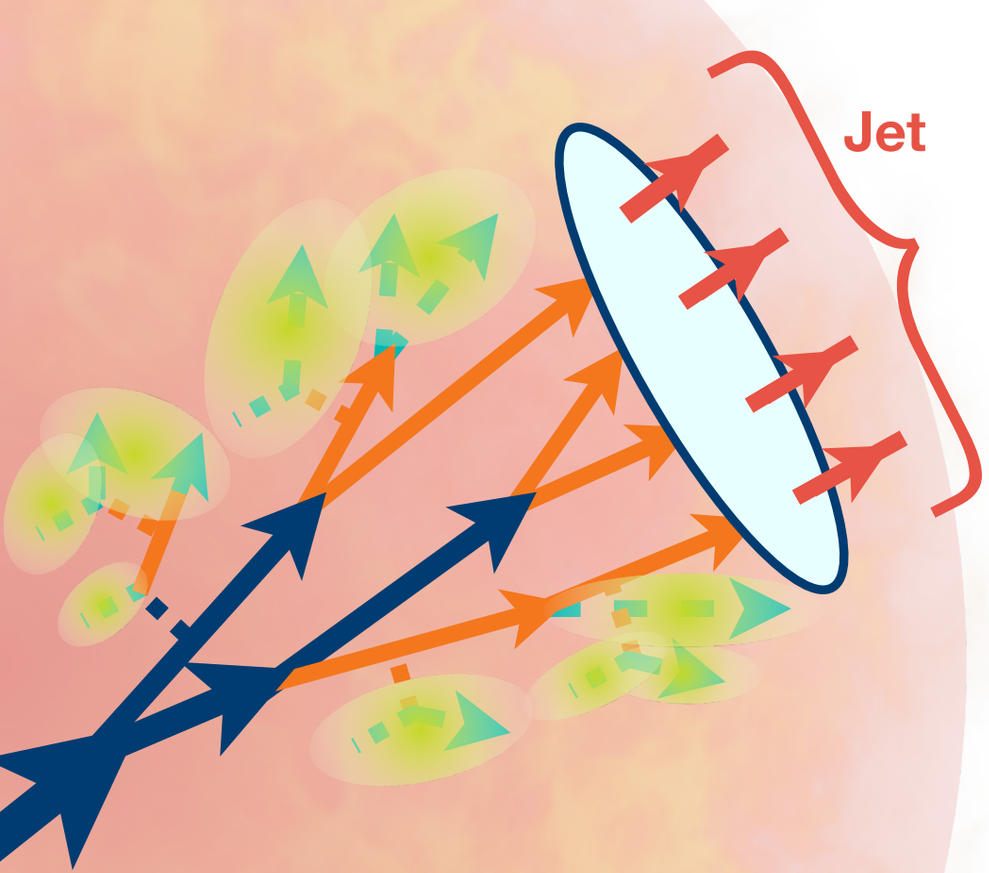
# The JETSCAPE and XSCAPE Framework

## Lecture




**Joern Putschke**
**(Wayne State University)**

***JETSCAPE Online School***
**July 17 2023**

If you have issues with installing/running JETSCAPE/X-SCAPE in Docker, ask questions/get support in slack channel:

**#software-install-problems**

Ask questions in slack channel:

**#july17-18-xscape-framework**

# JETSCAPE/X-SCAPE

## Event generator

- A ***framework*** for general-purpose MC event generators in e-A, p/d-A and A-A collisions

**https://github.com/JETSCAPE/X-SCAPE**

## Statistical toolkit

- Extract model parameters via Bayesian analysis with Gaussian Process Emulators

**https://github.com/JETSCAPE/STAT**

**Topic for today**

# A general-purpose MC framework

**JETSCAPE/X-SCAPE is not just for jets!**
**It is a framework for *general-purpose* event generators**

1 **The JETSCAPE/X-SCAPE framework is modular**

The core framework decides how physics modules can interact with each other — but the modules themselves can be user-contributed

2 **Physics modules are open-source**

Key improvement in particular in heavy-ion physics — predictions can be checked against many observables simultaneously

**A unified framework has clear benefits when we want to compare models of one particular part of a multi-stage event evolution**

# The current status

**The framework(s) are available and ready for public use**

**Wide variety of physics available — additions ongoing**

**Ideal time to contribute additional physics modules**

README.md

## JETSCAPE 3.5.4 https://github.com/JETSCAPE/JETSCAPE

The JETSCAPE simulation framework is an overarching computational envelope for developing complete event generators for heavy-ion collisions. It allows for modular incorporation of a wide variety of existing and future software that simulates different aspects of a heavy-ion collision. For a full introduction to JETSCAPE, please see The JETSCAPE framework.

Please cite The JETSCAPE framework if you use this package for scientific work.

### Installation

Please see the Installation Instructions.

README.md

## X-SCAPE 1.0 https://github.com/JETSCAPE/X-SCAPE

The X-ion collisions with a Statistically and Computationally Advanced Program Envelope (X-SCAPE) is the enhanced (and 2nd) project of the JETSCAPE collaboration which extends the framework to include small systems created in p-A and p-p collisions, lower energy heavy-ion collisions and electron-Ion collisions. The new framework allows for novel functionality such as the ability of the main simulation clock to go backwards and forwards, to deal systematically with initial state and final state evolution. It allows for multiple bulk event generators to run concurrently while exchanging information via a new Bulk Dynamics Manager. The X-SCAPE framework can be run using the new functionality or in JETSCAPE mode allowing for full backwards compatibility. New modules can also run in a hybrid fashion, choosing to use or not use the new clock functionality. More documentation of the new X-SCAPE framework capabilities will be provided in the near future. For now, test examples showcasing the new X-SCAPE framework functionalities can be found in the `./examples/custom_examples/` directory (for example in `PythiaBDMTes.cc` and `PythiaBrickTest.cc`).

The JETSCAPE simulation framework is an overarching computational envelope for developing complete event generators for heavy-ion collisions. It allows for modular incorporation of a wide variety of existing and future software that simulates different aspects of a heavy-ion collision. For a full introduction to JETSCAPE, please see The JETSCAPE framework.

Please cite The JETSCAPE framework if you use this package for scientific work.

# JETSCAPE vs X-SCAPE

**X-SCAPE is fully backwards compatible to JETSCAPE!**
**→ X-SCAPE 1.0 includes and can be run in JETSCAPE mode exactly like JETSCAPE 3.5.x !**

**JETSCAPE release map:**
- 3.6.x last feature/physics release (fragmentation hadrons in SMASH)
- 4.x code optimization

**X-SCAPE release map:**
- 1.1 to include JETSCAPE 3.6.x
- 2.x Low beam energy AA
- 3.x EIC physics

In this lecture, if I refer to JETSCAPE (logo in upper right corner), I refer to running X-SCAPE 1.0 in JETSCAPE mode!

If I refer to X-SCAPE (logo in upper right corner), I refer to X-SCAPE 1.0 specific functionalities not part of JETSCAPE

# Installing X-SCAPE

## Doc.Installation

https://github.com/JETSCAPE/X-SCAPE/wiki/Doc.Installation

Joe Latessa edited this page on Apr 11 · 5 revisions

To run X-SCAPE, you will need to install several software pre-requisites, and then build X-SCAPE. Acquiring the pre-requisites using a container environment will be simpler than installing the pre-requisites manually.

### Instructions to Install and Run X-SCAPE Using Docker

**Preferred for local use and development**

We recommend to install X-SCAPE and its pre-requisites using Docker.

### Instructions to Install and Run X-SCAPE Using Singularity

**Usually needed to use container feature in large computing farms (some comments later)**

Another option is to install X-SCAPE and its pre-requisites using Singularity.

### Manual Installation (not recommended)

Please see the instructions here.

### External packages

To run certain external software (MUSIC, CLVisc, SMASH), you will need to explicitly download them, and you may need to re-run `cmake` with specific command-line options. Scripts to download and install the external packages are provided in `external_packages/`. Please see external packages for full details.

The available cmake options are:

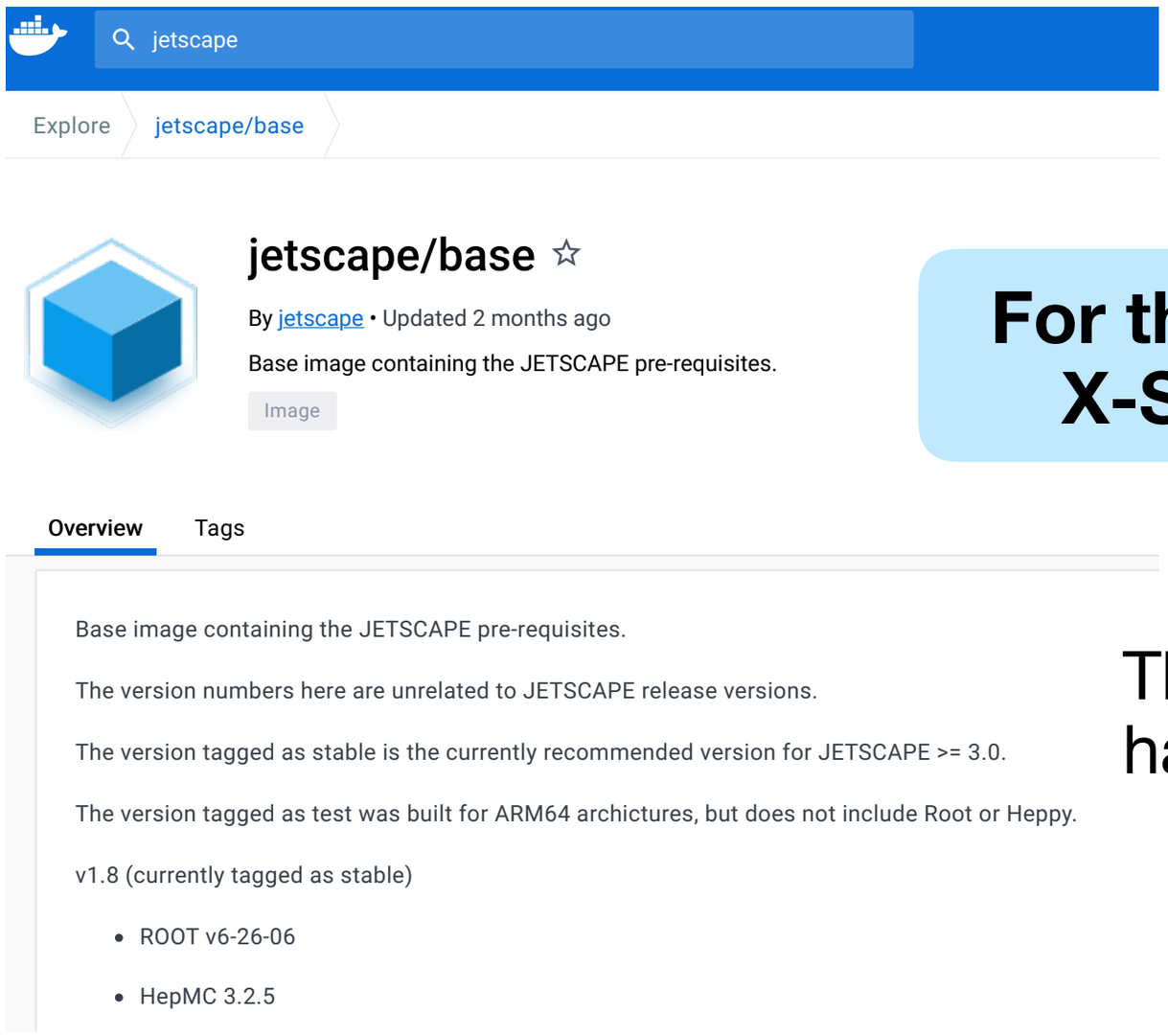
```
cmake .. -DUSE_MUSIC=ON -DUSE_ISS=ON -DUSE_FREESTREAM=ON -DUSE_SMASH=ON -DUSE_CLVISC=ON
```

# Installing X-SCAPE

## Docker support

One line of code to create environment with all software pre-reqs

For example: https://github.com/JETSCAPE/X-SCAPE/wiki/Doc.Installation.Docker.MacOS




**For this school, we require you to run X-SCAPE (JETSCAPE) via docker**

This allows everyone in the school to have a uniform software environment

# More Docker Containers …

**If you just want to use X-SCAPE/JETSCAPE as an out of the box event generator we provide fully compiled, ready to run Docker containers!**

**jetscape/xscape_full** ☆

By jetscape • Updated 3 months ago

Image containing a full implementation of X-SCAPE.

