Goals and Developments for MC Event Generators

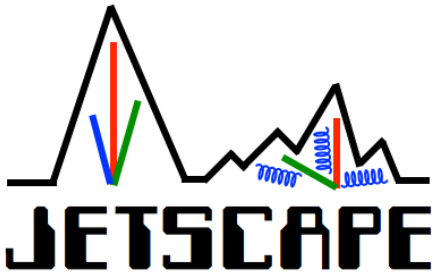
Discussion

Königspitze (3851m)

Goals and Developments for MC Event Generators

Discussion

in the context of JETSCAPE



```
[Info] *------------------------------------------------------------*
[Info] |                                                            |
[Info] |                                                            |
[Info] |                                                            |
[Info] |                                                            |
[Info] |                                                            |
[Info] |                                                            |
[Info] |                                                            |
[Info] |                                                            |
[Info] |                                                            |
[Info] |                                                            |
[Info] |                   JETSCAPE release 1.0 RC1                 |
[Info] |                                                            |
[Info] |     The Jet Energy-loss Tomography with a Statistically    |
[Info] |       and Computationally Advanced Program Envelope        |
[Info] |                http://jetscape.wayne.edu                   |
[Info] |                                                            |
[Info] | Please cite xxx if you use this package for scientific work. |
[Info] |                                                            |
[Info] | JETSCAPE is provided without warranty under the terms      |
[Info] | of the GNU GPLv3. It uses xxx code(s).                     |
[Info] | See COPYING file for details.                              |
[Info] |                                                            |
[Info] *------------------------------------------------------------*
```

Königspitze (3851m)

(Some) Goals of Monte Carlo Event Generators

Clearly we want to learn/study the fundamental physics processes of jet quenching/QCD/QGP medium, etc by comparing data to models (apples-to-apples)

[Systematic evaluations via Bayesian analysis]

Use MC to study detector performance/sensitivity to quenching observables. Easy access to different jet quenching implementations (or different MC generators) needed.

…

Some questions from previous Workshop …

Questions Discussed

- Is an experimental jet equivalent to a theoretical jet?
- (How) can a given experimental technique be applied to a theoretical calculation?
- Can a bias imposed by an experimental technique be (easily/well) calculated theoretically?
- Can a given theoretical definition of background be distinguished from the signal in experiment?
- What is the sensitivity of techniques to interesting physics?
- How should background and background fluctuations be treated for full Monte Carlo models?
- How low in $p_{T,Jet}$ are jets still well defined? Do we need low p_T jets?

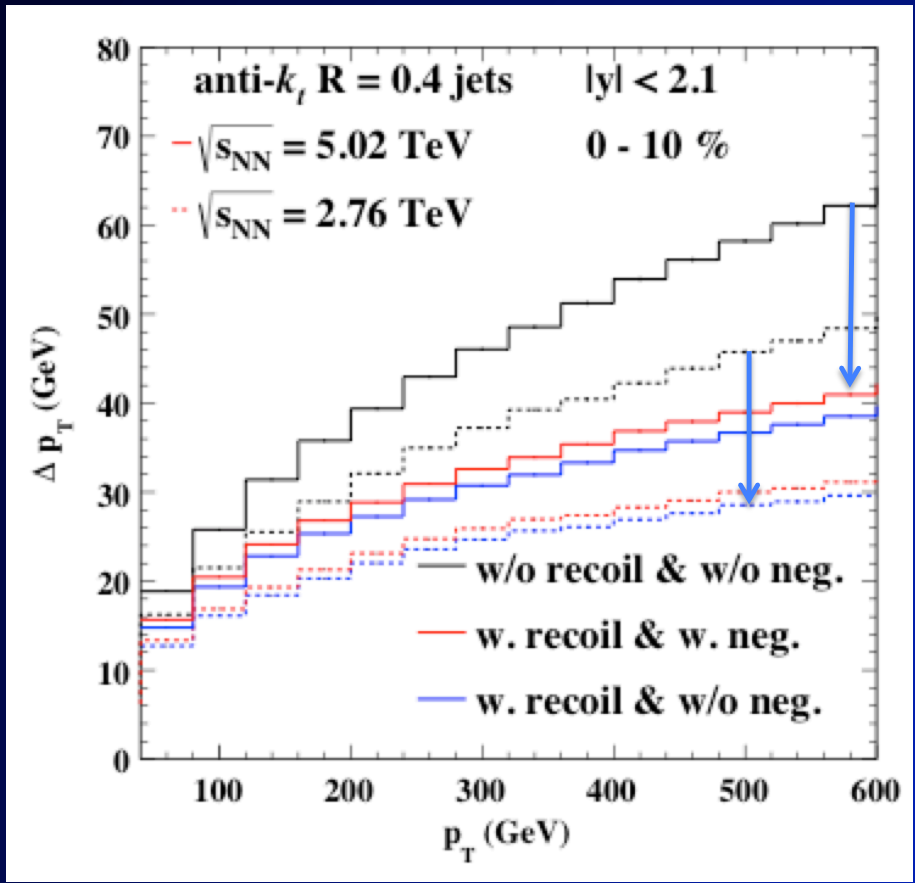
Some questions from previous Workshop …

Questions Discussed

- Is an experimental jet equivalent to a theoretical jet?
- (How) can a given experimental technique be applied to a theoretical calculation?
- Can a bias imposed by an experimental technique be (easily/well) calculated theoretically?
- Can a given theoretical definition of background be distinguished from the signal in experiment?
  - For analytical calculations: **regions of validity** should be assigned together with systematics due to underlying assumptions
- W
  ph
- Hc
  be
- Hc
  lo

Importance of Medium Response (LBT)

Medium response reduces jet energy loss



Recoil partons within the jet cone reduce the net jet energy loss

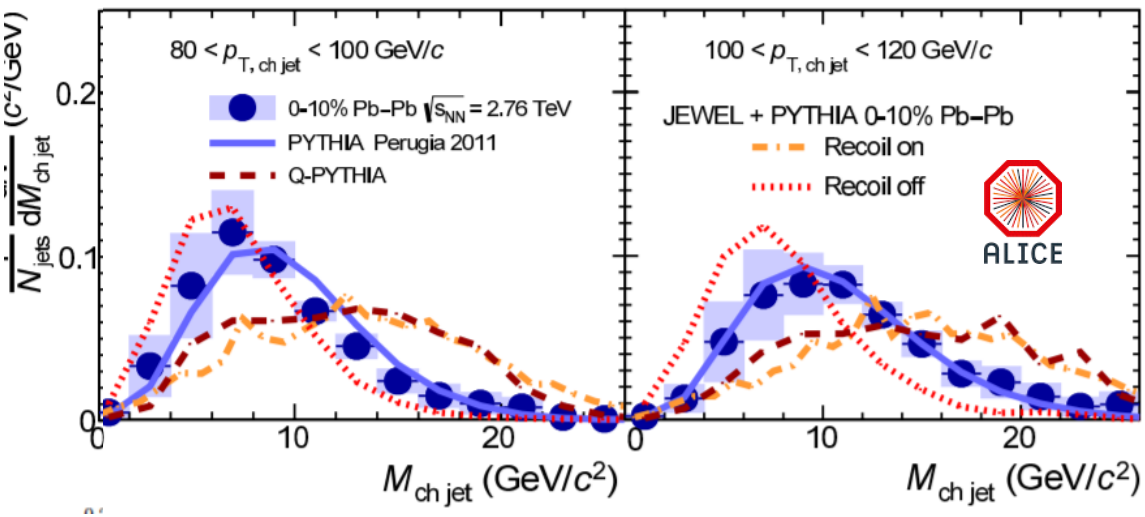
Diffusion wake (backreaction) reduces the thermal background, if taken into account, increase the net jet Energy loss with given cone-size

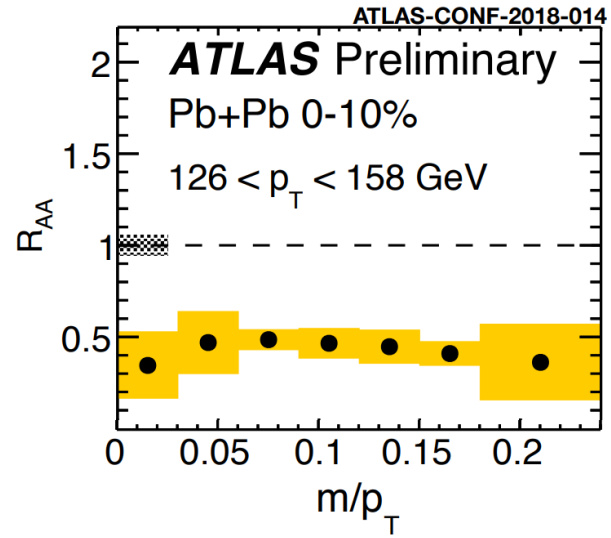
Depend on jet cone-size R
Sensitive to radial flow

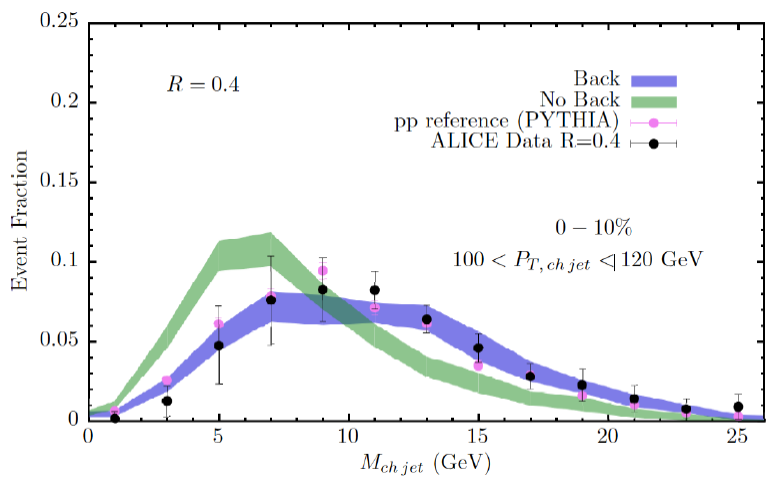
Can be estimated from elastic scattering: PRC96(2017)034903

Data vs Models (reproducibility)

"Ungroomed" jet mass



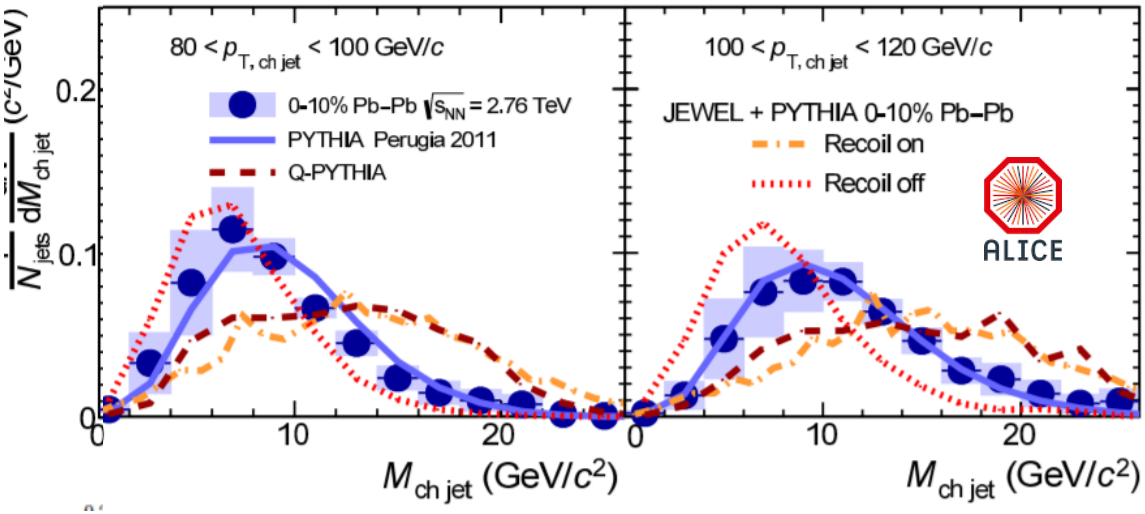


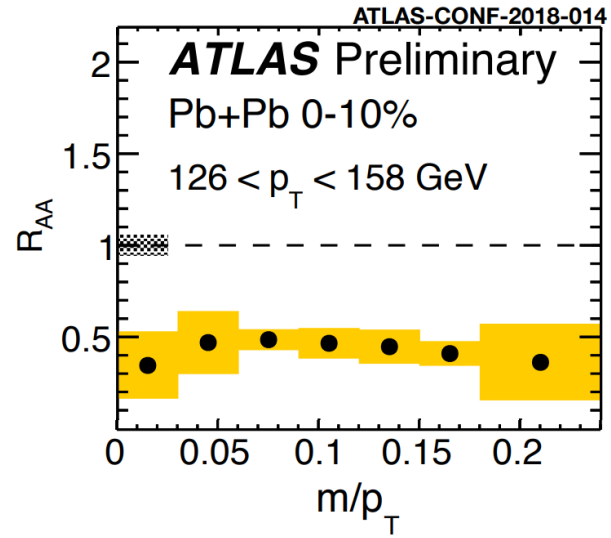


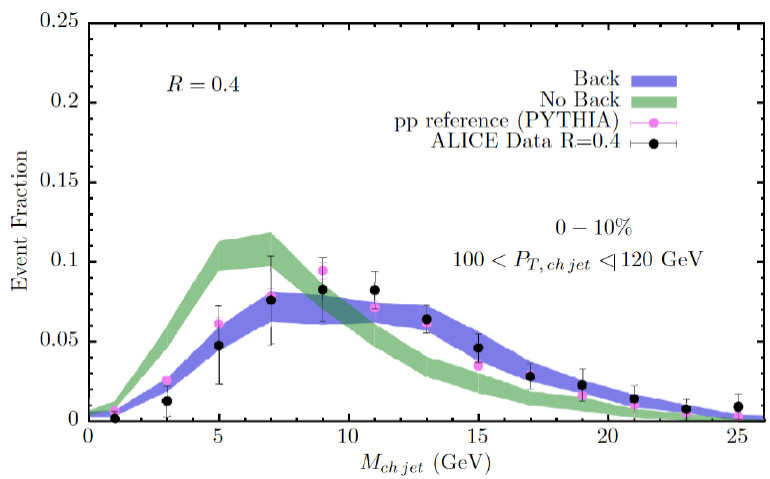
- No modification of the distribution is observed in **ALICE** and **ATLAS** data with respect to pp reference
- Cancelling effects from medium modifications of the shower and medium response

Data vs Models (reproducibility)

