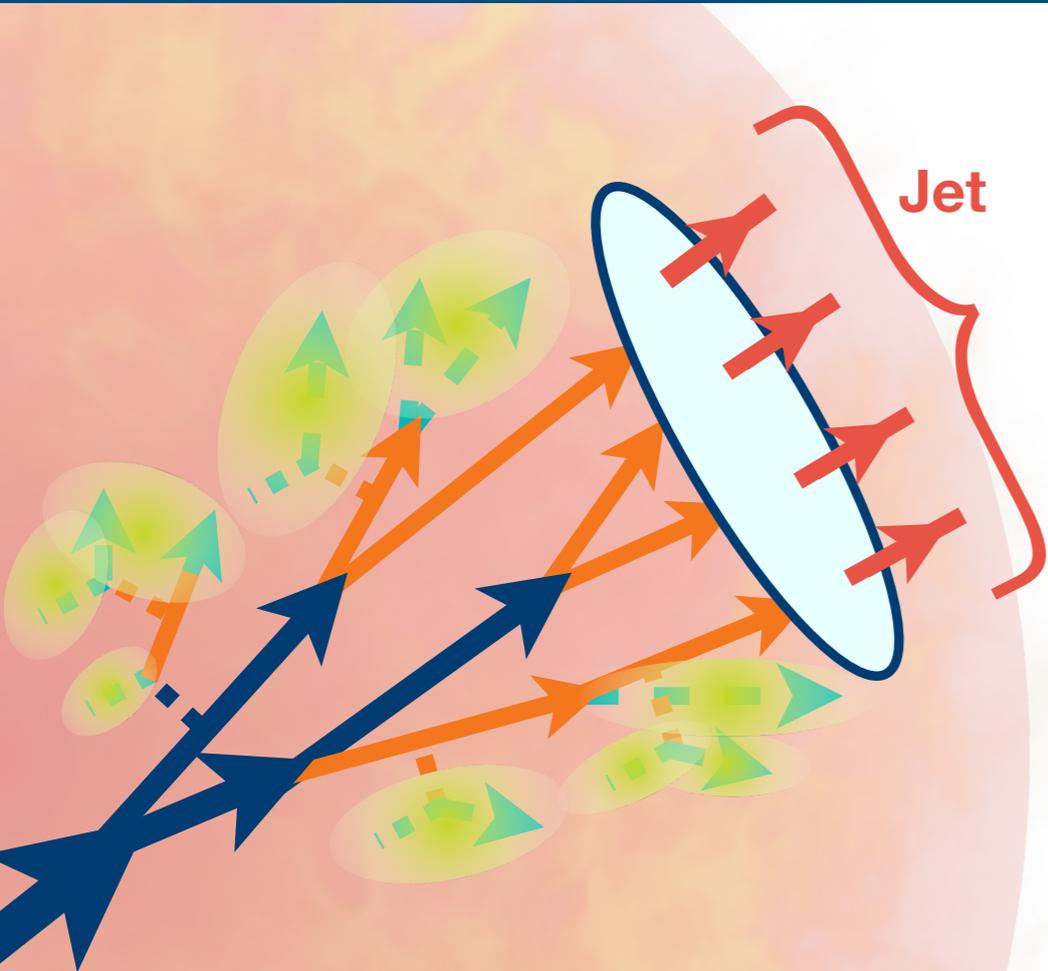


The JETSCAPE Framework



Lecture

Joern Putschke
(Wayne State University)

JETSCAPE Online School
July 25 2022

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

#software-install-problems

Ask questions in slack channel:
#july25-26-framework

Event generator

- A **framework** for general-purpose MC event generators in heavy-ion collisions

<https://github.com/JETSCAPE/JETSCAPE>

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 is not just for jets!
It is a framework for *general-purpose* event generators

1

The JETSCAPE framework is modular

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

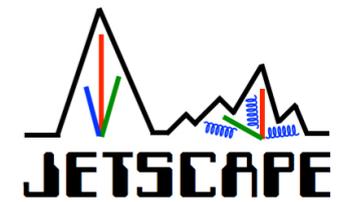
2

Physics modules are open-source

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

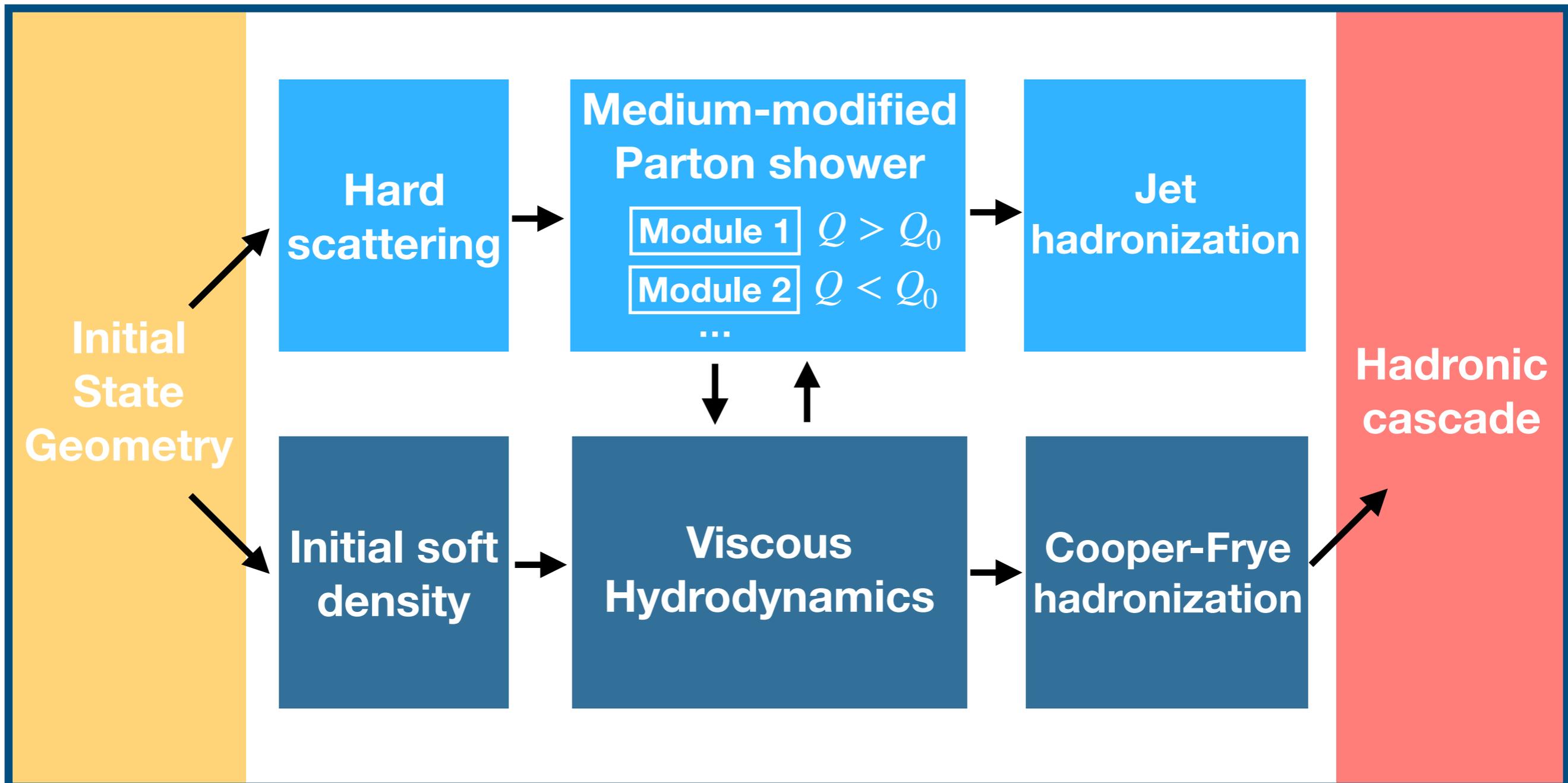
A unified framework has clear benefits when we want to compare models of one particular part of a multi-stage event evolution

