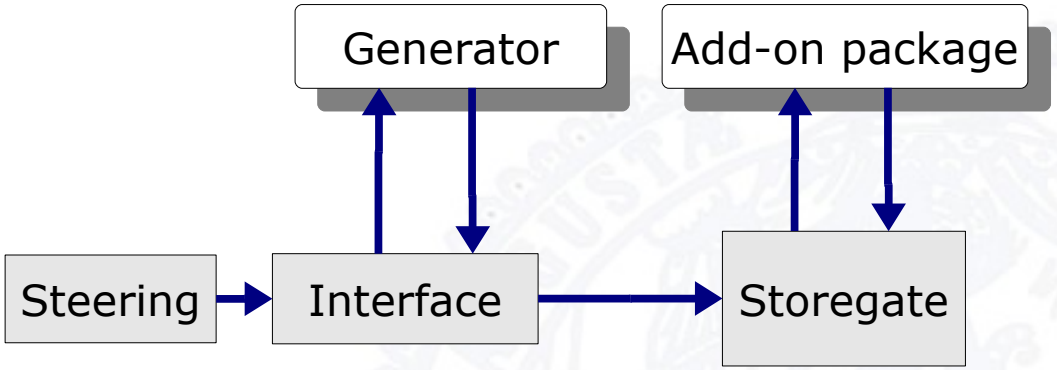
Generators in ATLAS

Long list of Generators and are still adding to it:

1. Full Generators including Parton showers and fragmentation:
   - Pythia
   - Herwig
   - Sherpa
   - Hijing (uses some Pythia Tools)
   - Pythia8
   - Herwig++



2. Specific purpose add-on packages to generators
   - Tauola
   - EvtGen
   - Photos

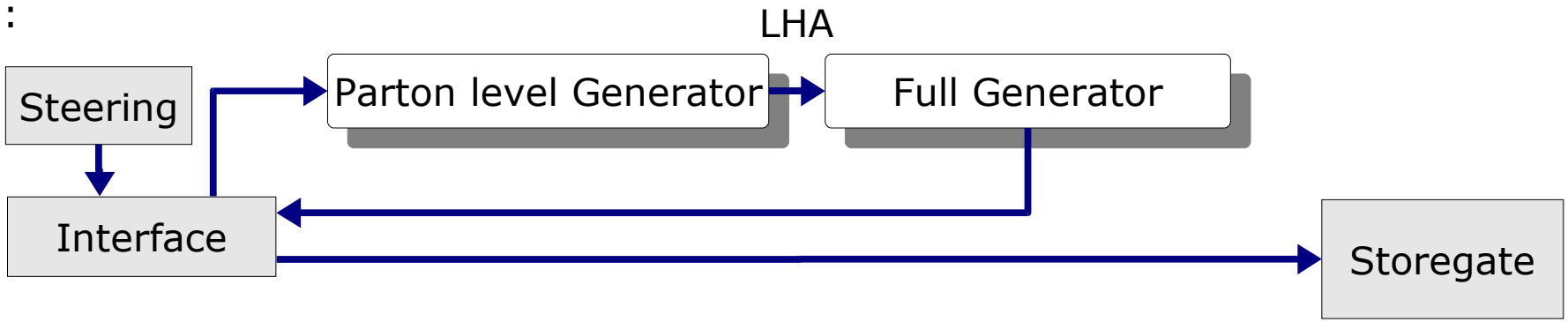


3. Parton level Generators to be interfaced
   - AcerMC
   - Alpgen
   - Cascade
   - ...

Generators in ATLAS II

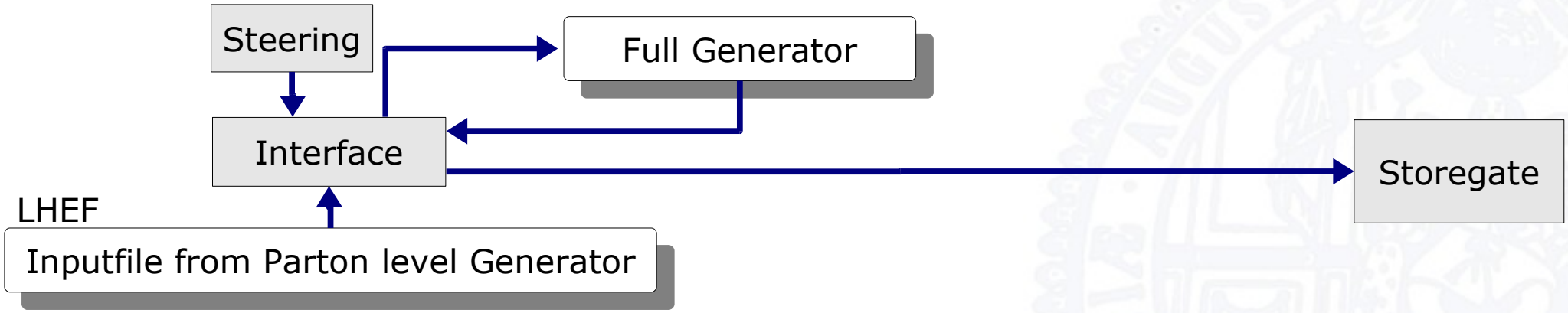
Two Possibilities to interface Parton level Generators:

Possibility 1:



Example: AcerMC, Cascade, ...

