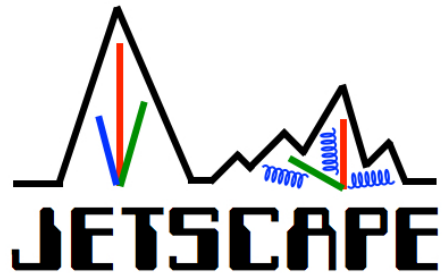
Implementation of the JetScape Framework version 1.0

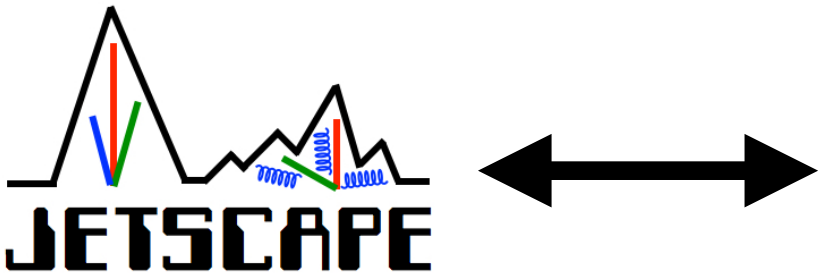
Joern Putschke (WSU)



JetScape Winter School January 2018, LBNL

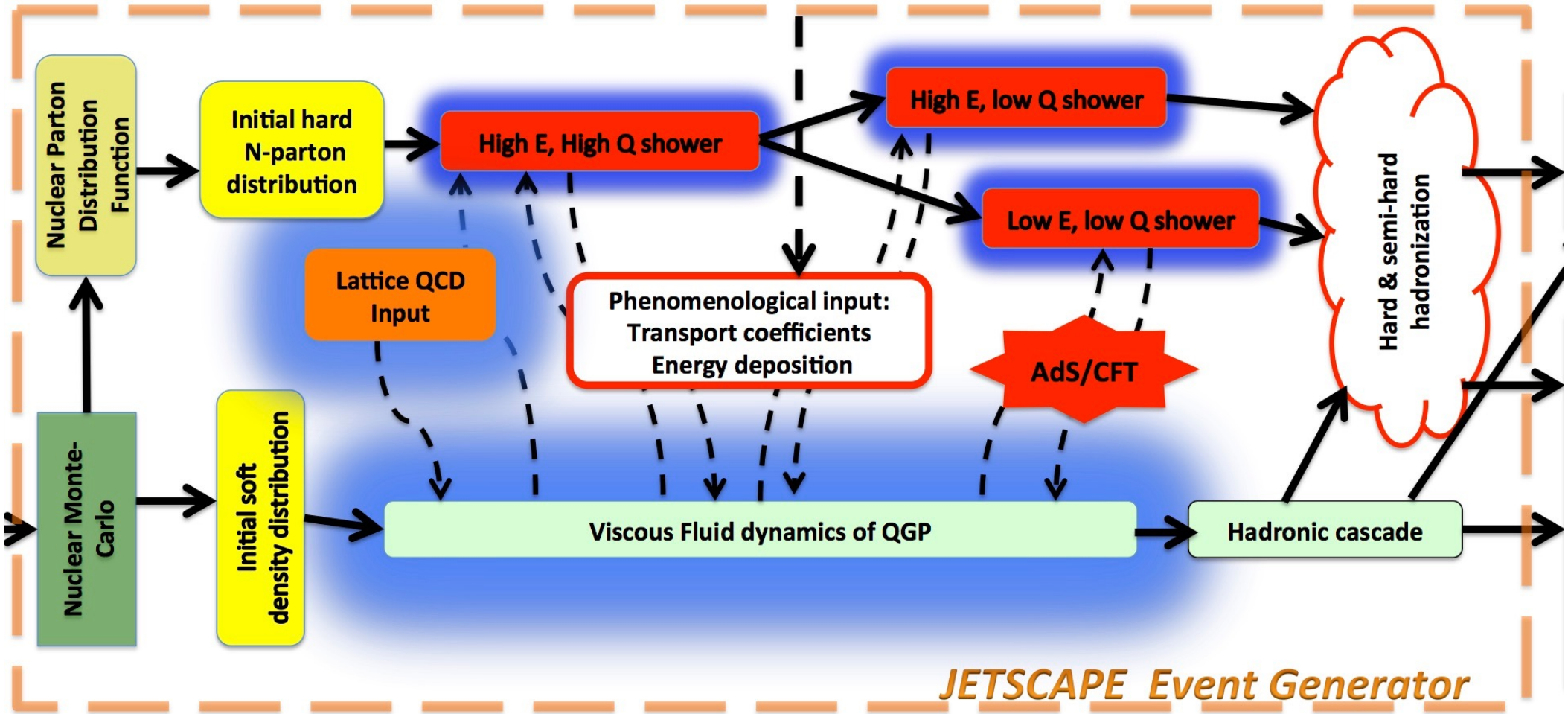
Implementation of the JetScape Framework version 1.0

Joern Putschke (WSU)




```
[Info] *------------------------------------------------------------*
[Info] |                                                            |
[Info] |                    /\                                      |
[Info] |                   /  \                                     |
[Info] |                  / |  \                                    |
[Info] |                 /  |   \              /\                   |
[Info] |                /   |    \          /\/  \                  |
[Info] |               / \  |  /  \      /\/     | \/\             |
[Info] |              /   \ | /    \    /       %  |  %  \          |
[Info] |           __/     \|/      \__/        %|%     \/\__      |
[Info] |                                                            |
[Info] |                  JETSCAPE release 1.0                      |
[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] *------------------------------------------------------------*
```

Reminder: JetScape Event Generator (Physics View)

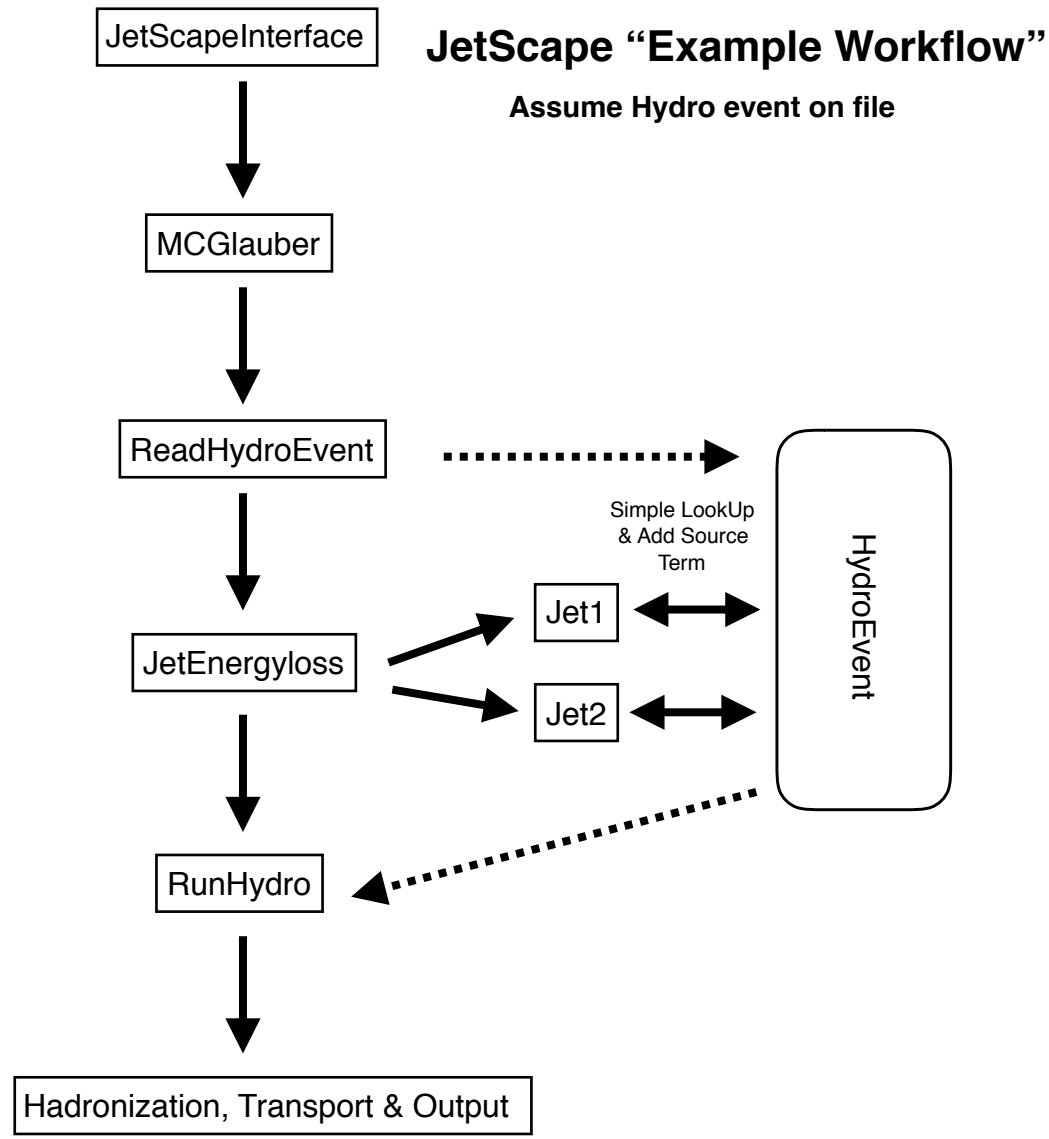


In general: Software Engineering Principles

Software Engineering Principles

- Rigor and formality
- Separation of concerns
  - Modularity, decomposition, abstraction
  - Cohesion and coupling
- Anticipation of change
- Generality
- Incrementality
- Scalability
- Heterogeneity

Our approach: JetScape Workflow and Considerations

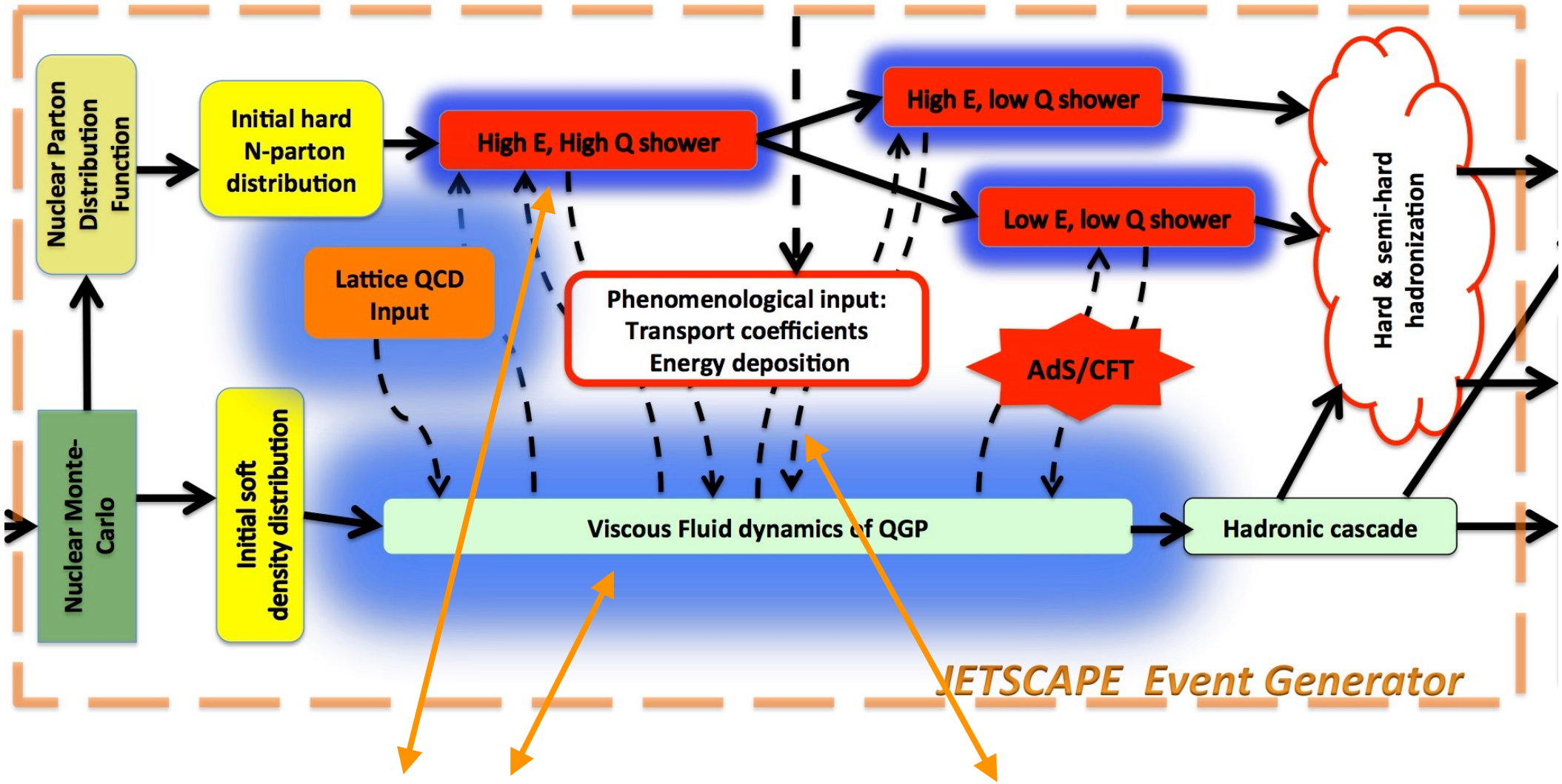


Our approach: JetScape Workflow and Considerations

Considerations and implementation/design philosophy:

- **minimize dependency on external libraries** concerning the core framework code (all used external sources are “small” and are all part of the “core framework”)
- **C++11 style** —> no new/delete **smart pointers** (shared, weak and unique)
  —> simplifies memory management (reduces chances of memory leaks)
- **Object Orientated (OO) Framework (C++ class inheritance)**
- **Tasked-based** implementation: *Init(), Exec(), Clear() and Finish()*
  Helpful for “further” parallelization (see JetScapeEnergyLossManager as a test case)
- **Strict data encapsulation** between modules (only “share” what is needed!)
- **Signal/Slot** mechanism to *ensure data encapsulation*.
  Also elegant solution concerning switching between energy loss modules and “communication” with hydro
- **Clear and easy interface** for further “end-user” developer, inherit from proper base class and overload the “JetScape interface functions” and “data structures”
  —> **No real knowledge** of the framework itself is needed and importantly is hidden
  —> **Safety!!!**

Reminder: JetScape Event Generator (Code Design)

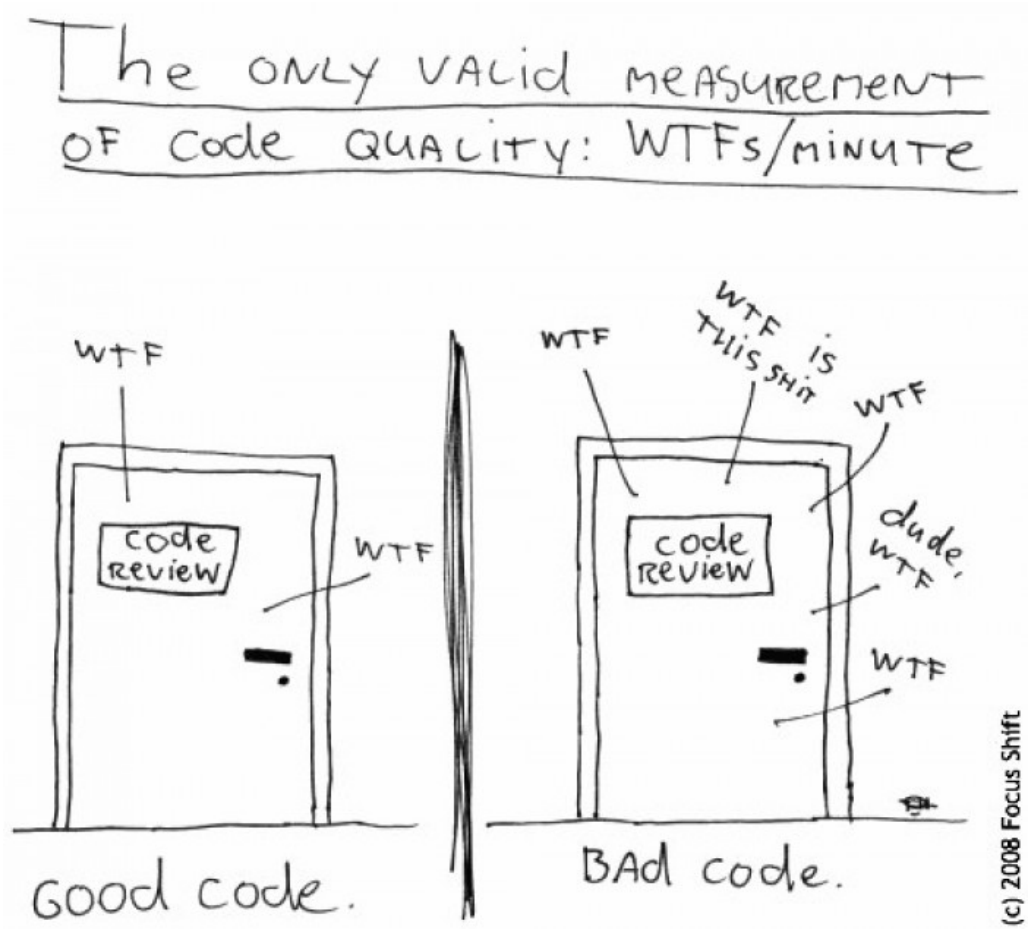


Tasks —> User defined *Modules*

Communication: *Signal/Slots*

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

Code Quality: Lets see how we will stack up …



A quick word about C++11 (more in working session)

Some new useful features:

- Automatic Type Deduction
- Null Pointer
- **Smart Pointers**
- Range-based for Loops
- (many more …)

A quick word about C++11 (more in working session)