JETSCAPE Event Generator



JETSCAPE Manual: 1903.07706

<https://github.com/JETSCAPE/JETSCAPE>

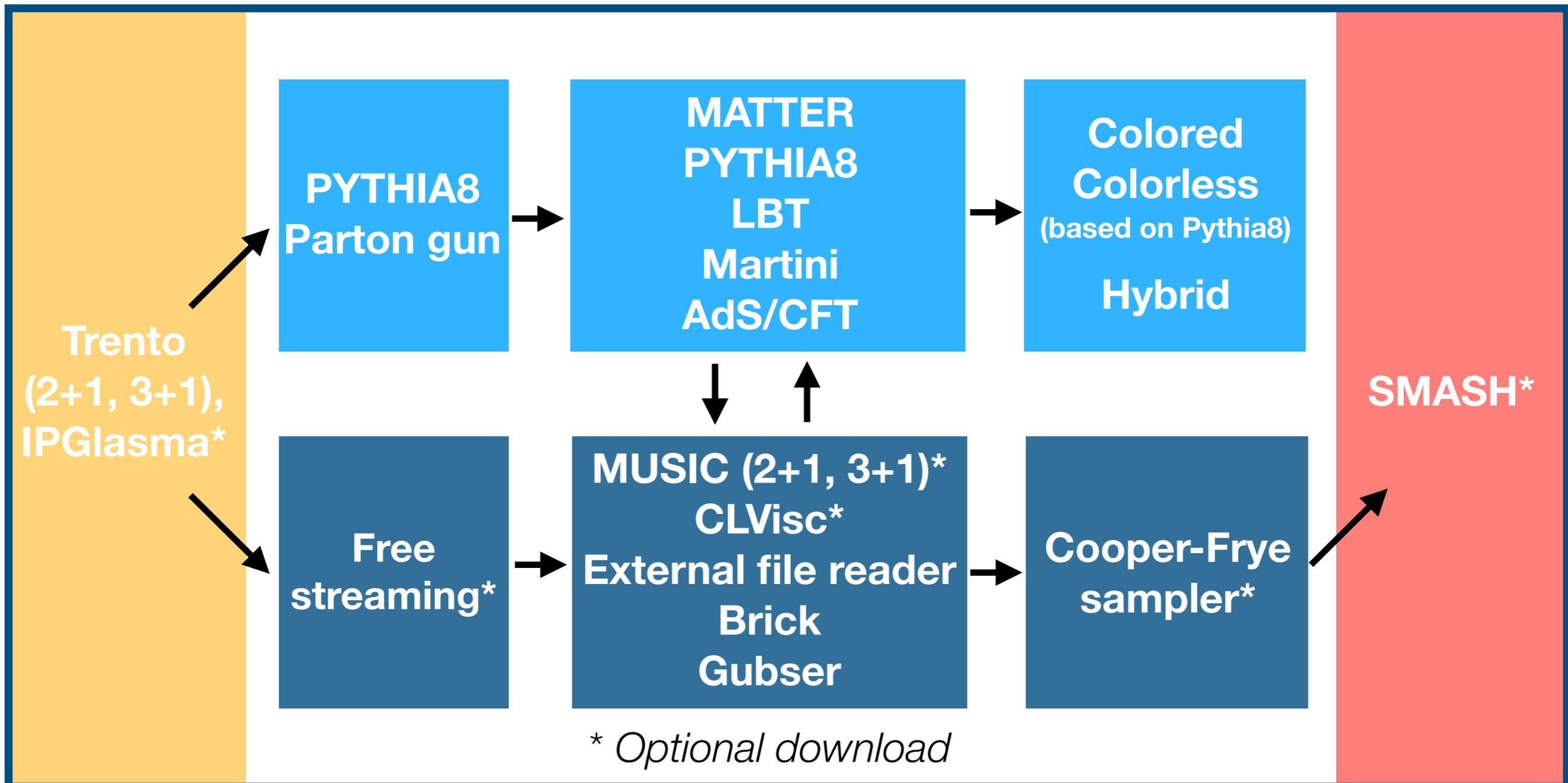


JETSCAPE Event Generator

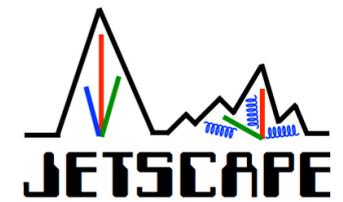


JETSCAPE Manual: 1903.07706

<https://github.com/JETSCAPE/JETSCAPE>



The current status



**The framework is 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

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

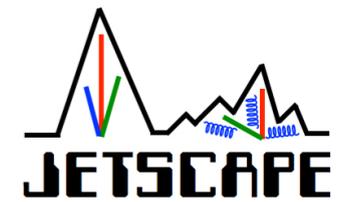
Please see the [Installation Instructions](#).

Running JETSCAPE

The main executable to generate JETSCAPE events is `runJetscape`, located in the `build/` directory. To generate JETSCAPE events, you should pass an XML file specifying the settings with which you would like to run:

```
./runJetscape ../config/jetscape_user.xml
```

The current status



**The framework is 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

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](#).

Running JETSCAPE

The main executable to generate JETSCAPE events is `runJetscape`, located in the `build/` directory. To generate JETSCAPE events, you should pass an XML file specifying the settings with which you would like to run:

```
./runJetscape ../config/jetscape_user.xml
```

What this means for physics comparisons

First physics results of “out-of-the-box” models from the JETSCAPE Collaboration

Use responsibly — it is a framework — you get out the physics you put in

There is a lot of theoretical work inside — but currently only slices of the theory landscape

➔ **We are at the start of the exciting phase — well-controlled theory comparisons**

Steps to run JETSCAPE



1. **Install and build 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!

Installing JETSCAPE



<https://github.com/JETSCAPE/JETSCAPE/wiki/Doc.Installation>

JETSCAPE Installation

Joe Latessa edited this page on Jun 3 · 2 revisions

Instructions to Install and Run JETSCAPE Using Docker

Preferred for local use and development

We recommend to [install JETSCAPE](#) and its pre-requisites using Docker.

Instructions to Install and Run JETSCAPE Using Singularity

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

Another option is to [install JETSCAPE](#) 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 JETSCAPE



Docker support



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

README.md

Using JETSCAPE via Docker

Docker is a software tool that allows one to deploy an application in a portable environment. A docker "image" can be created for the application, allowing any user to run a docker "container" from this image. We have prepared a docker image for the JETSCAPE environment, which allows you to use JETSCAPE on macOS or linux without installing a long list of pre-reqs or worrying about interference with software you already have installed. Step-by-step instructions are provided below.

Create and start a docker container that contains all of the JETSCAPE pre-reqs:

macOS:

```
docker run -it -v ~/jetscape-docker:/home/jetscape-user --name myJetscape jetscape/base:v1
```

linux:

```
docker run -it -v ~/jetscape-docker:/home/jetscape-user --name myJetscape --user $(id -u):$(id -g)
```

<https://github.com/JETSCAPE/JETSCAPE/tree/main/docker>

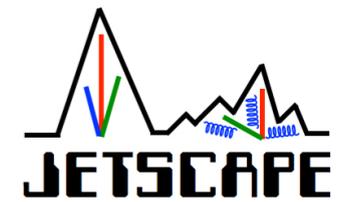
For this school, we require you to run JETSCAPE via docker