“Ungroomed” jet mass



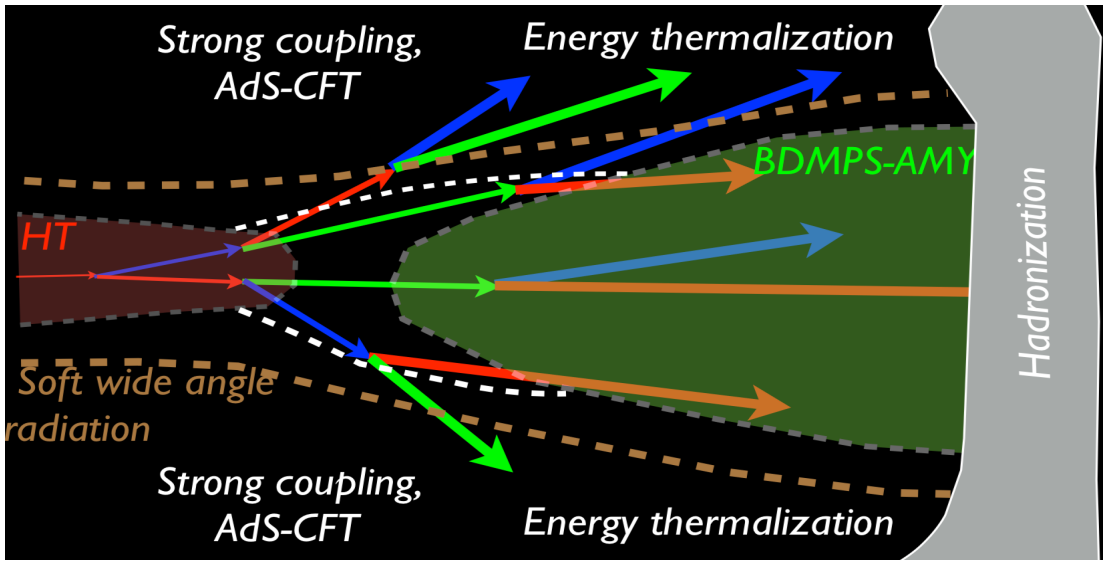




Treatment of Medium, Hadronization etc very different …
—> Hard to determine which underlying energy loss mechanism is the “correct” one.
Need for comparing different energy loss mechanism while keeping everything else the same …

Why is JETSCAPE a potential solution …

A. Majumder, Hard Probes '15

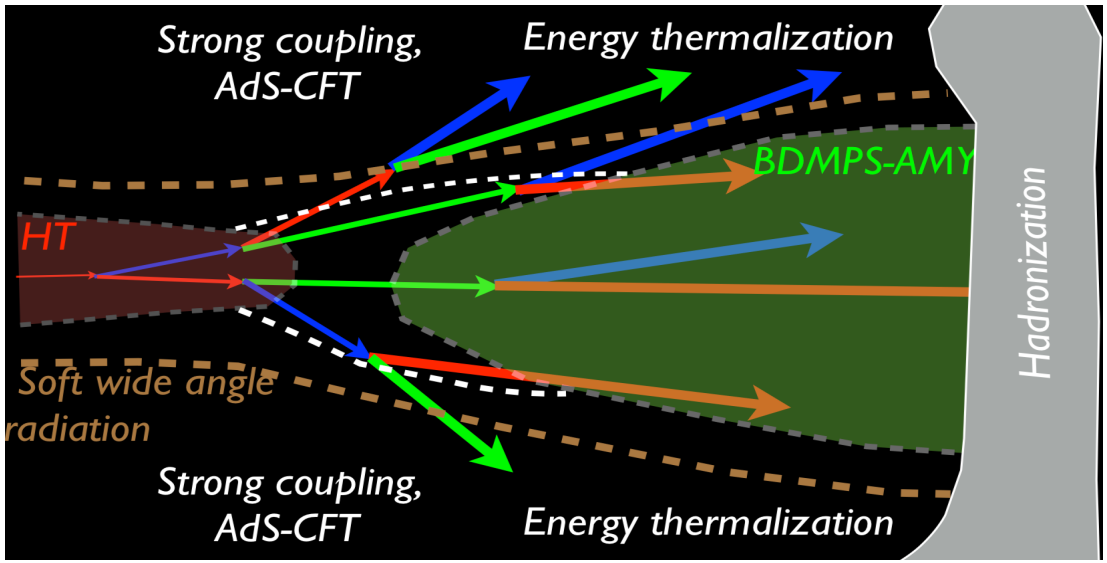


JETSCAPE:
Theoretical and experimental physicists, computer scientists, statisticians

- Multi-Stage Energy Loss
- … no one group can do it all
- Mission Statement: **Extensive, extensible event generator**
  - General. Modular. Self-contained. State-of-the-art.
- Note: Framework is agnostic to "multi-stage", "energy loss"

Why is JETSCAPE a potential solution …

A. Majumder, Hard Probes '15



JETSCAPE:
Theoretical and experimental physicists, computer scientists, statisticians

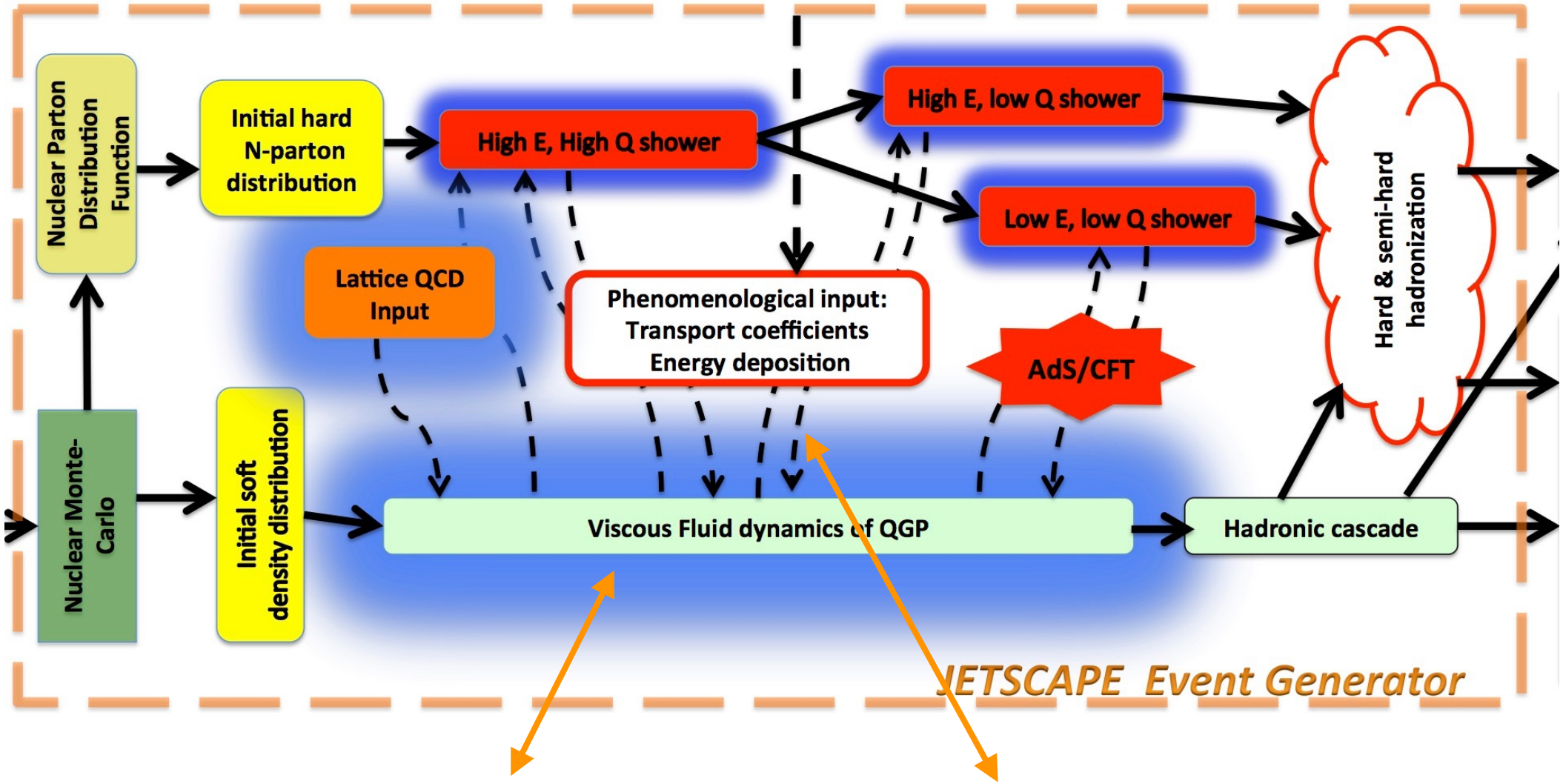
- Multi-Stage Energy Loss
- … no one group can do it all
- Mission Statement: **Extensive,**
  - General. Modular. Self-cont
- Note: Framework is agnostic

Two main goals of JETSCAPE:

i) **Event-Generator for end users**

ii) **Generic framework and extensions to allow easy integration of new physics (think a bit about like fastjet-contrib)**

Why is JETSCAPE a potential solution - Code Design

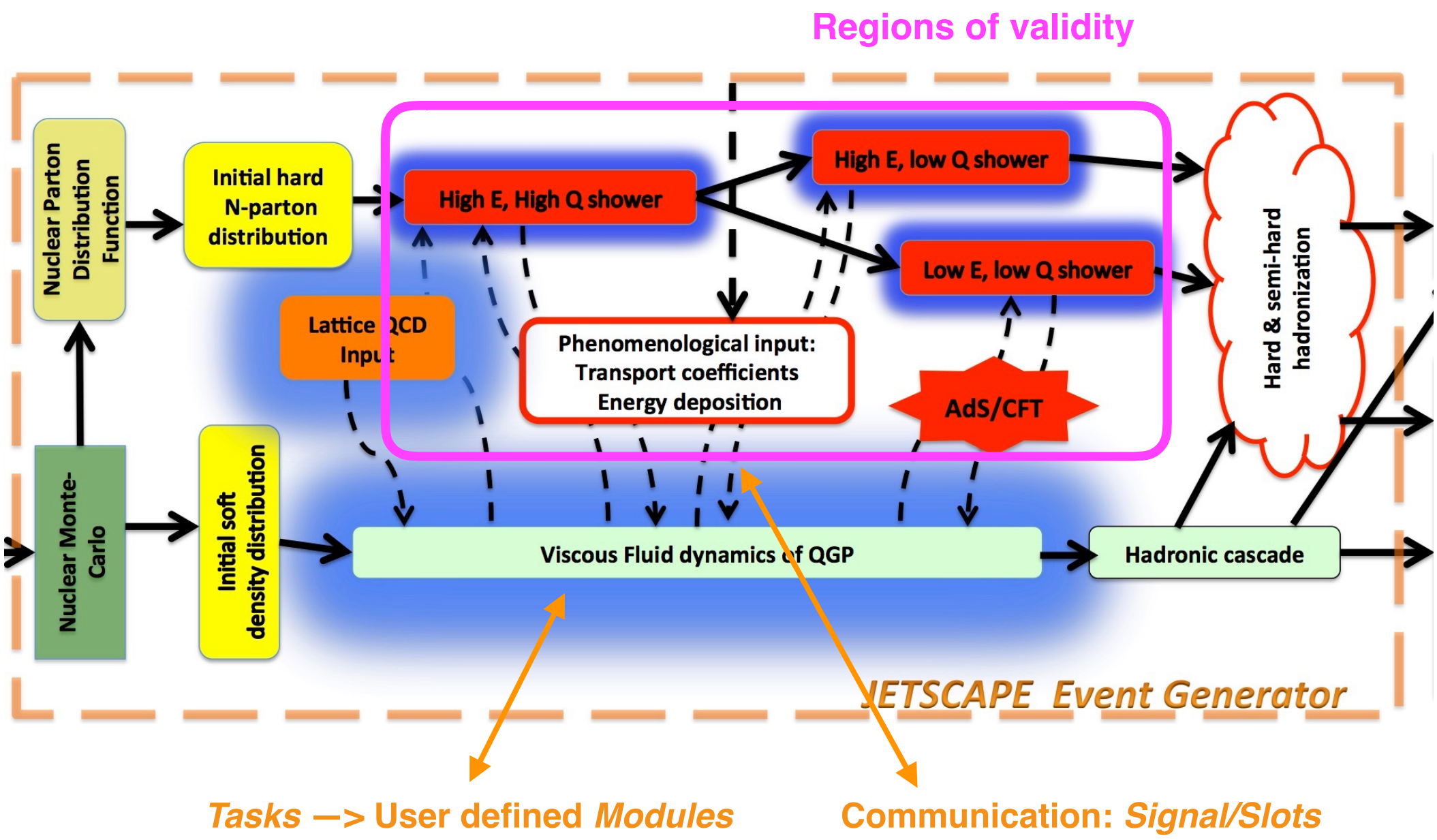


Tasks —> User defined *Modules*

Communication: *Signal/Slots*

Future: (Lego) *Blocks* for further modularization (more later)

Why is JETSCAPE a potential solution - Code Design



Tasks —> User defined *Modules*

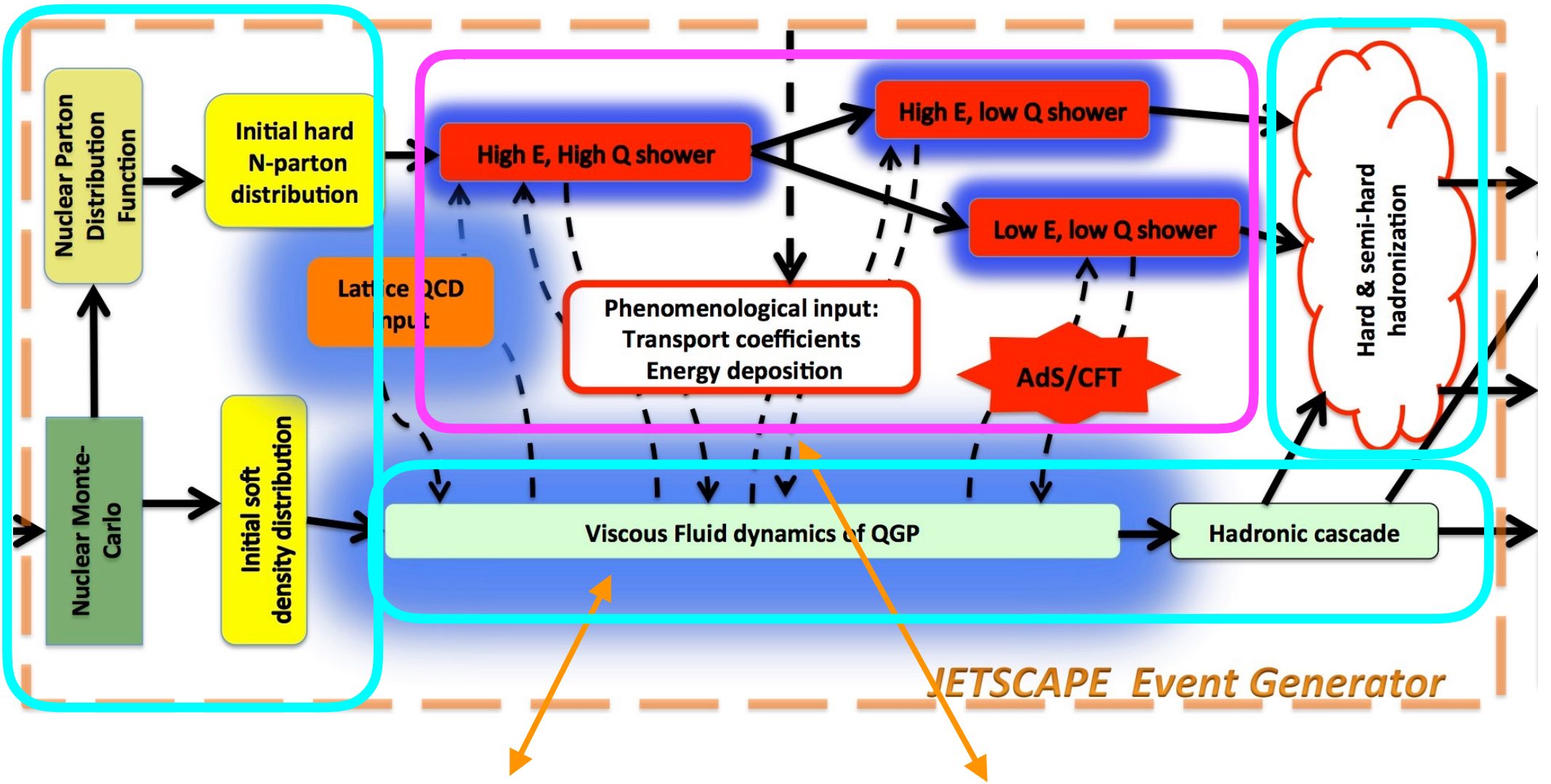
Communication: *Signal/Slots*

Future: (Lego) *Blocks* for further modularization (more later)