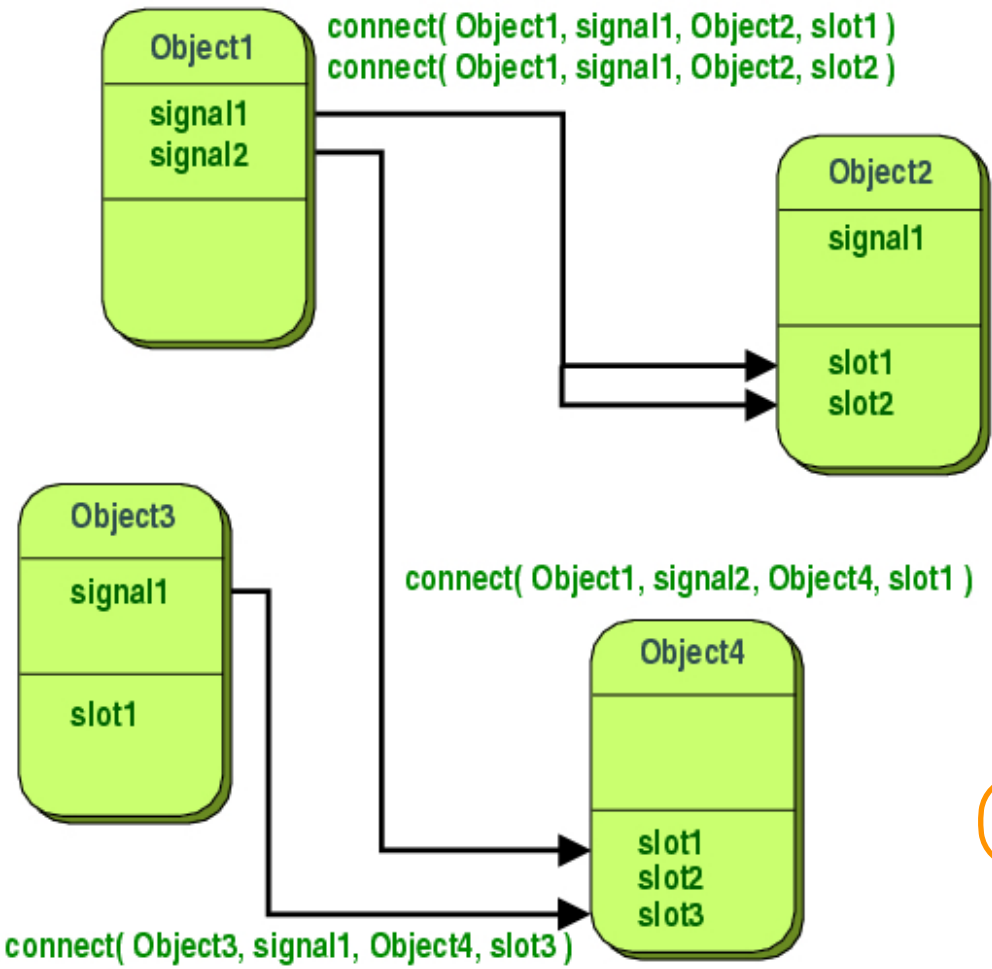
Some new useful features:

- Automatic Type Deduction
- Null Pointer
- **Smart Pointers**
- Range-based for Loops
- (many more …)

- **A *wrapper class* for a pointer with operators like * and -> overloaded**
- **The objects of smart pointer class look like pointer, but can do more like *automatic destruction via reference counting***
  —> we don't have to explicitly use *delete* (and *new*)
- **Destructor is automatically called when an object goes out of scope**
  —> *the dynamically allocated memory is automatically deleted*

A quick word about Signal/Slots

http://doc.qt.io/qt-4.8/signalsandslots.html



Developed by Qt for GUI.

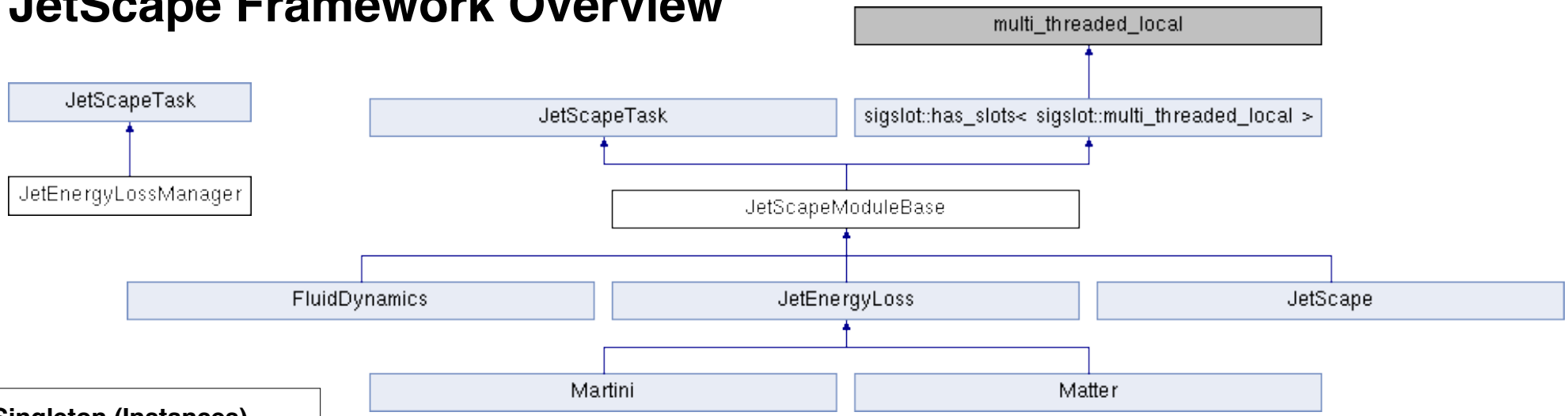
Qt: Preprocessor needed (Moc) (dependencies on large external libraries/license)

Alternatives more or less pure C++ based (and/or Boost):

http://sigslot.sourceforge.net

http://www.boost.org/doc/libs/../signals2.html

JetScape Framework Overview



Singleton (Instances)

- JetScapeXML
- JetScapeLogger
- JetScapeSignalManager
- JetScapeTaskSupport

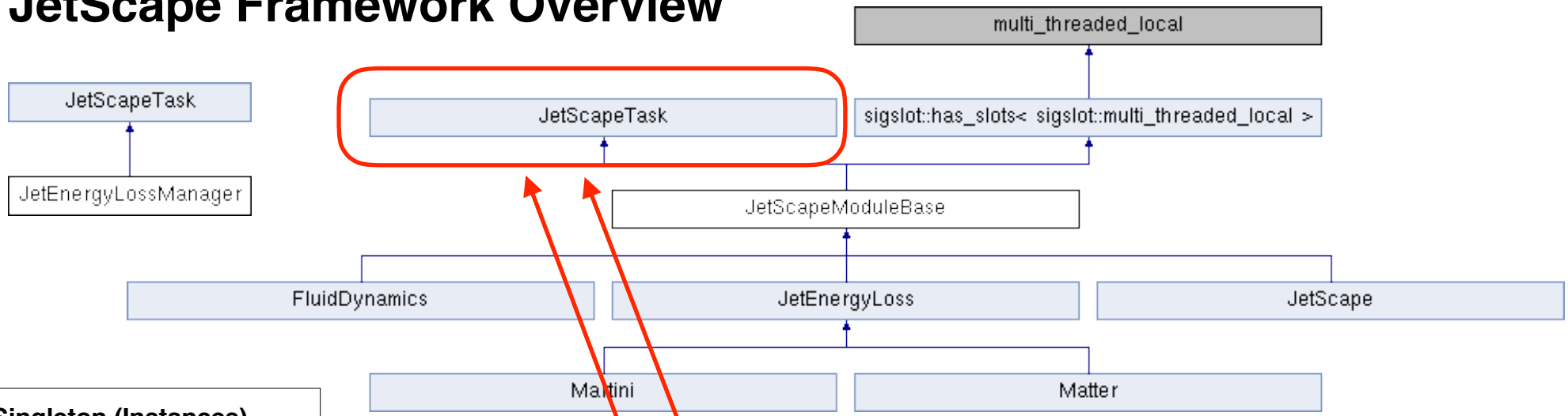
External

- sigslot.h
- tinyxml2.h
- gzstream.h
- GTL graph Library

Small or Header only. Build directly into the lib.

```
…

auto jetscape = make_shared<JetScape>("./jetscape_init.xml",3);
auto jlossmanager = make_shared<JetEnergyLossManager> ();
auto jloss = make_shared<JetEnergyLoss> ();
auto hydro = make_shared<FluidDynamics> ();

auto matter = make_shared<Matter> ();
auto martini = make_shared<Martini> ();

jetscape->Add(hydro);

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

jlossmanager->Add(jloss);

jetscape->Add(jlossmanager);

jetscape->Init();

jetscape->Exec();

…
```

JetScape Framework Overview



Singleton (Instances)

- JetScapeXML
- JetScapeLogger
- JetScapeSignalManager
- JetScapeTaskSupport

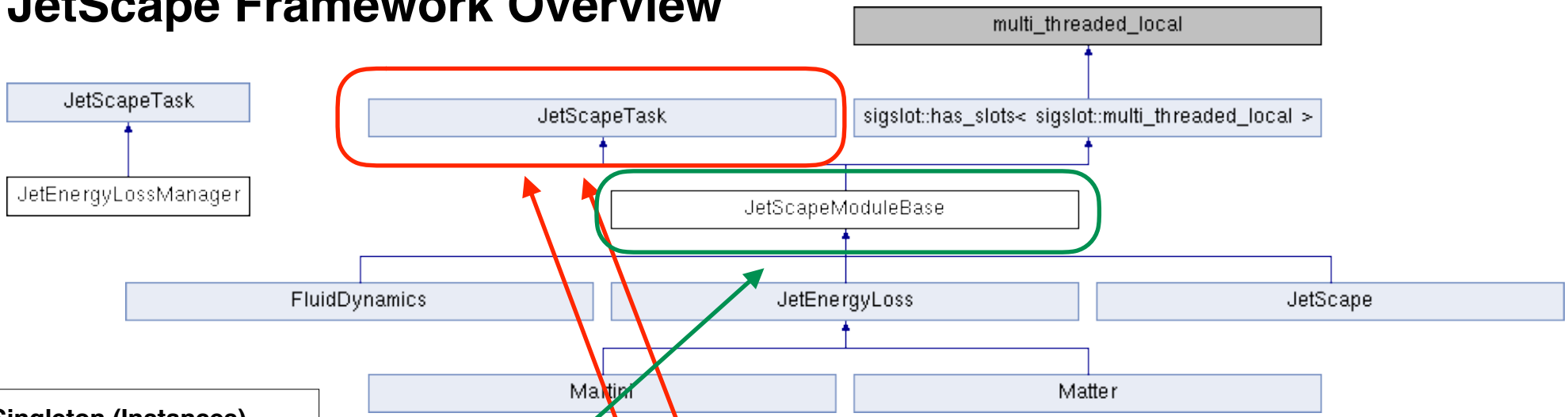
External

- sigslot.h
- tinyxml2.h
- gzstream.h
- GTL graph Library

Small or Header only. Build directly into the lib.

```
…

auto jetscape = make_shared<JetScape>("./jetscape_init.xml",3);
auto jlossmanager = make_shared<JetEnergyLossManager> ();
auto jloss = make_shared<JetEnergyLoss> ();
auto hydro = make_shared<FluidDynamics> ();

auto matter = make_shared<Matter> ();
auto martini = make_shared<Martini> ();

jetscape->Add(hydro);

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

jlossmanager->Add(jloss);

jetscape->Add(jlossmanager);

jetscape->Init();

jetscape->Exec();

…
```

JetScape Framework Overview



Singleton (Instances)

- JetScapeXML
- JetScapeLogger
- JetScapeSignalManager
- JetScapeTaskSupport

Provides seed and random number generator to ensure reproducibility

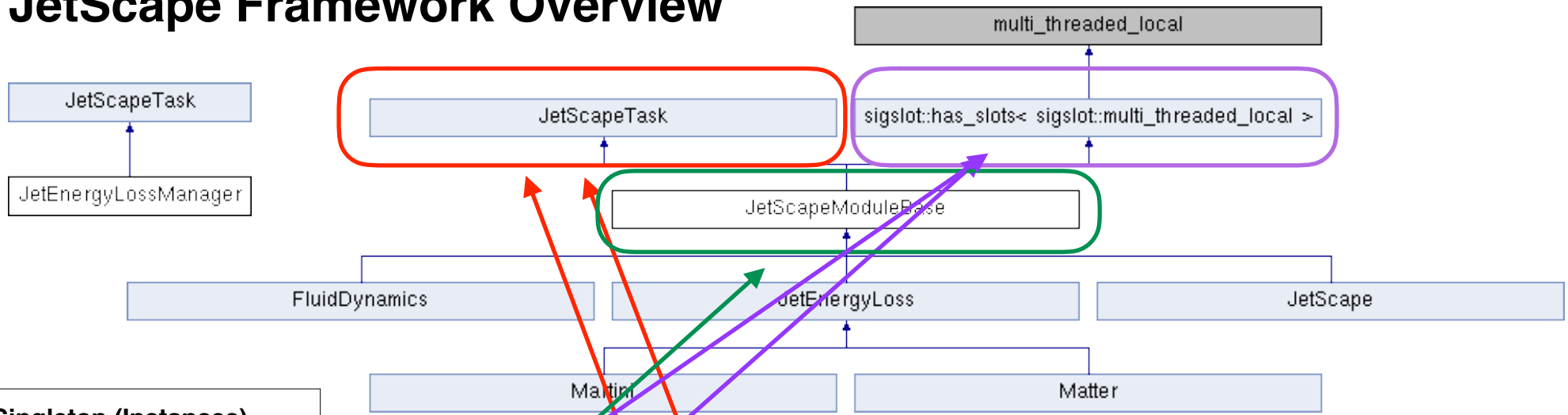
External

- sigslot.h
- tinyxml2.h
- gzstream.h
- GTL graph Library

Small or Header only. Build directly into the lib.

```
…

auto jetscape = make_shared<JetScape>("./jetscape_init.xml",3);
auto jlossmanager = make_shared<JetEnergyLossManager> ();
auto jloss = make_shared<JetEnergyLoss> ();
auto hydro = make_shared<FluidDynamics> ();

auto matter = make_shared<Matter> ();
auto martini = make_shared<Martini> ();

jetscape->Add(hydro);

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

jlossmanager->Add(jloss);

jetscape->Add(jlossmanager);

jetscape->Init();

jetscape->Exec();

…
```

JetScape Framework Overview



Singleton (Instances)

- JetScapeXML
- JetScapeLogger
- JetScapeSignalManager
- JetScapeTaskSupport

External

- sigslot.h
- tinyxml2.h
- gzstream.h
- GTL graph Library

Provides seed and random number generator to ensure reproducibility

Small or Header only. Build directly into the lib.

```
…

auto jetscape = make_shared<JetScape>("./jetscape_init.xml",3);
auto jlossmanager = make_shared<JetEnergyLossManager> ();
auto jloss = make_shared<JetEnergyLoss> ();
auto hydro = make_shared<FluidDynamics> ();

auto matter = make_shared<Matter> ();
auto martini = make_shared<Martini> ();

jetscape->Add(hydro);

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

jlossmanager->Add(jloss);

jetscape->Add(jlossmanager);

jetscape->Init();

jetscape->Exec();

…
```

JetScape Instances: Logger, XML Reader, Signal Manager, Task Support

JetScapeXML Class Reference

`#include <JetScapeXML.h>`

Easy access to XML file (read parameters …)

Public Member Functions

tinyxml2::XMLElement *	GetXMLRoot ()
tinyxml2::XMLDocument &	GetXMLDocument ()
void	SetXMLFileName (string m_name)
string	GetXMLFileName ()
bool	IsOpen ()
void	OpenXMLFile ()
void	OpenXMLFile (string

Static Public Member Functions

static JetScapeXML *	Instance ()

JetScapeLogger Class Reference

`#include <JetScapeLogger.h>`

Standardized and thread safe output

Public Member Functions

LogStreamer	Info ()
LogStreamer	InfoNice ()
LogStreamer	Warn ()
LogStreamer	Debug ()
LogStreamerThread	DebugThread ()
LogStreamer	Remark ()
LogStreamer	Verbose (unsigned short m_vlevel)
LogStreamer	VerboseShower (unsigned short m_vlevel)
LogStreamer	VerboseParton (unsigned short m_vlevel, Parton &
LogStreamer	VerboseVertex (unsigne
void	SetDebug (bool m_debu
void	SetRemark (bool m_rem
void	SetVerboseLevel (unsig
bool	GetDebug ()
bool	GetRemark ()
unsigned short	GetVerboseLevel ()

Static Public Member Functions

static JetScapeLogger *	Instance ()

JetScapeSignalManager Class Reference

`#include <JetScapeSignalManager.h>`

Basically a helper instance to QA signal/slots. Provides easy access for quick developments.

Public Member Functions

void	SetHydroPointer (shared_ptr< FluidDynamics > m_hydro)
weak_ptr< FluidDynamics >	GetHydroPointer ()
void	SetJetEnergyLossManagerPointer (shared_ptr< JetEnergyLossManager > m_jloss)
weak_ptr< JetEnergyLossManager >	GetJetEnergyLossManagerPointer ()
void	SetHardProcessPointer (shared_ptr< HardProcess > m_hardp)
weak_ptr< HardProcess >	GetHardProcessPointer ()
void	SetWriterPointer (shared_ptr< JetScapeWriter > m_writer)
weak_ptr< JetScapeWriter >	GetWriterPointer ()
void	ConnectJetSignal (shared_ptr< JetEnergyLoss > j)
void	ConnectEdensitySignal (shared_ptr< JetEnergyLoss > j)
void	ConnectAddJetSourceSignal (shared_ptr< JetEnergyLoss > j)
void	ConnectGetTemperatureSignal (shared_ptr< JetEnergyLoss > j)
void	ConnectGetHydroCellSignal (shared_ptr< JetEnergyLoss > j)
void	ConnectGetHardPartonListSignal (shared_ptr< JetEnergyLossManager > jm)
void	ConnectSentInPartonsSignal (shared_ptr< JetEnergyLoss > j, shared_ptr< JetEnergyLoss > j2)
void	ConnectGetOutPartons (shared_ptr< JetEnergyLoss > j, shared_ptr< JetEnergyLoss > j2)
void	DisconnectSignal ()
void	CleanUp ()

Jetscape::JetScapeTaskSupport Class Reference

Provides random number generator (reproducibility)

`#include <JetScapeTaskSupport.h>`

Public Member Functions

int	RegisterTask ()
shared_ptr< std::mt19937 >	get_mt19937_generator (int TaskId)

Static Public Member Functions

static JetScapeTaskSupport *	Instance ()
static void	ReadSeedFromXML ()
	Initialize random engine functionality from the XML file. More...