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

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 (master) 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, pbbp2760, pbbp5020, 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>
```

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 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

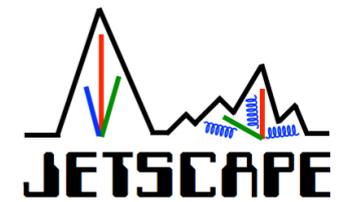
HepMC3

Standard event format (larger size)

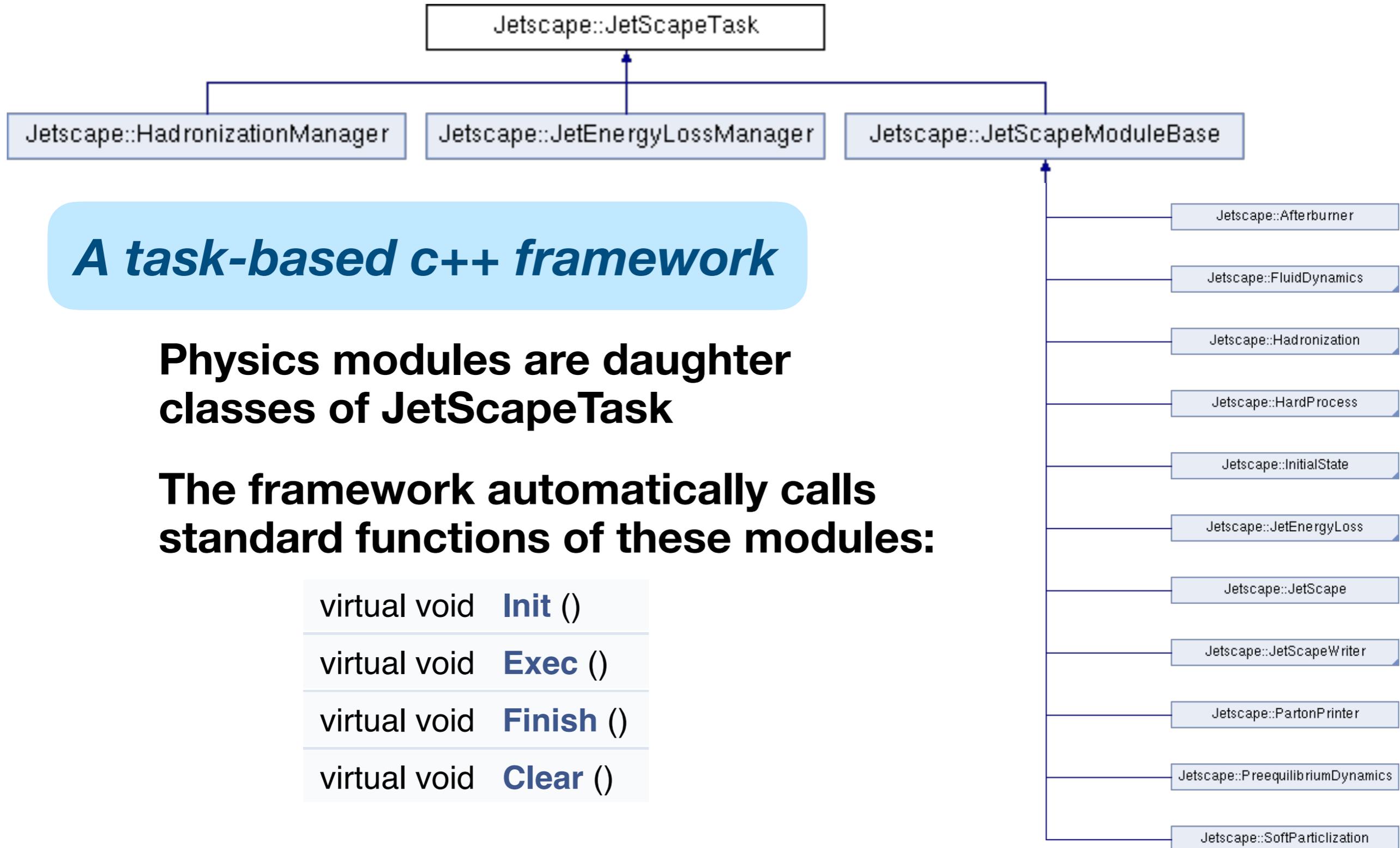
Compatible with Rivet

* You can easily write your own output class, for example using ROOT trees, 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 **Init** ()
- virtual void **Exec** ()
- virtual void **Finish** ()
- virtual void **Clear** ()

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

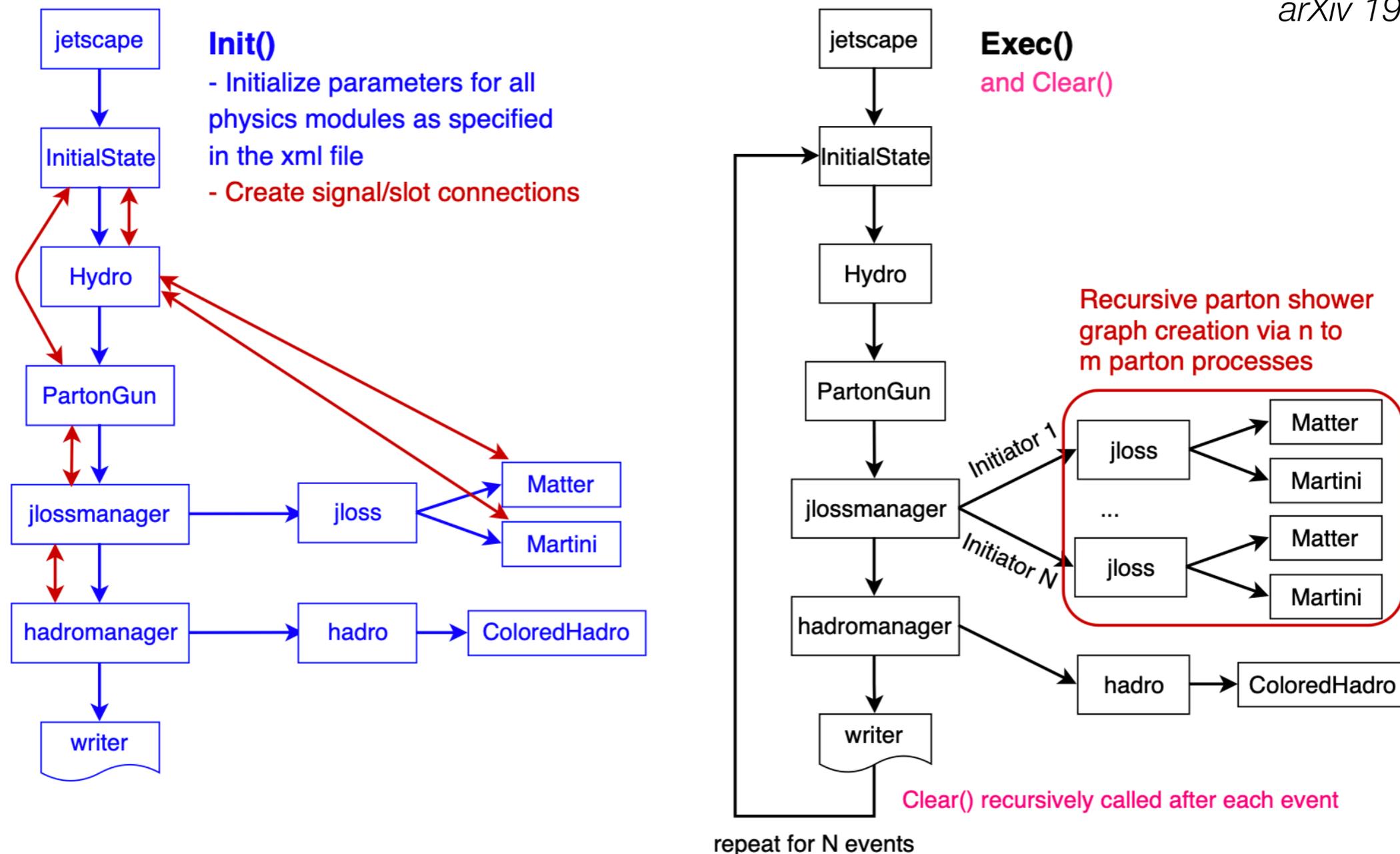
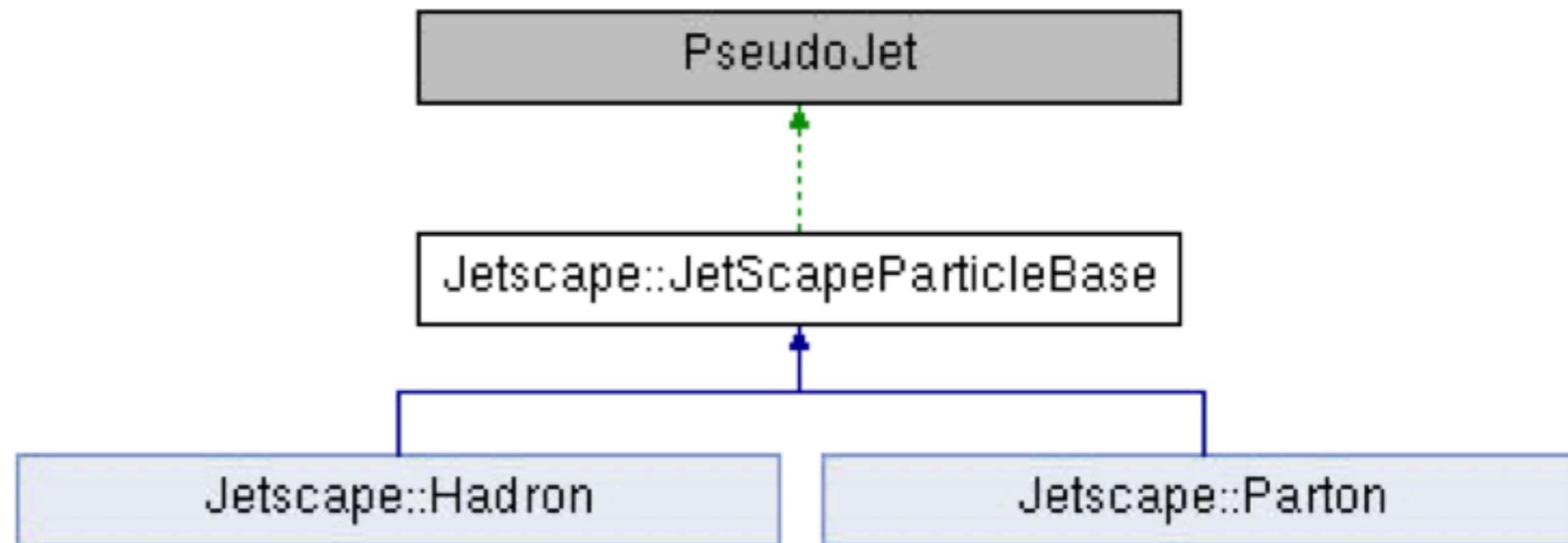


Figure 4: Example workflow of the `Init()` (left side) and `Exec()` (and `Clear()`) (right side) phase of the task based JETSCAPE framework (the not extensively used `Finish()` phase is omitted). One should be aware that the created signal/slot connections in the `Init()` phase are of course present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

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



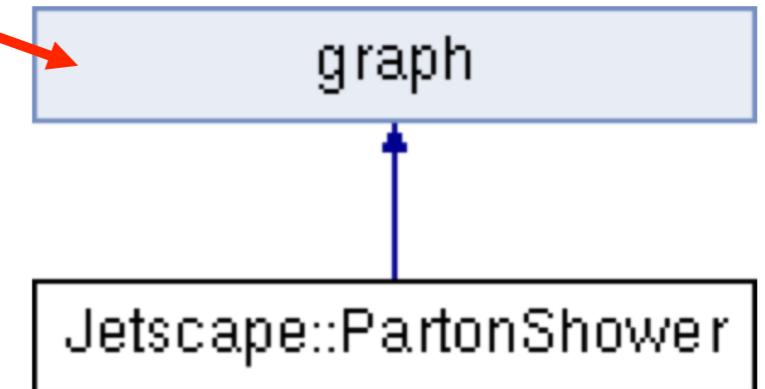
```
Parton (int label, int id, int stat, const FourVector& p, const FourVector& x);  
Parton (int label, int id, int stat, double pt, double eta, double phi, double e, double* x=0);  
...
```

In addition: Photon class (inherits from Parton)

GTL Graph Template Library

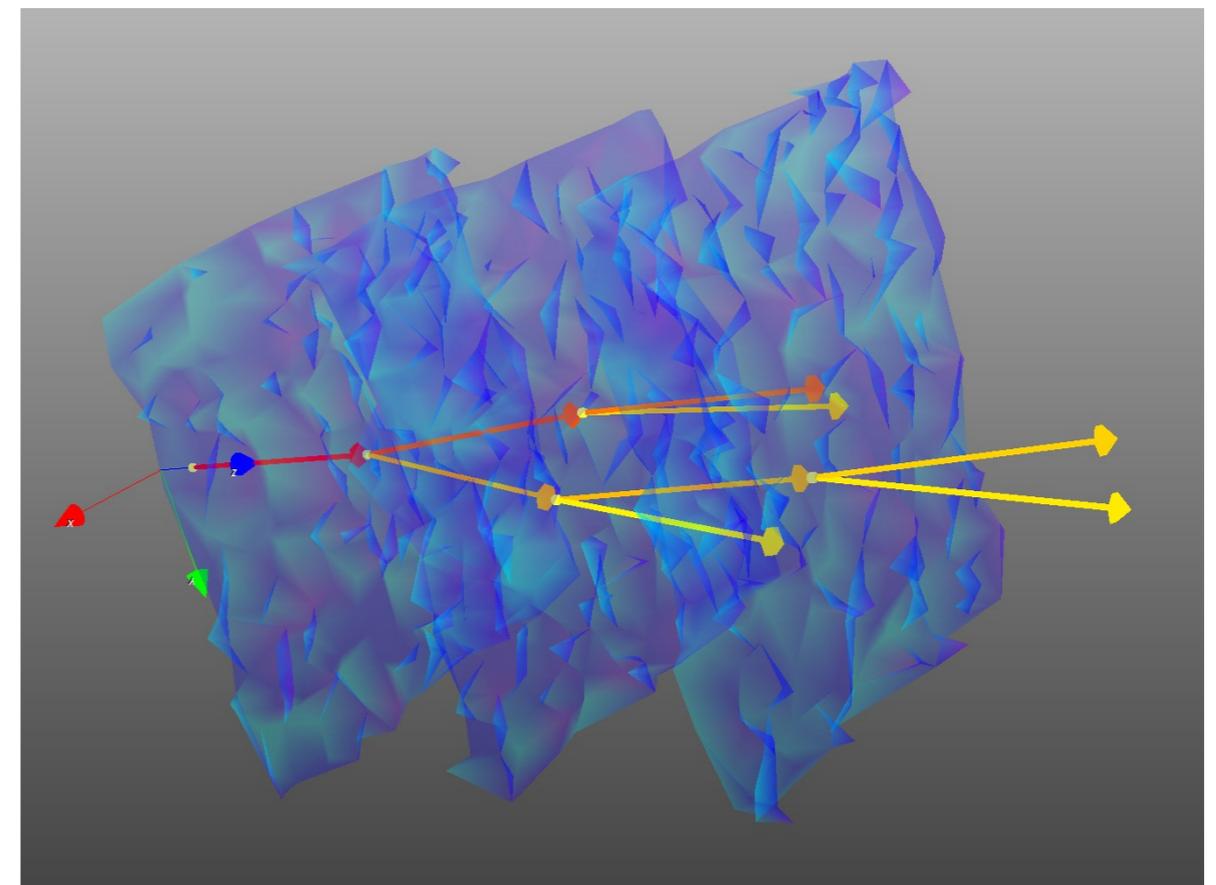
[<https://github.com/rdmpage/graph-template-library>]

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