Why is JETSCAPE a potential solution - Code Design

"Keep everything else the same …"

Regions of validity



Tasks —> User defined *Modules*

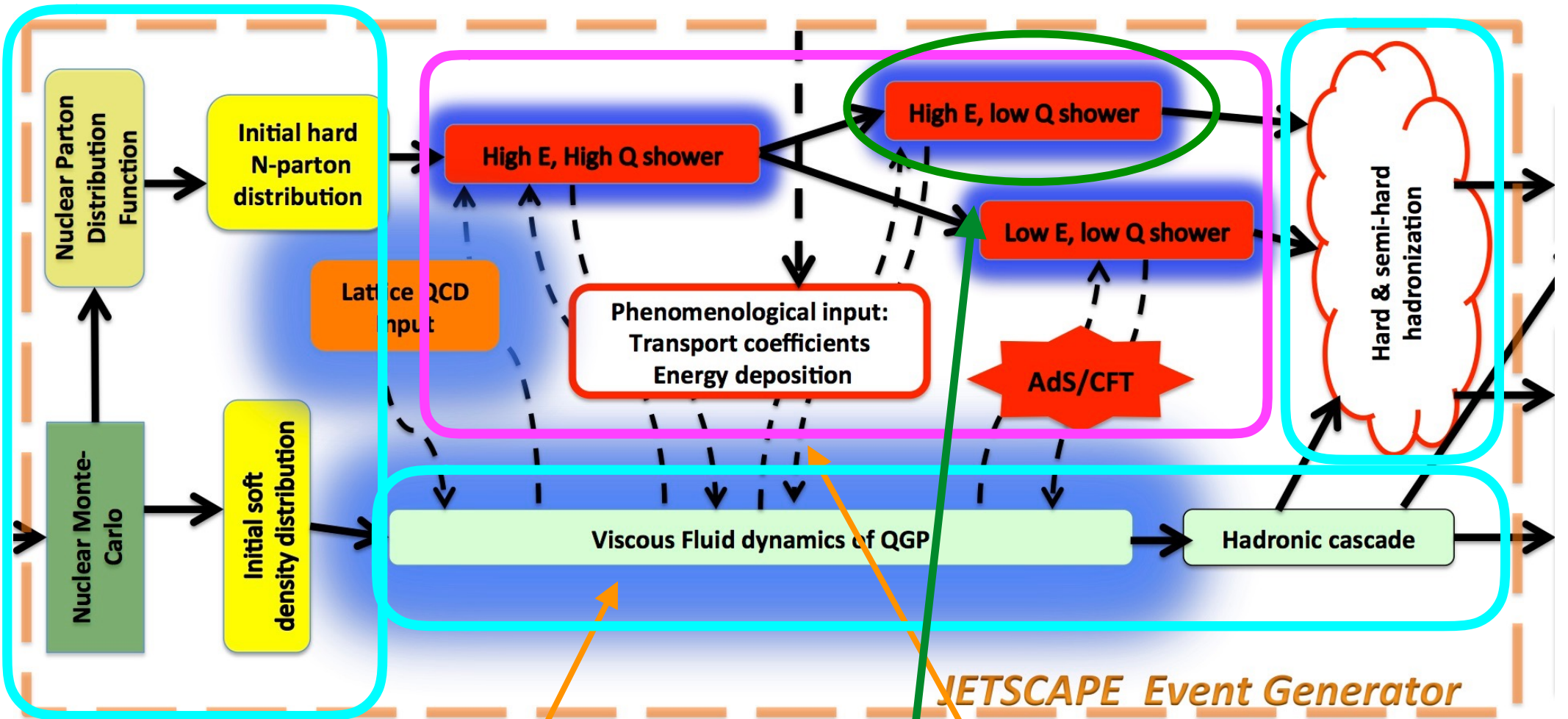
Communication: *Signal/Slots*

Future: (Lego) *Blocks* for further modularization (more later)

Why is JETSCAPE a potential solution - Code Design

"Keep everything else the same …"

Regions of validity



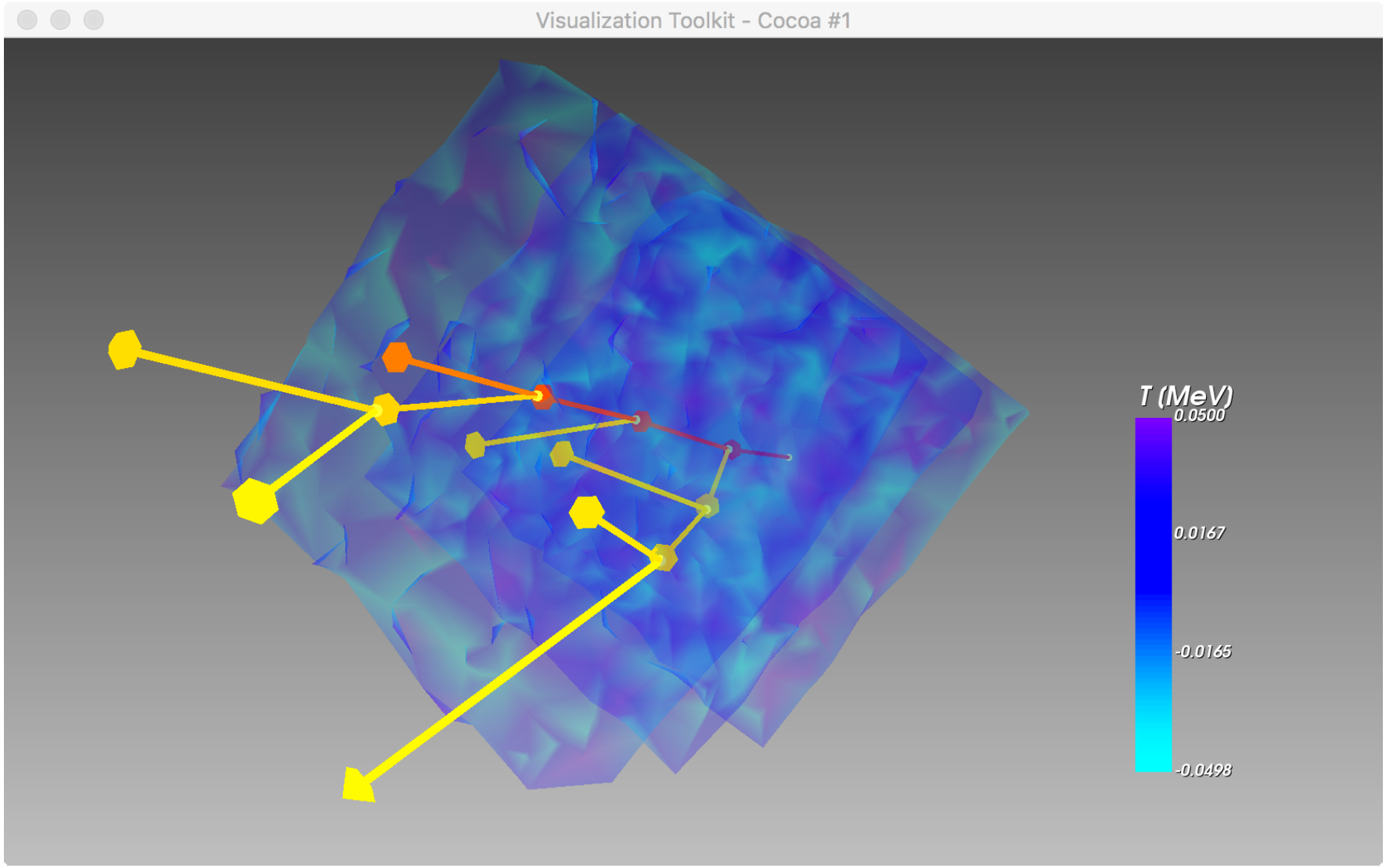
Tasks —> User defined *Modules*

Communication: *Signal/Slots*

Future: (Lego) *Blocks* for further modularization (more later)

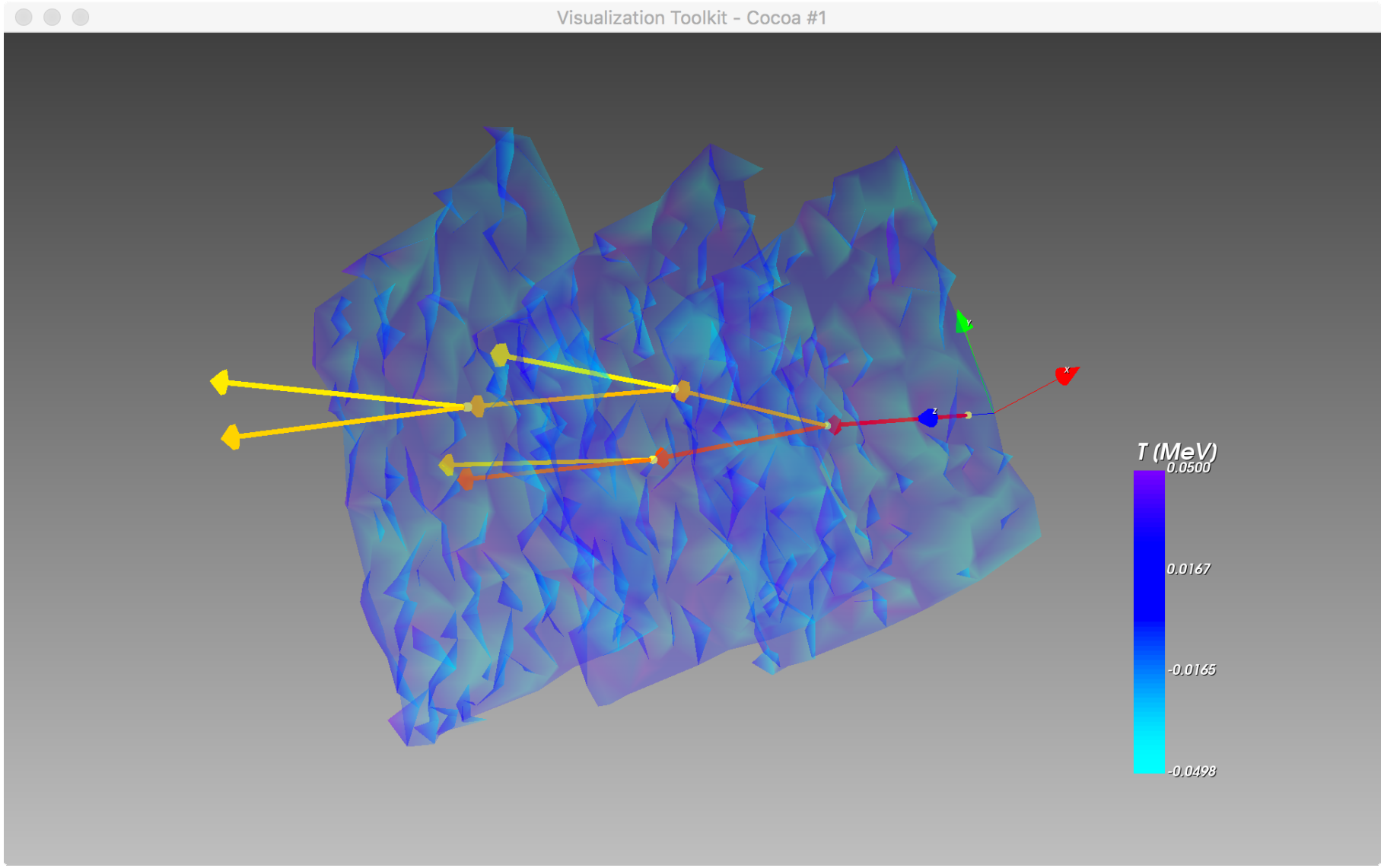
"User friendly": Focus on your physics, leave the rest to the JETSCAPE framework …

Computational: Graphs



For 3d visualization and movies (besides ROOT) we can use VTK!
(Used by hydro and SMASH, so consistent integration)