Possibility 2:



Example: MadGraph, Baur, Winhac, MC@NLO, ...

Les Houches Event File

- LHA Common Block
  - FORTRAN common blocks HEPRUP and HEPUP - API
  - implement interface as Pythia / Herwig user process
  - Lives in Memory – well defined standard

- "old style LHEF"
  - ASCII dump of the LHA Common Block
  - generator specific order of variables / initialization
  - Not a standard

Not Preferred by Atlas

- LHEF
  - A standard format for Les Houches Event Files J. Alwall et al., September 3, 2006, arXiv:hep-ph/0609017
  - XML dump of the LHA Common Block
  - comments, headers, extensible, ...
  - well-defined standard

Generic reading of LHEF format preferred by Atlas

Interface packages – run Athena

Example: generate Pythia Events

jobOption.py

```
from AthenaCommon.AppMgr import ServiceMgr
ServiceMgr.MessageSvc.OutputLevel = DEBUG

theApp.EvtMax = 5

from AthenaServices.AthenaServicesConf import AtRndmGenSvc
ServiceMgr += AtRndmGenSvc()
ServiceMgr.AtRndmGenSvc.Seeds = ["PYTHIA 4789899 989240512",
                                 "PYTHIA_INIT 820021 2347532"]

from AthenaCommon.AlgSequence import AlgSequence
job=AlgSequence()

from Pythia_i.Pythia_iConf import Pythia
job +=Pythia()
job.Pythia.PythiaCommand = ["pysubs msel 13","pysubs ckin 3 18.","pypars mstp 43 2"]
job.Pythia.PythiaCommand += ["pypars mstp 51 19070", "pypars mstp 52 2",
                             "pypars mstp 53 19070", "pypars mstp 54 2",
                             "pypars mstp 55 19070", "pypars mstp 56 2"]

from TruthExamples.TruthExamplesConf import DumpMC
job += DumpMC()
```

Interface packages – run Athena

Example: generate Pythia Events

athena jobOption.py

jobOption.py

```
from AthenaCommon.AppMgr import ServiceMgr
ServiceMgr.MessageSvc.OutputLevel = DEBUG

theApp.EvtMax = 5

from AthenaServices.AthenaServicesConf import AtRndmGenSvc
ServiceMgr += AtRndmGenSvc()
ServiceMgr.AtRndmGenSvc.Seeds = ["PYTHIA 4789899 989240512",
                                 "PYTHIA_INIT 820021 2347532"]

from AthenaCommon.AlgSequence import AlgSequence
job=AlgSequence()

from Pythia_i.Pythia_iConf import Pythia
job +=Pythia()
job.Pythia.PythiaCommand = ["pysubs msel 13","pysubs ckin 3 18.",
                            "pypars mstp 43 2"]
job.Pythia.PythiaCommand += ["pypars mstp 51 19070", "pypars mstp 52 2",
                             "pypars mstp 53 19070", "pypars mstp 54 2",
                             "pypars mstp 55 19070", "pypars mstp 56 2"]

from TruthExamples.TruthExamplesConf import DumpMC
job += DumpMC()
```

Application Manager

Service Manager

- Message Svc
- Random Seed Svc
- ...

Algorithm sequence

- Pythia
- DumpMC
- ...

Transformations

For production tasks use 'transformations' instead of hundreds of jobOption files!

Transform Inputfile into Outpufile
with steering possibility

- Base features of Transformations
  - Pass command line arguments to a skeleton jobOptions file
  - Extensible and flexible framework
  - Produce a summary jobReport at the end
- Available transformations
  - Event generation (any type)
  - Geant4 simulation, Digitization and Combined
  - Reconstruction (ESD and AOD making)

Components of JobTransform

- python executable script defining the transform: scripts/*_trf.py
- skeleton joboptions file used to run athena: share/skeleton.*.py
- runargs joboptions file, created on-the-fly at runtime to make the command line arguments available to the skeleton
- configuration object (a python object) to steer the skeleton joboptions file (make it Configurable)

```
from PyJobTransformsCore.trf import JobTransform
from PyJobTransformsCore.full_trfarg import *
from MyPackage.MyConfig import myConfig

class MyTransform(JobTransform):
    def __init__(self):
        JobTransform.__init__(self, skeleton='myskeleton.py', config=myConfig )
        self.add ( MaxEvents() )

# execute it if run as a script
if __name__ == '__main__':
    # make transform object
    trf = MyTransform()

import sys
sys.exit( trf.exeSysArgs().exitCode() )
```