```
int GetNumberOfParents (int n)
int GetNumberOfChilds (int n)
shared_ptr< Parton > GetFinalPartons ()
shared_ptr< fjcore::PseudoJet > GetFinalPartonsForFastJet ()
int GetNumberOfPartons () const
int GetNumberOfVertices () const
```

Provides functionalities to query shower



Contributing modules



arXiv 1903.07706

JETSCAPE 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 Init ()
```

```
virtual void Exec ()
```

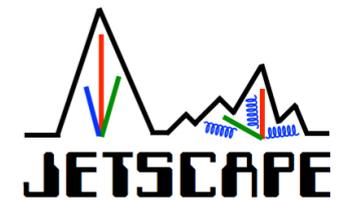
```
virtual void Finish ()
```

```
virtual void Clear ()
```

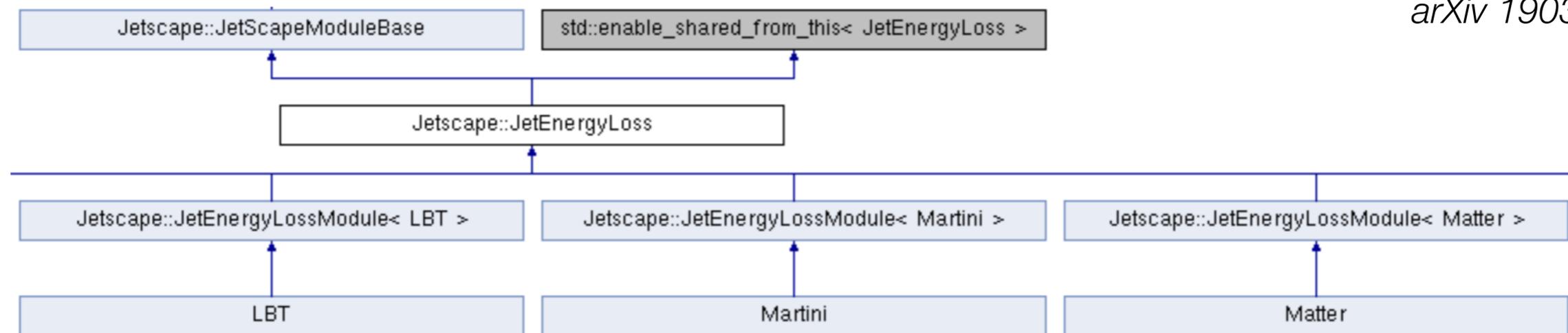
Use appropriate signal/slot info to interact with other modules

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()`

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).