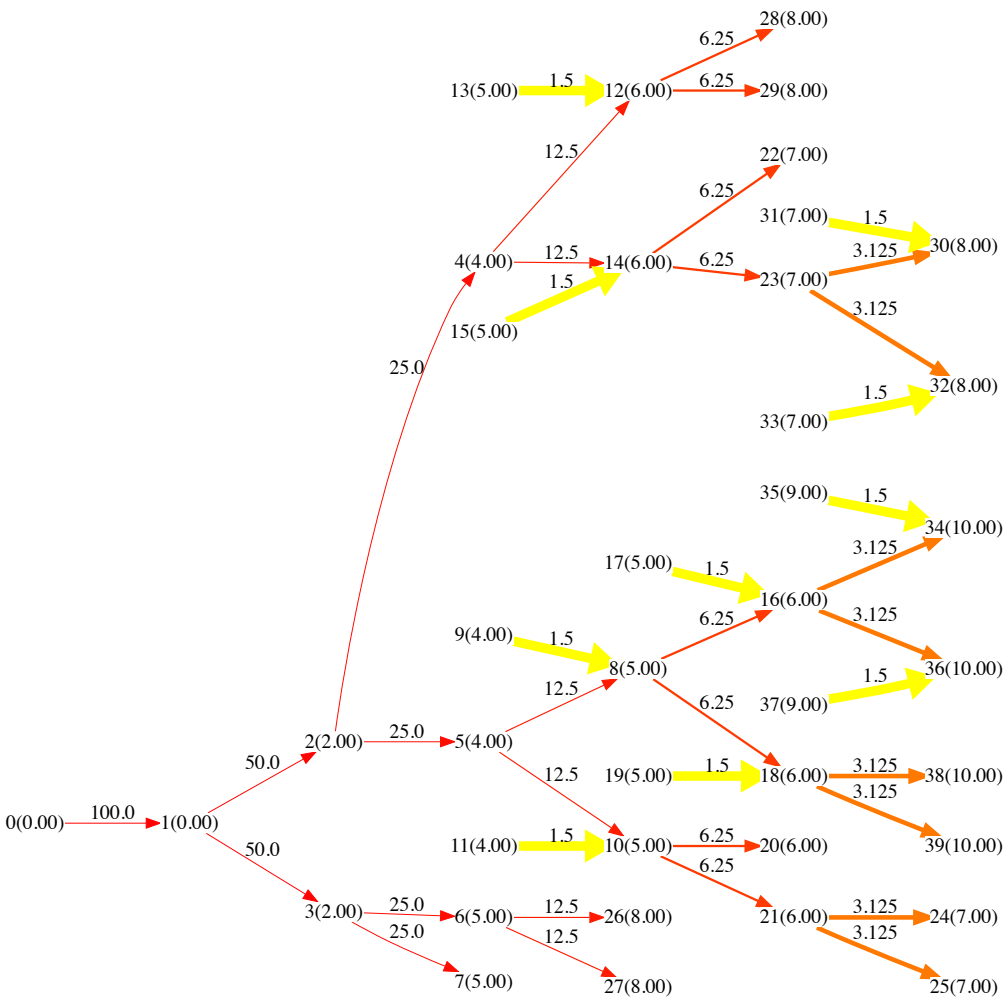
Computational: Graphs



For 3d visualization and movies (besides ROOT) we can use VTK!
(Used by hydro and SMASH, so consistent integration)

Computational: Graphs

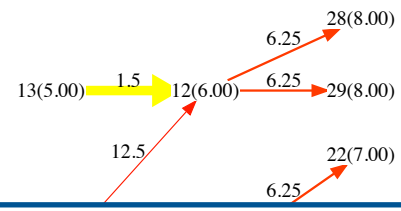
JetScape "toy" Parton Shower as a Graph



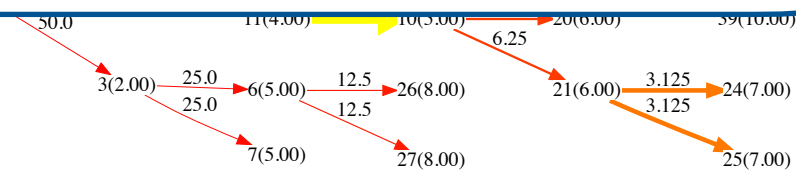
- Internal shower structure is **graph-based** (GTL)
- A lot of very well established mathematical/computer science algorithms to search/analyze/look for similarities between graphs (easy user interface)
- Allow novel approaches to analyze partonic E-loss
- Direct connection to computer science → develop/extend graphs for physics, for example multi-layer graphs …
- Graphs are ideal to "search" for connectivity. What hadron belongs to what parton(s). What part of the shower is affected by "medium" partons/initial state …

Computational: Graphs

JetScape "toy" Parton Shower as a Graph

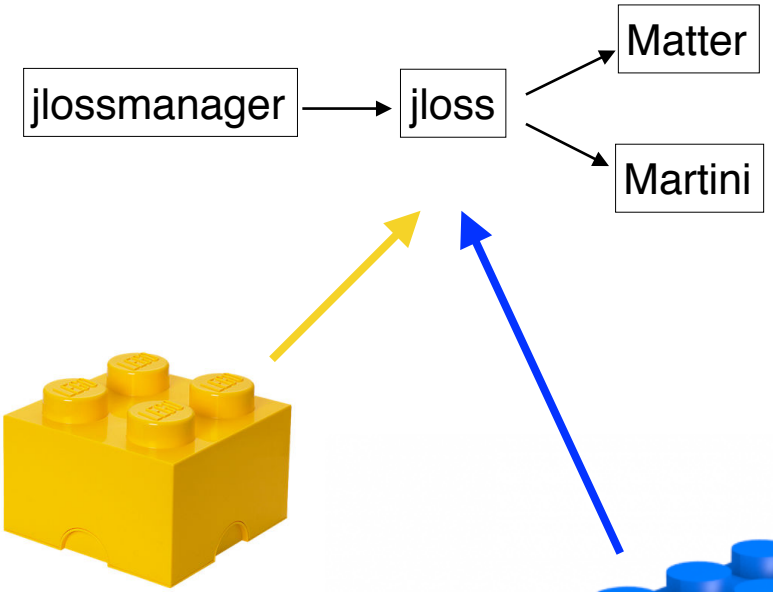


Why stop at the parton shower?
QGP = the social network of QCD?

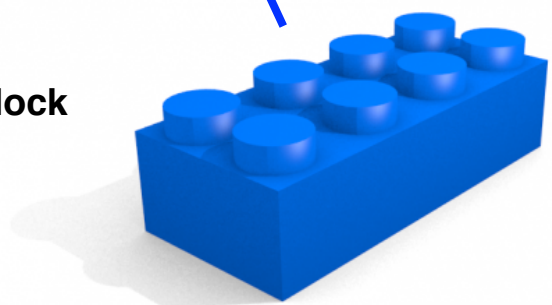


- Internal shower structure is **graph-based** (GTL)
- A lot of very well established mathematical/computer science algorithms to search/analyze/look for similarities between graphs (easy user interface)
- Allow novel approaches to analyze partonic E-loss
- Direct connection to computer science → develop/extend graphs for physics, for example multi-layer graphs …
- Graphs are ideal to "search" for connectivity. What hadron belongs to what parton(s). What part of the shower is affected by "medium" partons/initial state …

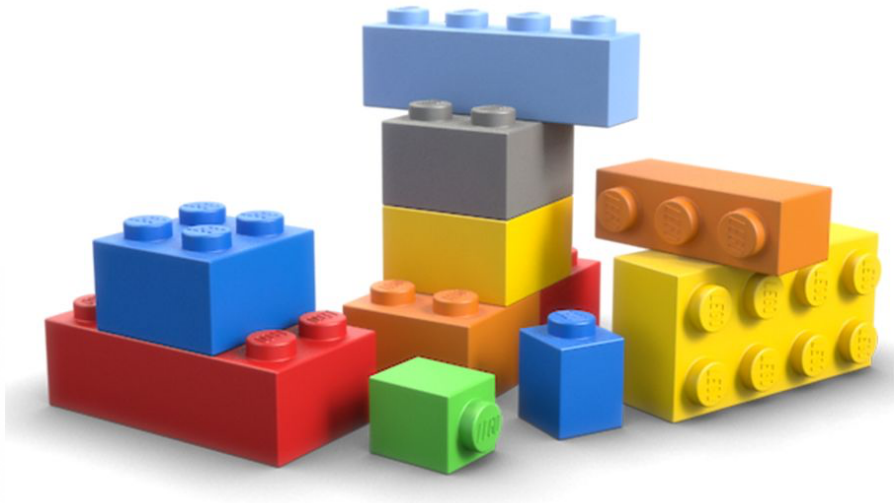
Future: More modular via "JetScape Blocks"/Plugins



Virtuality Calculation Block



Color Coherence/Antenna Block

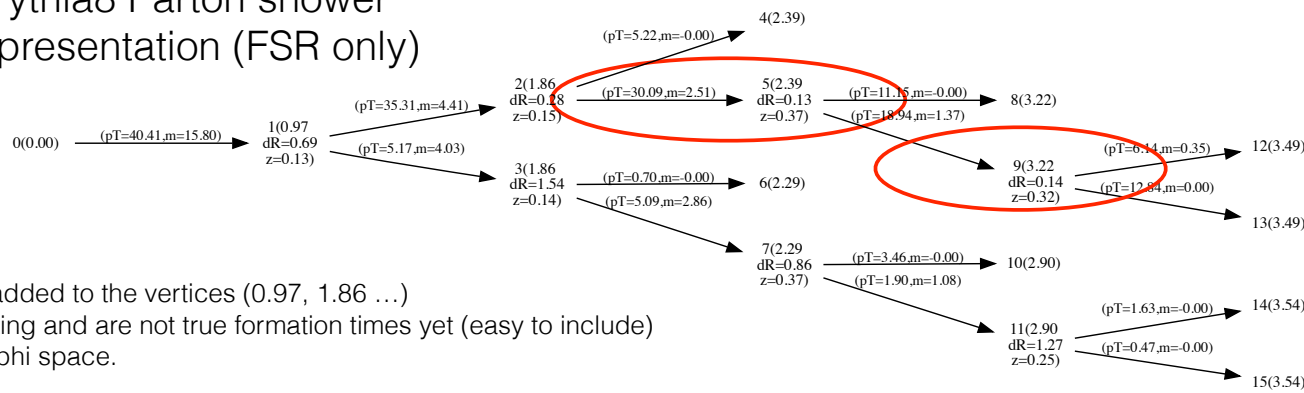


...

Remark: Some of these features will be available for testing in a DEV (developer branch at github)

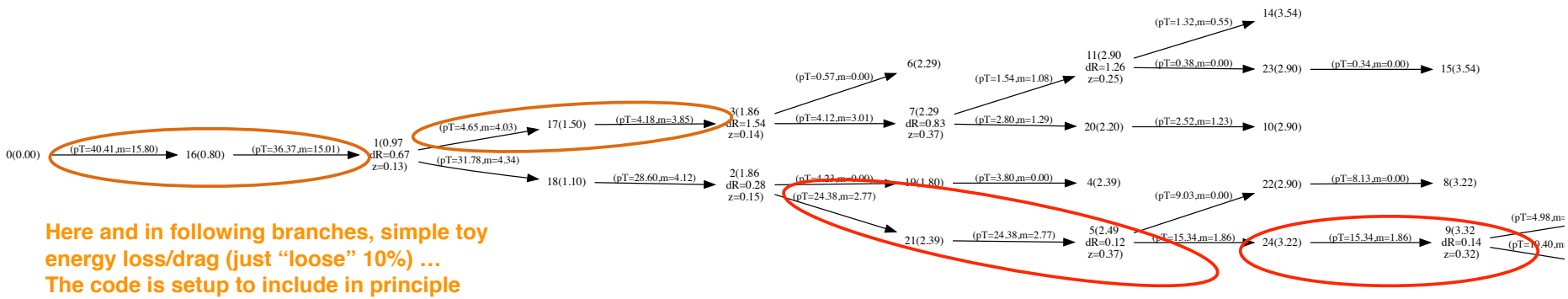
One example (hopefully to be followed up): Coherence …

"Original" Pythia8 Parton shower
in graph representation (FSR only)



Remark: The times added to the vertices (0.97, 1.86 …)
are made up for testing and are not true formation times yet (easy to include)
dR is "angle" in eta/phi space.

We can now traverse the original Pythia8 graph and modify (change time via coherence)
and add (energy loss/drag) accordingly



**Here and in following branches, simple toy energy loss/drag (just "loose" 10%) …
The code is setup to include in principle all Eloss implemented in JetScape, with that said, how to use a Pythia8 shower and include at early times, if needed, splitting to be seen, but generically the code is in principle there.**

**Attempt of a dummy *"coherence module"*:
Just choose "random" numbers for illustration.
If dR<0.15 of a split then "delay" the split by 0.1 in time.
This module has access to all medium/hydro information to include a realistic medium.**

Let me know if you think that this would be a good starting point,
anything obvious missing, suggestions …

Future: Even more modular …

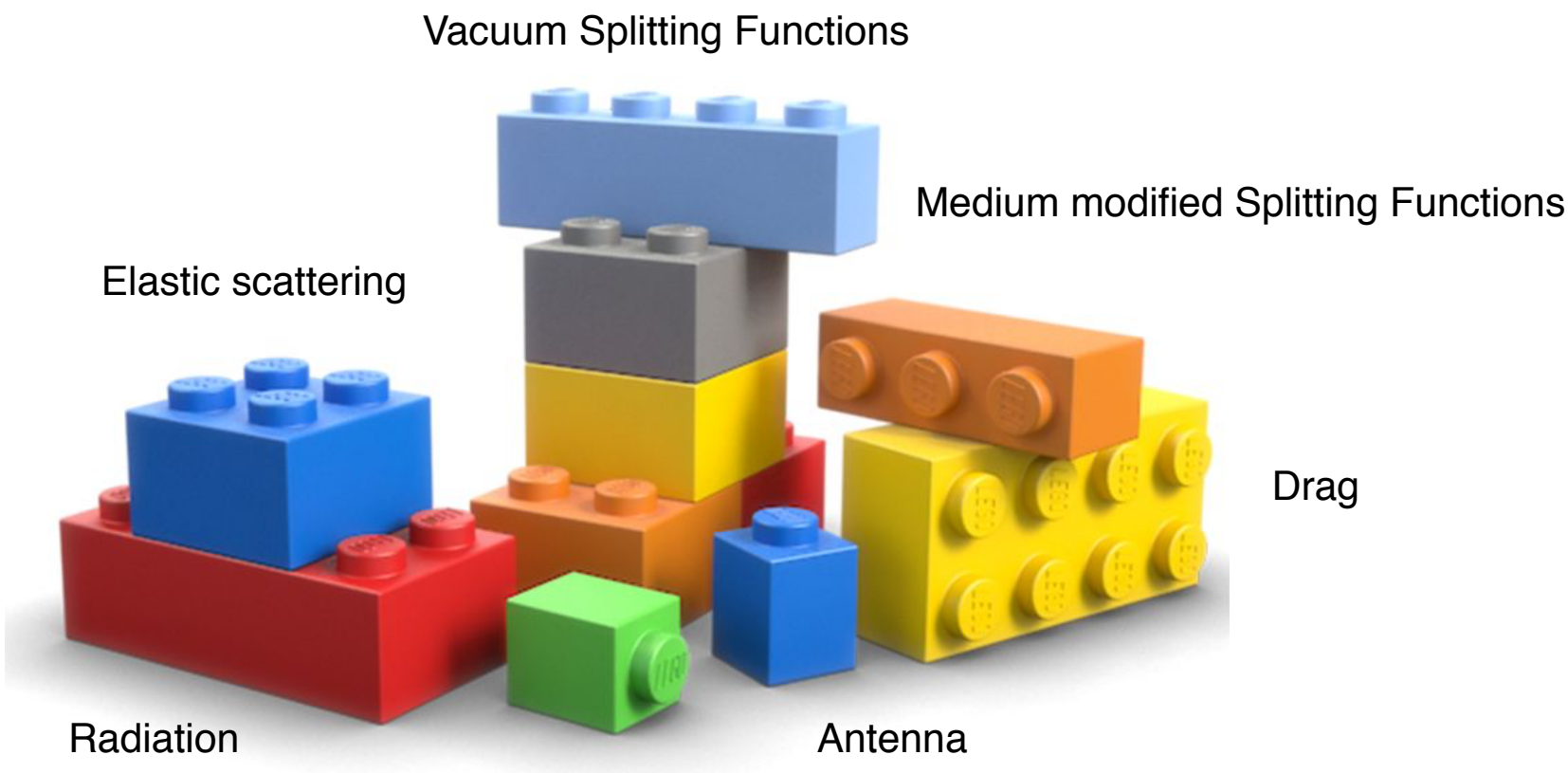
Why would this be desirable?