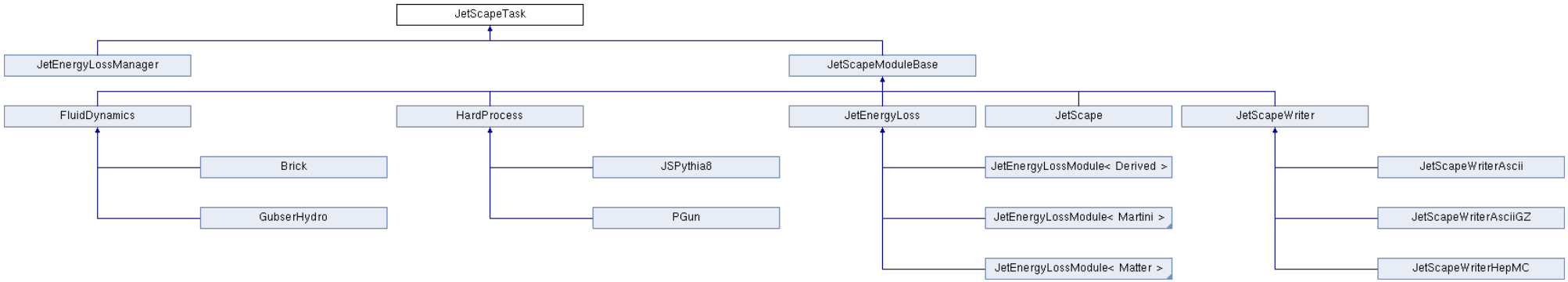
Static Protected Attributes

static bool	one_generator_per_task_ =false

Singleton (Instances)

- JetScapeXML
- JetScapeLogger
- JetScapeSignalManager
- JetScapeTaskSupport

The JetScape Task and the derived Module Base Classes



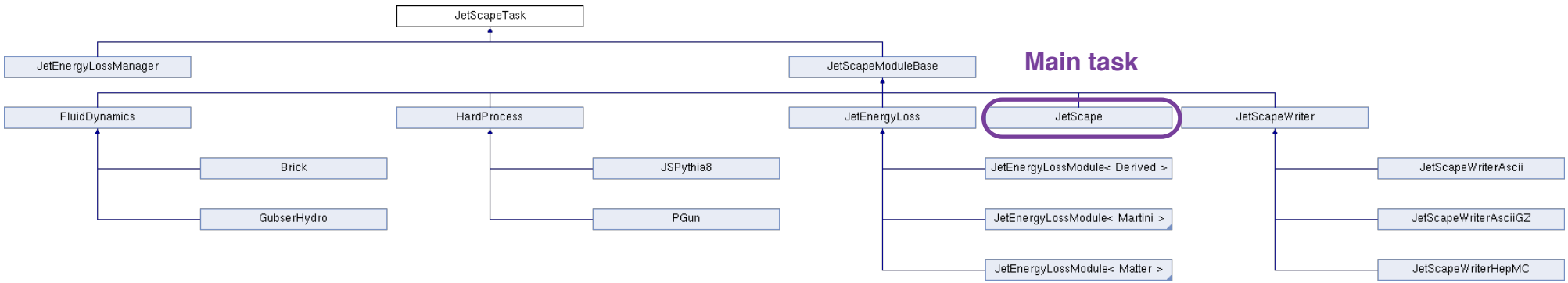
Public Member Functions

	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 (see JetEnergyLoss later)

Remark: I will not go through every (technical) detail, and all (external) dependencies at this point (pro/cons). Optimizations and changes might be necessary in the future if more "physics" will be included …

The JetScape Task and the derived Module Base Classes



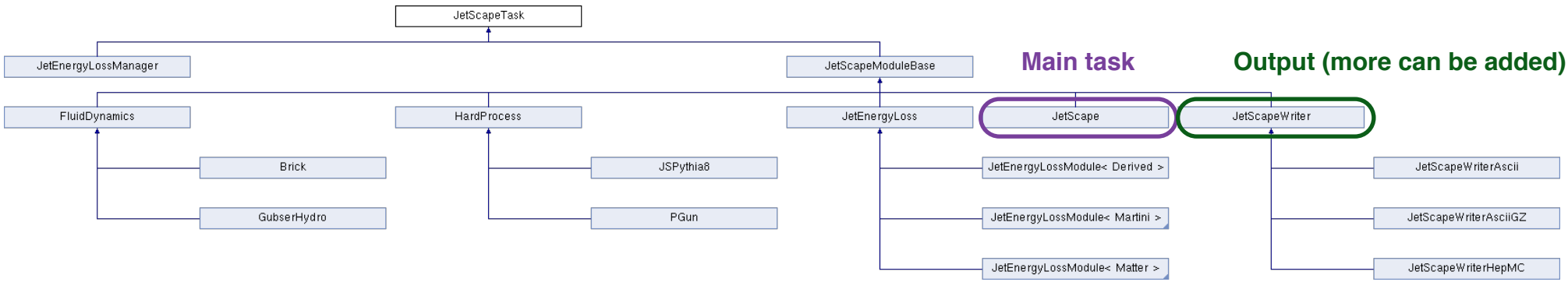
Public Member Functions

	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 (see JetEnergyLoss later)

Remark: I will not go through every (technical) detail, and all (external) dependencies at this point (pro/cons). Optimizations and changes might be necessary in the future if more "physics" will be included …

The JetScape Task and the derived Module Base Classes



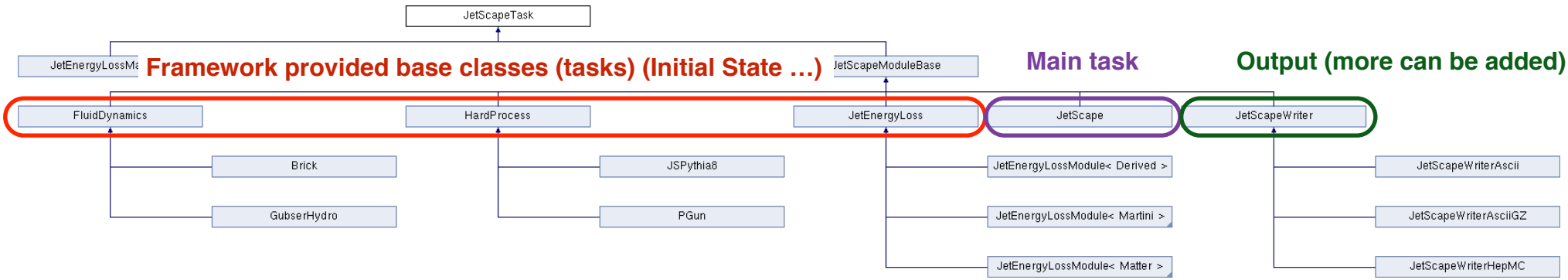
Public Member Functions

	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 (see JetEnergyLoss later)

Remark: I will not go through every (technical) detail, and all (external) dependencies at this point (pro/cons). Optimizations and changes might be necessary in the future if more "physics" will be included …

The JetScape Task and the derived Module Base Classes



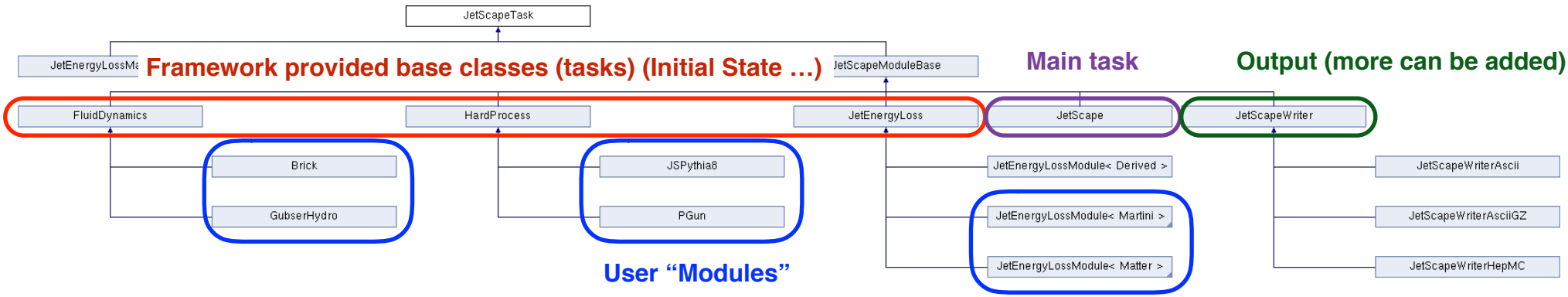
Public Member Functions

	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 (see JetEnergyLoss later)

Remark: I will not go through every (technical) detail, and all (external) dependencies at this point (pro/cons). Optimizations and changes might be necessary in the future if more "physics" will be included …

The JetScape Task and the derived Module Base Classes



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 (see JetEnergyLoss later)

Remark: I will not go through every (technical) detail, and all (external) dependencies at this point (pro/cons). Optimizations and changes might be necessary in the future if more "physics" will be included …

Init and XML … (JetScapeXML instance)

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

  Matter();
  virtual ~Matter();
  void Init();

  …
};
```

```
<?xml version="1.0"?>
<!-- Just for test purposes! -->
<!-- More details and final format to be determined ... -->
<jetscape>

  <debug> on </debug>
  <remark> off </remark>
  <vlevel> 0 </vlevel>

  <!--  If for example one wants to run JetScape purely via xml ... -->
   <Tasks>
     <task>Matter</task>
     <task>Hydro1</task>
   </Tasks>

   <!-- Inital State Module ... -->
   <IS>

     <Trento>
     </Trento>

   </IS>

   <!-- Hard Process/Pythia to be included -->

   <Hard>

     <!-- Parton Gun test with fixed pT and fixed IS pos = 0 ... -->
     <PGun>
       <name>PGun</name>
       <pT>100</pT>
     </PGun>

   </Hard>


   <!--Eloss Module ... -->
   <Eloss>

     <deltaT>1</deltaT>
     <maxT>10</maxT>

     <!-- Individual Eloss Modules run Eloss and Eloss Manager ... -->
     <!-- Just a test here (no physics implemented in Matter and Martini ... -->
     <Matter>
       <name>Matter</name>
       <param1> 2 </param1>
       <qhat> 1.1 </qhat>
     </Matter>

     <Martini>
       <name > Martini </name>
       <param1> 3 </param1>
     </Martini>

   </Eloss>
…
```

Init and XML … (JetScapeXML instance)

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

  Matter();
  virtual ~Matter();
  void Init();

  …
};
```

```
<?xml version="1.0"?>
<!-- Just for test purposes! -->
<!-- More details and final format to be determined ... -->
<jetscape>

  <debug> on </debug>
  <remark> off </remark>
  <vlevel> 0 </vlevel>

  <!--  If for example one wants to run JetScape purely via xml ... -->
   <Tasks>
     <task>Matter</task>
     <task>Hydro1</task>
   </Tasks>

   <!-- Inital State Module ... -->
   <IS>
```

```
// As example: Toy Matter Eloss
void Matter::Init()
{
  INFO<<"Intialize Matter ...";

  tinyxml2::XMLElement *eloss= JetScapeXML::Instance()->GetXMLRoot()->FirstChildElement("Eloss" );
  tinyxml2::XMLElement *matter=eloss->FirstChildElement("Matter");

  if (matter)
    {
      string s = matter->FirstChildElement( "name" )->GetText();

      DEBUG << s << " to be initilizied ...";

      double m_qhat=-99.99;
      matter->FirstChildElement("qhat")->QueryDoubleText(&m_qhat);
      SetQhat(m_qhat);

      DEBUG  << s << " with qhat = "<<GetQhat();
    }
  else
    {
      WARN << " : Matter not properly initialized in XML file ...";
      exit(-1);
    }
}
```

Init and XML … (JetScapeXML instance)

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

  Matter();
  virtual ~Matter();
  void Init();

  …
};
```

```
<?xml version="1.0"?>
<!-- Just for test purposes! -->
<!-- More details and final format to be determined ... -->
<jetscape>

  <debug> on </debug>
  <remark> off </remark>
  <vlevel> 0 </vlevel>

  <!--  If for example one wants to run JetScape purely via xml ... -->
   <Tasks>
     <task>Matter</task>
     <task>Hydro1</task>
   </Tasks>

   <!-- Inital State Module ... -->
   <IS>
```

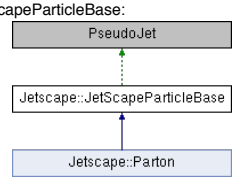
```
// As example: Toy Matter Eloss
void Matter::Init()
{
  INFO<<"Intialize Matter ...";

  tinyxml2::XMLElement *eloss= JetScapeXML::Instance()->GetXMLRoot()->FirstChildElement("Eloss" );
  tinyxml2::XMLElement *matter=eloss->FirstChildElement("Matter");

  if (matter)
    {
      string s = matter->FirstChildElement( "name" )->GetText();

      DEBUG << s << " to be initilizied ...";

      double m_qhat=-99.99;
      matter->FirstChildElement("qhat")->QueryDoubleText(&m_qhat);
      SetQhat(m_qhat);