JETSCAPE is a **framework for general-purpose heavy-ion event generators**

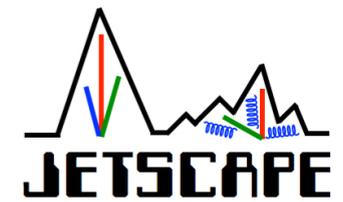
- Modular, extensible — please contribute modules!

JETSCAPE is a tool for the community

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

JETSCAPE has been successfully used and tested in large scale simulation efforts

Remarks on large scale simulations



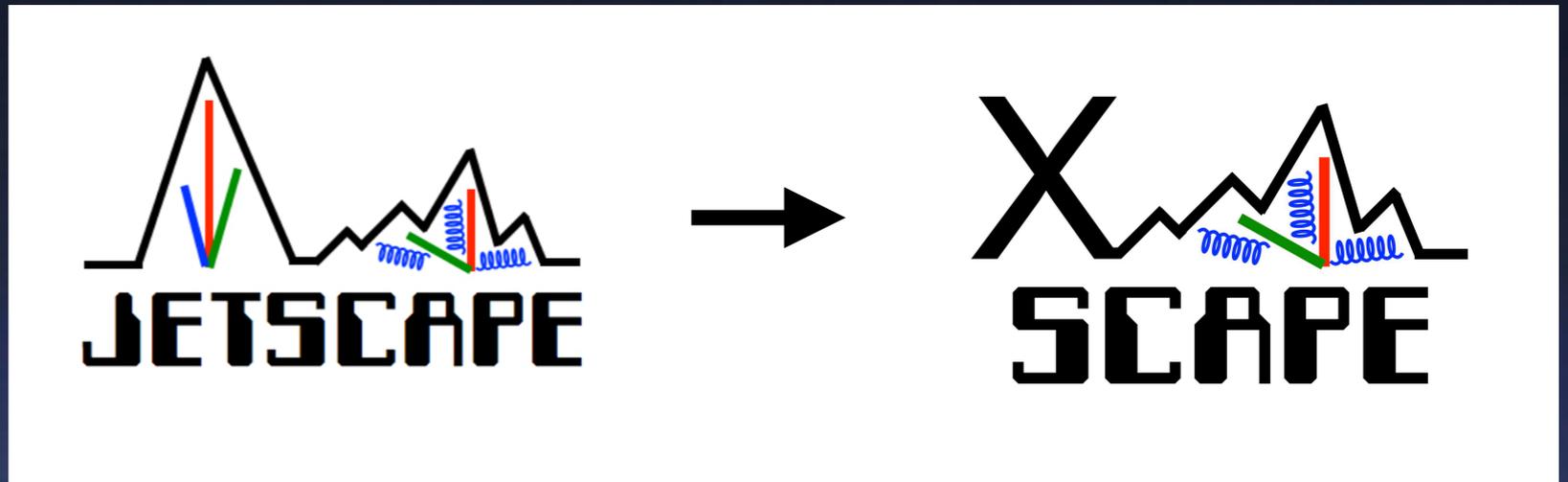
JETSCAPE has used **tens of millions of core-hours running simulations** for Bayesian analyses (covered next week) which are especially computationally expensive. When you run simulations at scale, here are some details to consider:

- 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

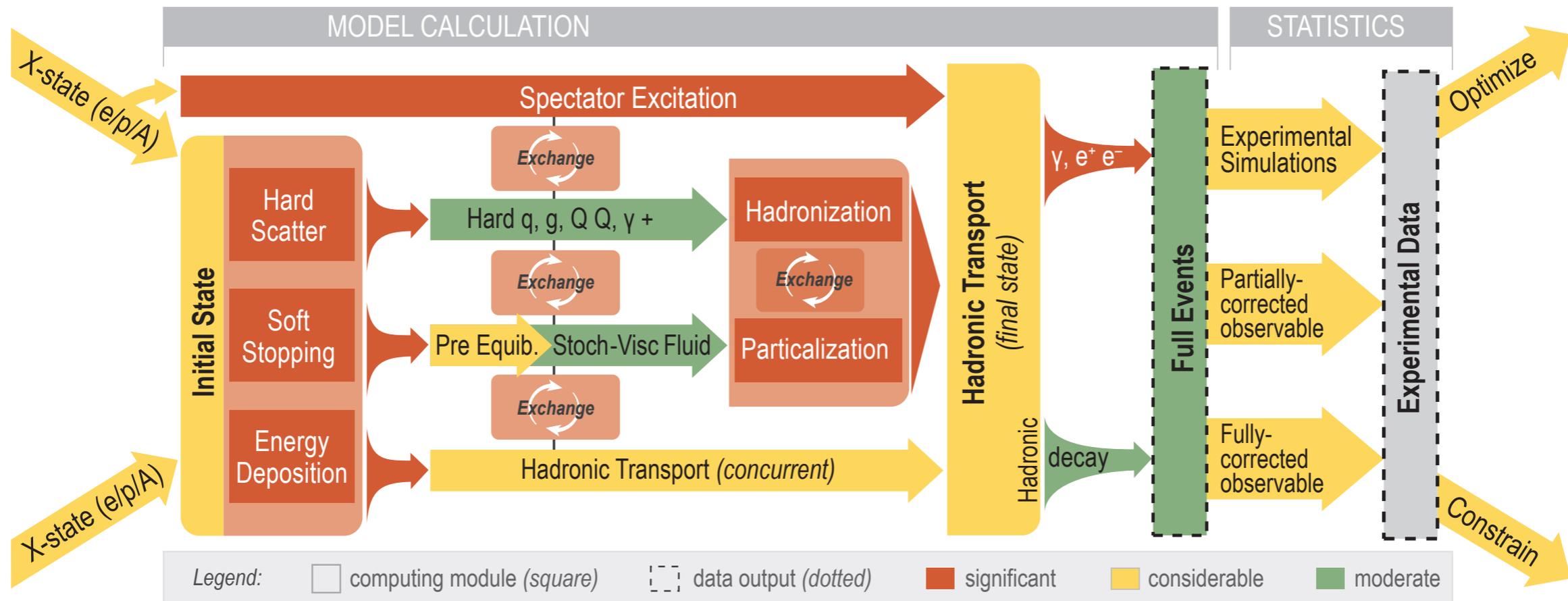
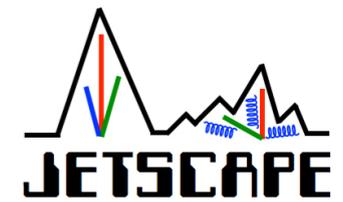
One more thing...



One more thing...