Minimizes even further “non-physics” related code development!

Even easier to combine different physics mechanisms (simultaneously if needed)!

Easier to develop and test new mechanisms/ideas!

Ensures modern/flexible and safe code!



Assembling/Discussing a wish list …

What is the need from theorists to develop their physics in JetScape?

What do experimentalists need/expect from JetScape?

Any concerns using an overarching framework?
If so, what are they?

I truly think that JetScape will make life significantly easier for exploring/implementing (new) jet quenching physics in MC, but in order to succeed feedback and of course criticism is needed …

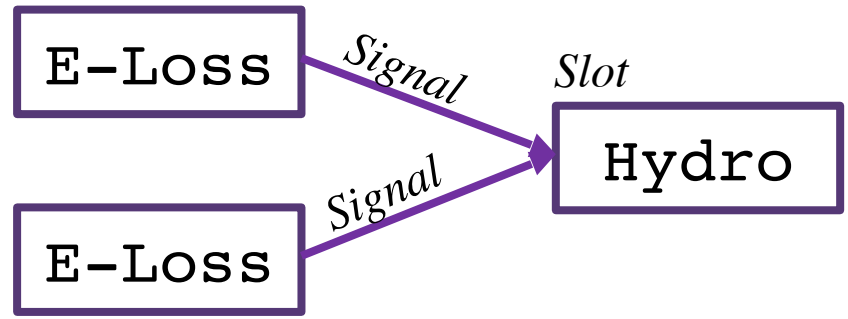
Longer Term Future: JETSCAPE and EIC?

The current JETSCAPE framework is extremely modular and flexible and provides a lot of infrastructure which could be used as the basis for an EIC MC event generator.

It would be helpful to discuss and specify the needs of an EIC MC early on to identify if the JETSCAPE framework/generator could be used for the EIC efforts to provide a uniform code base for high energy nuclear physics in the US.

Advanced Concepts: Code Design

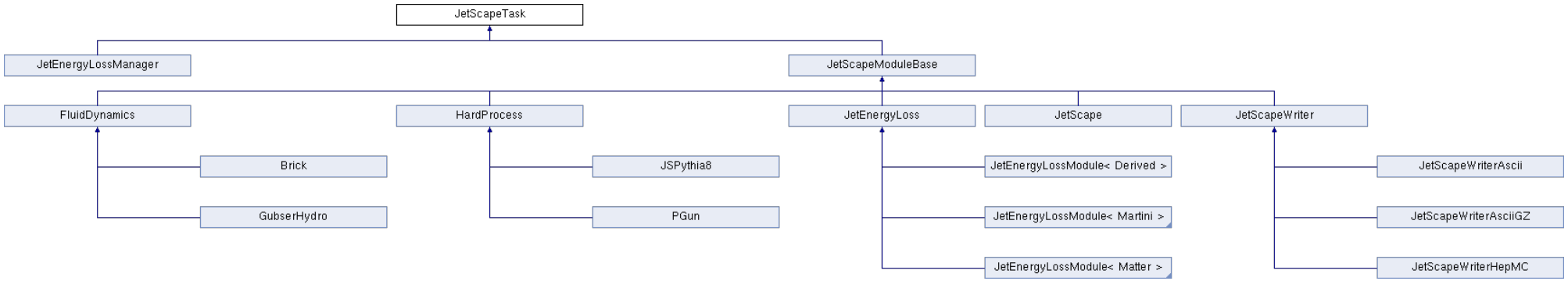
- C++11
  - *Smart pointers* (easy and safe memory management)
  - *Inheritance*
  - *Encapsulation*
- Infrastructure:
  - XML reader
  - thread-safe logger
  - Unified random numbers across modules to ensure reproducibility!
  - Signal Managers
- Communication via ***Signals & Slots***



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

All under the hood!

Advanced Concepts: Tasks

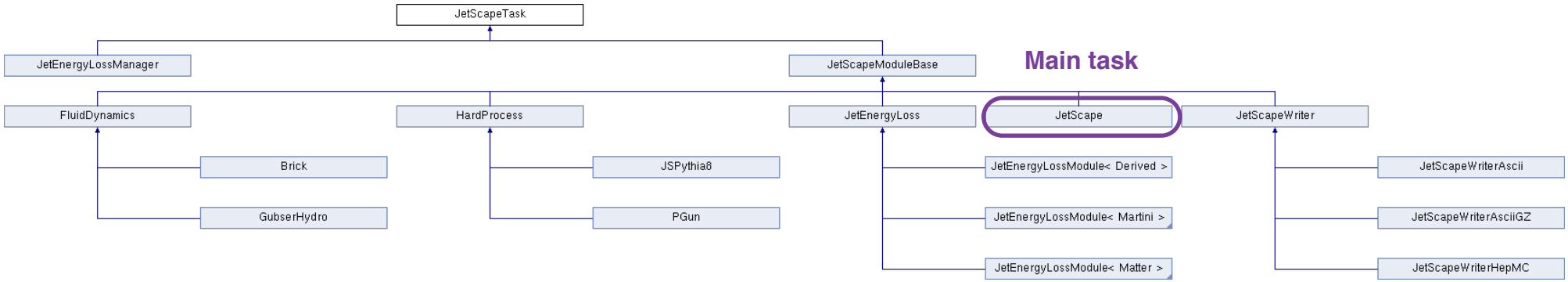


Public Member Functions

	JetScapeTask ()
virtual	**~JetScapeTask** ()
virtual void	**Init** ()
virtual void	**Exec** ()
virtual void	**Finish** ()
virtual void	**Clear** ()
virtual void	**ExecuteTasks** ()
virtual void	**ExecuteTask** ()
virtual void	**InitTask** ()
virtual void	**InitTasks** ()
virtual void	**ClearTasks** ()
virtual void	**ClearTask** ()
virtual void	**FinishTask** ()
virtual void	**FinishTasks** ()
virtual void	**WriteTasks** (weak_ptr< **JetScapeWriter** > w)
virtual void	**WriteTask** (weak_ptr< **JetScapeWriter** > w)
virtual void	**Add** (shared_ptr< **JetScapeTask** > m_tasks)
const vector< shared_ptr< **JetScapeTask** > >	**GetTaskList** () const
shared_ptr< **JetScapeTask** >	**GetTaskAt** (int i)
void	**EraseTaskLast** ()
void	**EraseTaskAt** (int i)
void	**ResizeTaskList** (int i)
void	**ClearTaskList** ()
int	**GetNumberOfTasks** ()
const bool	**GetActive** () const
void	**SetActive** (bool m_active_exec)
void	**SetId** (string m_id)
const string	**GetId** () const

All of theses functions will be called recursively in a task based framework, so once added there is no problem that it will get initialized, executed (if specified) … Of course a task in itself is a natural unit which can be easily parallelized if necessary.

Advanced Concepts: Tasks

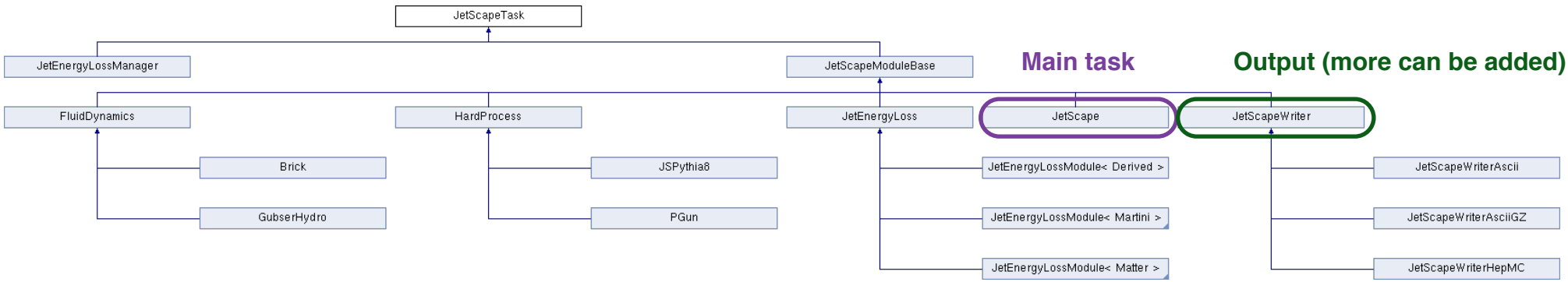


Public Member Functions

	JetScapeTask ()
virtual	**~JetScapeTask** ()
virtual void	**Init** ()
virtual void	**Exec** ()
virtual void	**Finish** ()
virtual void	**Clear** ()
virtual void	**ExecuteTasks** ()
virtual void	**ExecuteTask** ()
virtual void	**InitTask** ()
virtual void	**InitTasks** ()
virtual void	**ClearTasks** ()
virtual void	**ClearTask** ()
virtual void	**FinishTask** ()
virtual void	**FinishTasks** ()
virtual void	**WriteTasks** (weak_ptr< **JetScapeWriter** > w)
virtual void	**WriteTask** (weak_ptr< **JetScapeWriter** > w)
virtual void	**Add** (shared_ptr< **JetScapeTask** > m_tasks)
const vector< shared_ptr< **JetScapeTask** > >	**GetTaskList** () const
shared_ptr< **JetScapeTask** >	**GetTaskAt** (int i)
void	**EraseTaskLast** ()
void	**EraseTaskAt** (int i)
void	**ResizeTaskList** (int i)
void	**ClearTaskList** ()
int	**GetNumberOfTasks** ()
const bool	**GetActive** () const
void	**SetActive** (bool m_active_exec)
void	**SetId** (string m_id)
const string	**GetId** () const

All of theses functions will be called recursively in a task based framework, so once added there is no problem that it will get initialized, executed (if specified) … Of course a task in itself is a natural unit which can be easily parallelized if necessary.

Advanced Concepts: Tasks

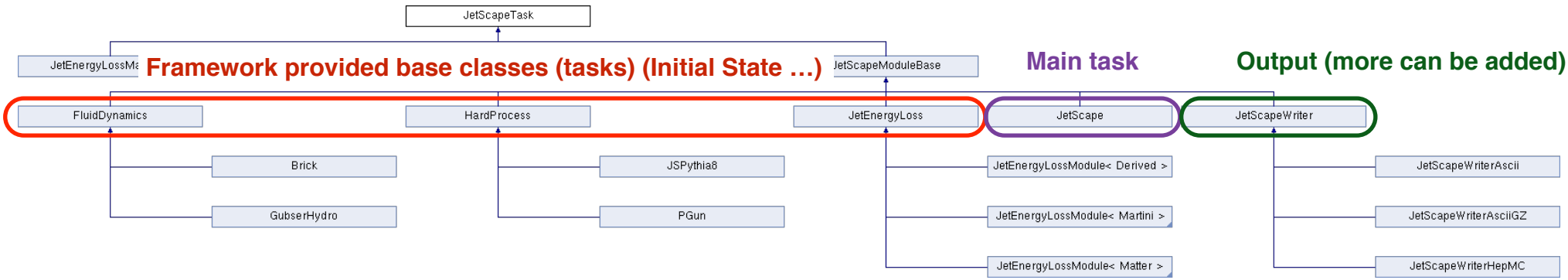


Public Member Functions

	JetScapeTask ()
virtual	**~JetScapeTask** ()
virtual void	**Init** ()
virtual void	**Exec** ()
virtual void	**Finish** ()
virtual void	**Clear** ()
virtual void	**ExecuteTasks** ()
virtual void	**ExecuteTask** ()
virtual void	**InitTask** ()
virtual void	**InitTasks** ()
virtual void	**ClearTasks** ()
virtual void	**ClearTask** ()
virtual void	**FinishTask** ()
virtual void	**FinishTasks** ()
virtual void	**WriteTasks** (weak_ptr< **JetScapeWriter** > w)
virtual void	**WriteTask** (weak_ptr< **JetScapeWriter** > w)
virtual void	**Add** (shared_ptr< **JetScapeTask** > m_tasks)
const vector< shared_ptr< **JetScapeTask** > >	**GetTaskList** () const
shared_ptr< **JetScapeTask** >	**GetTaskAt** (int i)
void	**EraseTaskLast** ()
void	**EraseTaskAt** (int i)
void	**ResizeTaskList** (int i)
void	**ClearTaskList** ()
int	**GetNumberOfTasks** ()
const bool	**GetActive** () const
void	**SetActive** (bool m_active_exec)
void	**SetId** (string m_id)
const string	**GetId** () const

All of theses functions will be called recursively in a task based framework, so once added there is no problem that it will get initialized, executed (if specified) … Of course a task in itself is a natural unit which can be easily parallelized if necessary.

Advanced Concepts: Tasks

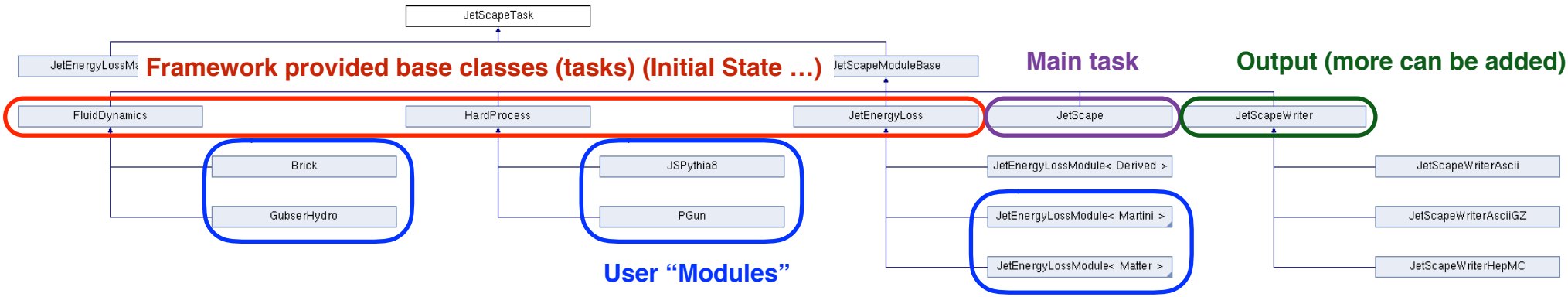


Public Member Functions

	JetScapeTask ()
virtual	**~JetScapeTask** ()
virtual void	**Init** ()
virtual void	**Exec** ()
virtual void	**Finish** ()
virtual void	**Clear** ()
virtual void	**ExecuteTasks** ()
virtual void	**ExecuteTask** ()
virtual void	**InitTask** ()
virtual void	**InitTasks** ()
virtual void	**ClearTasks** ()
virtual void	**ClearTask** ()
virtual void	**FinishTask** ()
virtual void	**FinishTasks** ()
virtual void	**WriteTasks** (weak_ptr< **JetScapeWriter** > w)
virtual void	**WriteTask** (weak_ptr< **JetScapeWriter** > w)
virtual void	**Add** (shared_ptr< **JetScapeTask** > m_tasks)
const vector< shared_ptr< **JetScapeTask** > >	**GetTaskList** () const
shared_ptr< **JetScapeTask** >	**GetTaskAt** (int i)
void	**EraseTaskLast** ()
void	**EraseTaskAt** (int i)
void	**ResizeTaskList** (int i)
void	**ClearTaskList** ()
int	**GetNumberOfTasks** ()
const bool	**GetActive** () const
void	**SetActive** (bool m_active_exec)
void	**SetId** (string m_id)
const string	**GetId** () const

All of theses functions will be called recursively in a task based framework, so once added there is no problem that it will get initialized, executed (if specified) … Of course a task in itself is a natural unit which can be easily parallelized if necessary.