      DEBUG  << s << " with qhat = "<<GetQhat();
    }
  else
    {
      WARN << " : Matter not properly initialized in XML file ...";
      exit(-1);
    }
}
```

JetScape Framework Data Types

Jetscape::JetScapeParticleBase Class Reference

`#include <JetScapeParticles.hpp>`

Inheritance diagram for Jetscape::JetScapeParticleBase:



FastJet (core) protected inheritance

Why? Next slide …

Public Member Functions

void	**reset_momentum** (const double px, const double py, const double pz, const double e)
void	**reset_momentum** (const **FourVector** &p)
fjcore::PseudoJet	**GetPseudoJet** () const No implicit cast to PseudoJet is allowed, provide a conversion. More...
	JetScapeParticleBase ()
	JetScapeParticleBase (int label, int id, int stat, const **FourVector** &p, const **FourVector** &x)
	JetScapeParticleBase (int label, int id, int stat, double pt, double eta, double phi, double e, double *x=0)
	JetScapeParticleBase (const **JetScapeParticleBase** &srp)
virtual	**~JetScapeParticleBase** ()
void	**clear** ()
void	**set_label** (int label)
void	**set_id** (int id)
void	**set_stat** (int stat)
void	**set_x** (double x[4])
void	**init_jet_v** ()
void	**set_jet_v** (double v[4])
const int	**pid** ()
const int	**pstat** ()
const int	**plabel** ()
const double	**time** ()
FourVector &	**x_in** ()
FourVector &	**jet_v** ()
const double	**restmass** ()
const double	**p** (int i)
double	**pl** ()
const double	**nu** ()
const double	**t_max** ()
virtual **JetScapeParticleBase** &	**operator=** (**JetScapeParticleBase** &c)
virtual **JetScapeParticleBase** &	**operator=** (const **JetScapeParticleBase** &c)

Protected Member Functions

void	**set_restmass** (double mass_input) shouldn't be called from the outside, needs to be consistent with PID More...

Jetscape::Vertex Class Reference

`#include <JetClass.hpp>`

Public Member Functions

	Vertex ()
	Vertex (double x, double y, double z, double t)
	Vertex (**FourVector** &x)
virtual	**~Vertex** ()
void	**set_location** (**FourVector** &x)
FourVector &	**x_in** ()

Protected Attributes

FourVector	**x_in_**

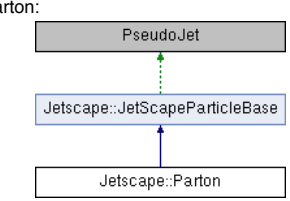
Friends

ostream &	**operator<<** (ostream &output, **Vertex** &vertex)

Jetscape::Parton Class Reference

`#include <JetScapeParticles.hpp>`

Inheritance diagram for Jetscape::Parton:



Parton Showers as Graph (container) more later …

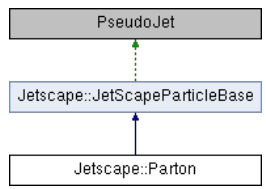
More data types, like Hadrons …
as more modules/physics get added

The Parton Class in more detail …

Jetscape::Parton Class Reference

`#include <JetScapeParticles.hpp>`

Inheritance diagram for Jetscape::Parton:



FastJet (core) protected inheritance

Public Member Functions

virtual void	**set_mean_form_time** ()
virtual void	**set_form_time** (double **form_time**)
virtual double	**form_time** ()
virtual const double	**mean_form_time** ()
	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)
	Parton (const **Parton** &srp)
Parton &	**operator=** (**Parton** &c)
Parton &	**operator=** (const **Parton** &c)
const double	**t** ()
void	**set_t** (double **t**) virtuality of particle, : rescales the spatial component More...

▸ Public Member Functions inherited from **Jetscape::JetScapeParticleBase**

Protected Member Functions

void	**initialize_form_time** ()

▸ Protected Member Functions inherited from **Jetscape::JetScapeParticleBase**

Protected Attributes

double	**mean_form_time_** Mean formation time. More...
double	**form_time_** event by event formation time More...

▸ Protected Attributes inherited from **Jetscape::JetScapeParticleBase**

A **JetScapeParticleBase** derives PRIVARTELY from FastJet PseudoJet and has additional information:

- PID (from PDG) and rest mass (these should eventually be coupled and only PID kept track of internally)
- A location (creation point) 4-vector
- a label and a status
- currently additional information that should be moved to derived classes or only used as UserInfo

The design choice of protected inheritance is due to a disconnect between available packages. The overwhelming majority of the theory community expects the 0 component to be time/energy, whereas FastJet (and others, like ROOT) prefer t,e to be the fourth component. Private inheritance means we can inherit and make accessible safe methods (with C++11 using), while protecting users from unsafe (explicit component access) ones. Note that this is only necessary because otherwise it's impossible to disallow constructors and getters that explicitly assume indices! IF we could get rid of those or change to the fastjet convention, we could derive publicly and get a true Is_A relationship. Alas.

You can in principle use the base class directly, but it's recommended to use the derived classes **Parton** and/or (todo) Hadron, Lepton, ...

Warning

PseudoJet doesn't have a concept of rest mass, any mass related functions literally assume $M^2 = E^2 - p * p$. Especially in the case of off-shell partons, the correct interpretation is "$M^{2"} = E^2 - p^2 == M0^2 + Q^2 == M0^2 + t$ Therefore, there is the dangerous possibility that functions like mass(), reset_PtYPhiM(...) can be used by unwitting users and display unexpected behavior. For protection against this possibility, "mass"-related functions in PseudoJet are not made available

Future considerations:

- We should consider making a Pythia8 installation mandatory; with pythia guaranteed to be present, the rest mass lookup could be automatically done using PDG data.
- If ROOT were a mandatory part, TLorentzVector would be a good replacement for the homebrewed **FourVector**
- If HepMc were a mandatory part, HepMc::FourVector would be a good replacement for the homebrewed **FourVector**

Protected Member Functions

void	**set_restmass** (double mass_input) shouldn't be called from the outside, needs to be consistent with PID More...

Protected Attributes

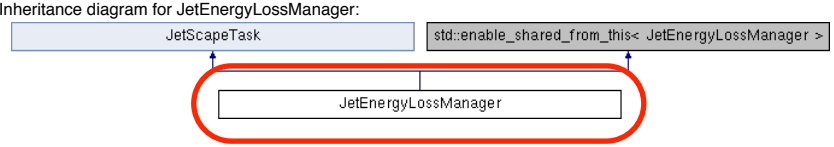
int	**pid_** particle id More...
int	**pstat_** status of particle More...
int	**plabel_** the line number in the event record More...
double	**mass_** rest mass of the particle More...
FourVector	**x_in_** position of particle More...
FourVector	**jet_v_** jet four vector, without gamma factor (so not really a four vector) More...

Focus on: JetEnergyLoss and JetEnergyLossManager

JetEnergyLossManager Class Reference

`#include <JetEnergyLossManager.h>`

Inheritance diagram for JetEnergyLossManager:



Public Member Functions

	JetEnergyLossManager ()
virtual	**~JetEnergyLossManager** ()
virtual void	**Init** ()
virtual void	**Exec** ()
virtual void	**Clear** ()
virtual void	**WriteTask** (weak_ptr< **JetScapeWriter** > w)
int	**GetNumSignals** ()
void	**CreateSignalSlots** ()
void	**SetGetHardPartonListConnected** (bool m_GetHardPartonListConnected)
const bool	**GetGetHardPartonListConnected** ()

▸ Public Member Functions inherited from **JetScapeTask**

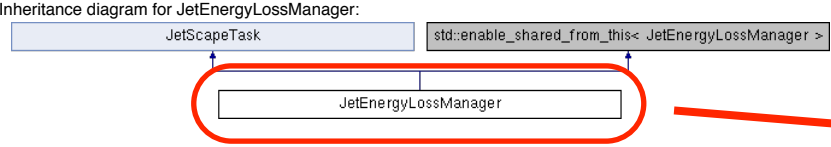
jlossmanager

Focus on: JetEnergyLoss and JetEnergyLossManager

JetEnergyLossManager Class Reference

`#include <JetEnergyLossManager.h>`

Inheritance diagram for JetEnergyLossManager:

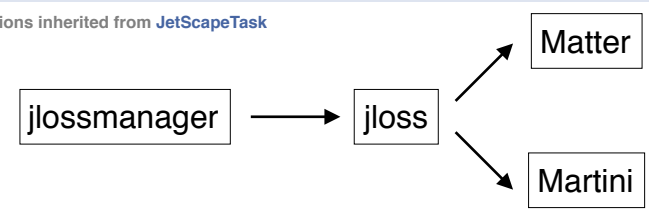


Public Member Functions

	JetEnergyLossManager ()
virtual	~JetEnergyLossManager ()
virtual void	Init ()
virtual void	Exec ()
virtual void	Clear ()
virtual void	WriteTask (weak_ptr< JetScapeWriter > w)
int	GetNumSignals ()
void	CreateSignalSlots ()
void	SetGetHardPartonListConnected (bool m_GetHardP
const bool	GetGetHardPartonListConnected ()

Public Member Functions inherited from JetScapeTask

Creates as many copies as needed (based on input from IS/HardProcess). Including all relevant signal/ slots. Simple to parallelize, checked with C++11 <thread>. Works fine!



shared_ptr< Parton >	GetShowerInitiatingParton ()
void	PrintShowerInitiatingParton ()
double	GetDeltaT ()
double	GetMaxT ()
shared_ptr< PartonShower >	GetShower ()

Public Member Functions inherited from JetScapeModuleBase

Public Member Functions inherited from JetScapeTask

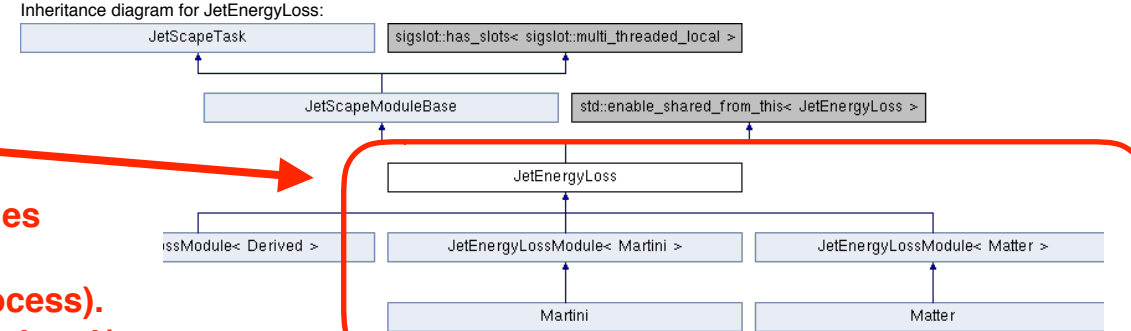
Public Attributes

sigslot::signal2< int, double, multi_threaded_local >	jetSignal
sigslot::signal2< int, double &, multi_threaded_local >	edensitySignal
sigslot::signal5< double, double, double, double, JetSource, multi_threaded_local >	AddJetSourceSignal
sigslot::signal5< double, double, double, double, double &, multi_threaded_local >	GetTemperatureSignal
sigslot::signal5< double, double, double, double, FluidCellInfo *, multi_threaded_local >	GetHydroCellSignal
sigslot::signal4< double, double, vector< Parton > &, vector< Parton > &, multi_threaded_local >	SentInPartons
sigslot::signal1< vector< Parton > &, multi_threaded_local >	GetOutPartons

JetEnergyLoss Class Reference

`#include <JetEnergyLoss.h>`

Inheritance diagram for JetEnergyLoss:

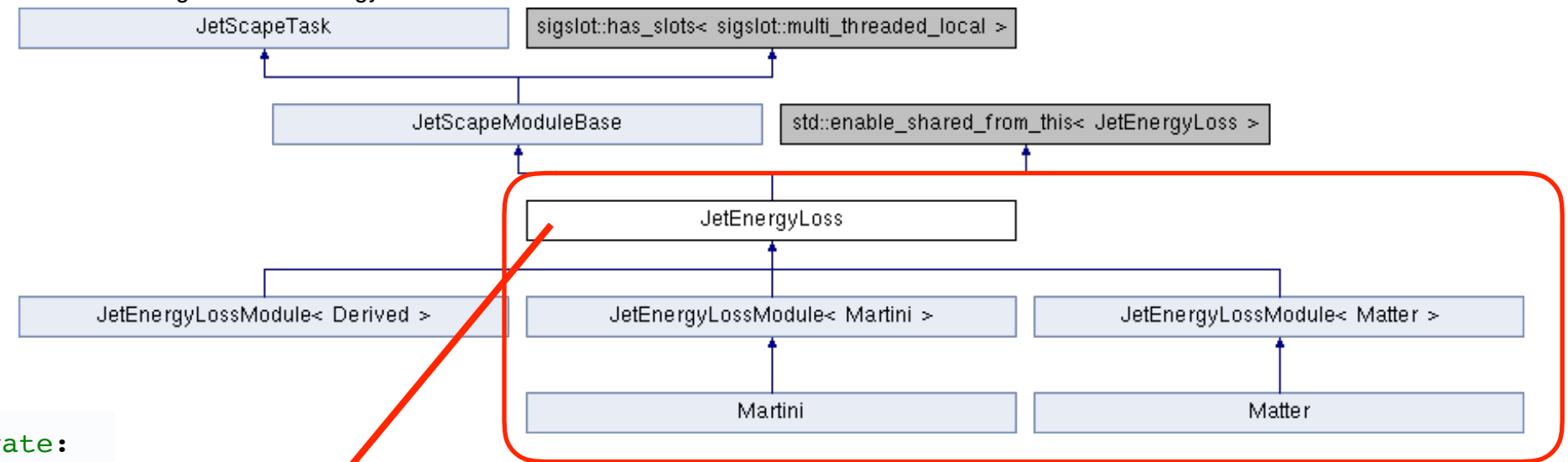


ber Functions

	JetEnergyLoss ()
	JetEnergyLoss (string m_name)
	JetEnergyLoss (const JetEnergyLoss &j)
virtual	~JetEnergyLoss ()
virtual shared_ptr< JetEnergyLoss >	Clone () const
virtual void	Init ()
virtual void	Exec () final
virtual void	WriteTask (weak_ptr< JetScapeWriter > w)
virtual void	Clear ()
virtual void	DoEnergyLoss (double deltaT, double Q2, vector< Parton > &pIn, vector< Parton > &pOut)
void	SetQhat (double m_qhat)
const double	GetQhat () const
void	SetJetSignalConnected (bool m_jetSignalConnected)
const bool	GetJetSignalConnected () const
void	SetEdensitySignalConnected (bool m_edensitySignalConnected)
const bool	GetEdensitySignalConnected () const
void	SetAddJetSourceSignalConnected (bool m_AddJetSourceSignalConnected)
const bool	GetAddJetSourceSignalConnected ()
void	SetGetTemperatureSignalConnected (bool m_GetTemperatureSignalConnected)
const bool	GetGetTemperatureSignalConnected ()
void	SetGetHydroCellSignalConnected (bool m_GetHydroCellSignalConnected)
const bool	GetGetHydroCellSignalConnected ()
void	SetSentInPartonsConnected (bool m_SentInPartonsConnected)
const bool	GetSentInPartonsConnected ()
void	SetGetOutPartonsConnected (bool m_GetOutPartonsConnected)
const bool	GetGetOutPartonsConnected ()
void	AddShowerInitiatingParton (shared_ptr< Parton > p)

Focus on: “Do” Parton Shower in JetScapeEnergyLoss

Inheritance diagram for JetEnergyLoss:

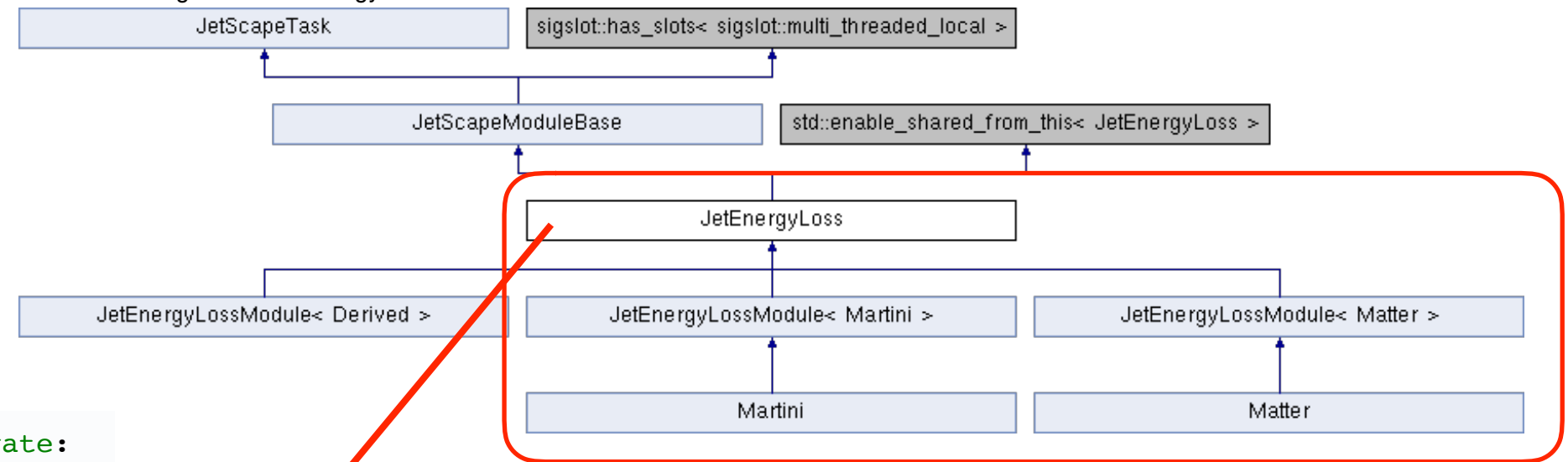



```
private:
...
shared_ptr<PartonShower> pShower;
node vStart;
node vEnd;
void DoShower();
```

(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

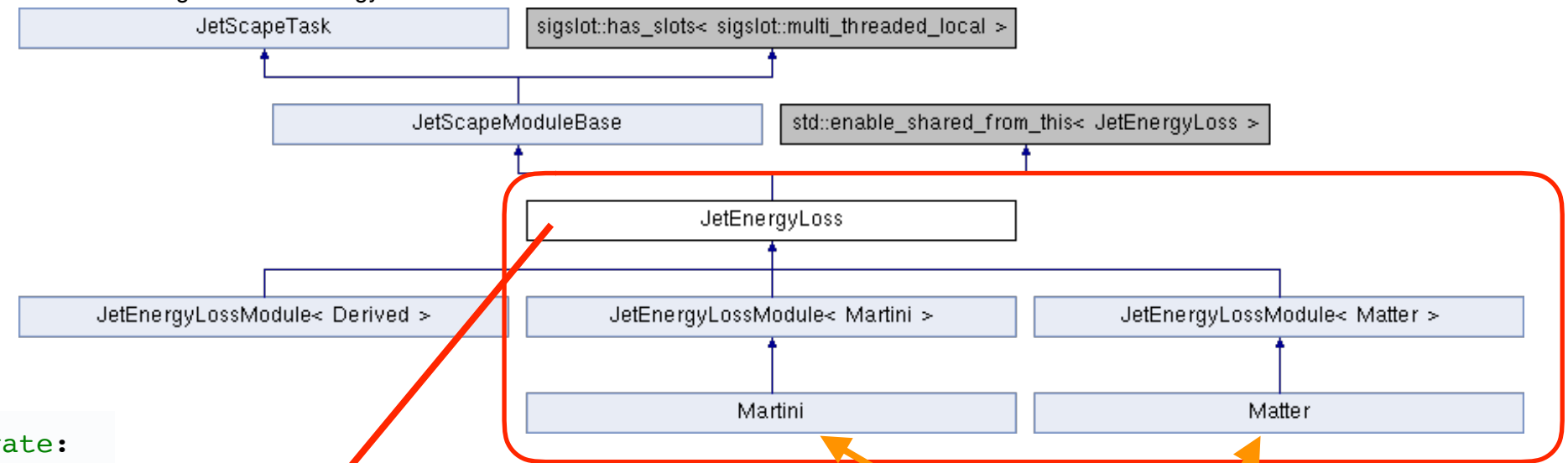
◆ SentInPartons

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

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

Slot(s)

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

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

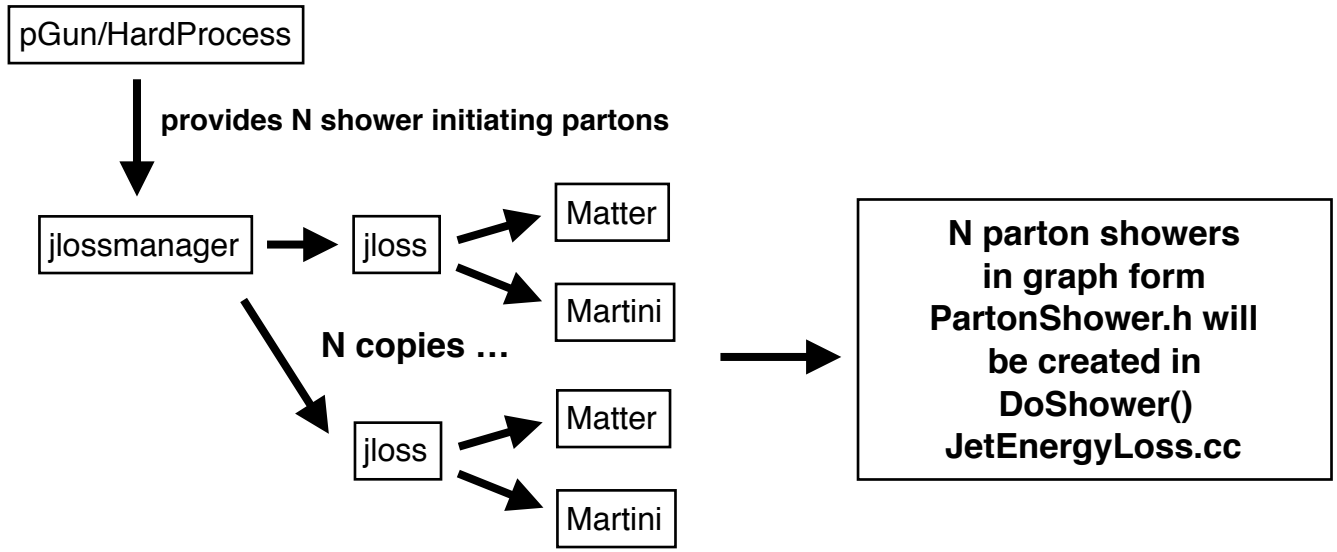
◆ SentInPartons

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

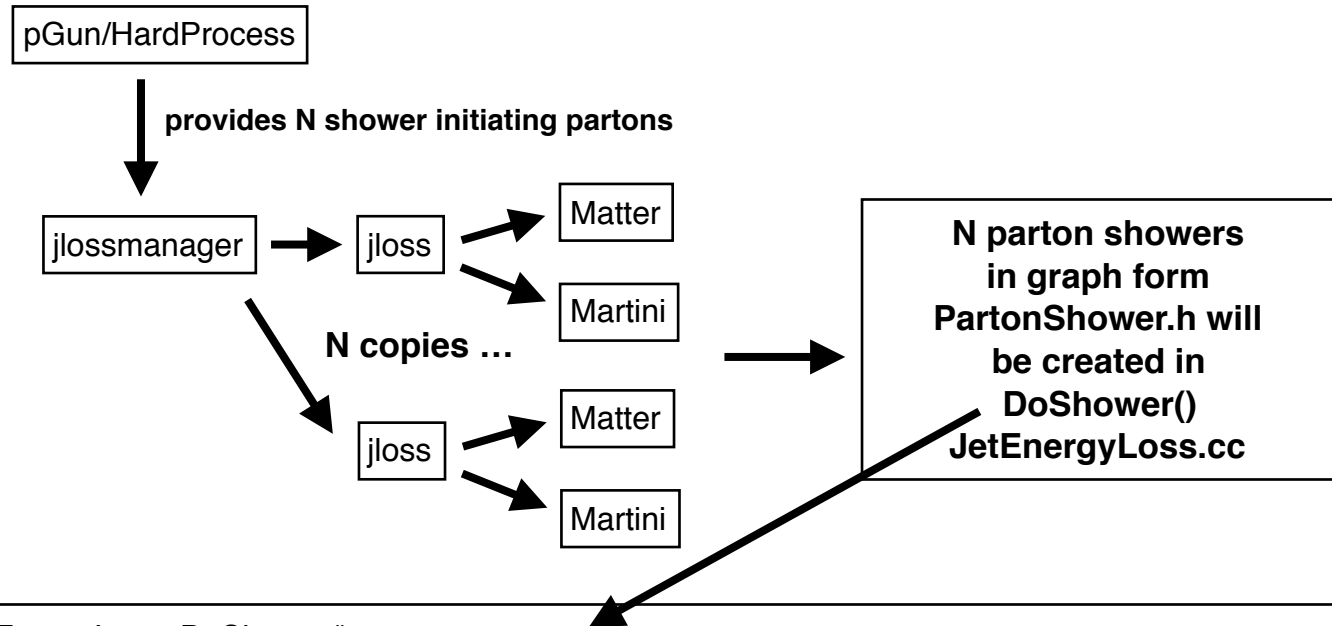
Definition at line **56** of file **JetEnergyLoss.h**.

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

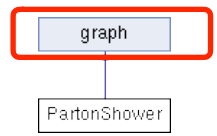
Dynamic generation of graph structure (Parton Shower.h) if something "happened" in an energy loss module …

JetScape Parton Showers as Graph "Containers" (more later)

PartonShower Class Reference

`#include <PartonShower.h>`

Inheritance diagram for PartonShower:



GTL: Graph Template Library

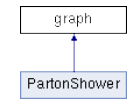
https://github.com/rdmpage/graph-template-library

graph Class Reference

A directed or undirected graph. More...

`#include <graph.h>`

Inheritance diagram for graph:



Public Member Functions

	PartonShower ()
virtual	~PartonShower ()
node	new_vertex (shared_ptr< VertexBase > v)
void	new_parton (node s, node t, shared_ptr< Parton > p)
shared_ptr< VertexBase >	GetVertex (node n)
shared_ptr< Parton >	GetParton (edge e)
shared_ptr< Parton >	GetPartonAt (int n)
shared_ptr< VertexBase >	GetVertexAt (int n)
node	GetNodeAt (int n)
edge	GetEdgeAt (int n)
int	GetNumberOfParents (int n)
int	GetNumberOfChilds (int n)
vector< shared_ptr< Parton > >	GetFinalPartons ()
vector< Parton >	GetFinalPartonsForFastJet ()
int	GetNumberOfPartons () const
int	GetNumberOfVertices () const
void	save_node_info_handler (ostream *o, node n) const
void	save_edge_info_handler (ostream *o, edge n) const
void	load_edge_info_handler (edge e, GML_pair *read)
void	load_node_info_handler (node n, GML_pair *read)
void	pre_clear_handler ()
void	PrintVertices ()
void	PrintPartons ()
void	PrintNodes (bool verbose=true)
void	PrintEdges (bool verbose=true)
void	SaveAsGML (string fName)
void	SaveAsGV (string fName)
void	SaveAsGraphML (string fName)

Public Member Functions inherited from graph

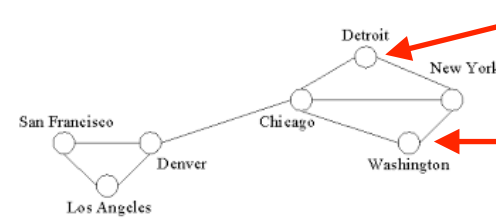
Public Types

typedef list< node >::const_iterator	node_iterator
typedef list< edge >::const_iterator	edge_iterator

Public Member Functions

	graph ()
	graph (const graph &G)
	graph (const graph &G, const list< node > &nodes)
	graph (const graph &G, list< node >::const_iterator it, list< node >::const_iterator end)
virtual	~graph ()
void	make_directed ()
void	make_undirected ()
bool	is_directed () const
bool	is_undirected () const
bool	is_bidirected (edge_map< edge > &rev) const
bool	is_connected () const
bool	is_acyclic () const
int	number_of_nodes () const
int	number_of_edges () const
node	center () const
virtual node	new_node ()
virtual edge	new_edge (node s, node t)
virtual edge	new_edge (const list< node > &sources, const list< node > &targets)
void	del_node (node n)
void	del_all_nodes ()

...



City=Node—> Vertex

Route=Edge—> Parton

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

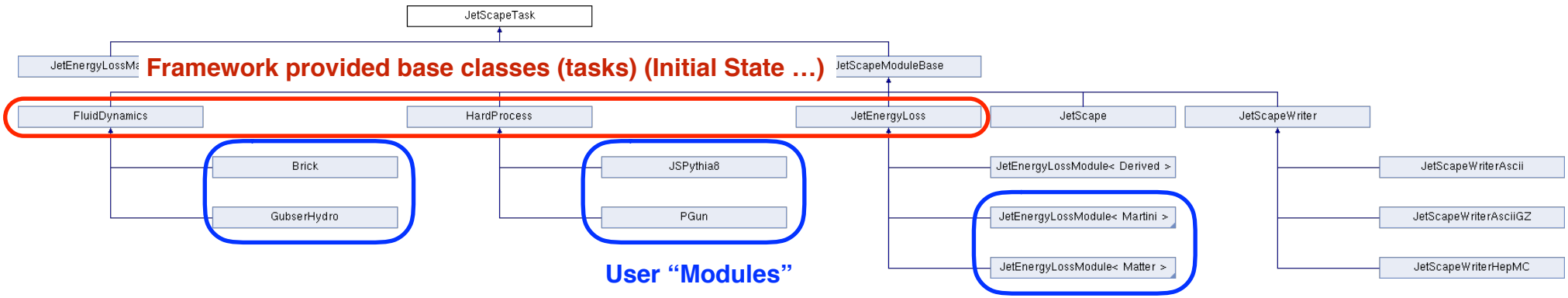
—> Safety!
other mo