Image

Pulls 10

Overview | Tags

xscape_full contains a full implementation of X-SCAPE.

The version numbers here are unrelated to X-SCAPE release versions.

beta_v0.2

- Implements X-SCAPE

**Docker Pull Command**

```
docker pull jetscape/xscape_full
```

**jetscape/jetscape_full** ☆

By jetscape • Updated 3 months ago

Image containing a full implementation of JETSCAPE.

Image

Pulls 5

Overview | Tags

jetscape_full contains a full implementation of JETSCAPE.

The version numbers here are unrelated to JETSCAPE release versions.

beta_v0.2

- Implements JETSCAPE 3.5.4

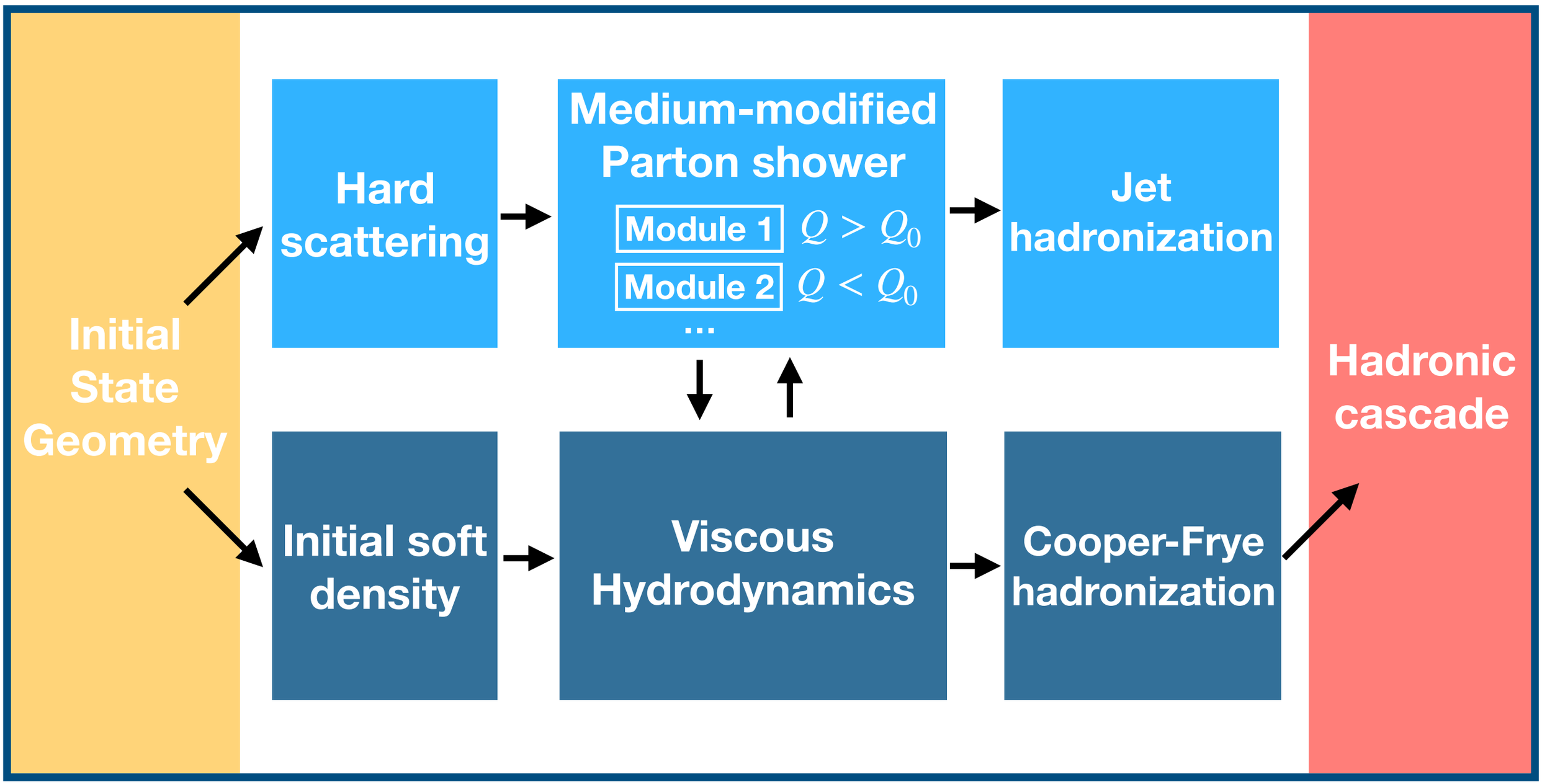
**Docker Pull Command**

```
docker pull jetscape/jetscape_full
```

# JETSCAPE Event Generator

**JETSCAPE Manual: 1903.07706**

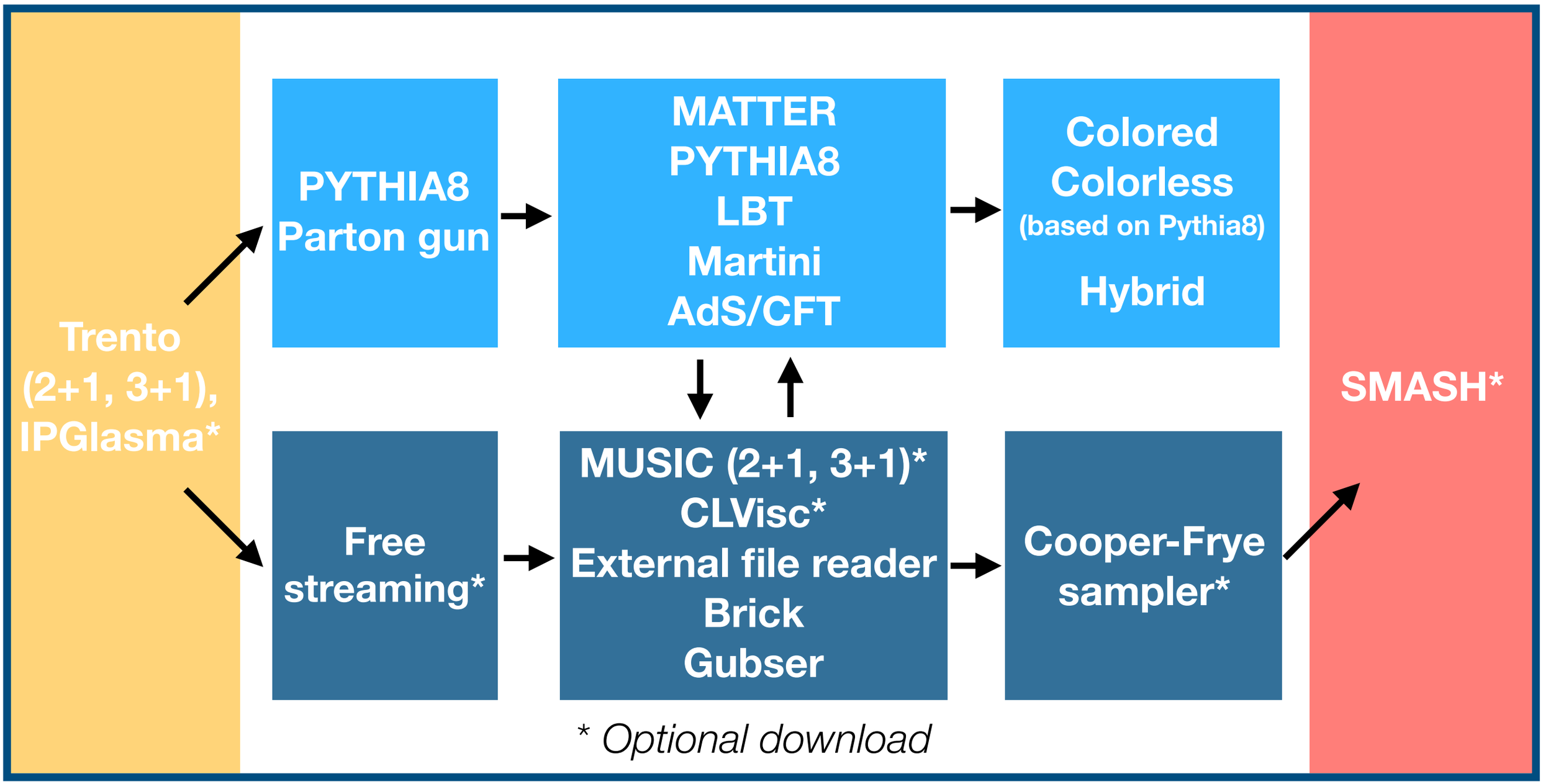
https://github.com/JETSCAPE/X-SCAPE



# JETSCAPE Event Generator

**JETSCAPE Manual: 1903.07706**

https://github.com/JETSCAPE/X-SCAPE



# Steps to run JETSCAPE

1. **Install and build X-SCAPE (JETSCAPE)**
2. **Create a configuration file: *config/jetscape_user.xml***
   - **List of which modules to run, and which model parameters to use**
   - See examples in: *config/*
3. **Execute *runJetscape***

***That's it!***

# Configuring JETSCAPE

**JETSCAPE is configured via two XML files**

- **Main XML file — *you don't modify this***
  - Contains default values for every possible module and parameter
- **User XML file — *you provide this***
  - List of which modules to run, and which default parameter values to override

# Configuring JETSCAPE

**Main XML file**

*you don't modify this*

**A "database" of all possible modules and parameters**

*All possible initial state module parameters*

...

config/jetscape_master.xml

```
<jetscape>

 <!-- General settings -->
 <nEvents> 100 </nEvents>
 <setReuseHydro> true </setReuseHydro>
 <nReuseHydro> 10 </nReuseHydro>

 <!-- Technical settings -->
 <debug> on </debug>
 <remark> off </remark>
 <vlevel> 0 </vlevel>
 <enableAutomaticTaskListDetermination> true </enableAutomaticTaskListDetermination>

 <!--  JetScape Writer Settings -->
 <outputFilename>test_out</outputFilename>
 <JetScapeWriterAscii> off </JetScapeWriterAscii>
 <JetScapeWriterAsciiGZ> off </JetScapeWriterAsciiGZ>
 <JetScapeWriterHepMC> off </JetScapeWriterHepMC>

 <!--  Random Settings. For now, just a global  seed. -->
 <Random>
   <seed>1</seed>
 </Random>

 <!-- Inital State Module  -->
 <IS>
   <!-- x range [-grid_max_x, grid_max_x] -->
   <!-- y range [-grid_max_y, grid_max_y]-->
   <!-- longitudinal range [-grid_max_z, grid_max_z]-->
   <!-- in units of [fm] -->
   <grid_max_x> 10 </grid_max_x>
   <grid_max_y> 10 </grid_max_y>
   <grid_max_z> 0 </grid_max_z>
   <grid_step_x> 0.2 </grid_step_x>
   <grid_step_y> 0.2 </grid_step_y>
   <grid_step_z> 0.2 </grid_step_z>

   <Trento use_module="pre_defined">
     <!-- pre-defined system: default collisions have auau200, pbpb2760, pbpb5020, more in
          the future -->
     <pre_defined collision_system="auau200" centrality_min="30" centrality_max="40" />
     <!-- user-defined system: to get one event in 0-100% centrality range -->
     <user_defined projectile="Au" target="Au" sqrts="200" cross_section="4.2" />
   </Trento>

   <!-- Options to read initial conditions from saved file  -->
   <initial_profile_path>../examples/test_hydro_files</initial_profile_path>
 </IS>
   ...
```