EvGenJobTransform

Use predefined jobFragments controlled via command line arguments or grid tools

→ Pass arguments and run on the local/Grid for supported Generators
→ Parameter centrally controlled and maintained
→ data reproducable

Grid tools:
Ganga
Pathena
panda

set arguments

JobTransform:
csc_evgen08_trf.py

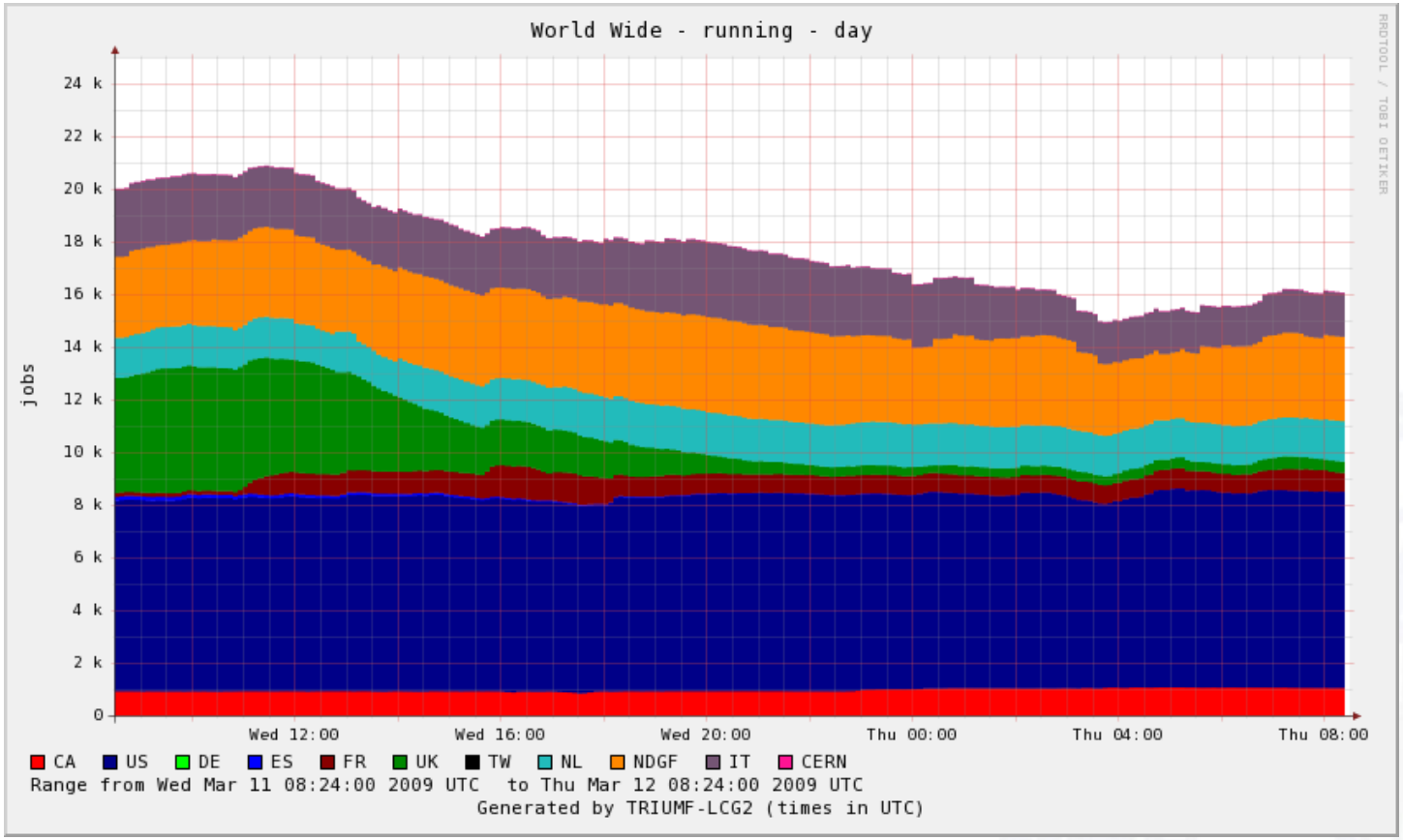
Skeleton joboption file:
skeleton.csc_evgen08.py

Config file:
evgenConfig

MC Production worldwide

Snapshot from:
http://panda.cern.ch:25880/server/pandamon/query?dash=prod

20k



- Thousands of Jobs every day worldwide
- Millions of Events produced

Summary

- Basic functionality of Atlas software
  - Managed by cmt and cvs
  - Controlled via python jobOption scripts
- Generators used in Atlas
  - Depending on Generator three concepts generating Events
  - Les Houches Event file format
- Example jobOption file for generate Pythia Events
- More flexible usage with JobTransform
  - Generate Events over Grid Tools