```
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!

Generically: How to create your own physics module?



Derive your physics module from the JetScape framework provided bases classes:
FluidDynamics, HardProcess, JetEnergyLoss, InitialState …

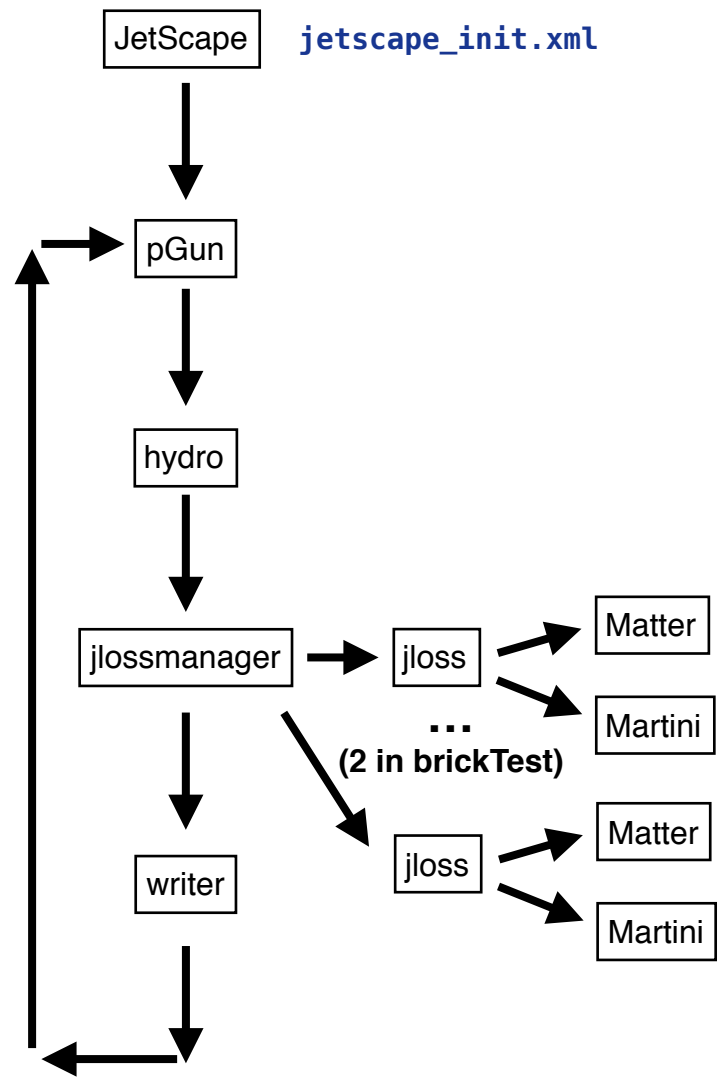
These base classes provide the ***interface and data types*** (defined by the JetScape framework)
—> **Override** these interface functions following the task based approached:
Init(), Exec(), Finish() … (some modules require to override different functions according to their physics needs. A comprehensive manual will be provided at release time …)

<u>Two different approaches concerning your physics code:</u>
a) Implement your **code directly** in your derived JetScape class
b) Use the derived JetScape class as a **wrapper** to your code (either source of library)

More details in the working sessions …

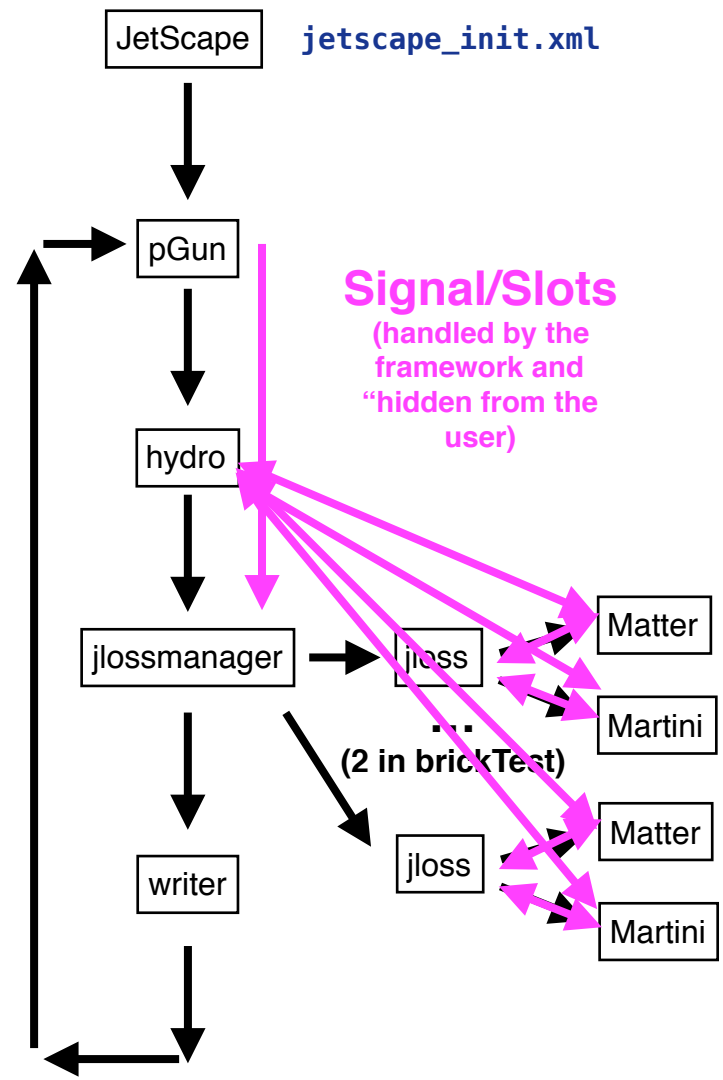
Our “Toy Brick” Test

```
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;
}
```



Our “Toy Brick” Test

```
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;
}
```



Our “Toy E

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

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

  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();

“Stress” Test …

```
...
[Info] 1MB  Run JetScape ...
[Info] 1MB  Number of Events = 1000
[Info] 1MB  Run Event # = 0
[Info] 1MB  Run Hard Process : PGun ...
[Info] 3MB  Run Hydro : Brick ...
[Info] 3MB  Run JetEnergyLoss Manager ...
[Info] 3MB   Number of Hard Partons = 10000
[Info] 32MB   Found 10000 Eloss Manager Tasks/Modules Execute them ...
[Info] 263MB   10000 Eloss Manager Tasks/Modules finished.
[Info] 263MB  Run JetScapeWriterAscii: Write event # 0 ...
[Info] 14MB  Run Event # = 1
[Info] 14MB  Run Hard Process : PGun ...
[Info] 14MB  Run Hydro : Brick ...
[Info] 14MB  Run JetEnergyLoss Manager ...
[Info] 14MB   Number of Hard Partons = 10000
[Info] 34MB   Found 10000 Eloss Manager Tasks/Modules Execute them ...
[Info] 263MB   10000 Eloss Manager Tasks/Modules finished.
[Info] 263MB  Run JetScapeWriterAscii: Write event # 1 ...
[Info] 17MB  Run Event # = 2
[Info] 17MB  Run Hard Process : PGun ...
[Info] 16MB  Run Hydro : Brick ...
[Info] 16MB  Run JetEnergyLoss Manager ...
[Info] 16MB   Number of Hard Partons = 10000
[Info] 34MB   Found 10000 Eloss Manager Tasks/Modules Execute them ...
[Info] 263MB   10000 Eloss Manager Tasks/Modules finished.
[Info] 263MB  Run JetScapeWriterAscii: Write event # 2 ...
[Info] 16MB  Run Event # = 3
[Info] 16MB  Run Hard Process : PGun ...
[Info] 16MB  Run Hydro : Brick ...
[Info] 16MB  Run JetEnergyLoss Manager ...
[Info] 16MB   Number of Hard Partons = 10000
[Info] 34MB   Found 10000 Eloss Manager Tasks/Modules Execute them ...
[Info] 260MB   10000 Eloss Manager Tasks/Modules finished.
[Info] 260MB  Run JetScapeWriterAscii: Write event # 3 ...
[Info] 14MB  Run Event # = 4
[Info] 14MB  Run Hard Process : PGun ...
[Info] 15MB  Run Hydro : Brick ...
[Info] 15MB  Run JetEnergyLoss Manager ...
[Info] 15MB   Number of Hard Partons = 10000
[Info] 34MB   Found 10000 Eloss Manager Tasks/Modules Execute them ...
...
```

In short:

No memory leaks!

All signal/slots working!

Also parallelization of Eloss tasks via C++11 <task> (not shown)

…

(more differential via further debug outputs or Xcode …)