*All possible basic settings*

# Configuring JETSCAPE

**User XML file**

*you provide this*

**Specify which modules to run**

**Specify parameter values (otherwise taken from master)**

See examples in: config/

*Set nEvents*

*Set Writer*

*Activate modules (in order)*

*Override values*

```
<jetscape>

  <nEvents> 200 </nEvents>

  <!-- Jetscape Writer -->
  <JetScapeWriterAscii> on </JetScapeWriterAscii>

  <!-- Inital State Module  -->
  <IS>
    <Trento use_module="user_defined"> </Trento>
  </IS>

  <!-- Hard Process -->
  <Hard>
    <PGun> </PGun>
  </Hard>

  <!-- Hydro  Module  -->
  <Hydro>
    <MUSIC> </MUSIC>
  </Hydro>

  <!--Eloss Modules -->
  <Eloss>
    <Matter>
      <Q0> 2.0 </Q0>
      <qhat0> 5.0 </qhat0>
    </Matter>
    <AdSCFT> </AdSCFT>
  </Eloss>

  <!-- Jet Hadronization Module -->
  <JetHadronization>
    <name>colored</name>
  </JetHadronization>

</jetscape>
```

# Running JETSCAPE

**There is one central executable*: runJetscape.cc**

- Modules are automatically added according to User XML
- You don’t ever need to re-compile this executable

**Pass your user configuration file as a command line argument:**

```
./runJetscape /path/to/my/user_config.xml
```

* For integration in other frameworks/different usage you can of course also write your own executable see in GitHub ./examples/costum_examples

# JETSCAPE/X-SCAPE Output

**JETSCAPE output contains:**

Final state hadrons

Final state partons

Full parton-shower history

**You can produce JETSCAPE output in two formats*:**

**Ascii**

Custom JETSCAPE format

Option to write (i) parton shower history, or (ii) only final-state particles

Can also directly write gzipped ascii

**HepMC3**

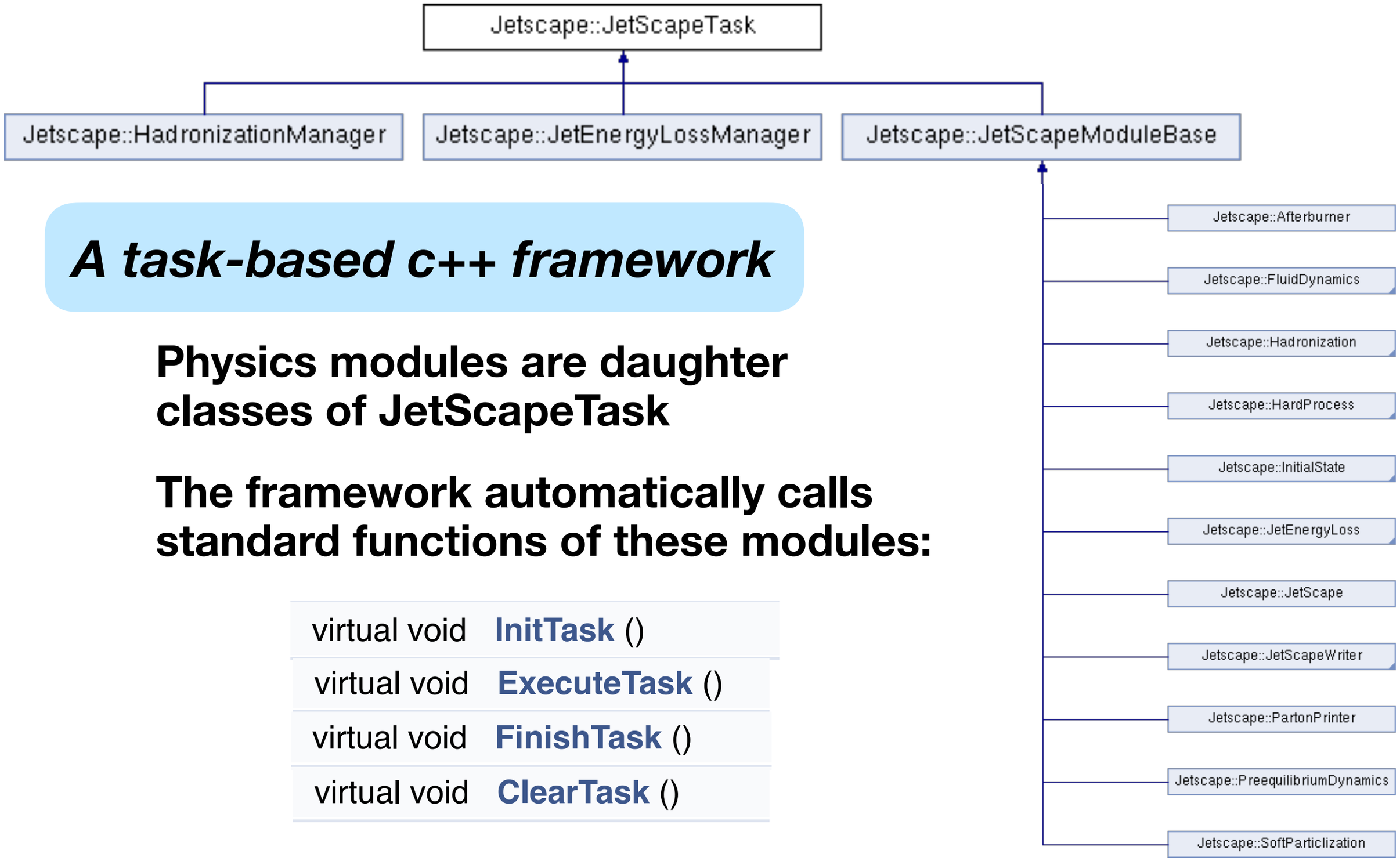
Standard event format Ascii or ROOT format supported

Compatible with Rivet

* You can easily write your own output class to tailor to your specific needs by inheriting from the JetScapeWriter

# Framework design: Inner workings

*arXiv 1903.07706*




***A task-based c++ framework***

**Physics modules are daughter classes of JetScapeTask**

**The framework automatically calls standard functions of these modules:**

| | |
|---|---|
| virtual void | **InitTask** () |
| virtual void | **ExecuteTask** () |
| virtual void | **FinishTask** () |
| virtual void | **ClearTask** () |

# Framework design: Inner workings

*arXiv 1903.07706*


**There is one significant API change between JETSCAPE and X-SCAPE (source codes on GitHub). It does not affect running, (X-SCAPE is fully backward compatible) but it is important if you develop your own new module:**

| JETSCAPE | X-SCAPE |
|---|---|
| virtual void **Init** () | virtual void **InitTask** () |
| virtual void **Exec** () | virtual void **ExecuteTask** () |
| virtual void **Finish** () | virtual void **FinishTask** () |
| virtual void **Clear** () | virtual void **ClearTask** () |

# Framework design: Inner workings

*arXiv 1903.07706*

**The framework defines how different types of modules can interact with each other**

For example: Jet energy loss module needs access to hydro info

**This is implemented in a "signal-slot" paradigm*:**

| Module 1 | Module 2 | Signal | Slot |
|---|---|---|---|
| `JetEnergyLossManager` | `HardProcess` | `GetHardPartonList()` | `GetHardPartonList()` |
| `JetEnergyLoss` | `FluidDynamics` | `jetSignal()` | `UpdateEnergyDeposit()` |
| `JetEnergyLoss` | `FluidDynamics` | `edensitySignal()` | `GetEnergyDensity()` |
| `JetEnergyLoss` | `FluidDynamics` | `GetHydroCellSignal()` | `GetHydroCell()` |
| `JetEnergyLoss` | `JetEnergyLoss` | `SentInPartons()` | `DoEnergyLoss()` |
| `Hadronization` | `Hadronization` | `TransformPartons()` | `DoHadronization()` |
| `HadronizationManager` | `HardProcess` | `GetHadronList()` | `GetHadronList()` |
| `HadronizationManager` | `JetEnergyLoss` | `GetFinalPartonList()` | `SendFinalStatePartons()` |

Table 3: The list of all connection methods between JETSCAPE modules provide by the `JetScapeSignalManager`.