XSCAPE: Extended Physics Scope



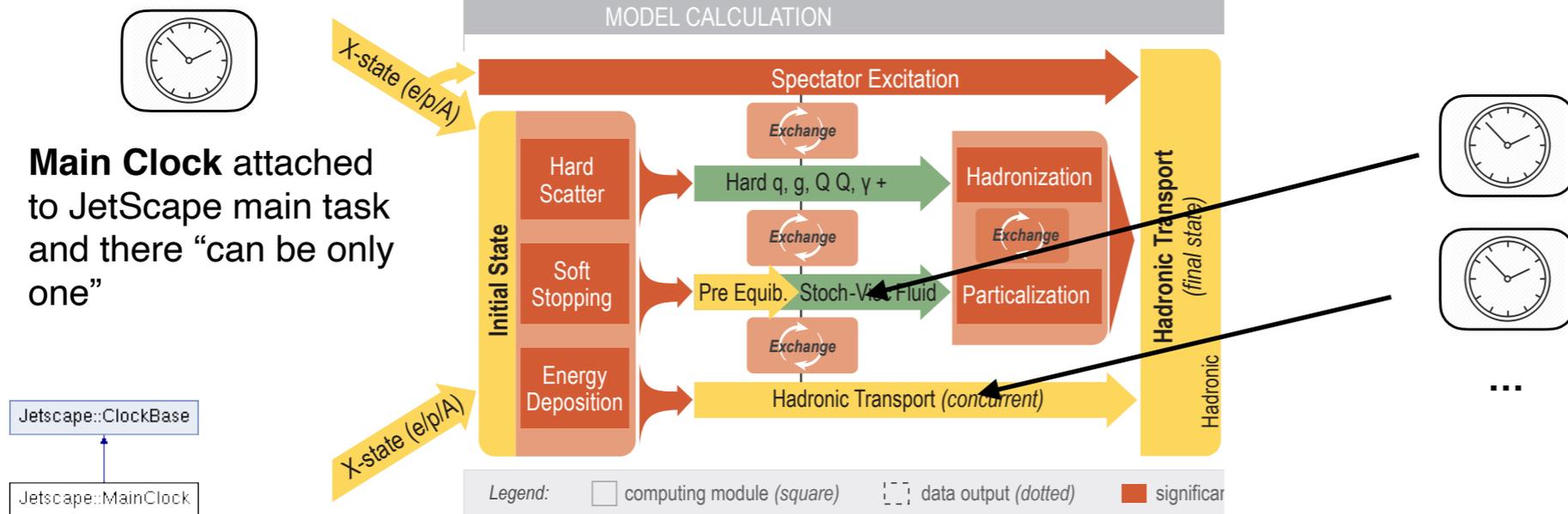
XSCAPE framework goal: To provide a *decentralized* framework, which will allow any user to attach his/her own modules and reorganize the flow of data between the modules. with the goal to simulate:

- 1) Any aspect of A-A collisions from FAIR to top LHC energy
- 2) To simulate p -A and p - p collisions at arbitrary multiplicity
- 3) To study e^+e^- and certain aspects of e -A collisions.

XSCAPE: Some feature highlights ...



Full backwards compatibility to JETSCAPE!



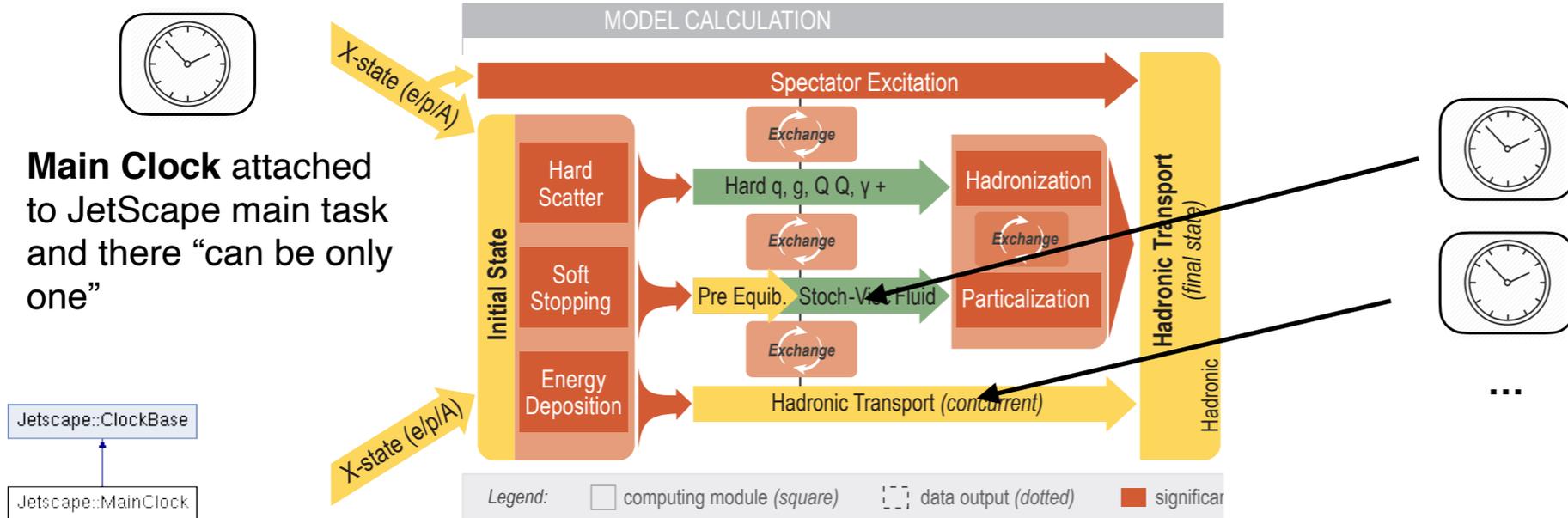
Module Clock(s) can be attached to any JetScape task

Execution of all Tasks/physics modules per time-step.

XSCAPE: Some feature highlights ...

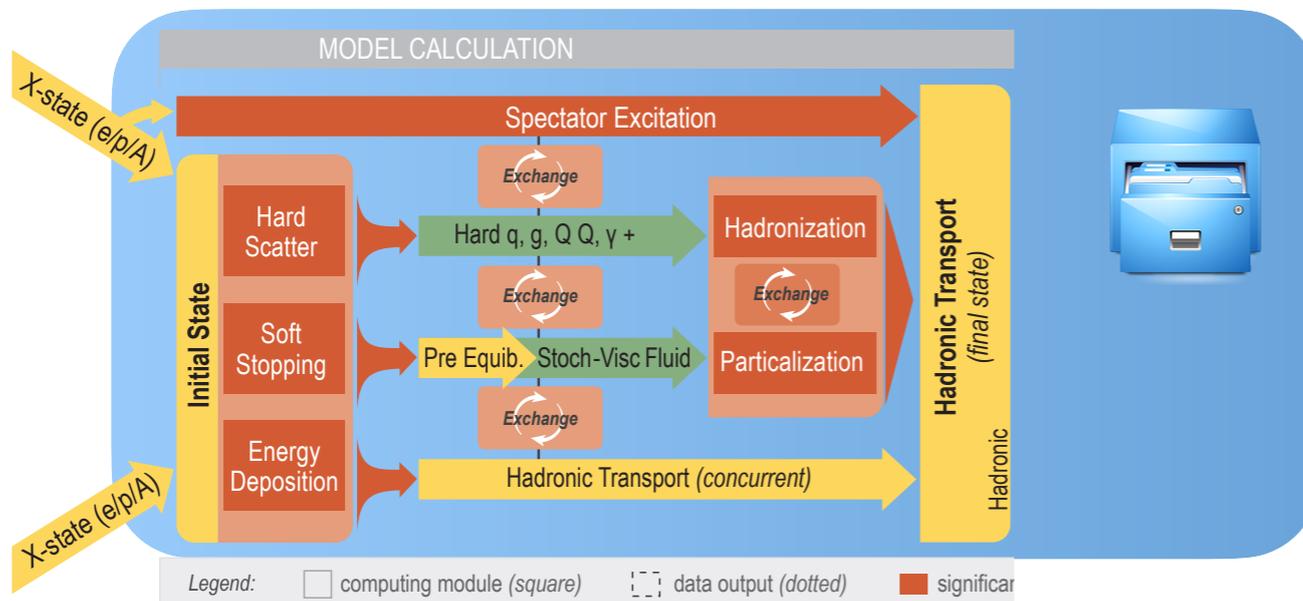


Full backwards compatibility to JETSCAPE!



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

Execution of all Tasks/physics modules per time-step.