“Stress” Test …

```
...
[Info
[Info
[Info
[Info
[Info
[Info
[Info
[Info
[Info
[Info
[Info
[Info
[Info
[Info
[Info
[Info
[Info
[Info
[Info
[Info
[Info
[Info
[Info
[Info
[Info
[Info
[Info
[Info
[Info
[Info
[Info
[Info
[Info
[Info
[Info
[Info
[Info
[Info
[Info
[Info
...
```

```
[Info] 8MB  Intialize JetScape ...
[Info] 8MB  Created JetScapeXML Instance
[Info] 8MB  Open XML file : ./jetscape_init.xml
[Debug] virtual void JetScape::Init() : JetScape Debug from XML =  on
[Debug] virtual void JetScape::Init() : JetScape Remark from XML =  off
[Info] 8MB  Set Hydro,JetEnergylossManager and IS Pointers for SignalManager to create Signal/Slots
[Info] 8MB  Created JetScapeSignalManager Instance
[Info] 8MB  Found 4 Modules Initialize them ...
[Verbose][7] 8MB virtual void JetScapeTask::InitTasks() :  : # Subtasks = 4
[Info] 8MB  Intialize HardProcess : PGun ...
[Verbose][8] 8MB virtual void HardProcess::Init() :
[Debug] virtual void PGun::InitTask() : Initialize PGun Brick (Test) ...
[Verbose][8] 8MB virtual void PGun::InitTask() :
[Debug] virtual void PGun::InitTask() : PGun to be initilizied ...
[Debug] virtual void PGun::InitTask() : PGun with fixed pT = 100
[Info] 8MB  Parton Gun with fixed pT = 100
[Verbose][7] 8MB virtual void JetScapeTask::InitTasks() :  : # Subtasks = 0
[Info] 8MB  Intialize FluidDynamics : Brick ...
[Verbose][8] 8MB virtual void FluidDynamics::Init() :
[Debug] virtual void Brick::InitTask() : Initialize Brick (Test) ...
[Verbose][8] 8MB virtual void Brick::InitTask() :
[Debug] virtual void Brick::InitTask() : Brick to be initilizied ...
[Debug] virtual void Brick::InitTask() : Brick with T = 0.3
[Info] 8MB  Brick Temperature T = 0.3
[Verbose][8] 8MB virtual void Brick::initialize_hydro(Parameter) : dummy
[Verbose][7] 8MB virtual void JetScapeTask::InitTasks() :  : # Subtasks = 0
[Info] 8MB  Intialize JetEnergyLoss Manager ...
[Info] 8MB  Found 1 Eloss Manager Tasks/Modules Initialize them ...
[Verbose][7] 8MB virtual void JetScapeTask::InitTasks() :  : # Subtasks = 1
[Info] 8MB  Intialize JetEnergyLoss ...
[Info] 8MB  Eloss shower with deltaT = 1 and maxT = 10
[Info] 8MB  Found 2 Eloss Tasks/Modules Initialize them ...
[Verbose][7] 8MB virtual void JetScapeTask::InitTasks() :  : # Subtasks = 2
[Info] 8MB  Intialize Matter ...
[Debug] virtual void Matter::Init() : Matter to be initilizied ...
[Debug] virtual void Matter::Init() : Matter with qhat = 1.1
[Info] 8MB  Intialize Martini ...
[Info] 8MB  Connect JetEnergyLossManager Signal to Hard Process ...
[Info] 8MB  JetScape Ascii Writer initialized with output file = test_out.dat
[Info] 8MB  Run JetScape ...
[Info] 8MB  Number of Events = 3
[Info] 8MB  Run Event # = 0
[Debug] virtual void JetScape::Exec() : Found 4 Modules Execute them ...
[Verbose][7] 8MB virtual void JetScapeTask::ExecuteTasks() :  : # Subtasks = 4
[Info] 8MB  Run Hard Process : PGun ...
[Verbose][8] 8MB virtual void PGun::Exec() : Current Event #0
[Verbose][6] Parton:  0 21 0 100 0 0.826476 100 0 0 0 0
virtual void PGun::Exec() :
[Verbose][6] Parton:  0 -3 0 100 0 3.34748 100 0 0 0 0
virtual void PGun::Exec() :
[Verbose][8] 8MB virtual void PGun::Exec() : 2
[Info] 8MB  Run Hydro : Brick ...
[Verbose][8] 8MB virtual void FluidDynamics::Exec() : Current Event #0
[Verbose][8] 8MB virtual void Brick::evolve_hydro() :
[Verbose][7] 8MB virtual void JetScapeTask::ExecuteTasks() :  : # Subtasks = 0
[Info] 8MB  Run JetEnergyLoss Manager ...
[Debug] virtual void JetEnergyLossManager::Exec() : Task Id = 0x7fff7a22e000
[Info] 8MB   Number of Hard Partons = 2
[Debug] virtual void JetEnergyLossManager::Exec() : Create the 1 th copy because number of intital hard partons = 2
[Verbose][8] 8MB JetEnergyLoss::JetEnergyLoss(const JetEnergyLoss &) : To be copied : # Subtasks = 2
[Verbose][8] 8MB JetEnergyLoss::JetEnergyLoss(const JetEnergyLoss &) : To be copied : # Subtasks = 0
[Verbose][8] 8MB JetEnergyLoss::JetEnergyLoss(const JetEnergyLoss &) : To be copied : # Subtasks = 0
[Info] 8MB   Found 2 Eloss Manager Tasks/Modules Execute them ...
[Debug] virtual void JetEnergyLossManager::Exec() : Check and Create Signal/Slots via JetScapeSignalManaher instance if needed ...
[Verbose][8] 8MB void JetScapeSignalManager::PrintGetHydroCellSignalMap() : [0:0x7ff90af00a28] Matter
[Verbose][8] 8MB void JetScapeSignalManager::PrintGetHydroCellSignalMap() : [1:0x7ff90af00df8] Martini
[Verbose][8] 8MB void JetScapeSignalManager::PrintGetHydroCellSignalMap() : [2:0x7ff90af02488] Matter
[Verbose][8] 8MB void JetScapeSignalManager::PrintGetHydroCellSignalMap() : [3:0x7ff90af02858] Martini
[Verbose][8] 8MB void JetEnergyLossManager::CreateSignalSlots() :
[Verbose][8] 8MB void JetScapeSignalManager::PrintSentInPartonsSignalMap() : [0:0x7ff90af00a28] Matter
[Verbose][8] 8MB void JetScapeSignalManager::PrintSentInPartonsSignalMap() : [1:0x7ff90af00df8] Martini
[Verbose][8] 8MB void JetScapeSignalManager::PrintSentInPartonsSignalMap() : [2:0x7ff90af02488] Matter
[Verbose][8] 8MB void JetScapeSignalManager::PrintSentInPartonsSignalMap() : [3:0x7ff90af02858] Martini
[Verbose][7] 8MB virtual void JetScapeTask::ExecuteTasks() :  : # Subtasks = 2
[Debug] virtual void JetEnergyLoss::Exec() : Run JetEnergyLoss ...
[Debug] virtual void JetEnergyLoss::Exec() : Found 2 Eloss Tasks/Modules Execute them ...
[Verbose][8] 8MB PartonShower::PartonShower() :
[Verbose][8] 8MB void JetEnergyLoss::DoShower() : Hard Parton from Initial Hard Process ...
[Verbose][6] Parton:  0 21 0 100 0 0.826476 100 0 0 0 0
void JetEnergyLoss::DoShower() :
[Verbose][7] 8MB void JetEnergyLoss::DoShower() : Current time = 0 with #Input 1
[Verbose][8] 8MB virtual void Matter::DoEnergyLoss(double, double, vector<Parton> &, vector<Parton> &) : SentInPartons Signal received : 1 100 0x7fff57048240
[Verbose][8] 8MB virtual void Matter::DoEnergyLoss(double, double, vector<Parton> &, vector<Parton> &) : Temperature from Brick (Signal) = 0.3
[Verbose][7] 8MB void JetEnergyLoss::DoShower() : Current time = 1 with #Input 1
[Verbose][8] 8MB virtual void Matter::DoEnergyLoss(double, double, vector<Parton> &, vector<Parton> &) : SentInPartons Signal received : 2 100 0x7fff57048240
[Verbose][8] 8MB virtual void Matter::DoEnergyLoss(double, double, vector<Parton> &, vector<Parton> &) : Temperature from Brick (Signal) = 0.3
[Verbose][7] 8MB void JetEnergyLoss::DoShower() : Current time = 2 with #Input 2
[Verbose][8] 8MB virtual void Matter::DoEnergyLoss(double, double, vector<Parton> &, vector<Parton> &) : SentInPartons Signal received : 3 50 0x7fff57048240
[Verbose][8] 8MB virtual void Matter::DoEnergyLoss(double, double, vector<Parton> &, vector<Parton> &) : Temperature from Brick (Signal) = 0.3
[Verbose][8] 8MB virtual void Matter::DoEnergyLoss(double, double, vector<Parton> &, vector<Parton> &) : SentInPartons Signal received : 3 50 0x7fff57048240
[Verbose][8] 8MB virtual void Matter::DoEnergyLoss(double, double, vector<Parton> &, vector<Parton> &) : Temperature from Brick (Signal) = 0.3
[Verbose][7] 8MB void JetEnergyLoss::DoShower() : Current time = 3 with #Input 2
[Verbose][8] 8MB virtual void Matter::DoEnergyLoss(double, double, vector<Parton> &, vector<Parton> &) : SentInPartons Signal received : 4 50 0x7fff57048240
[Verbose][8] 8MB virtual void Matter::DoEnergyLoss(double, double, vector<Parton> &, vector<Parton> &) : Temperature from Brick (Signal) = 0.3
[Verbose][8] 8MB virtual void Matter::DoEnergyLoss(double, double, vector<Parton> &, vector<Parton> &) : SentInPartons Signal received : 4 50 0x7fff57048240
[Verbose][8] 8MB virtual void Matter::DoEnergyLoss(double, double, vector<Parton> &, vector<Parton> &) : Temperature from Brick (Signal) = 0.3
[Verbose][7] 8MB void JetEnergyLoss::DoShower() : Current time = 4 with #Input 3
[Verbose][8] 8MB virtual void Matter::DoEnergyLoss(double, double, vector<Parton> &, vector<Parton> &) : SentInPartons Signal received : 5 50 0x7fff57048240
[Verbose][8] 8MB virtual void Matter::DoEnergyLoss(double, double, vector<Parton> &, vector<Parton> &) : Temperature from Brick (Signal) = 0.3
[Verbose][8] 8MB virtual void Matter::DoEnergyLoss(double, double, vector<Parton> &, vector<Parton> &) : SentInPartons Signal received : 5 25 0x7fff57048240
[Verbose][8] 8MB virtual void Matter::DoEnergyLoss(double, double, vector<Parton> &, vector<Parton> &) : Temperature from Brick (Signal) = 0.3
[Verbose][8] 8MB virtual void Matter::DoEnergyLoss(double, double, vector<Parton> &, vector<Parton> &) : SentInPartons Signal received : 5 25 0x7fff57048240
[Verbose][8] 8MB virtual void Matter::DoEnergyLoss(double, double, vector<Parton> &, vector<Parton> &) : Temperature from Brick (Signal) = 0.3
[Verbose][7] 8MB void JetEnergyLoss::DoShower() : 1 new root node(s) to be added ...
[Verbose][7] 8MB void JetEnergyLoss::DoShower() : 1 new root node(s) to be added ...
[Verbose][7] 8MB void JetEnergyLoss::DoShower() : Current time = 5 with #Input 5
```

In short:

No memory leaks!
All signal/slots working!

Also parallelization of Eloss tasks via C++11 <task> (not shown)

…

(more differential via further debug outputs or Xcode …)

“Output format” (Ascii, gzipped Ascii or HepMC)

```
0 Event
# HardProcess Parton List: PGun
0 21 0 100 0 0.826476 100 0 0 0 0
0 -3 0 100 0 3.34748 100 0 0 0 0
# Energy loss Shower Initating Parton: Je
0 21 0 100 0 0.826476 100 0 0 0 0
# Parton Shower in JetScape format to be
[0] 0 0 0 0
[1] 0 0 0 0
[2] 0 0 0 2
[3] 0 0 0 2
[4] 0 0 0 4
[5] 0 0 0 4
[6] 0 0 0 5
[7] 0 0 0 5
[8] 0 0 0 5
[9] 0 0 0 4
[10] 0 0 0 5
[11] 0 0 0 4
[12] 0 0 0 6
[13] 0 0 0 5
[14] 0 0 0 6
[15] 0 0 0 5
[16] 0 0 0 6
[17] 0 0 0 5
[18] 0 0 0 6
[19] 0 0 0 5
[20] 0 0 0 6
[21] 0 0 0 6
[22] 0 0 0 7
[23] 0 0 0 7
[24] 0 0 0 7
[25] 0 0 0 7
[26] 0 0 0 8
[27] 0 0 0 8
[28] 0 0 0 8
[29] 0 0 0 8
[30] 0 0 0 8
[31] 0 0 0 7
[32] 0 0 0 8
[33] 0 0 0 7
[34] 0 0 0 10
[35] 0 0 0 9
[36] 0 0 0 10
[37] 0 0 0 9
[38] 0 0 0 10
[39] 0 0 0 10
```

Node—> Vertex

```
[0]->[1] 0 21 0 100 0 0.826476 100 0 0 0 0
[1]-->[2] 0 21 0 50 0 0.826476 50 0 0 0 0
[1]-->[3] 0 21 0 50 0 0.826476 50 0 0 0 0
[2]-->[4] 0 21 0 25 0 0.826476 25 0 0 0 0
[2]-->[5] 0 21 0 25 0 0.826476 25 0 0 0 0
[3]-->[6] 0 21 0 25 0 0.826476 25 0 0 0 0
[3]-->[7] 0 21 0 25 0 0.826476 25 0 0 0 0
[5]-->[8] 0 21 0 12.5 0 0.826476 12.5 0 0 0 0
[9]-->[8] 0 21 0 1.5 0 0.826476 1.5 0 0 0 0
[5]-->[10] 0 21 0 12.5 0 0.826476 12.5 0 0 0 0
[11]-->[10] 0 21 0 1.5 0 0.826476 1.5 0 0 0 0
[4]-->[12] 0 21 0 12.5 0 0.826476 12.5 0 0 0 0
[13]-->[12] 0 21 0 1.5 0 0.826476 1.5 0 0 0 0
[4]-->[14] 0 21 0 12.5 0 0.826476 12.5 0 0 0 0
[15]-->[14] 0 21 0 1.5 0 0.826476 1.5 0 0 0 0
[8]-->[16] 0 21 0 6.25 0 0.826476 6.25 0 0 0 0
[17]-->[16] 0 21 0 1.5 0 0.826476 1.5 0 0 0 0
[8]-->[18] 0 21 0 6.25 0 0.826476 6.25 0 0 0 0
[19]-->[18] 0 21 0 1.5 0 0.826476 1.5 0 0 0 0
[10]-->[20] 0 21 0 6.25 0 0.826476 6.25 0 0 0 0
[10]-->[21] 0 21 0 6.25 0 0.826476 6.25 0 0 0 0
[14]-->[22] 0 21 0 6.25 0 0.826476 6.25 0 0 0 0
[14]-->[23] 0 21 0 6.25 0 0.826476 6.25 0 0 0 0
[21]-->[24] 0 21 0 3.125 0 0.826476 3.125 0 0 0 0
[21]-->[25] 0 21 0 3.125 0 0.826476 3.125 0 0 0 0
[6]-->[26] 0 21 0 12.5 0 0.826476 12.5 0 0 0 0
[6]-->[27] 0 21 0 12.5 0 0.826476 12.5 0 0 0 0
[12]-->[28] 0 21 0 6.25 0 0.826476 6.25 0 0 0 0
[12]-->[29] 0 21 0 6.25 0 0.826476 6.25 0 0 0 0
[23]-->[30] 0 21 0 3.125 0 0.826476 3.125 0 0 0 0
[31]-->[30] 0 21 0 1.5 0 0.826476 1.5 0 0 0 0
[23]-->[32] 0 21 0 3.125 0 0.826476 3.125 0 0 0 0
[33]-->[32] 0 21 0 1.5 0 0.826476 1.5 0 0 0 0
[16]-->[34] 0 21 0 3.125 0 0.826476 3.125 0 0 0 0
[35]-->[34] 0 21 0 1.5 0 0.826476 1.5 0 0 0 0
[16]-->[36] 0 21 0 3.125 0 0.826476 3.125 0 0 0 0
[37]-->[36] 0 21 0 1.5 0 0.826476 1.5 0 0 0 0
[18]-->[38] 0 21 0 3.125 0 0.826476 3.125 0 0 0 0
[18]-->[39] 0 21 0 3.125 0 0.826476 3.125 0 0 0 0
# ElossModule Parton List:Matter
# Energy loss to be implemented accordingly …
1 Event
…
```

Edge—> Parton

Conceptually similar to HepMC (also available as output)

Reading in, “Closure Test” and some “Analysis”: FastJet and Graph traversing

```
int main(int argc, char** argv)
{
  …
  cout<<endl;
  Show();

  //Do some dummy jetfinding ...
  fjcore::JetDefinition jet_def(fjcore::antikt_algorithm, 0.7);

  vector<shared_ptr<PartonShower>> mShowers;

  auto reader=make_shared<JetScapeReaderAscii>(“test_out.dat");

  while (!reader->Finished())
    {
      reader->Next();

      cout<<"Analyze current event = "<<reader->GetCurrentEvent()<<endl;
      mShowers=reader->GetPartonShowers();

      for (int i=0;i<mShowers.size();i++)
        {
          cout<<" Analyze parton shower = "<<i<<endl;

          mShowers[i]->PrintVertices();
          mShowers[i]->PrintPartons();

          //Anti-kT jet finding ... (see: JetDefinition jet_def(antikt_algorithm, 0.7);)
          //Already easily avilable via inclusion of fjcore ...
          fjcore::ClusterSequence cs(mShowers[i]->GetFinalPartonsForFastJet(), jet_def);
          vector<fjcore::PseudoJet> jets = fjcore::sorted_by_pt(cs.inclusive_jets(2));
          cout<<endl;
          cout<<jet_def.description()<<endl;
          // Output of found jets ...
          //cout<<endl;
          for (int k=0;k<jets.size();k++)
            cout<<"Anti-kT jet "<<k<<" : "<<jets[k]<<endl;
          cout<<endl;
          cout<<"Shower initiating parton : "<<*(mShowers[i]->GetPartonAt(0))<<endl;
          cout<<endl;

          AnalyzeGraph(mShowers[i]);

          if (i==0)
            {
              mShowers[i]->SaveAsGV("my_test.gv");
              …
            }
       …
```

Reading in, “Closure Test” and some “Analysis”: FastJet and Graph traversing

```
int ma
{
  …
  cout
  Show

  //Do
  fjco

  vect

  auto

  whil
    {
```

```
…

[Info] 17MB  Open Input File = test_out.dat
[Info] 17MB  File opened
[Info] 17MB  Current Event = 0
Analyze current event = 0
 Analyze parton shower = 0
Vertex list : [0]=0 [1]=0 [2]=2 [3]=2 [4]=4 [5]=4 [6]=5 [7]=5 [8]=5 [9]=4 [10]=5 [11]=4 [12]=6 [13]=5 [14]=6
[15]=5 [16]=6 [17]=5 [18]=6 [19]=5 [20]=6 [21]=6 [22]=7 [23]=7 [24]=7 [25]=7 [26]=8 [27]=8 [28]=8 [29]=8 [30]=8
[31]=7 [32]=8 [33]=7 [34]=10 [35]=9 [36]=10 [37]=9 [38]=10 [39]=10
Parton list : [0]-->[1]=100 [1]-->[2]=50 [1]-->[3]=50 [2]-->[4]=25 [2]-->[5]=25 [3]-->[6]=25 [3]-->[7]=25 [5]--
>[8]=12.5 [9]-->[8]=1.5 [5]-->[10]=12.5 [11]-->[10]=1.5 [4]-->[12]=12.5 [13]-->[12]=1.5 [4]-->[14]=12.5 [15]--
>[14]=1.5 [8]-->[16]=6.25 [17]-->[16]=1.5 [8]-->[18]=6.25 [19]-->[18]=1.5 [10]-->[20]=6.25 [10]-->[21]=6.25
[14]-->[22]=6.25 [14]-->[23]=6.25 [21]-->[24]=3.125 [21]-->[25]=3.125 [6]-->[26]=12.5 [6]-->[27]=12.5 [12]--
>[28]=6.25 [12]-->[29]=6.25 [23]-->[30]=3.125 [31]-->[30]=1.5 [23]-->[32]=3.125 [33]-->[32]=1.5 [16]-->[34]=3.125
[35]-->[34]=1.5 [16]-->[36]=3.125 [37]-->[36]=1.5 [18]-->[38]=3.125 [18]-->[39]=3.125
#------------------------------------------------------------------------
#                      FastJet release 3.2.1 [fjcore]
#                  M. Cacciari, G.P. Salam and G. Soyez
#     A software package for jet finding and analysis at colliders
#                           http://fastjet.fr
#
# Please cite EPJC72(2012)1896 [arXiv:1111.6097] if you use this package
# for scientific work and optionally PLB641(2006)57 [hep-ph/0512210].
#
# FastJet is provided without warranty under the terms of the GNU GPLv2.
# It uses T. Chan's closest pair algorithm, S. Fortune's Voronoi code
# and 3rd party plugin jet algorithms. See COPYING file for details.
#------------------------------------------------------------------------

Longitudinally invariant anti-kt algorithm with R = 0.7 and E scheme recombination
Anti-kT jet 0 :  pt = 106 m = -1.90735e-06 y = 1.11022e-16 phi = 0.826476

Shower initiating parton : 0 21 0 100 0 0.826476 100 0 0 0 0

[Info] 17MB  Some GTL graph/shower analysis/dfs search output:

DFS graph search feature from GTL:
Number of Nodes reached from node 0 = 40
Node/Vertex ordering result from DFS:
[0] [1] [3] [7] [6] [27] [26] [2] [5] [10] [21] [25] [24] [20] [8] [18] [39] [38] [16] [36] [34] [4] [14] [23]
[32] [30] [22] [12] [29] [28] [9] [11] [13] [15] [17] [19] [31] [33] [35] [37]
Edge/Parton ordering result from DFS:
[0]-->[1][1]-->[3][3]-->[7][3]-->[6][6]-->[27][6]-->[26][1]-->[2][2]-->[5][5]-->[10][10]-->[21][21]-->[25][21]--
>[24][10]-->[20][5]-->[8][8]-->[18][18]-->[39][18]-->[38][8]-->[16][16]-->[36][16]-->[34][2]-->[4][4]-->[14]
[14]-->[23][23]-->[32][23]-->[30][14]-->[22][4]-->[12][12]-->[29][12]-->[28]
List of root nodes found in graph/shower : [0][9][11][13][15][17][19][31][33][35][37]

…
```