* Introduced by the QT framework

# Framework design: Inner workings

arXiv 1903.07706

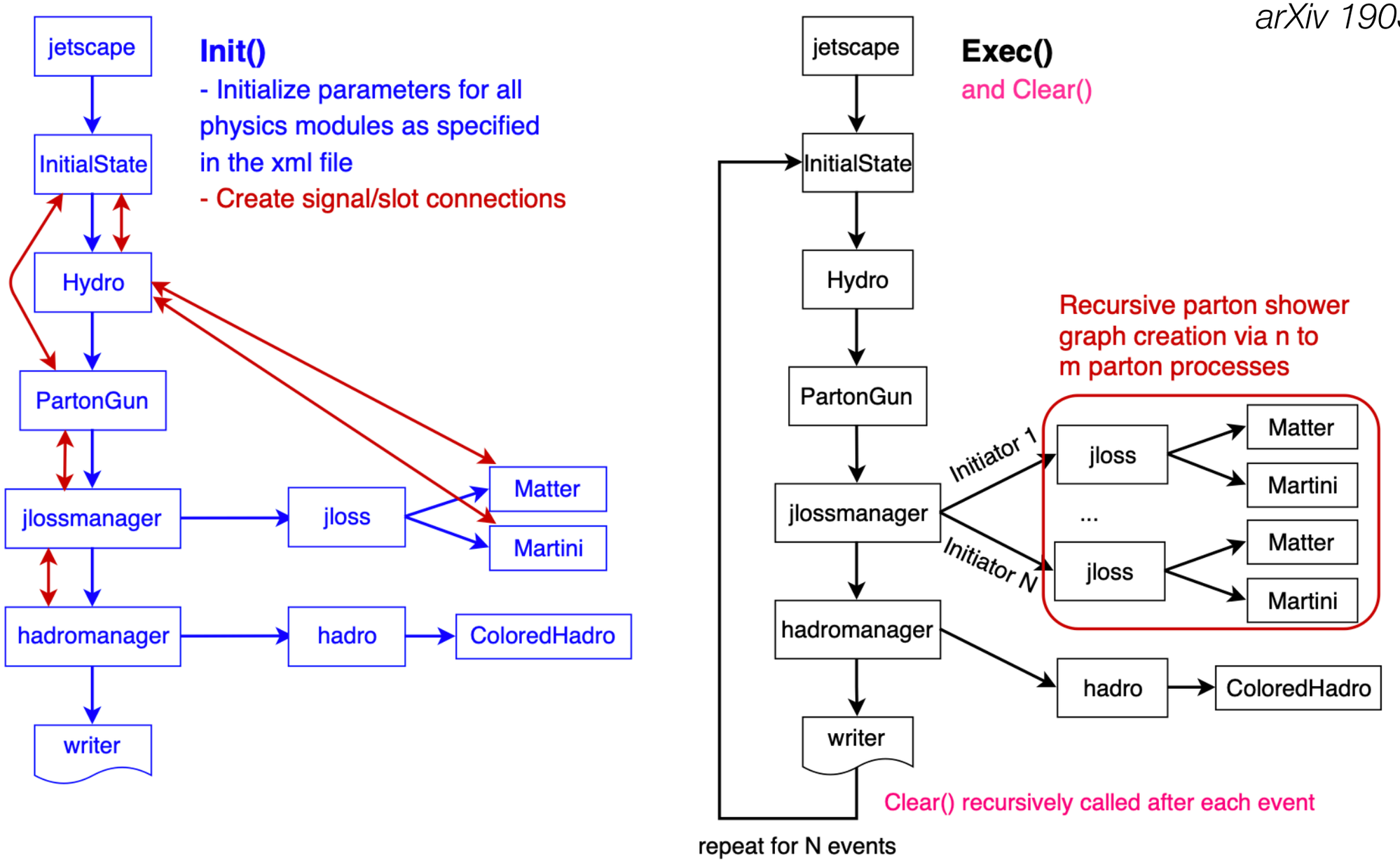


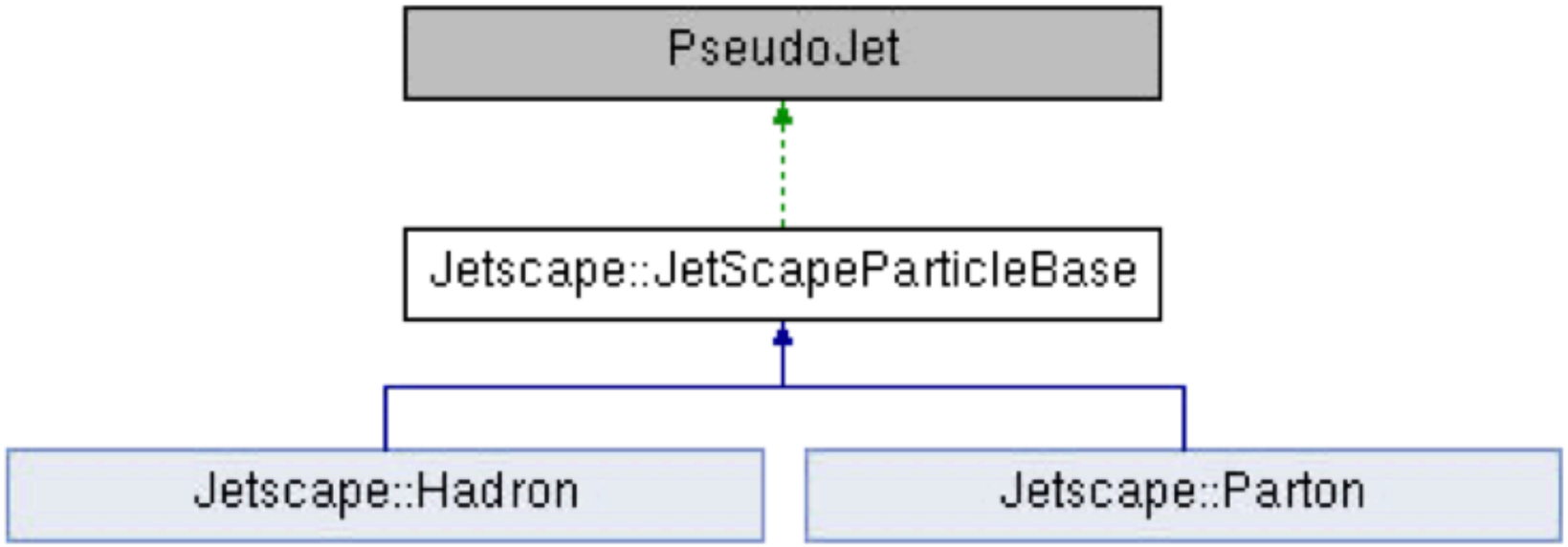

Figure 4: Example workflow of the `Init()` (left side) and `Exec()` (and `Clear()`) (right side) phase of the task based JETSCAPE framework (the not extensively used Finish() phase is omitted). One should be aware that the created signal/slot connections in the `Init()` phase are of course present and utilized in the `Exec()` phase, but are not show in the figure for simplicity.

# Data structures

*arXiv 1903.07706*

Class **JetScapeParticleBase**

- The base class for all the JETSCAPE particles
- Privately inherits from FastJet PseudoJet and has
  - PID and rest mass
  - A location (4-vector)
  - Label and status
- Derived classes so far:
  - Parton and Hadron

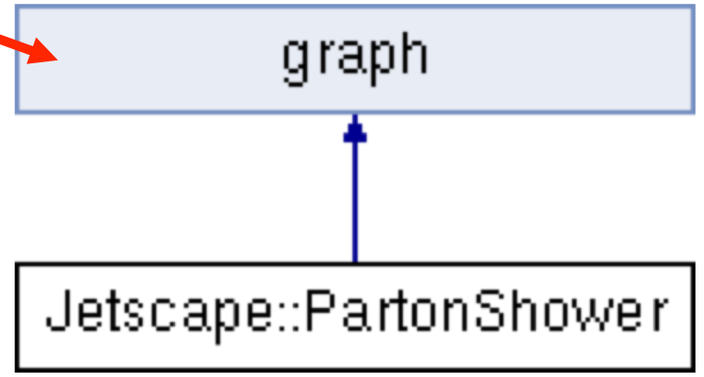



```
Parton (int label, int id, int stat, const FourVector& p, const FourVector& x);
Parton (int label, int id, int stat, double pt, double eta, double phi, double e, double* x=0);
...
```

In addition: Photon class (inherits from Parton)

# Data structures

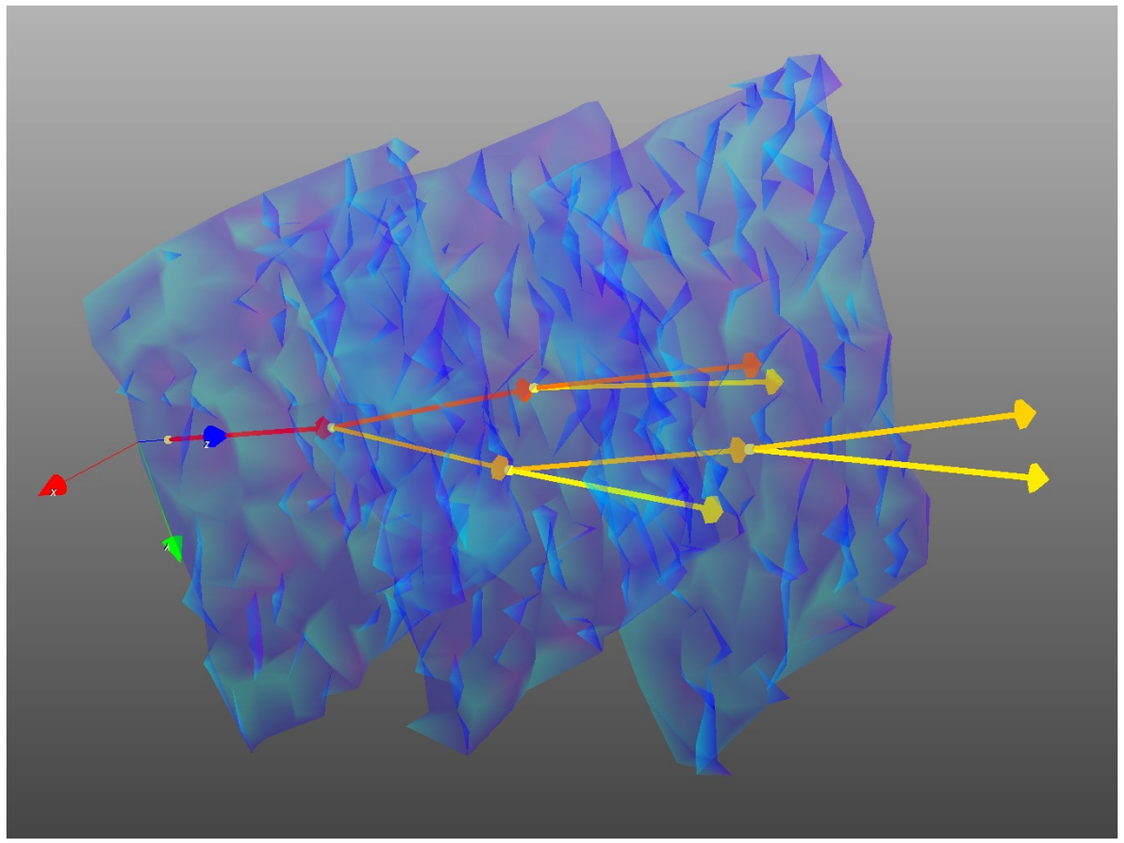
**GTL Graph Template Library**
[https://github.com/rdmpage/graph-template-library]