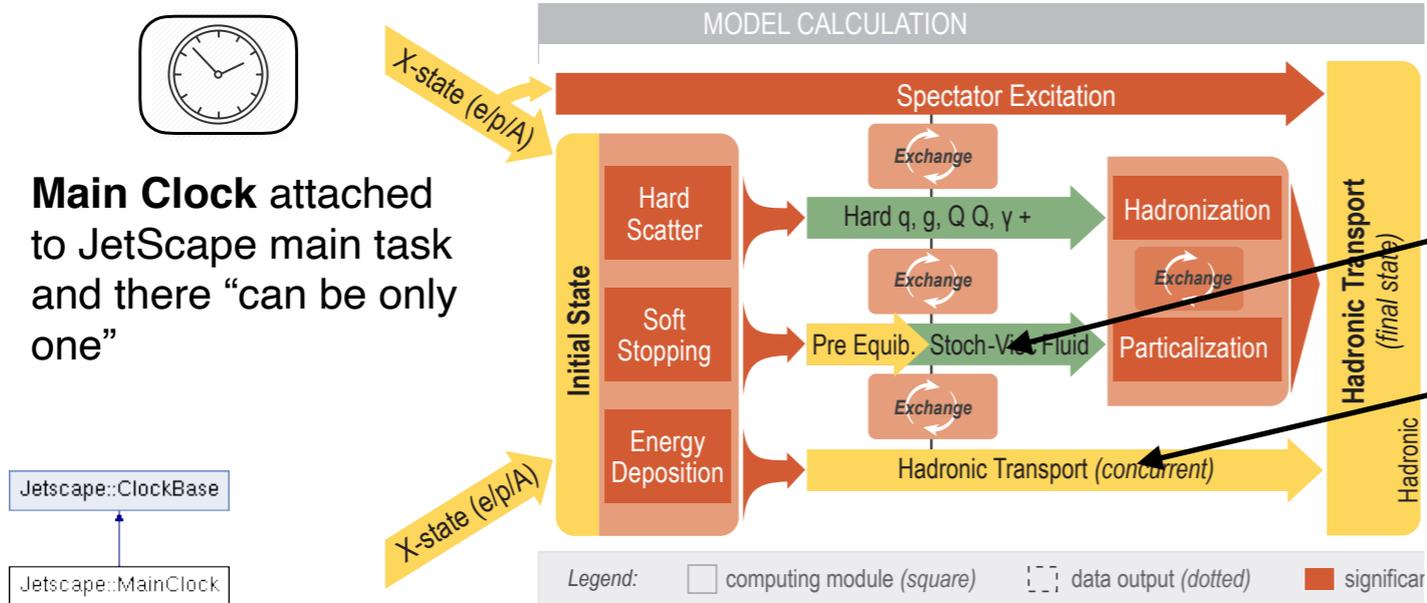


"Database" query approach using any/variant datatype via **QueryHistory** instance

XSCAPE: Some feature highlights ...



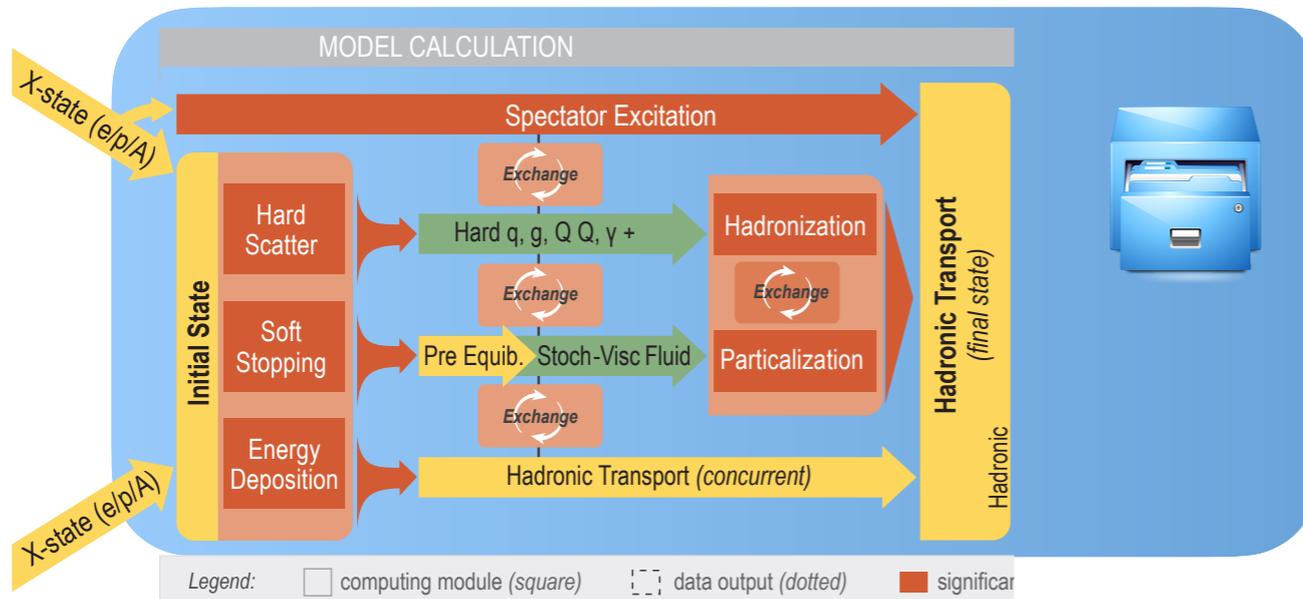
Full backwards compatibility to JETSCAPE!



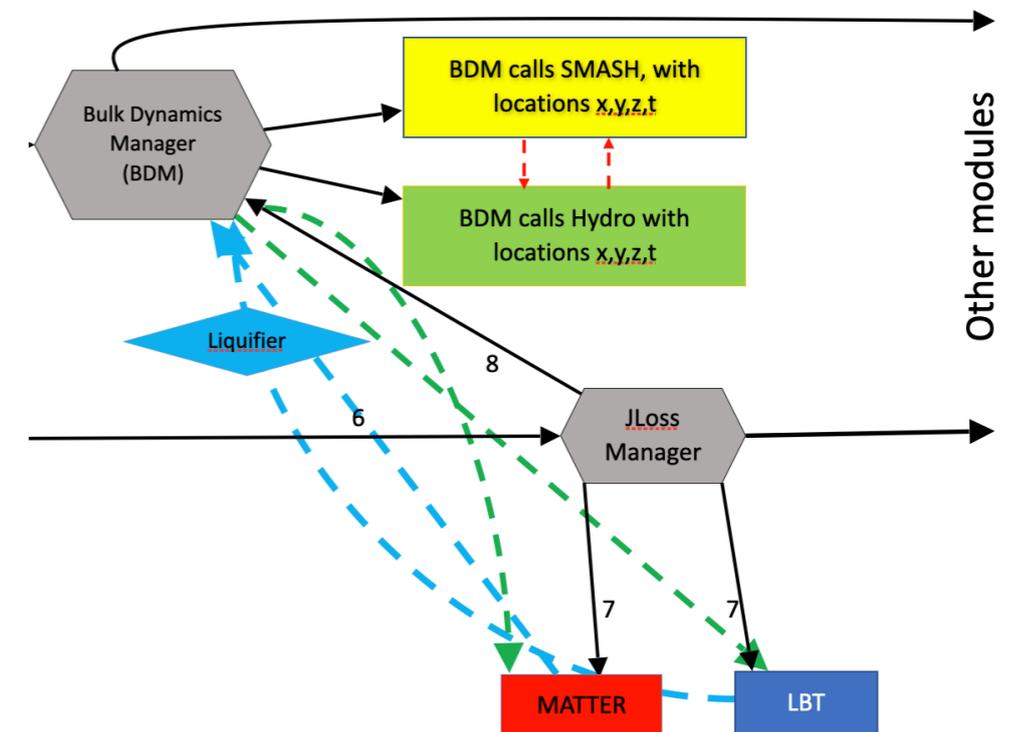
Module Clock(s) can be attached to any JetScape task

Execution of all Tasks/physics modules per time-step.