Advanced Concepts: Tasks

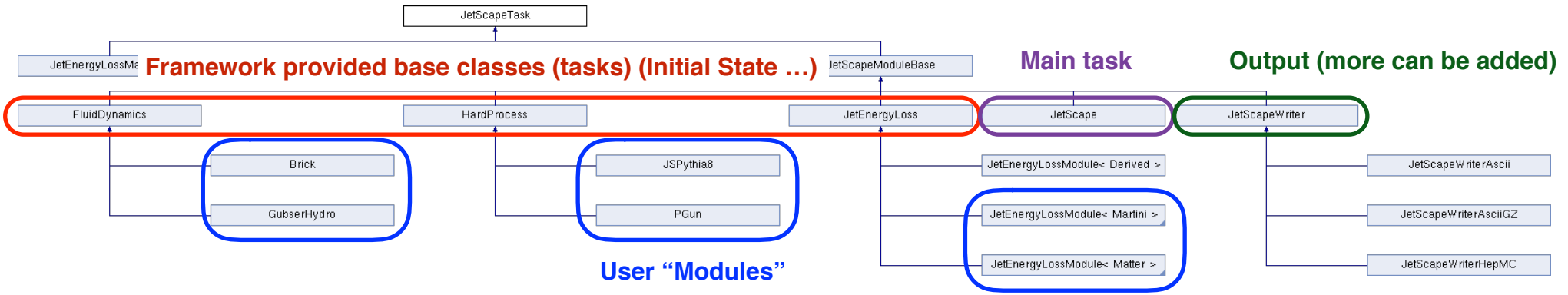


Public Member Functions

	JetScapeTask ()
virtual	**~JetScapeTask** ()
virtual void	**Init** ()
virtual void	**Exec** ()
virtual void	**Finish** ()
virtual void	**Clear** ()
virtual void	**ExecuteTasks** ()
virtual void	**ExecuteTask** ()
virtual void	**InitTask** ()
virtual void	**InitTasks** ()
virtual void	**ClearTasks** ()
virtual void	**ClearTask** ()
virtual void	**FinishTask** ()
virtual void	**FinishTasks** ()
virtual void	**WriteTasks** (weak_ptr< **JetScapeWriter** > w)
virtual void	**WriteTask** (weak_ptr< **JetScapeWriter** > w)
virtual void	**Add** (shared_ptr< **JetScapeTask** > m_tasks)
const vector< shared_ptr< **JetScapeTask** > >	**GetTaskList** () const
shared_ptr< **JetScapeTask** >	**GetTaskAt** (int i)
void	**EraseTaskLast** ()
void	**EraseTaskAt** (int i)
void	**ResizeTaskList** (int i)
void	**ClearTaskList** ()
int	**GetNumberOfTasks** ()
const bool	**GetActive** () const
void	**SetActive** (bool m_active_exec)
void	**SetId** (string m_id)
const string	**GetId** () const

All of theses functions will be called recursively in a task based framework, so once added there is no problem that it will get initialized, executed (if specified) … Of course a task in itself is a natural unit which can be easily parallelized if necessary.

Advanced Concepts: Tasks

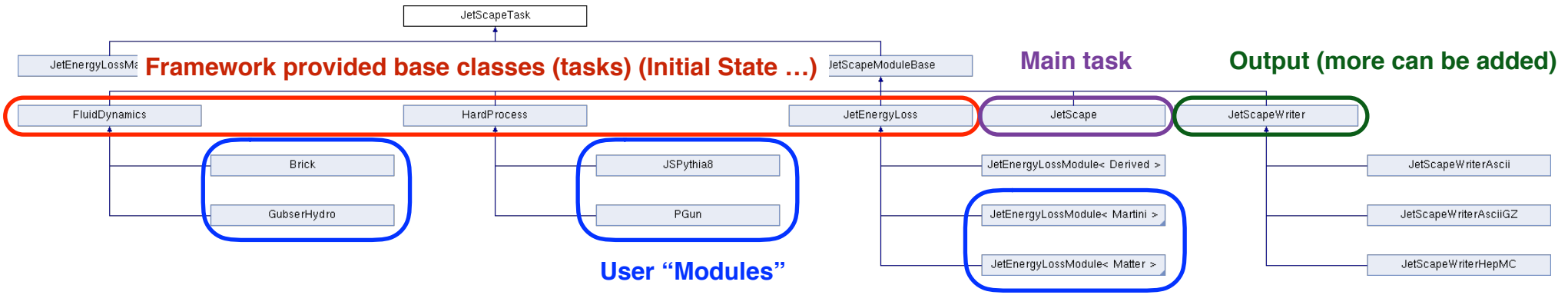


Public Member Functions

	JetScapeTask ()
virtual	**~JetScapeTask** ()
virtual void	**Init** ()
virtual void	**Exec** ()
virtual void	**Finish** ()
virtual void	**Clear** ()
virtual void	**ExecuteTasks** ()
virtual void	**ExecuteTask** ()
virtual void	**InitTask** ()
virtual void	**InitTasks** ()
virtual void	**ClearTasks** ()
virtual void	**ClearTask** ()
virtual void	**FinishTask** ()
virtual void	**FinishTasks** ()
virtual void	**WriteTasks** (weak_ptr< **JetScapeWriter** > w)
virtual void	**WriteTask** (weak_ptr< **JetScapeWriter** > w)
virtual void	**Add** (shared_ptr< **JetScapeTask** > m_tasks)
const vector< shared_ptr< **JetScapeTask** > >	**GetTaskList** () const
shared_ptr< **JetScapeTask** >	**GetTaskAt** (int i)
void	**EraseTaskLast** ()
void	**EraseTaskAt** (int i)
void	**ResizeTaskList** (int i)
void	**ClearTaskList** ()
int	**GetNumberOfTasks** ()
const bool	**GetActive** () const
void	**SetActive** (bool m_active_exec)
void	**SetId** (string m_id)
const string	**GetId** () const

```
int main(int argc, char** argv)
{
  …

  auto jetscape = make_shared<JetScape>("./jetscape_init.xml",3);

  auto jlossmanager = make_shared<JetEnergyLossManager> ();
  auto jloss = make_shared<JetEnergyLoss> ();
  auto matter = make_shared<Matter> ();
  auto martini = make_shared<Martini> ();

  auto pGun= make_shared<PGun> ();
  auto hydro = make_shared<Brick> ();
  auto writer= make_shared<JetScapeWriterAscii> ("test_out.dat");

  jetscape->Add(pGun);

  jetscape->Add(hydro);

  jloss->Add(matter);
  jloss->Add(martini);
  jlossmanager->Add(jloss);

  jetscape->Add(jlossmanager);

  jetscape->Add(writer);


  // Intialize all modules tasks
  jetscape->Init();

  // Run JetScape with all task/modules as specified ...
  jetscape->Exec();

  jetscape->Finish();

  …
}
```

Advanced Concepts: Tasks



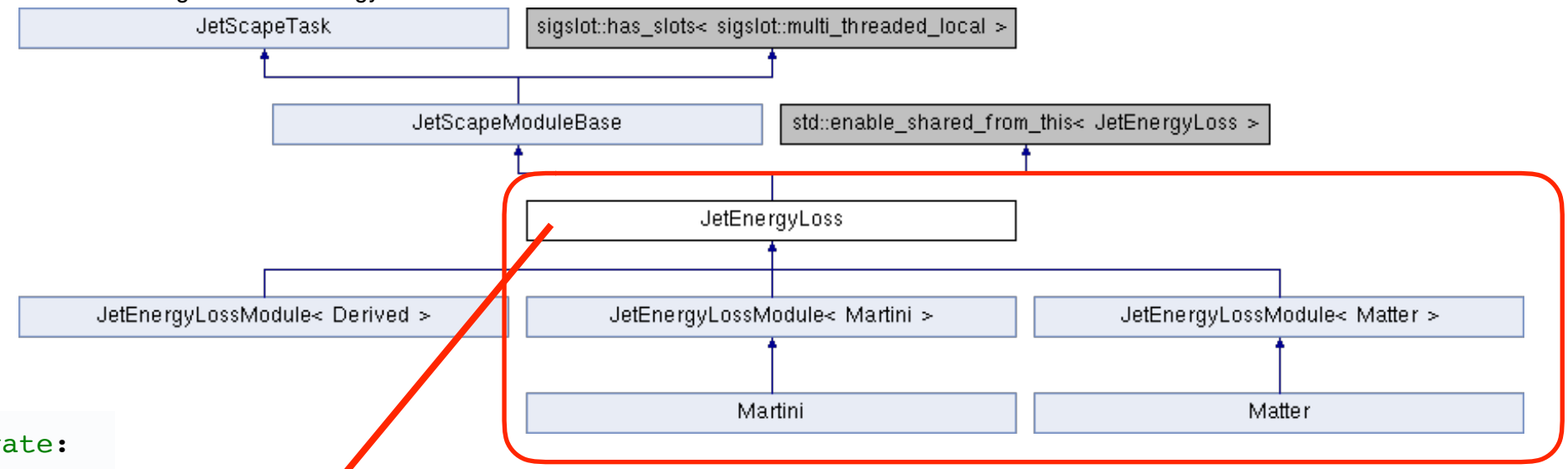
Public Member Functions

	JetScapeTask ()
virtual	**~JetScapeTask** ()
virtual void	**Init** ()
virtual void	**Exec** ()
virtual void	**Finish** ()
virtual void	**Clear** ()
virtual void	**ExecuteTasks** ()
virtual void	**ExecuteTask** ()
virtual void	**InitTask** ()
virtual void	**InitTasks** ()
virtual void	**ClearTasks** ()
virtual void	**ClearTask** ()
virtual void	**FinishTask** ()
virtual void	**FinishTasks** ()
virtual void	**WriteTasks** (weak_ptr< **JetScapeWriter** > w)
virtual void	**WriteTask** (weak_ptr< **JetScapeWriter** > w)
virtual void	**Add** (shared_ptr< **JetScapeTask** > m_tasks)
const vector< shared_ptr< **JetScapeTask** > >	**GetTaskList** () const
shared_ptr< **JetScapeTask** >	**GetTaskAt** (int i)
void	**EraseTaskLast** ()
void	**EraseTaskAt** (int i)
void	**ResizeTaskList** (int i)
void	**ClearTaskList** ()
int	**GetNumberOfTasks** ()
const bool	**GetActive** () const
void	**SetActive** (bool m_active_exec)
void	**SetId** (string m_id)
const string	**GetId** () const

```
int main(int argc, char** argv)
{
  …
  auto jetscape = make_shared<JetScape>("./jetscape_init.xml",3);
  auto jlossmanager = make_shared<JetEnergyLossManager> ();
  auto jloss = make_shared<JetEnergyLoss> ();
  auto matter = make_shared<Matter> ();
  auto martini = make_shared<Martini> ();

  auto pGun= make_shared<PGun> ();
  auto hydro = make_shared<Brick> ();
  auto writer= make_shared<JetScapeWriterAscii> ("test_out.dat");

  jetscape->Add(pGun);
  jetscape->Add(hydro);
  jloss->Add(matter);
  jloss->Add(martini);
  jlossmanager->Add(jloss);

  jetscape->Add(jlossmanager);
  jetscape->Add(writer);

  // Intialize all modules tasks
  jetscape->Init();

  // Run JetScape with all task/modules as specified ...
  jetscape->Exec();

  jetscape->Finish();

  …
}
```

Focus on: “Do” Parton Shower in JetScapeEnergyLoss

Inheritance diagram for JetEnergyLoss:

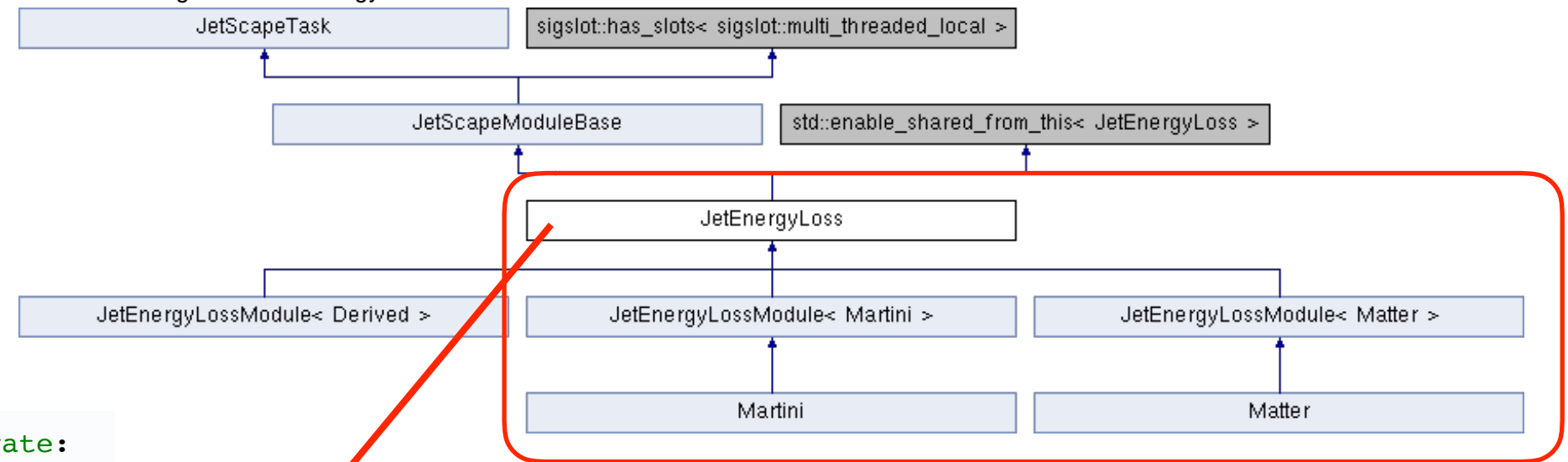



```
private:
...
shared_ptr<PartonShower> pShower;
node vStart;
node vEnd;
void DoShower();
```