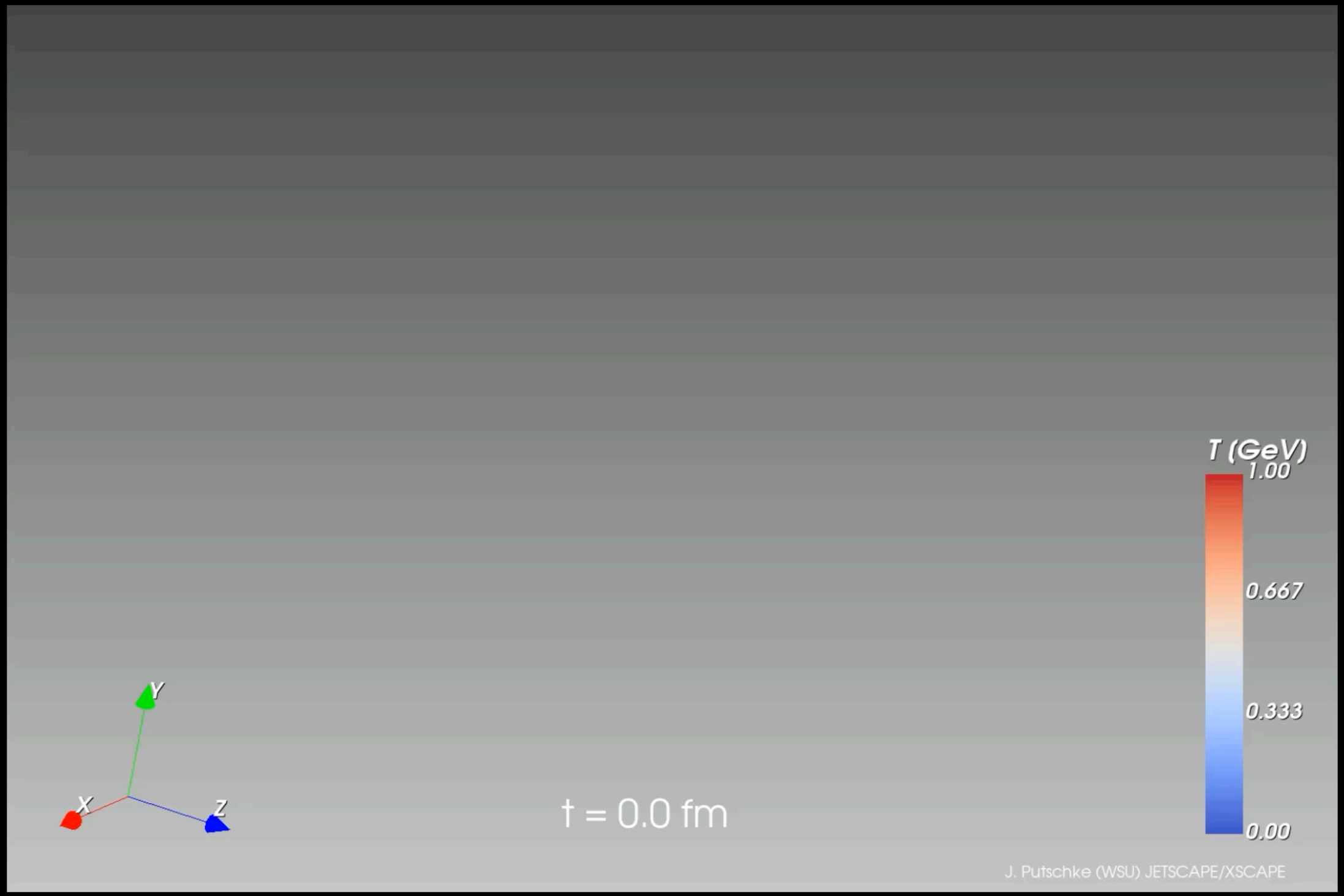


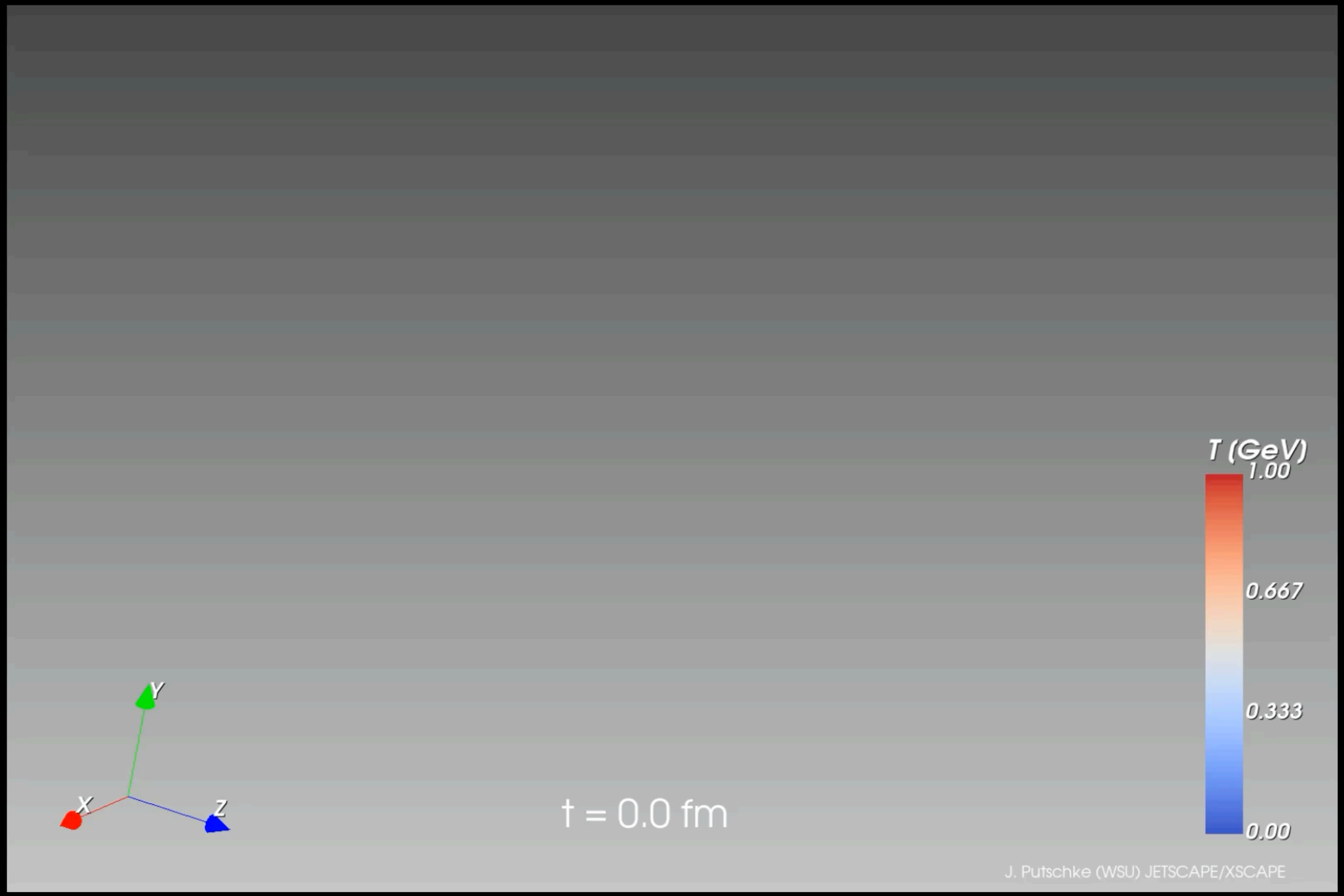

- Class **PartonShower**
- Models parton showering as a **Graph**
  - Partons are the **edges**
  - Partons split at **vertices**

| | |
|---|---|
| int | **GetNumberOfParents** (int n) |
| int | **GetNumberOfChilds** (int n) |
| ared_ptr< **Parton** > > | **GetFinalPartons** () |
| < fjcore::PseudoJet > | **GetFinalPartonsForFastJet** () |
| int | **GetNumberOfPartons** () const |
| int | **GetNumberOfVertices** () const |

**Provides functionalities to query shower**




T (GeV)
1.00
0.667
0.333
0.00
Y
X
Z
t = 0.0 fm
J. Putschke (WSU) JETSCAPE/XSCAPE


T (GeV)
1.00
0.667
0.333
0.00
Y
X
Z
t = 0.0 fm
J. Putschke (WSU) JETSCAPE/XSCAPE

# Contributing modules

*arXiv 1903.07706*

**JETSCAPE/X-SCAPE is designed for users to contribute new physics modules**

Framework connects those modules together

**To contribute modules, just need to interface to JETSCAPE:**

Implement appropriate standard functions:

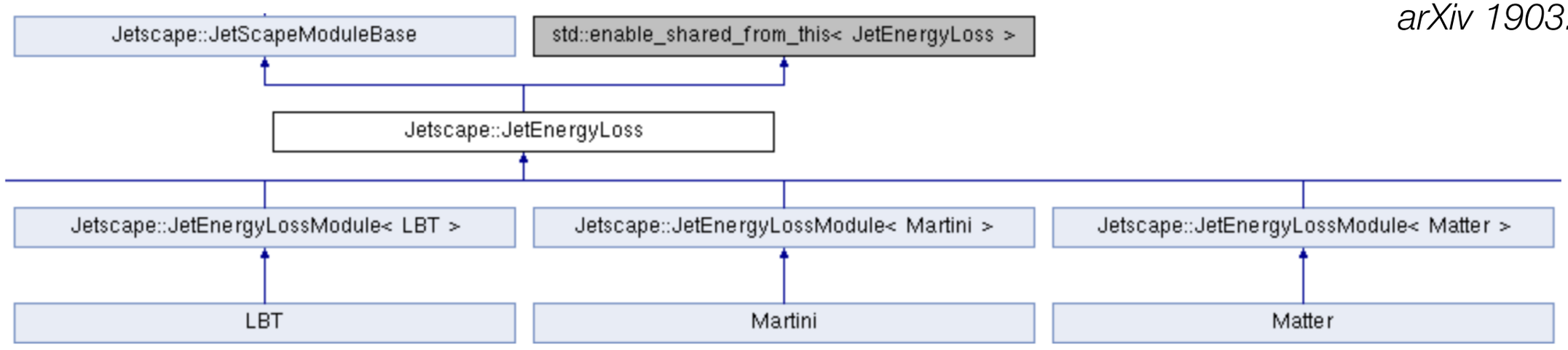
| | |
|---|---|
| virtual void | **InitTask** () |
| virtual void | **ExecuteTask** () |
| virtual void | **FinishTask** () |
| virtual void | **ClearTask** () |

Use appropriate signal/slot info to interact with other modules

See existing modules for examples

# Example: Jet energy loss

*arXiv 1903.07706*




- User's code must be a subclass of **JetEnergyLossModule**

- User's code must override **init()** method for initializations

- User's code must override **DoEnergyLoss()** method

  - The actual energy loss calculations happen in this method

**Note:** Jet energy loss modules are "special" in that DoEnergyLoss() is called by the framework **per-parton**. Most modules are called **per-event** with Exec()

# XML Reader

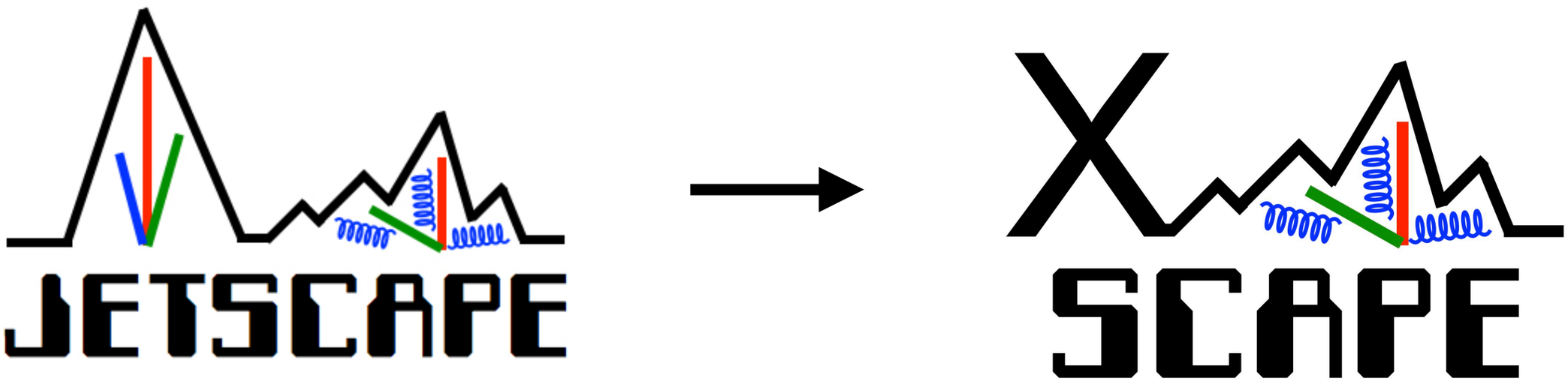
*arXiv 1903.07706*

**An XML singleton class is provided to allow easy initialization of your module parameters from the XML files**

- **JetScapeXML** provides functionality to first examine the User file for a given parameter, and if it is not found, it takes the value from the Master file.