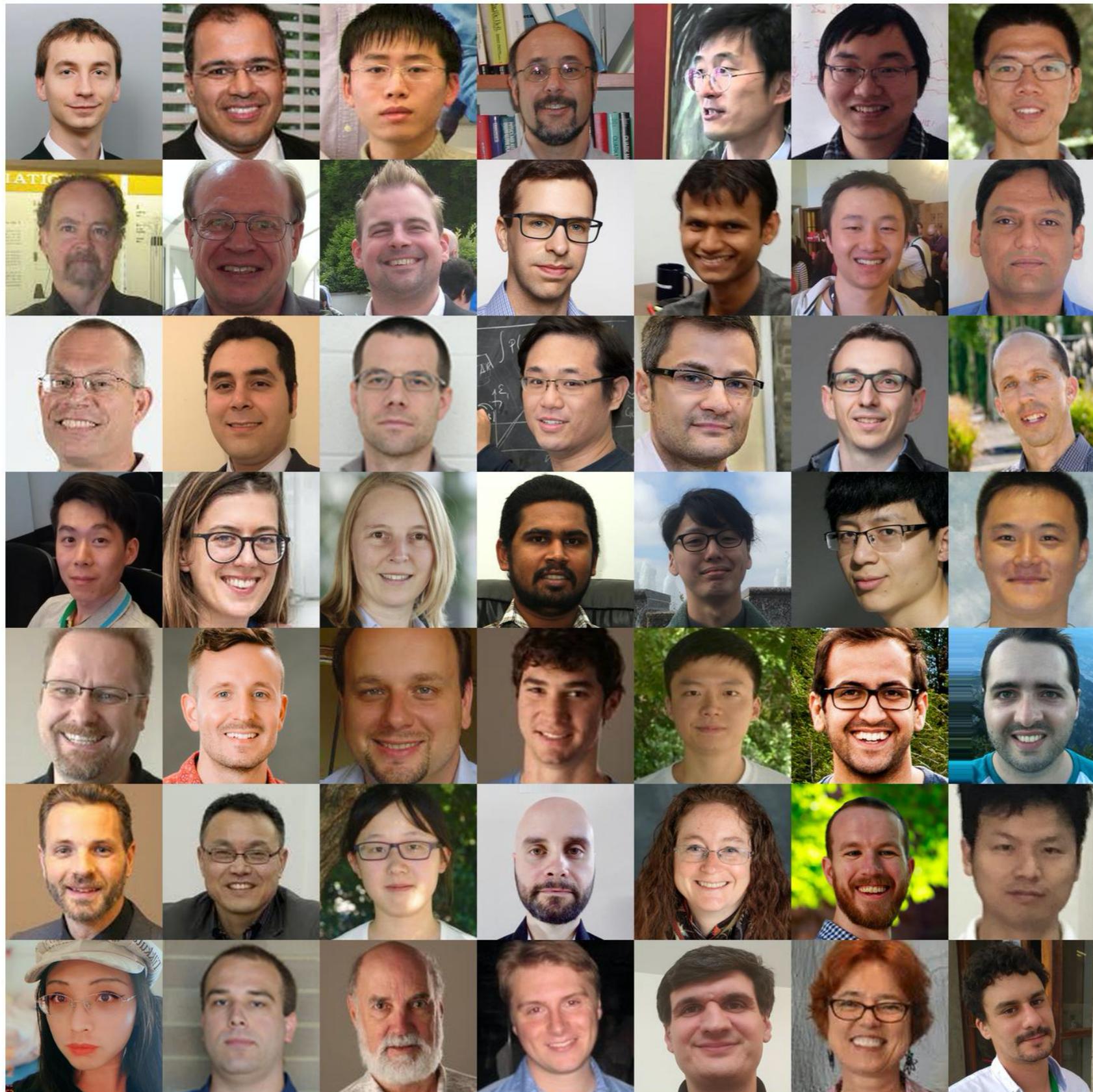
BulkDynamicsManager (BDM): Organizes/allows concurrent running of several different bulk media implementations



“Database” query approach using any/variant datatype via **QueryHistory** instance



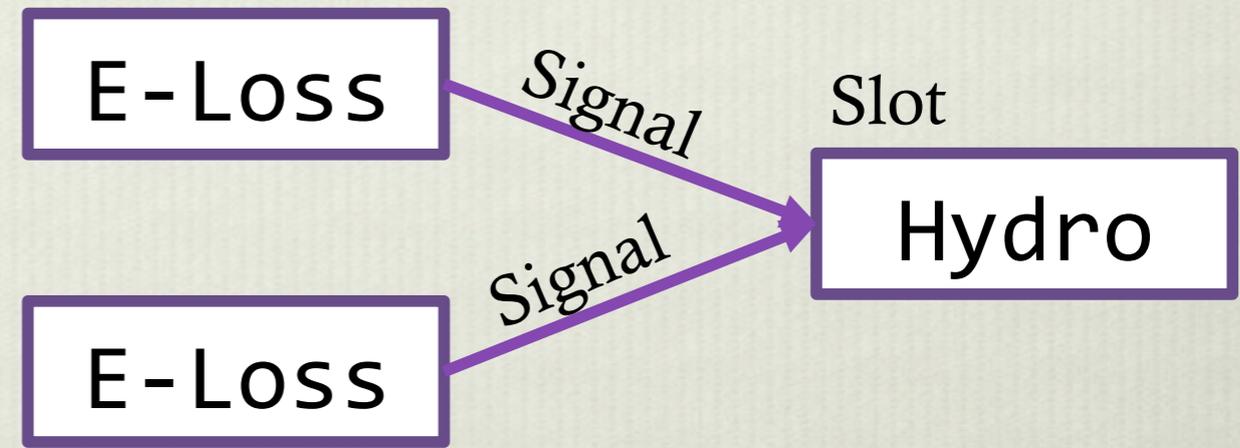
Thank you!



Backup

- ❖ C++11
 - ❖ Smart pointers
 - no memory leaks!
 - ❖ Encapsulation
- ❖ Infrastructure:
 - ❖ XML reader
 - ❖ thread-safe logger
 - ❖ Unified random numbers across modules
 - reproducibility!
 - ❖ Signal Managers

- ❖ Communication via **Signals & Slots**

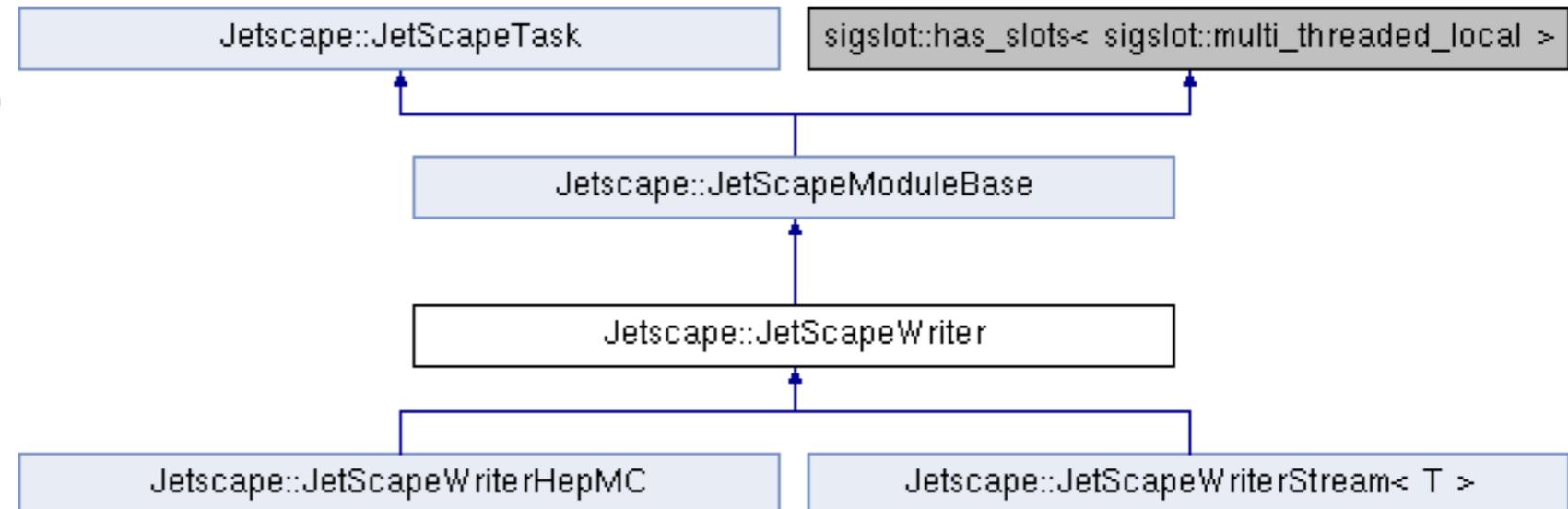


- ❖ Developed by the Qt project,
- ❖ Using light **sigslot** by Sarah Thompson
- ❖ Manage module switching



All under the hood!

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)

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 ()  
shared_ptr< Hadron >> GetHadrons ()  
vector< fjcore::PseudoJet > GetHadronsForFastJet ()
```

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