(n) —> (m) parton splitting in time steps deltaT

Focus on: “Do” Parton Shower in JetScapeEnergyLoss

Inheritance diagram for JetEnergyLoss:




```
private:
...
shared_ptr<PartonShower> pShower;
node vStart;
node vEnd;
void DoShower();
```

(n) —> (m) parton splitting
in time steps deltaT

DoShower() sends a Signal

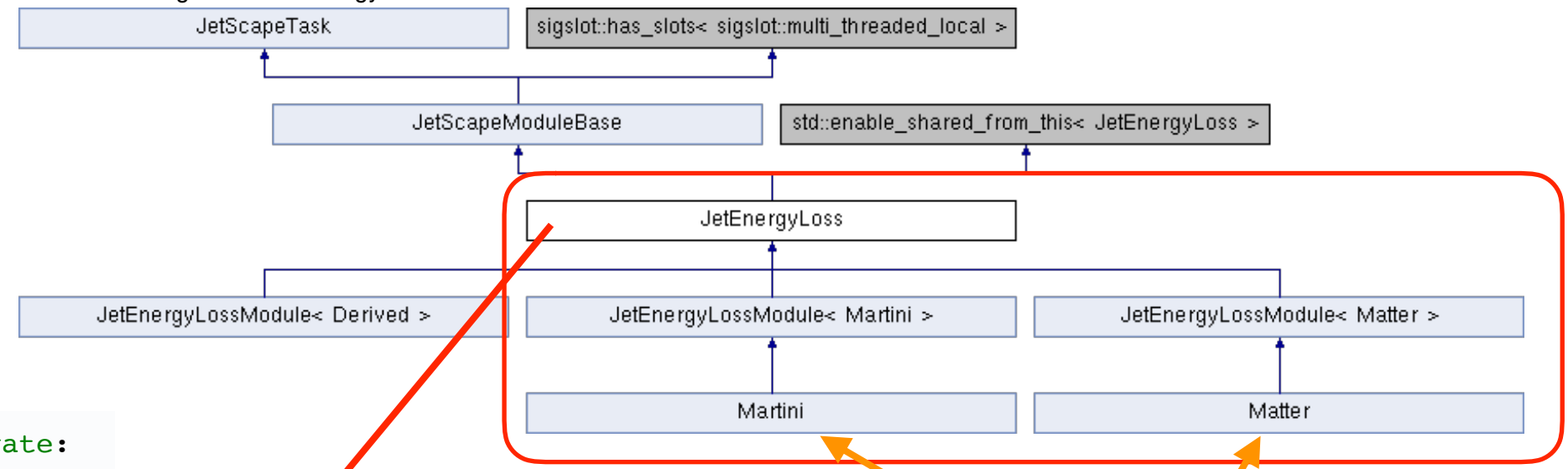
◆ SentInPartons

sigslot::signal4<double, double, vector<**Parton**>&, vector<**Parton**>&, multi_threaded_local> JetEnergyLoss::SentInPartons

Definition at line **56** of file **JetEnergyLoss.h**.

Focus on: “Do” Parton Shower in JetScapeEnergyLoss

Inheritance diagram for JetEnergyLoss:




```
private:
...
shared_ptr<PartonShower> pShower;
node vStart;
node vEnd;
void DoShower();
```

(n) —> (m) parton splitting in time steps deltaT

DoShower() sends a Signal

◆ SentInPartons

sigslot::signal4<double, double, vector<**Parton**>&, vector<**Parton**>&, multi_threaded_local> JetEnergyLoss::SentInPartons

Definition at line **56** of file **JetEnergyLoss.h**.

Slot(s)

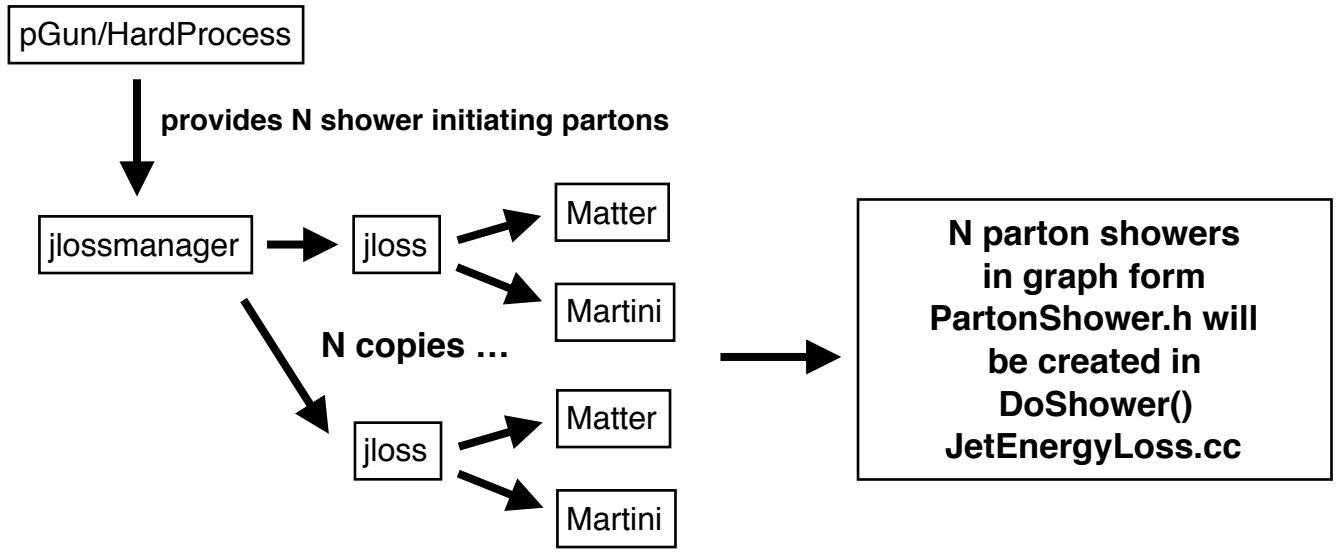
virtual void JetEnergyLoss::DoEnergyLoss (double deltaT,
double Q2,
vector< **Parton** > & pIn,
vector< **Parton** > & pOut
)

Reimplemented in **Martini**, and **Matter**.

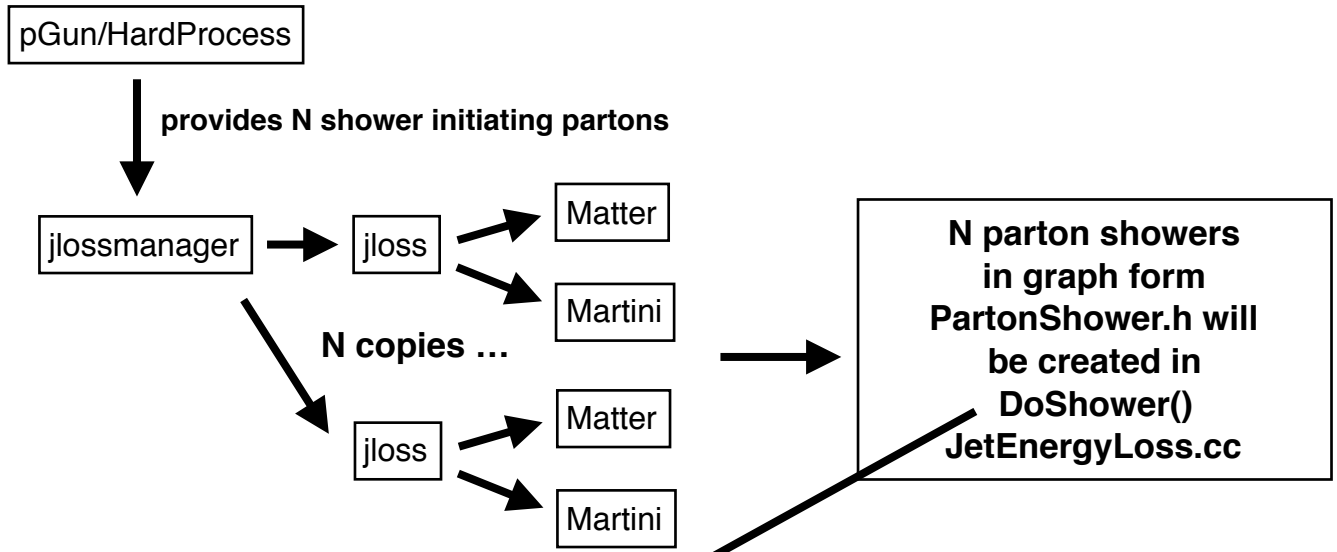
Definition at line **41** of file **JetEnergyLoss.h**.

JetEnergyLoss has signal/slots with Hydro: get energy density and sent energy deposition

In more detail: DoShower() and Graph Generation …



In more detail: DoShower() and Graph Generation …




```
void::JetEnergyLoss::DoShower()
{
  vector<Parton> pIn; vector<Parton> pOut;
   …
   do
    {
      for (int i=0;i<pIn.size();i++)
      {
       …
       SentInPartons(currentTime,pIn[i].pt(),pInTempModule,pOutTemp); //Signal
       …
         for (int k=0;k<pOutTemp.size();k++)
         {
           vEnd=pShower->new_vertex(make_shared<VertexBase>(0,0,0,currentTime));
           pShower->new_parton(vStart,vEnd,make_shared<Parton>(pOutTemp[k]));
           …
         }
    }
   while (currentTime<maxT);
}
```

vector of partons used in DoShower() to recursively create the shower (Graph structure not directly used, "just" for storage)

Only vector of partons as input/output for loss modules

Dynamic generation of graph structure (Parton Shower.h) if something "happened" in an energy loss module …

Looks complicated, but “User” does not need to know and care …

A specific (only relevant functionalities for your module), clean and clear (developer) “end-user” interface, take “Matter” for example:

```
class Matter : public JetEnergyLossModule<Matter>
{
 public:

  Matter();
  virtual ~Matter();

  void Init();

  void DoEnergyLoss(double deltaT, double Q2,
    vector<Parton>& pIn, vector<Parton>& pOut);

  void WriteTask(weak_ptr<JetScapeWriter> w);

 private:

};
```

Optional, only if you want to write out additional informations. The graph/partons will be written by the base class.

Everything else is handled and hidden from the user. You only have to know and care/understand your module: “Partons” and a way to get and sent information to hydro, these interfaces are defined by us.
No additional confusion and exposure to data/implementation of the framework

—> **Safety!** The user can not (by accident) override/change anything in other modules (event class …), only access via our interfaces!

Looks complicated, but “User” does not need to know and care …

A specific (or
(developer) “

```
class Matter :
{
 public:

  Matter();
  virtual ~Matt

  void Init();

  void DoEnergy
    vector<Part

  void WriteTas

 private:

};
```

Everything el
and care/unc
information t
No additiona

—> Safety!
other mo

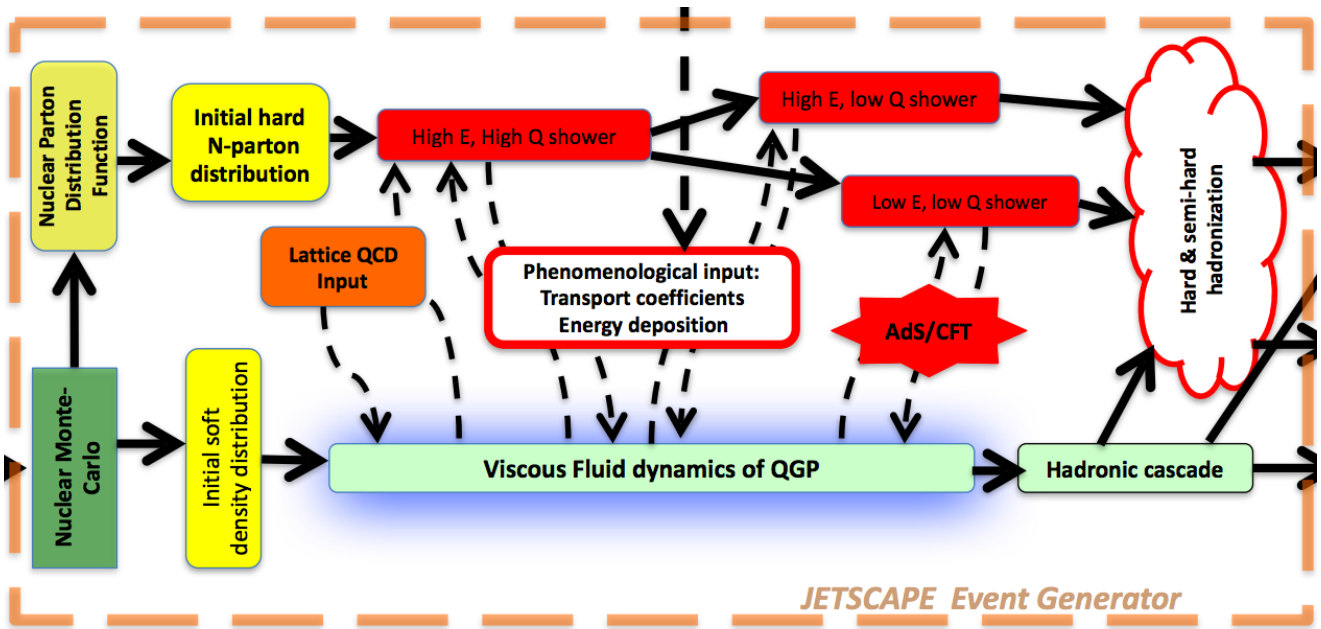
```
void Matter::Init()
{
  INFO<<"Intialize Matter ...";

  tinyxml2::XMLElement *eloss= JetScapeXML::Instance()->GetXMLRoot()->FirstChildElement("Eloss" );
  tinyxml2::XMLElement *matter=eloss->FirstChildElement("Matter");
  …
}
…
void Matter::DoEnergyLoss(double deltaT, double Q2, vector<Parton>& pIn, vector<Parton>& pOut)
{
  double z=0.5;

  if (Q2>5)
    {
      VERBOSESHOWER(8)<< MAGENTA << "SentInPartons Signal received : "<<deltaT<<" "<<Q2<<" "<<&pIn;

      FluidCellInfo* check_fluid_info_ptr = new FluidCellInfo;
      GetHydroCellSignal(1, 1.0, 1.0, 0.0, check_fluid_info_ptr);

      VERBOSE(8)<< MAGENTA<<"Temperature from Brick (Signal) = "<<check_fluid_info_ptr->temperature;

      delete check_fluid_info_ptr;

      double rNum;

      for (int i=0;i<pIn.size();i++)
        {
          rNum=distribution(generator);

          // simulate a "random" split 50/50 in pT
          if (rNum>0.7)
            {
              double newPt=pIn[i].pt()*z;
              double newPt2=pIn[i].pt()*(1-z);

              pOut.push_back(Parton(0,21,0,newPt,pIn[i].eta(),pIn[i].phi(),newPt));
              pOut.push_back(Parton(0,21,0,newPt2,pIn[i].eta(),pIn[i].phi(),newPt));
            }
        }

      if (rNum>0.9)
        {
          pIn.push_back(Parton(0,21,0,1.5,0,pIn[0].phi(),1.5);
        }
}
…
```

Toy splitting, no physics for testing/demo.