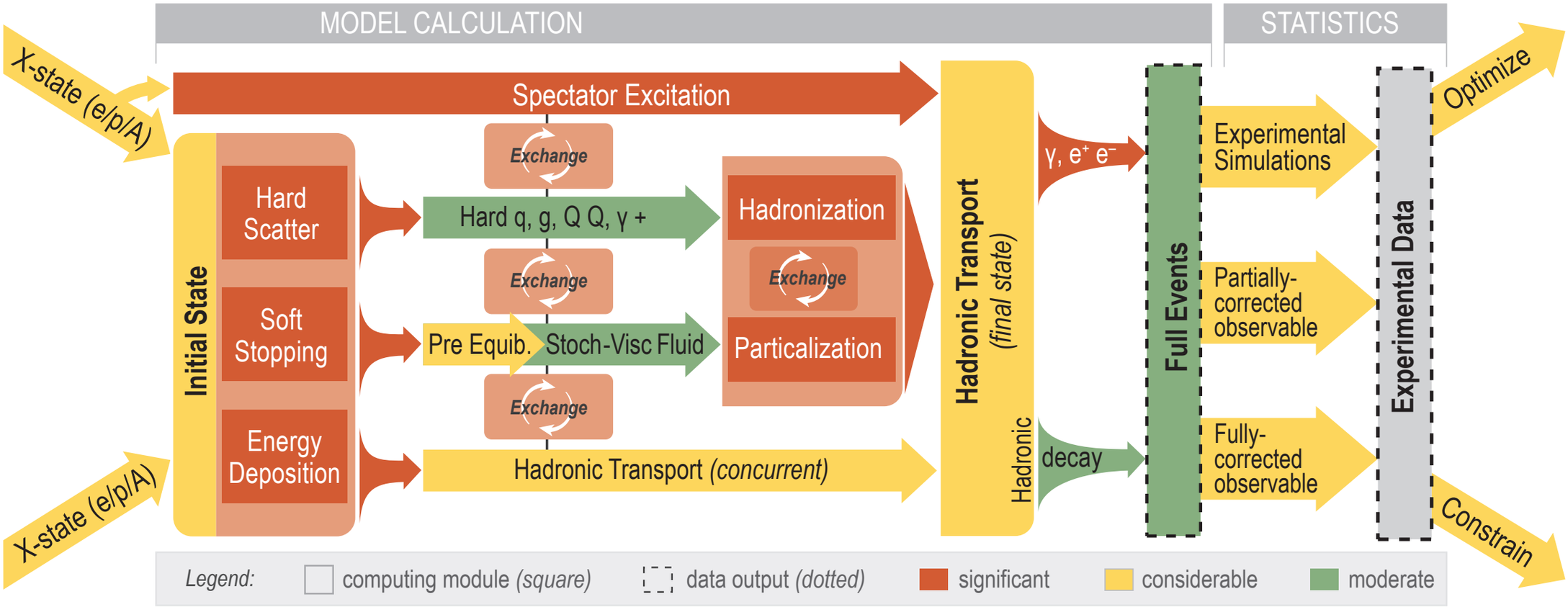
- To init a parameter, call ***GetElementDouble*({"Eloss", "Martini", "x"})**
  - No need to keep track of XML elements in modules! Just call the function!
  - Similar functions ***GetElementText*, *GetElementInt*, *GetElement***

- An optional second argument in these functions allows the parameter to be optional in the XML file (by default, a parameter is required to be present or else the program will crash when it is not found).




Remark: X-SCAPE 1.0 is not fully steered via XML, you have use a separate executable (see documentation for details). Will be changed in upcoming releases.

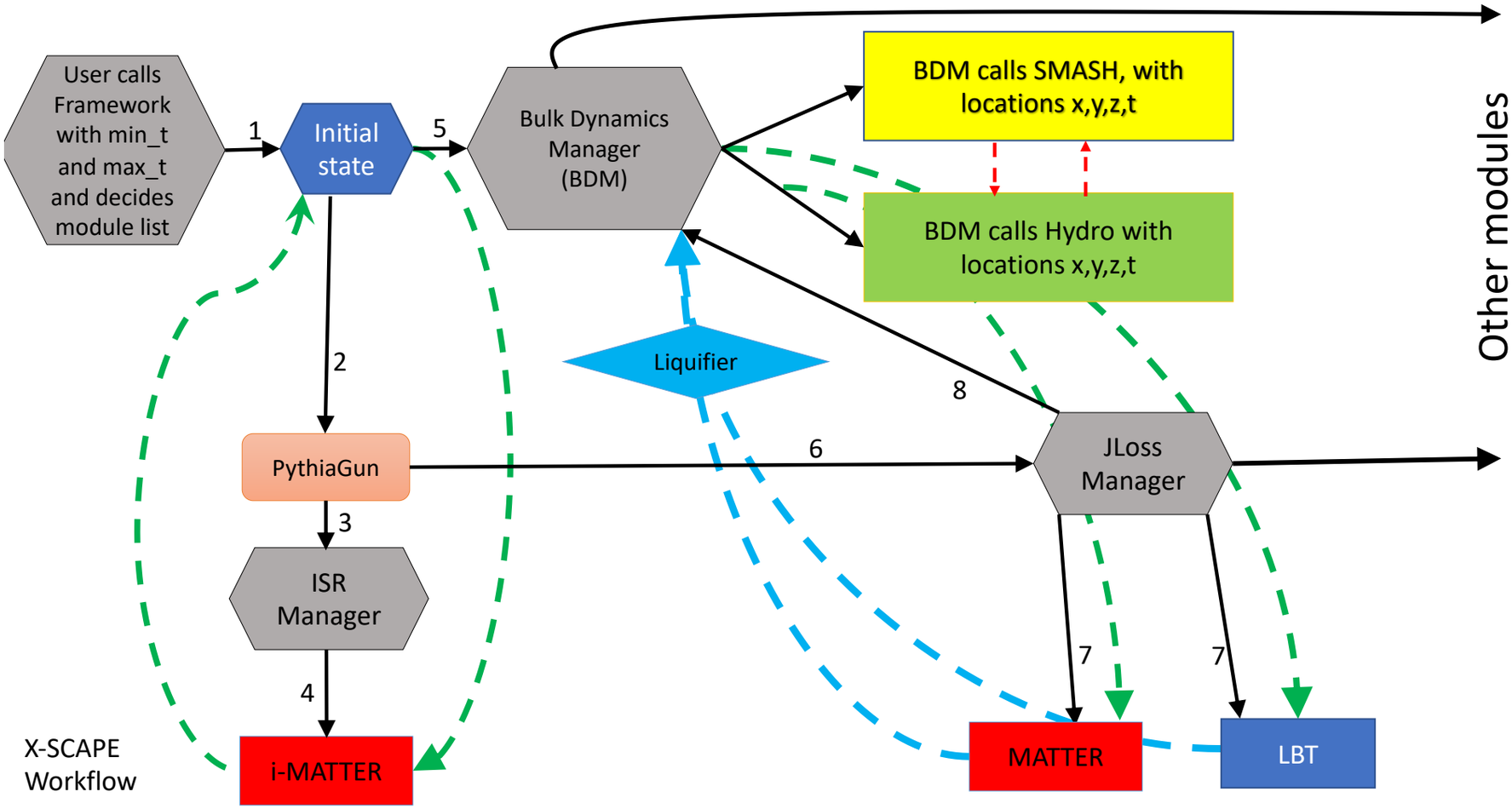
# X-SCAPE: Extended Physics Scope




**XSCAPE framework goal: To provide a *decentralized and synchronized* framework, which will allow any user to attach his/her own modules and reorganize the flow of data between the modules with the goal to simulate:**

1) To simulate *p*-*A* and *p*-*p* collisions at arbitrary multiplicity (X-SCAPE 1.0)
2) Any aspect of *A*-*A* collisions from FAIR to top LHC energy (X-SCAPE 2.0)
3) To study $e^+e^-$and certain aspects of *e*-*A* collisions (X-SCAPE 3.0)

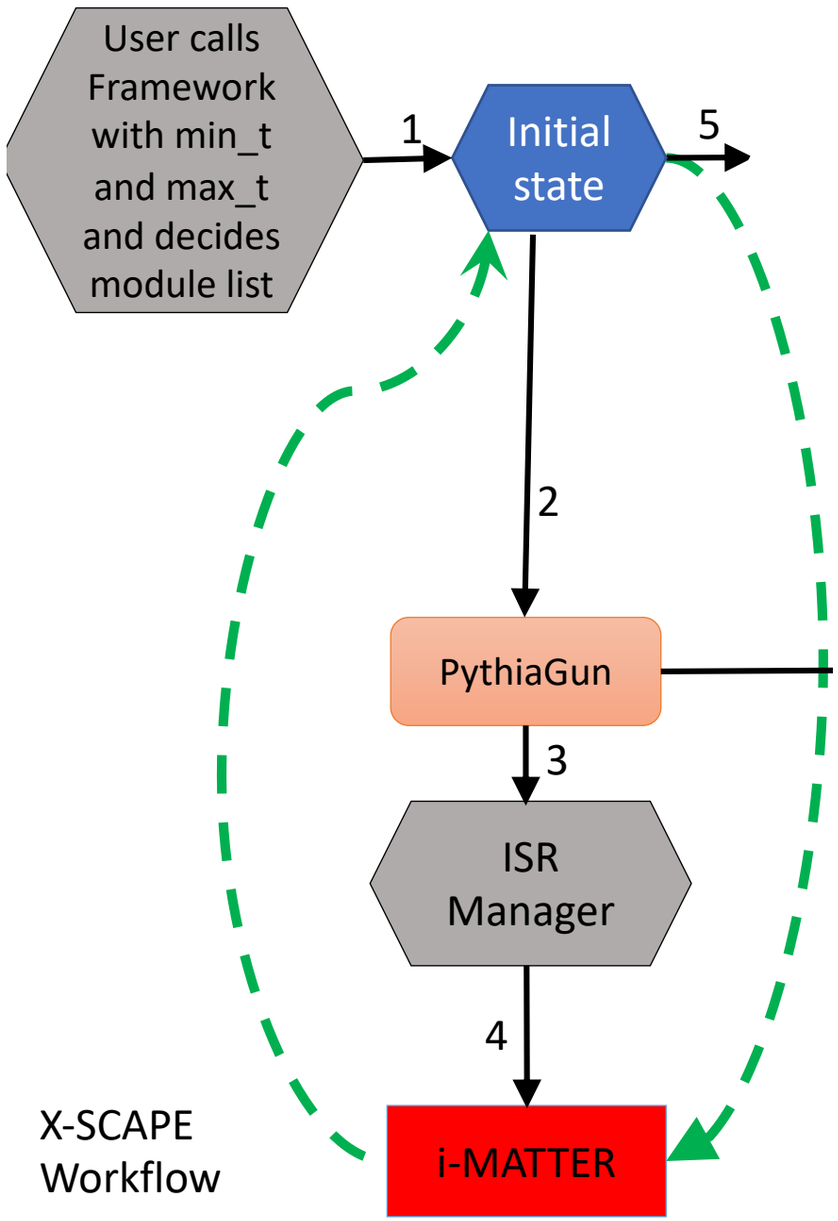
# X-SCAPE: Extended Physics Scope




**XSCAPE framework goal: To provide a *decentralized and synchronized* framework, which will allow any user to attach his/her own modules and reorganize the flow of data between the modules with the goal to simulate:**

1) To simulate *p*-*A* and *p*-*p* collisions at arbitrary multiplicity (X-SCAPE 1.0)
2) Any aspect of *A*-*A* collisions from FAIR to top LHC energy (X-SCAPE 2.0)