Reading in, “Closure Test” and some “Analysis”: FastJet and Graph traversing

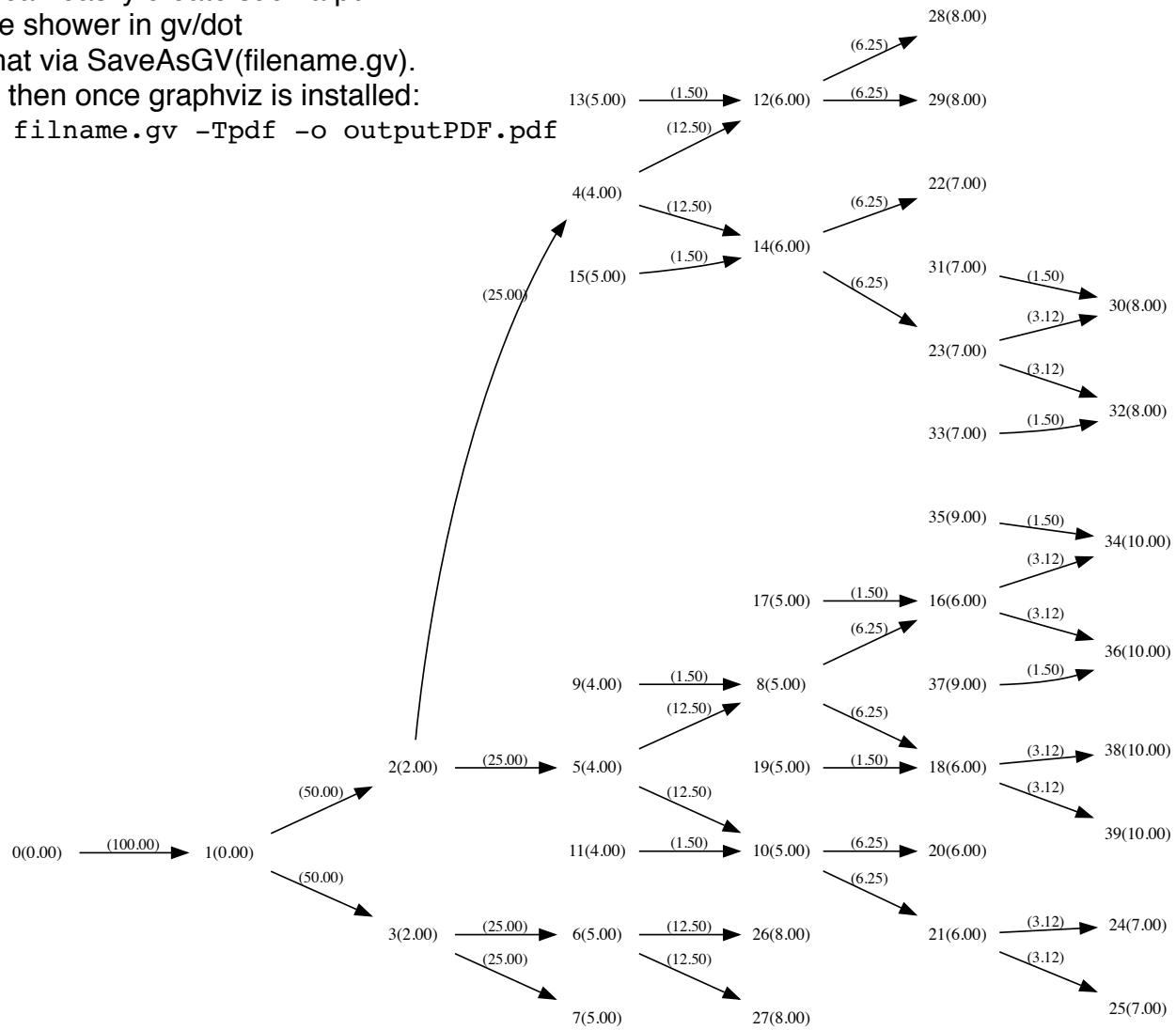
```
int ma
{
  …
  cout
  Show

  //Do
  fjco

  vect

  auto

  whil
    {
```

```
…

[Info] 17MB  Open Input File = test_out.dat
[Info] 17MB  File opened
[Info] 17MB  Current Event = 0
Analyze current event = 0
 Analyze parton shower = 0
Vertex list : [0]=0 [1]=0 [2]=2 [3]=2 [4]=4 [5]=4 [6]=5 [7]=5 [8]=5 [9]=4 [10]=5 [11]=4 [12]=6 [13]=5 [14]=6
[15]=5 [16]=6 [17]=5 [18]=6 [19]=5 [20]=6 [21]=6 [22]=7 [23]=7 [24]=7 [25]=7 [26]=8 [27]=8 [28]=8 [29]=8 [30]=8
[31]=7 [32]=8 [33]=7 [34]=10 [35]=9 [36]=10 [37]=9 [38]=10 [39]=10
Parton list : [0]-->[1]=100 [1]-->[2]=50 [1]-->[3]=50 [2]-->[4]=25 [2]-->[5]=25 [3]-->[6]=25 [3]-->[7]=25 [5]--
>[8]=12.5 [9]-->[8]=1.5 [5]-->[10]=12.5 [11]-->[10]=1.5 [4]-->[12]=12.5 [13]-->[12]=1.5 [4]-->[14]=12.5 [15]--
>[14]=1.5 [8]-->[16]=6.25 [17]-->[16]=1.5 [8]-->[18]=6.25 [19]-->[18]=1.5 [10]-->[20]=6.25 [10]-->[21]=6.25
[14]-->[22]=6.25 [14]-->[23]=6.25 [21]-->[24]=3.125 [21]-->[25]=3.125 [6]-->[26]=12.5 [6]-->[27]=12.5 [12]--
>[28]=6.25 [12]-->[29]=6.25 [23]-->[30]=3.125 [31]-->[30]=1.5 [23]-->[32]=3.125 [33]-->[32]=1.5 [16]-->[34]=3.125
[35]-->[34]=1.5 [16]-->[36]=3.125 [37]-->[36]=1.5 [18]-->[38]=3.125 [18]-->[39]=3.125
#------------------------------------------------------------------------
#                      FastJet release 3.2.1 [fjcore]
#                  M. Cacciari, G.P. Salam and G. Soyez
#     A software package for jet finding and analysis at colliders
#                           http://fastjet.fr
#
# Please cite EPJC72(2012)1896 [arXiv:1111.6097] if you use this package
# for scientific work and optionally PLB641(2006)57 [hep-ph/0512210].
#
# FastJet is provided without warranty under the terms of the GNU GPLv2.
# It uses T. Chan's closest pair algorithm, S. Fortune's Voronoi code
# and 3rd party plugin jet algorithms. See COPYING file for details.
#------------------------------------------------------------------------

Longitudinally invariant anti-kt algorithm with R = 0.7 and E scheme recombination
Anti-kT jet 0 :  pt = 106 m = -1.90735e-06 y = 1.11022e-16 phi = 0.826476

Shower initiating parton : 0 21 0 100 0 0.826476 100 0 0 0 0

[Info] 17MB  Some GTL graph/shower analysis/dfs search output:

DFS graph search feature from GTL:
Number of Nodes reached from node 0 = 40
Node/Vertex ordering result from DFS:
[0] [1] [3] [7] [6] [27] [26] [2] [5] [10] [21] [25] [24] [20] [8] [18] [39] [38] [16] [36] [34] [4] [14] [23]
[32] [30] [22] [12] [29] [28] [9] [11] [13] [15] [17] [19] [31] [33] [35] [37]
Edge/Parton ordering result from DFS:
[0]-->[1][1]-->[3][3]-->[7][3]-->[6][6]-->[27][6]-->[26][1]-->[2][2]-->[5][5]-->[10][10]-->[21][21]-->[25][21]--
>[24][10]-->[20][5]-->[8][8]-->[18][18]-->[39][18]-->[38][8]-->[16][16]-->[36][16]-->[34][2]-->[4][4]-->[14]
[14]-->[23][23]-->[32][23]-->[30][14]-->[22][4]-->[12][12]-->[29][12]-->[28]
List of root nodes found in graph/shower : [0][9][11][13][15][17][19][31][33][35][37]

…
```

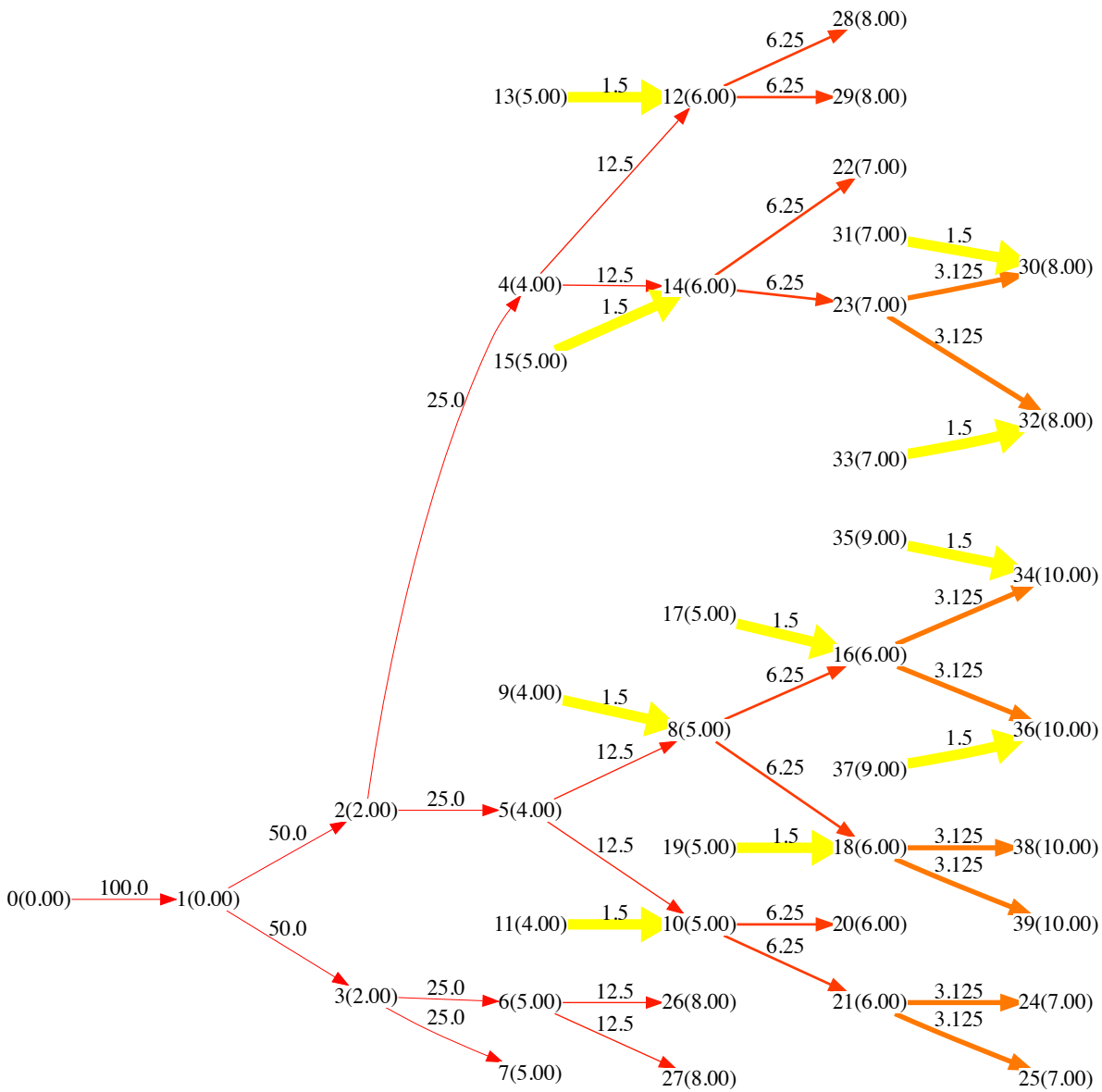
A non-physical JetScape Parton Shower as a Graph …

With graphViz installed on your computer,
you can easily create such a pdf.
Save shower in gv/dot
format via SaveAsGV(filename.gv).
And then once graphviz is installed:

```
dot filname.gv -Tpdf -o outputPDF.pdf
```



A non-physical JetScape Parton Shower as a Graph …

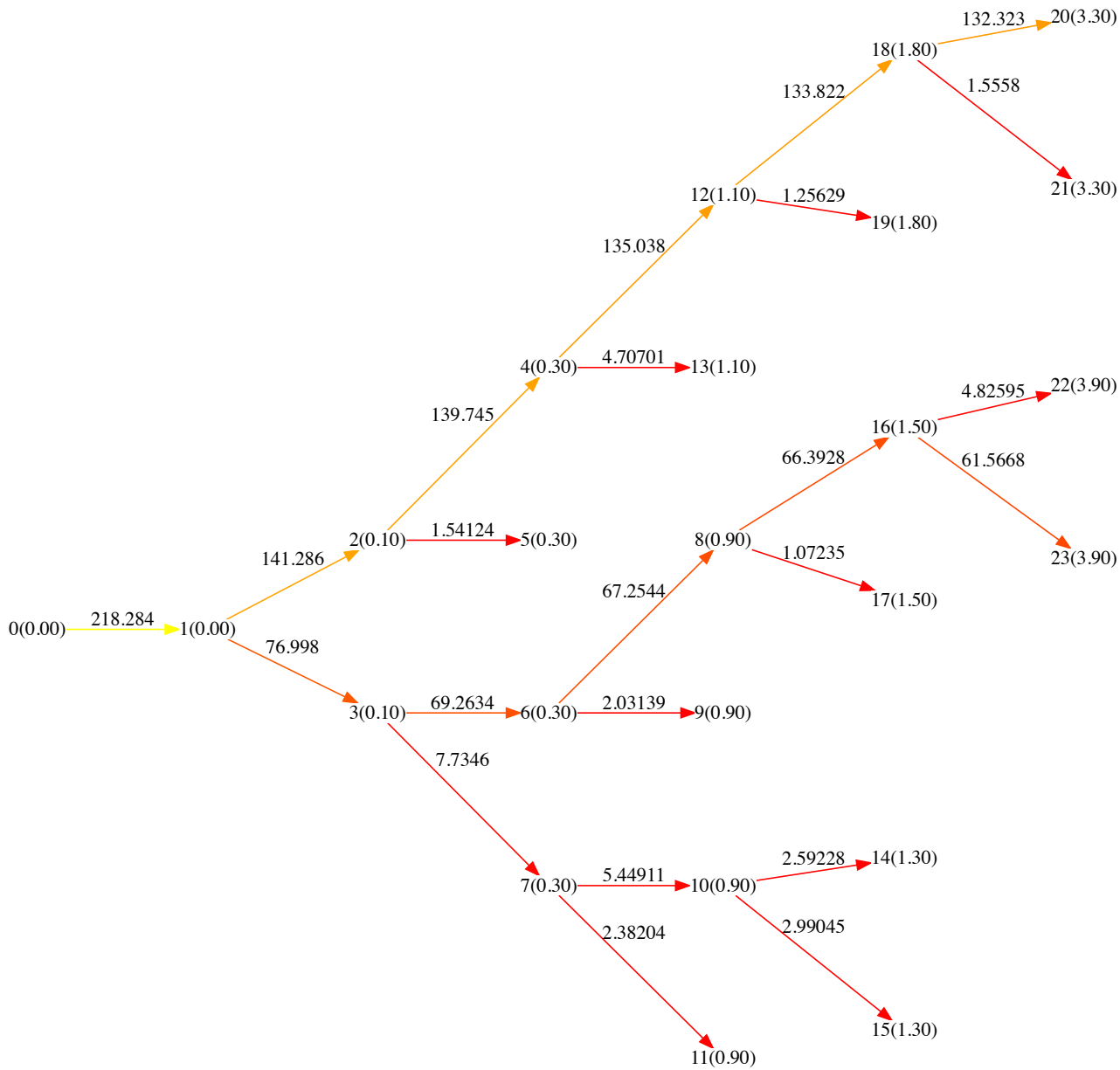


Why graphs?

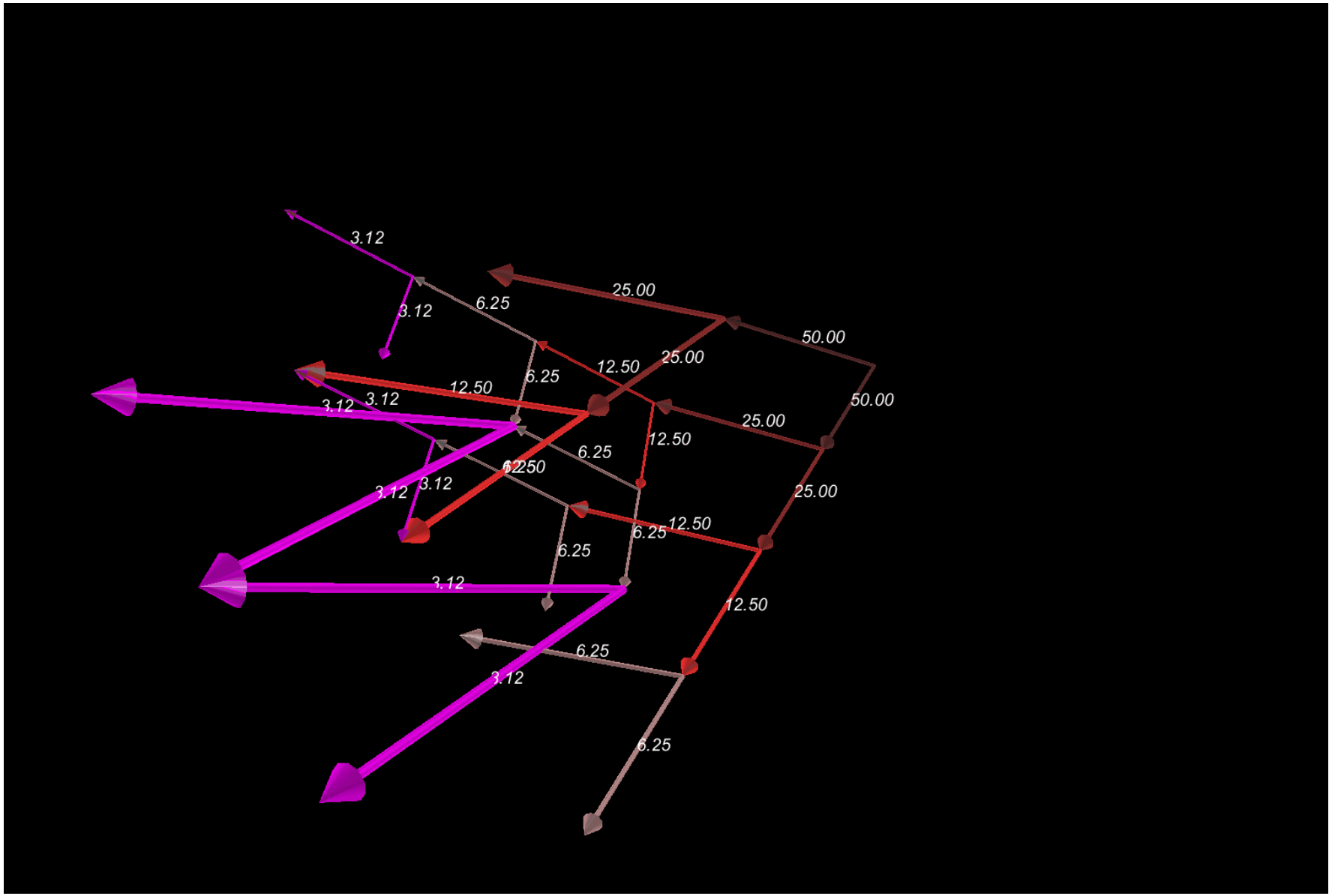
- Graphs are the most generic way of storing “tree”-like informations. A tree = directed acyclic graph. (Used in HepMc but w/o access to “direct graph” features like DFS,BFS … search)
- Graphs are ideal to “search” for connectivity. What hadron belongs to what parton(s). What part of the shower is affected by “medium” partons/initial state …
- A lot of very well established mathematical/computer science algorithms to search/analyze graphs. No need for us to reinvent the wheel!
- Easier interface/usability than having to go through the event record hierarchy on you own.
- Good for visualization (again a lot of tools available) —> Movies!
- Allows us to ask different/extend physics questions?
- Nice connection to computer science —> develop/extend graphs for physics; multi-layer graphs …

And if we do need any other aspects we can always reduce the graph to vectors of partons … at a later stage in our interface. So nothing lost, only potential gains!