From a Users perspective: YOUR PHYSICS GOES HERE!

Program: JETSCAPE 1.0 RC1

What's included in the event generator (RC1)
(as optional download)



- ✓ Trento (2+1)
- ✓ Free Streaming
- ✓ MUSIC (2+1, 3+1), external reader, brick, Gubser,
- ✓ Pythia8, parton gun
- ✓ MATTER, Martini, AdS/CFT, LBT
- ✓ Cooper Frye
- ✓ Pythia8 string fragmentation
- ✓ Custom and HepMC output

Getting Started ...

- Download - JETSCAPE is now available on github! www.github.com/JETSCAPE/JETSCAPE
- Few prerequisites, as little as C++11 support, cmake v3, zlib, Pythia8
- Build in build directory
- Run
- Analyze!

```
% mkdir JETSCAPE/build
% cd JETSCAPE/build
% cmake .. && make -j 4
% ./PythiaBrickTest
% ./readerTest
```

- Give feedback: jetscape.org@gmail.com, bugs.jetscape@gmail.com, https://github.com/JETSCAPE/JETSCAPE/issues
- Get involved!

```
class MyEloss : public JetEnergyLossModule<MyEloss>
void DoEnergyLoss(){…
```

Getting Started ...

- Download - JETSCAPE is now available on github! www.github.com/JETSCAPE/JETSCAPE
- Few prerequisites, as little as C++11 support, cmake v3, zlib, Pythia8
- Build in build directory
- Run
- Analyze!

```
% mkdir JETSCAPE/build
% cd JETSCAPE/build
% cmake .. && make -j 4
% ./PythiaBrickTest
% ./readerTest
```

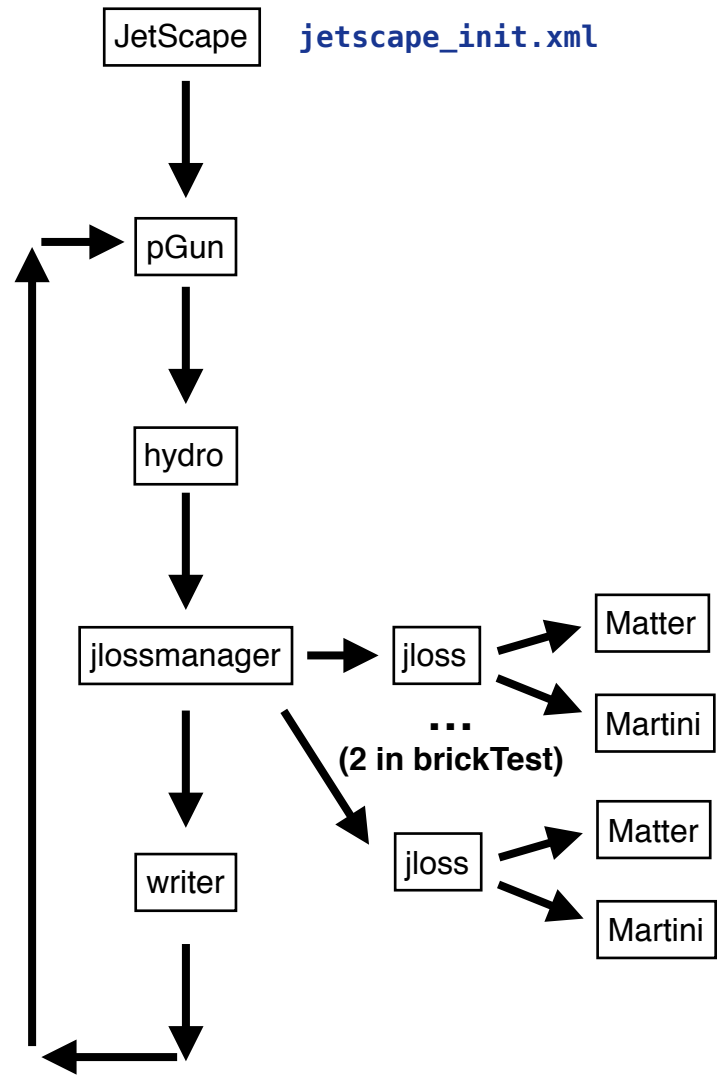
- Give feedback: jetscape.org@gmail.com, bugs.jetscape@gmail.com, https://github.com/JETSCAPE/JETSCAPE/issues
- Get involved!

```
class MyEloss : public JetEnergyLossModule<MyEloss>
void DoEnergyLoss(){…
```

Your input and feedback is needed!

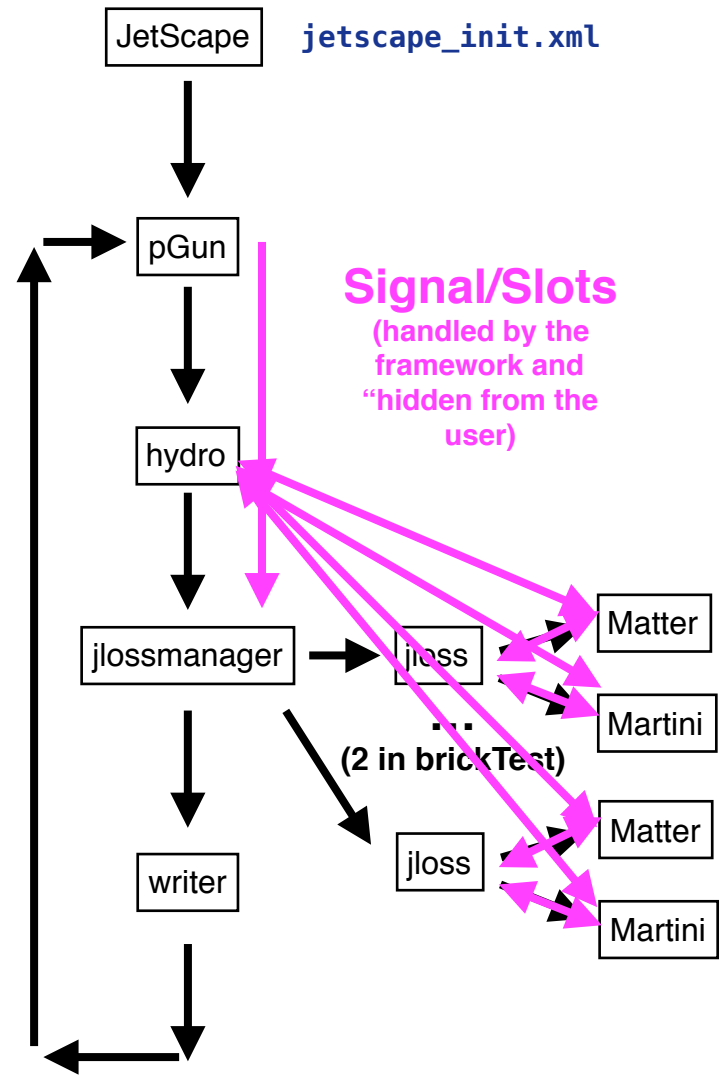
Running JETSCAPE ...

```
int main(int argc, char** argv)
{
  …
  JetScapeLogger::Instance()->SetDebug(false);
  JetScapeLogger::Instance()->SetVerboseLevel(0);

  auto jetscape = make_shared<JetScape>("./jetscape_init.xml",3);

  auto jlossmanager = make_shared<JetEnergyLossManager> ();
  auto jloss = make_shared<JetEnergyLoss> ();
  auto matter = make_shared<Matter> ();
  auto martini = make_shared<Martini> ();

  auto pGun= make_shared<PGun> ();
  auto hydro = make_shared<Brick> ();
  auto writer= make_shared<JetScapeWriterAscii> ("test_out.dat");

  jetscape->Add(pGun);

  jetscape->Add(hydro);

  jloss->Add(matter);
  jloss->Add(martini);
  jlossmanager->Add(jloss);

  jetscape->Add(jlossmanager);

  jetscape->Add(writer);


  // Intialize all modules tasks
  jetscape->Init();

  // Run JetScape with all task/modules as specified ...
  jetscape->Exec();

  jetscape->Finish();

  …
  return 0;
}
```



Running JETSCAPE ...

```
int main(int argc, char** argv)
{
  …
  JetScapeLogger::Instance()->SetDebug(false);
  JetScapeLogger::Instance()->SetVerboseLevel(0);

  auto jetscape = make_shared<JetScape>("./jetscape_init.xml",3);

  auto jlossmanager = make_shared<JetEnergyLossManager> ();
  auto jloss = make_shared<JetEnergyLoss> ();
  auto matter = make_shared<Matter> ();
  auto martini = make_shared<Martini> ();

  auto pGun= make_shared<PGun> ();
  auto hydro = make_shared<Brick> ();
  auto writer= make_shared<JetScapeWriterAscii> ("test_out.dat");

  jetscape->Add(pGun);
  jetscape->Add(hydro);
  jloss->Add(matter);
  jloss->Add(martini);
  jlossmanager->Add(jloss);

  jetscape->Add(jlossmanager);
  jetscape->Add(writer);

  // Intialize all modules tasks
  jetscape->Init();

  // Run JetScape with all task/modules as specified ...
  jetscape->Exec();

  jetscape->Finish();

  …
  return 0;
}
```



Running JETSCAPE ...

```
int main(int argc, char** argv)
{
  …
  JetScapeLogger::Instance()->SetDebug(false);
  JetScapeLogger::Instance()->SetVerboseLevel(0);

  auto jetscape = make_shared<JetScape>("./jetscape_init.xml",3);

  auto jlossmanager = make_shared<JetEnergyLossManager> ();
  auto jloss = make_shared<JetEnergyLoss> ();
  auto matter = make_shared<Matter> ();
  auto martini = make_shared<Martini> ();

  auto pGun= make_shared<PGun> ();
  auto hydro = make_shared<Brick> ();
  auto writer= make_shared<JetScapeWriterAscii> ("test_out.dat");

  jetscape->Add(pGun);

  jetscape->Add(hydro);

  jloss->Add(matter);
  jloss->Add(martini);
  jlossmanager->Add(jloss);

  jetscape->Add(jlossmanager);

  jetscape->Add(writer);


  // Intialize all modules tasks
  jetscape->Init();

  // Run JetScape with all task/modules as specified ...
  jetscape->Exec();

  jetscape->Finish();

  …
  return 0;
}
```

```
[Info] ------------------------------------
[Info] | Brick Test JetScape Framework ... |
[Info] ------------------------------------
[Info]
[Info] *-------------------------------------------------------------*
[Info] |                                                             |
[Info] |                  /\                                         |
[Info] |                 /  \                                        |
[Info] |                / |  \                                       |
[Info] |               /  |   \             /\                       |
[Info] |              /   |    \          /\/  \                     |
[Info] |             / \  |  /  \       /\/    |  \/\                |
[Info] |            /   \ | /    \     /    %  | %   \               |
[Info] |         __/     \|/      \__/      %|%     \/\__           |
[Info] |                                                             |
[Info] |                  JETSCAPE beta release 0.1                  |
[Info] |                                                             |
[Info] |     The Jet Energy-loss Tomography with a Statistically     |
[Info] |       and Computationally Advanced Program Envelope         |
[Info] |                http://jetscape.wayne.edu                    |
[Info] |                                                             |
[Info] | Please cite xxx if you use this package for scientific work. |
[Info] |                                                             |
[Info] | JETSCAPE is provided without warranty under the terms       |
[Info] | of the GNU GPLv3. It uses xxx code(s).                      |
[Info] | See COPYING file for details.                               |
[Info] |                                                             |
[Info] *-------------------------------------------------------------*
[Info] 18MB  Intialize JetScape ...
[Info] 18MB  Created JetScapeXML Instance
[Info] 18MB  Open XML file : ./jetscape_init.xml
[Info] 18MB  Set Hydro,JetEnergylossManager and IS Pointers for SignalManager to create Signal/Slots
[Info] 18MB  Created JetScapeSignalManager Instance
[Info] 18MB  Found 4 Modules Initialize them ...
[Info] 18MB  Intialize HardProcess : PGun ...
[Info] 18MB  Parton Gun with fixed pT = 100
[Info] 18MB  Intialize FluidDynamics : Brick ...
[Info] 18MB  Brick Temperature T = 0.3
[Info] 18MB  Intialize JetEnergyLoss Manager ...
[Info] 18MB  Found 1 Eloss Manager Tasks/Modules Initialize them ...
[Info] 18MB  Intialize JetEnergyLoss ...
[Info] 18MB  Eloss shower with deltaT = 1 and maxT = 10
[Info] 18MB  Found 2 Eloss Tasks/Modules Initialize them ...
[Info] 18MB  Intialize Matter ...
[Info] 18MB  Intialize Martini ...
[Info] 18MB  Connect JetEnergyLossManager Signal to Hard Process ...
[Info] 18MB  JetScape Ascii Writer initialized with output file = test_out.dat
[Info] 18MB  Run JetScape ...
[Info] 18MB  Number of Events = 3
[Info] 18MB  Run Event # = 0
[Info] 18MB  Run Hard Process : PGun ...
[Info] 18MB  Run Hydro : Brick ...
[Info] 18MB  Run JetEnergyLoss Manager ...
[Info] 18MB   Number of Hard Partons = 2
[Info] 18MB   Found 2 Eloss Manager Tasks/Modules Execute them ...
[Info] 18MB   2 Eloss Manager Tasks/Modules finished.
[Info] 18MB  Run JetScapeWriterAscii: Write event # 0 ...
[Info] 18MB  Run Event # = 1
[Info] 18MB  Run Hard Process : PGun ...
[Info] 18MB  Run Hydro : Brick ...
[Info] 18MB  Run JetEnergyLoss Manager ...
[Info] 18MB   Number of Hard Partons = 2
[Info] 18MB   Found 2 Eloss Manager Tasks/Modules Execute them ...
[Info] 18MB   2 Eloss Manager Tasks/Modules finished.
[Info] 18MB  Run JetScapeWriterAscii: Write event # 1 ...
[Info] 18MB  Run Event # = 2
[Info] 18MB  Run Hard Process : PGun ...
[Info] 18MB  Run Hydro : Brick ...
[Info] 18MB  Run JetEnergyLoss Manager ...
[Info] 18MB   Number of Hard Partons = 2
[Info] 18MB   Found 2 Eloss Manager Tasks/Modules Execute them ...
[Info] 18MB   2 Eloss Manager Tasks/Modules finished.
[Info] 18MB  Run JetScapeWriterAscii: Write event # 2 ...
[Info] 18MB  JetScape finished after 3 events!
[Info] Finished!

CPU time: 0.004522 seconds.
Real time: 0.000000 seconds.
```

jetscape->Init();

jetscape->Exec();

jetscape->Finish();