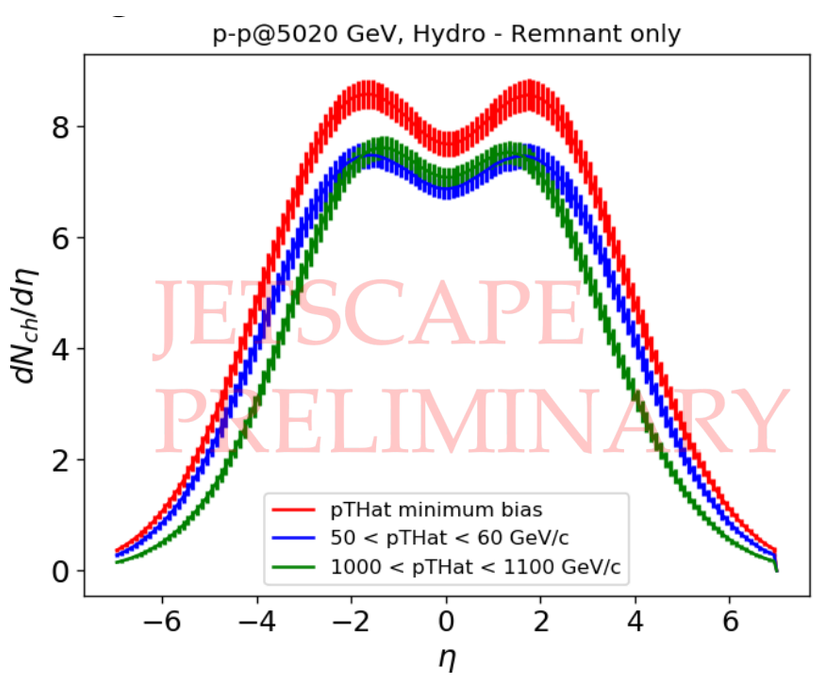
# X-SCAPE 1.0: Small Systems




X-SCAPE Workflow

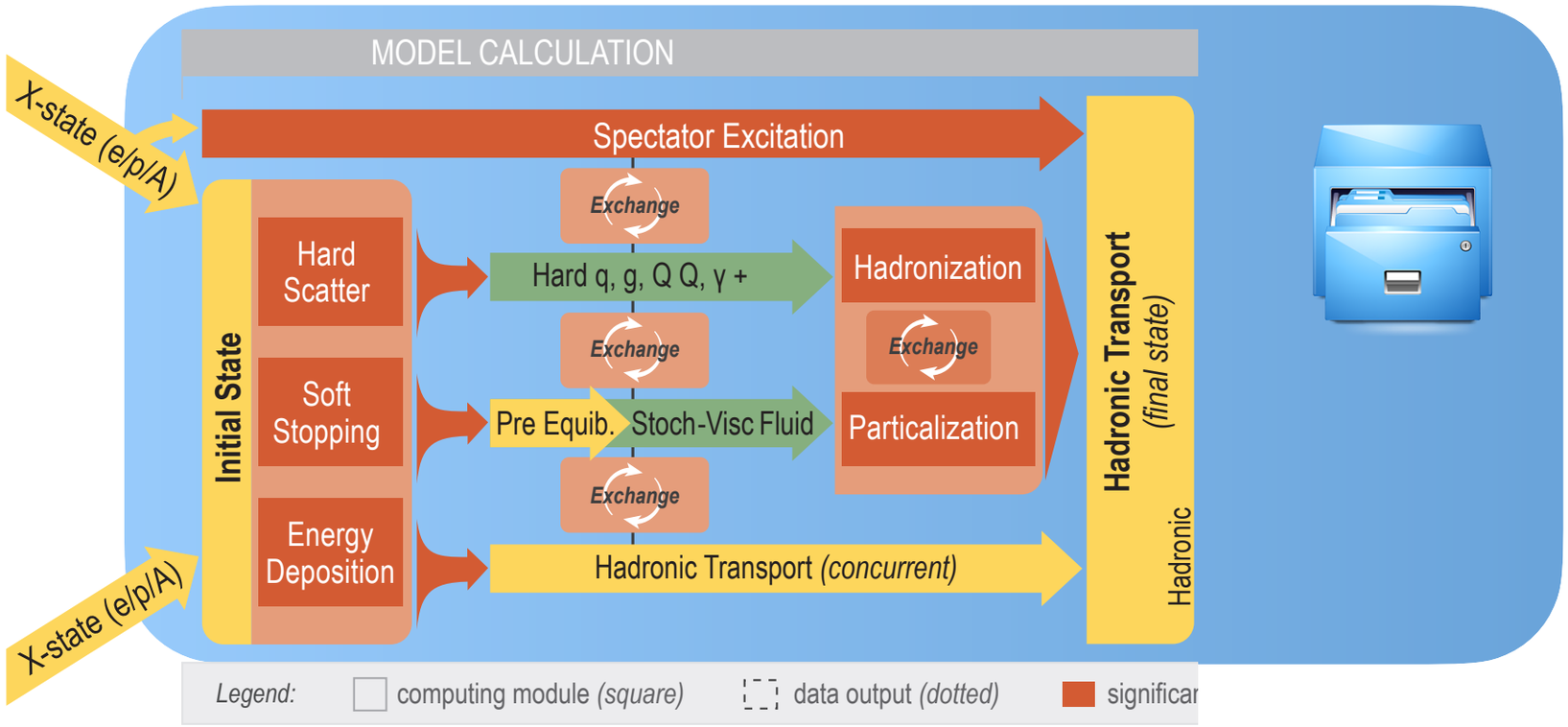
Provide new **ISR Manager** module, inherited from the JetEnergyLossManager, but evolves "backward" in time* to simulate the initial state radiation and allows exchange/interaction with the InitialState module (3dGlauber for example).

***Hard energy removed from nucleons, not available for hydro evolution! (Hard Probes 2023)***




*Remark: For now ISR Manager and backward time evolution is utilized for simplicity on a per-event basis (clock and per time-step evolution is working, but not utilized as default)

# Simplified/more flexible data exchange




“Database” query approach using `any(variant)` datatype via **QueryHistory instance**

**JetScapeModuleBase** ()

**JetScapeModuleBase** (string m_name)

virtual any **GetHistory** ()

Implement `GetHistory()` in Jetscape Module(s). Can hold “`any`” datatype, in particular allows non framework datatypes*!

*Caveat: Since an `any` datatype has to explicitly be casted, this new flexible data exchange might break the stringent API and only framework datatype approach of “every module works with every other module” of JETSCAPE. Might change in the future, using `variant` or might be defunct.

# Simplified/more flexible data exchange

## Jetscape::QueryHistory Class Reference

`#include <QueryHistory.h>`

### Public Member Functions

| | |
|---|---|
| void | **AddMainTask** (std::shared_ptr< **JetScapeTask** > m_main_task) |
| void | **UpdateTaskMap** () |
| void | **PrintTasks** () |
| void | **PrintTaskMap** () |
| std::unordered_multimap< std::string, std::weak_ptr< **JetScapeTask** > > | **GetTaskMap** () |
| any | **GetHistoryFromModule** (string mName) |
| vector< any > | **GetHistoryFromModules** (string mName) |

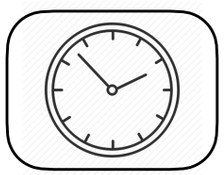
### Static Public Member Functions

| | |
|---|---|
| static **QueryHistory** * | **Instance** () |

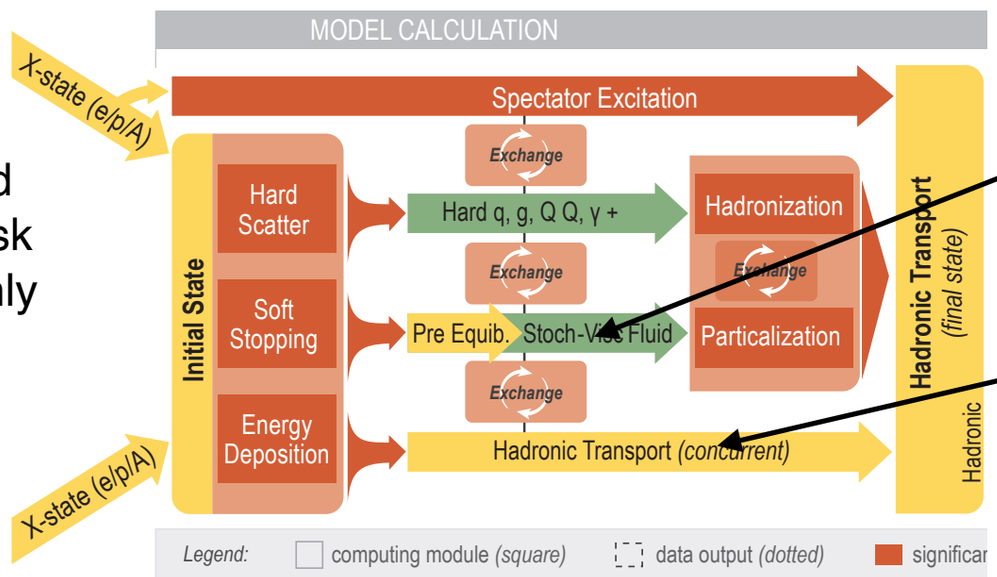
`QueryHistory` Instance queries tasks via `string` identifier and retrieves information from `GetHistory()` if implemented.

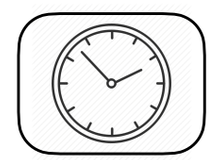
```
…
vector<any> eLossHistories =
QueryHistory::Instance()->
GetHistoryFromModules
(“JetEnergyLoss”);
…
```

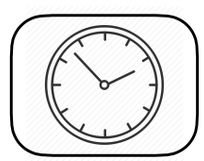
# Clock(s) in X-SCAPE …



**Main Clock** attached to JetScape main task and there “can be only one”



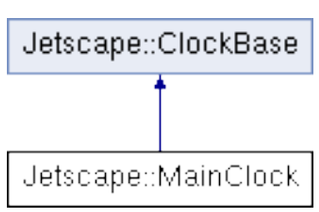





…

**Module Clock(s)** can be attached to any JetScape task

They are not independent clocks, but rather provide the “transformation” wrt the main clock time frame




**JetScapeModuleBase** ()

**JetScapeModuleBase** (string m_name)

bool **IsTimeStepped** () const
Returns whether the module evolves in time steps. More...

void **SetTimeStepped** (bool m_time_stepped)
Sets whether the module evolves in time steps. More...

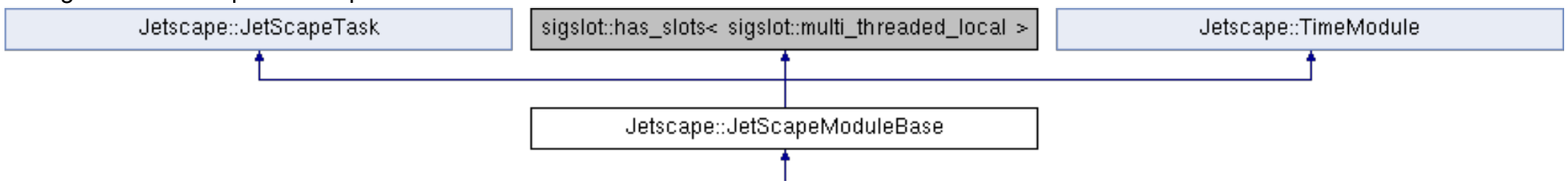
Remark: Has to be set explicitly to `true`, default is `false`, so per event “JETSCAPE” like execution.

If a main clock is attached to the main task, physics modules can be executed (no module clock needs to be provided, by default main clock will be used) on a per time-step basis! In general, one can mix per event and per time-step, if it makes sense from a physics perspective.

# Attaching Clock(s) to Modules …

```
#include <JetScapeModuleBase.h>
```

Inheritance diagram for Jetscape::JetScapeModuleBase:




## Public Member Functions

| | |
|---|---|
| | **TimeModule** () |
| | **TimeModule** (double t1, double t2) |
| virtual | **~TimeModule** () |
| void | **ClockInfo** () |
| void | **AddModuleClock** (shared_ptr< **ModuleClock** > m_mClock) |
| shared_ptr< **ModuleClock** > | **GetModuleClock** () const |
| void | **AddMainClock** (shared_ptr< **MainClock** > m_mainClock) |
| bool | **UseModuleClock** () |
| double | **GetModuleCurrentTime** () |
| double | **GetModuleDeltaT** () |
| bool | **IsValidModuleTime** () |
| void | **SetTimeRange** (double t1, double t2) |
| const double | **GetTStart** () const |
| const double | **GetTEnd** () const |

```
…
auto mClock =
make_shared<MainClock>("SpaceTime",
-0.1,0.1,0.1);

auto mMilneClock =
make_shared<MilneClock>();
mMilneClock->setEtaMax(5.0);
…
auto jetscape =
make_shared<JetScape>();
jetscape->AddMainClock(mClock);
…
auto hydro = make_shared<Brick> ();
hydro->SetTimeRange(0.6,10);
hydro->AddModuleClock(mMilneClock);

hydro->SetTimeStepped(true);
…
```

# Concurrent running of Modules per time-step ...

| | |
|---|---|
| | **JetScapeModuleBase** () |
| | **JetScapeModuleBase** (string m_name) |

| | |
|---|---|
| bool | **IsTimeStepped** () const<br>Returns whether the module evolves in time steps. More... |
| void | **SetTimeStepped** (bool m_time_stepped)<br>Sets whether the module evolves in time steps. More... |

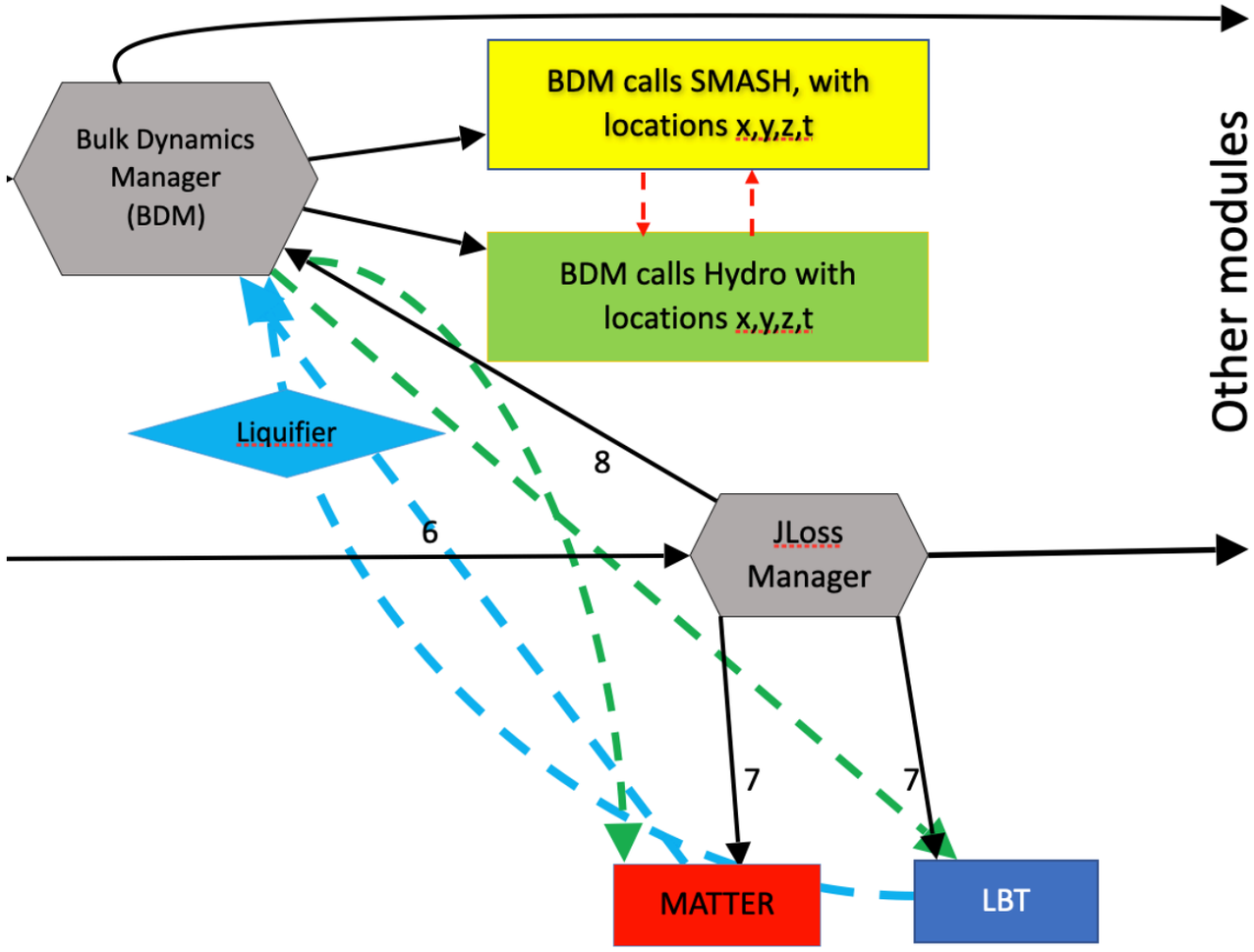
| | |
|---|---|
| virtual void | **CalculateTime** () |
| virtual void | **CalculateTimeTasks** () |
| virtual void | **CalculateTimeTask** () |
| virtual void | **ExecTime** () |
| virtual void | **ExecTimeTasks** () |
| virtual void | **ExecTimeTask** () |
| virtual void | **InitPerEvent** () |
| virtual void | **InitPerEventTasks** () |
| virtual void | **FinishPerEvent** () |
| virtual void | **FinishPerEventTasks** () |

`JetScapeModuleBase` provides an additional per time-step workflow:

1) `CalculateTime()` phase: Module(s) simulate the next time-step. No communication between modules -> can be trivially multi-threaded if needed.
2) `ExecTime()` phase: Communication/data exchange etc. between modules needed before next time-step

Remark: If you want to develop modules which can be executed per time-step you have to provide these functions, analogous to `InitTask()`, `ExecTask()`,... in the JETSCAPE workflow.

# X-SCAPE 2.0: Low energy AA …




**BulkDynamicsManager (BDM)**:

Organizes/allows concurrent running of multiple QCD bulk media implementations!

Initially for X-SCAPE 2.0 (release expected end of this year) hydro (MUSIC) and hadron cascade (SMASH), focus on low beam energy AA collisions!

First proof-of-principle implementation of the BDM and other example/test programs utilizing the new X-SCAPE  per time-step workflow can be found in the custom example directory!

# Summary

**JETSCAPE/X-SCAPE is a framework for general-purpose e-A, p/d-A and A-A event generators**

- Modular, extensible — please contribute modules!

**JETSCAPE/X-SCAPE is a tool for the community**

- To enable well-controlled event generator comparisons
- As a testbed for theoretical and experimental development

**JETSCAPE (X-SCAPE) has been successfully used and tested in large scale simulation efforts**

# Remarks on large scale simulations

JETSCAPE has used **tens of millions of core-hours running simulations** for Bayesian analyses (covered next week) which are especially computationally expensive.
When you run simulations at scale, here are some details to consider:

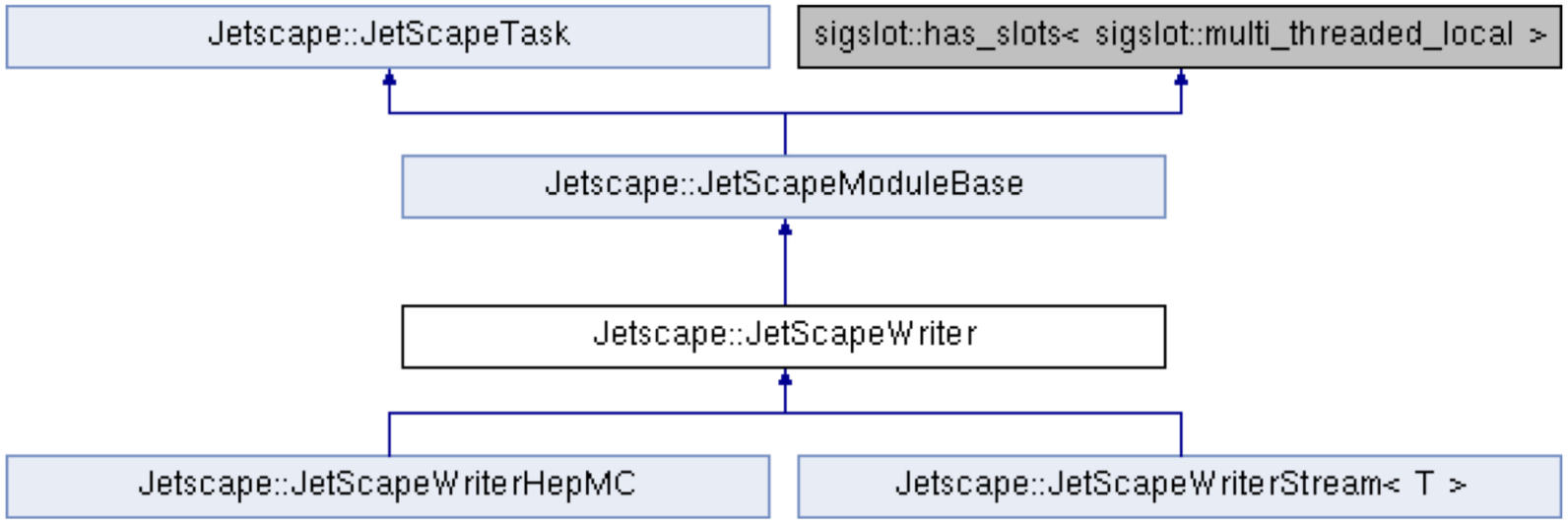
- Containers simplify deployment when running simulations at scale
- Utilize **singularity containers** (see GitHub installation instructions)
  - Most High Performance Computing (HPC) sites will not allow you to use docker containers
  - You can directly convert the docker container into a format suitable for singularity
- Make sure you build your container in a **reproducible** way by explicating setting the versions of dependencies
  - If you take the most recent version, it may change next time you build the container
  - That can change your physics!
- Be aware of HPC facility architecture, capabilities, requirements, and best practices
- Particularly watch out for file **I/O limits**; for example, reading pre-computed hydro profiles for use in jet energy loss calculations can be a high IO activity
- Write output files to the **temporary storage** of the computing node, and then copy only the files you need back to permanent storage; consider calculating observables directly from outputs
- **Optimize** for the type of simulation that you are running:
  - Hydro utilizes multi-core capabilities → Request many cores on a single node
  - Jet energy loss uses a core at time (massively parallel) → Cores can be from any node

# Thank you!

# Backup

# Output

## Class **JetScapeWriter**




Subclass of JetScapeModuleBase

- Can be attached as a task

Derived classes so far:

- HepMC writer
- ASCII writer

**Each task overrides its own write method**

| | |
|---|---|
| virtual void | **FinishTask** () |
| virtual void | **FinishTasks** () |
| virtual void | **WriteTasks** (weak_ptr< **JetScapeWriter** > w) |
| virtual void | **WriteTask** (weak_ptr< **JetScapeWriter** > w) |
| virtual void | **CollectHeader** (weak_ptr< **JetScapeWriter** > w) |
| virtual void | **CollectHeaders** (weak_ptr< **JetScapeWriter** > w) |

# Output

Class **JetScapeReader**

- Base class for reading JETSCAPE output files

Not a JETSCAPE task

- To be used after producing output

Derived classes so far:

- JetScapeReaderAscii

```
void   Next ()
bool   Finished ()
int    GetCurrentEvent ()
int    GetCurrentNumberOfPartonShowers ()
ptr< PartonShower > >   GetPartonShowers ()
hared_ptr< Hadron > >   GetHadrons ()
or< fjcore::PseudoJet >   GetHadronsForFastJet ()
```

**Provides access to PartonShower for final state Partons, and Hadrons**