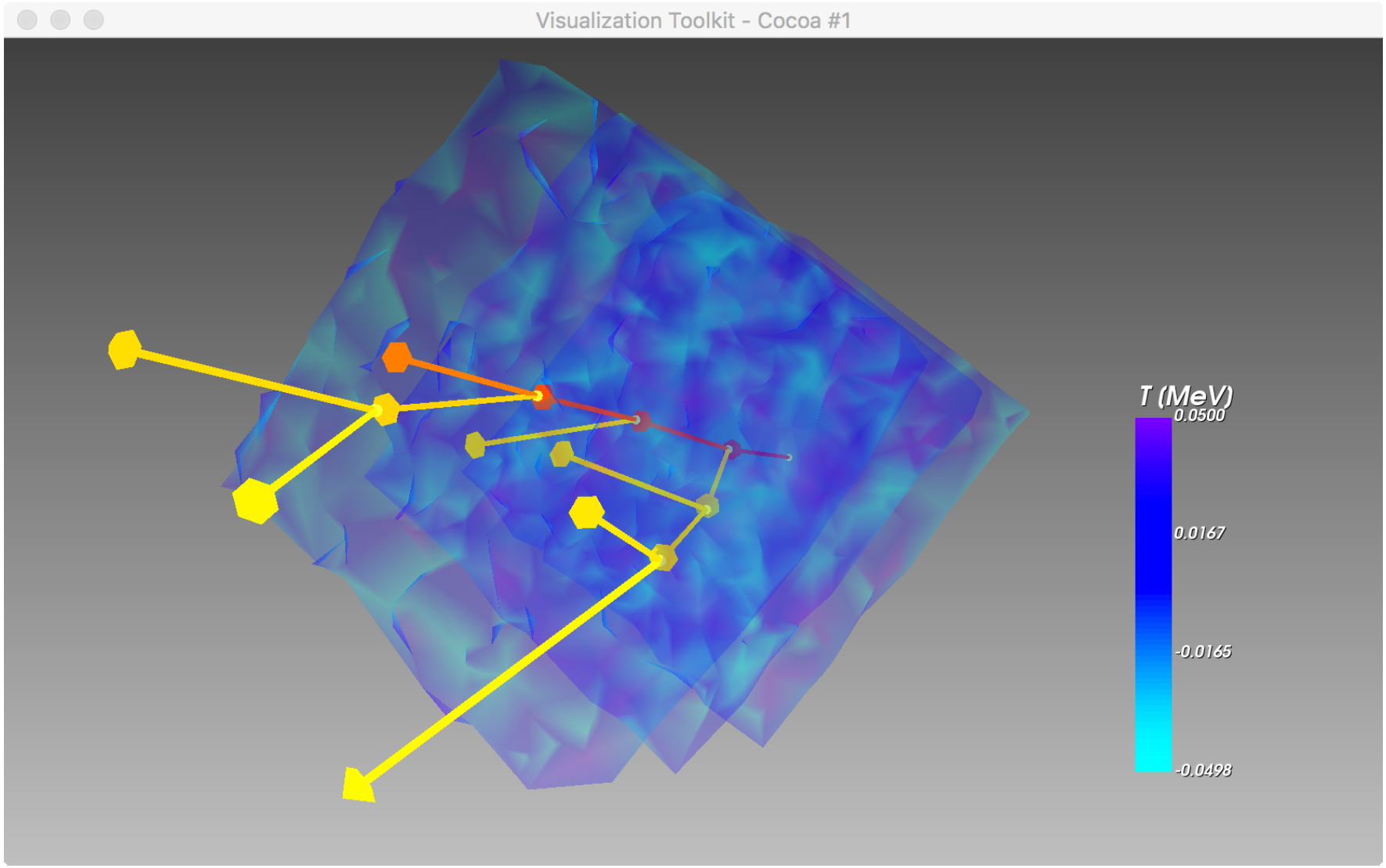
(Hot of the press) A physical Matter Shower as a Graph



Visualization: Toy shower in ROOT

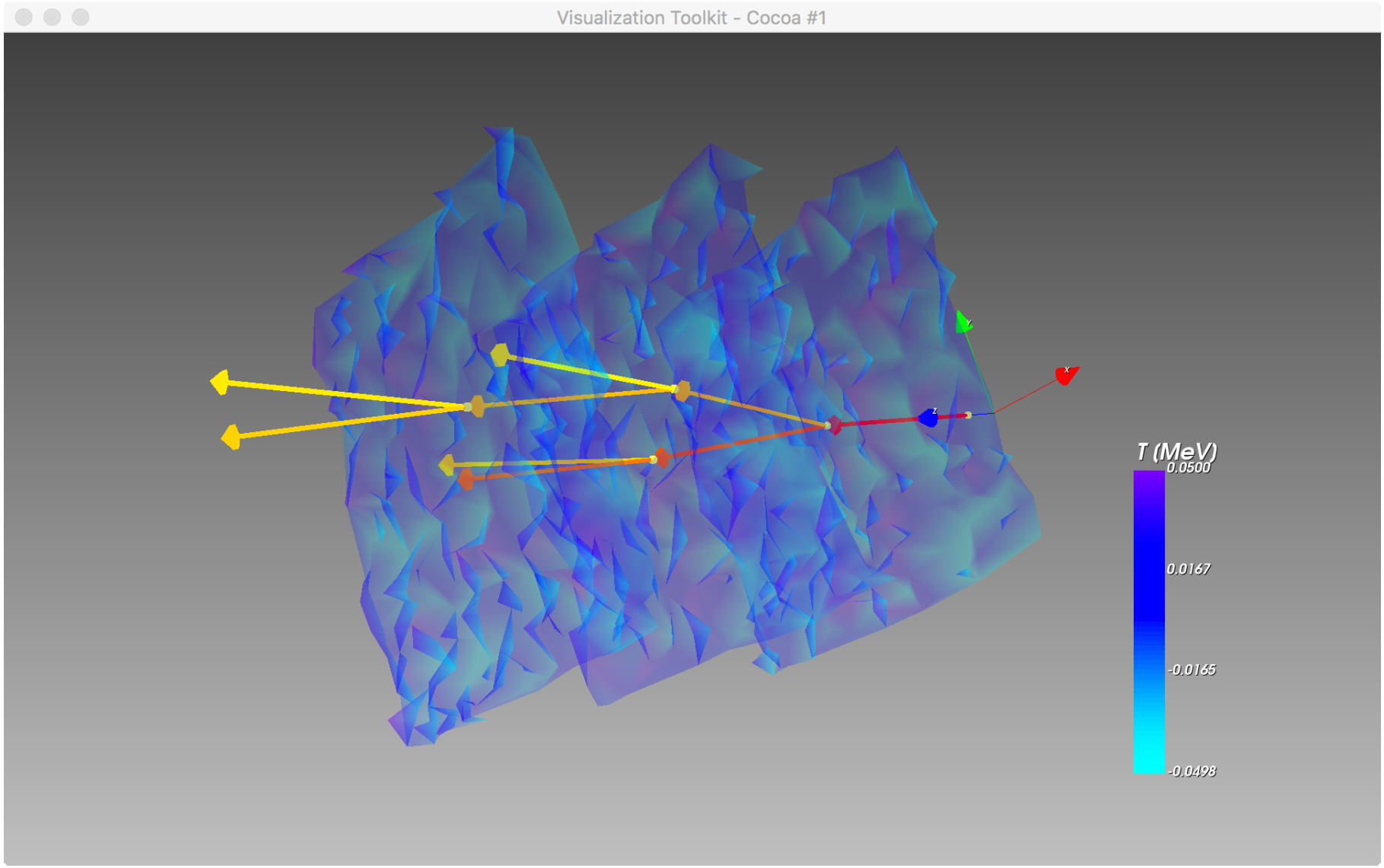


Visualization: A Pythia8 Shower (with fake Medium) …



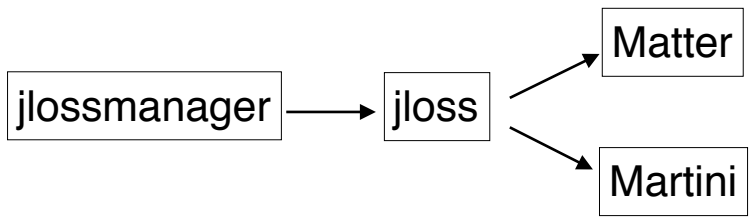
For 3d visualization and movies (besides ROOT) we can use VTK!
(Used by hydro and SMASH, so consistent integration)

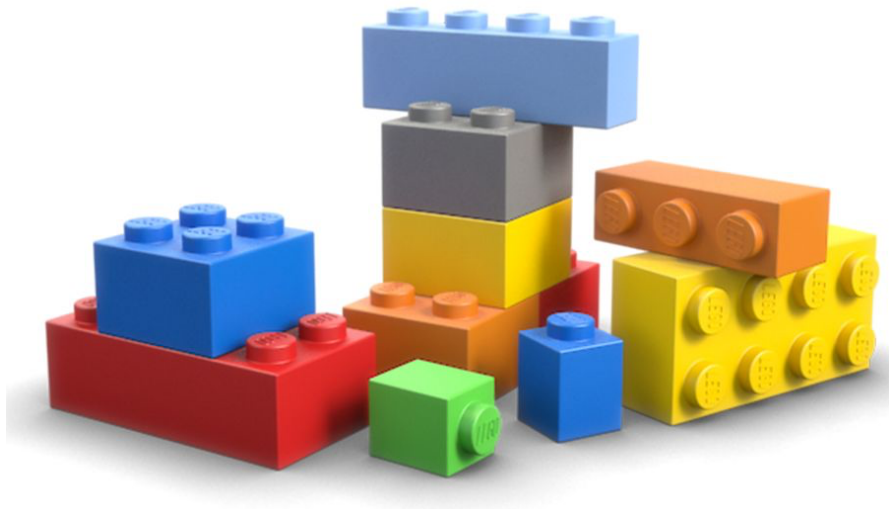
Visualization: A Pythia8 Shower (with fake Medium) …



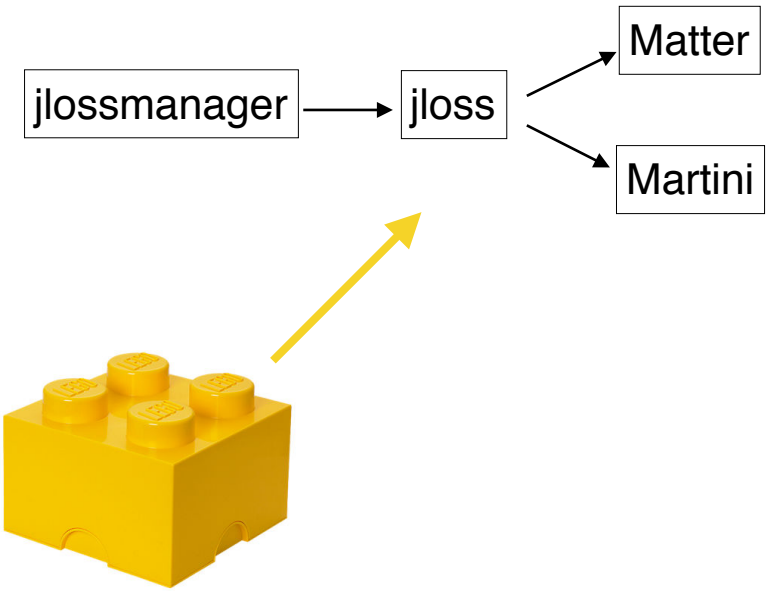
For 3d visualization and movies (besides ROOT) we can use VTK!
(Used by hydro and SMASH, so consistent integration)

Extension/more modular/including more physics via “JetScape Blocks”

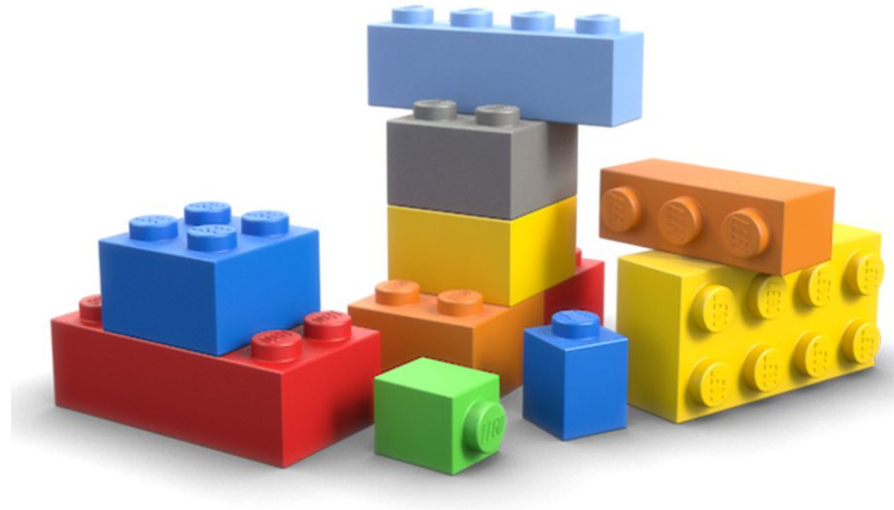




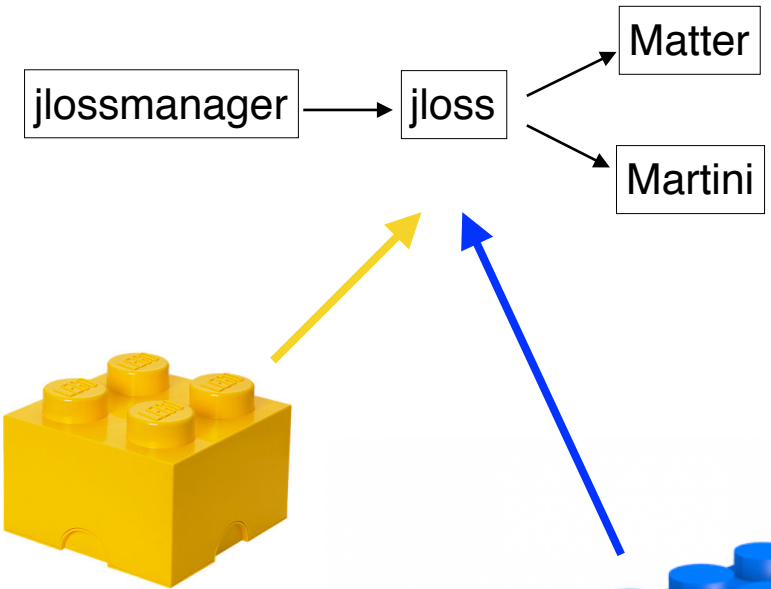
Extension/more modular/including more physics via “JetScape Blocks”



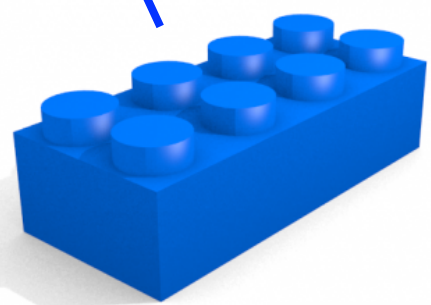
Virtuality Calculation Block



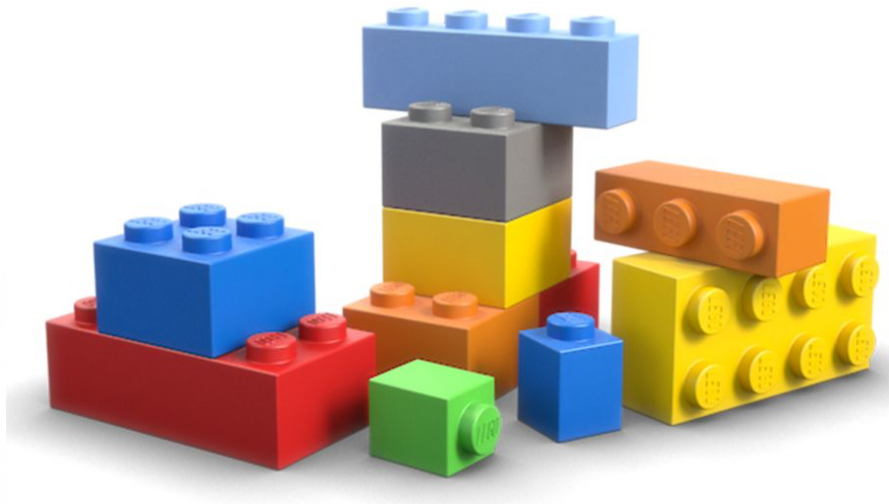
Extension/more modular/including more physics via “JetScape Blocks”



Virtuality Calculation Block

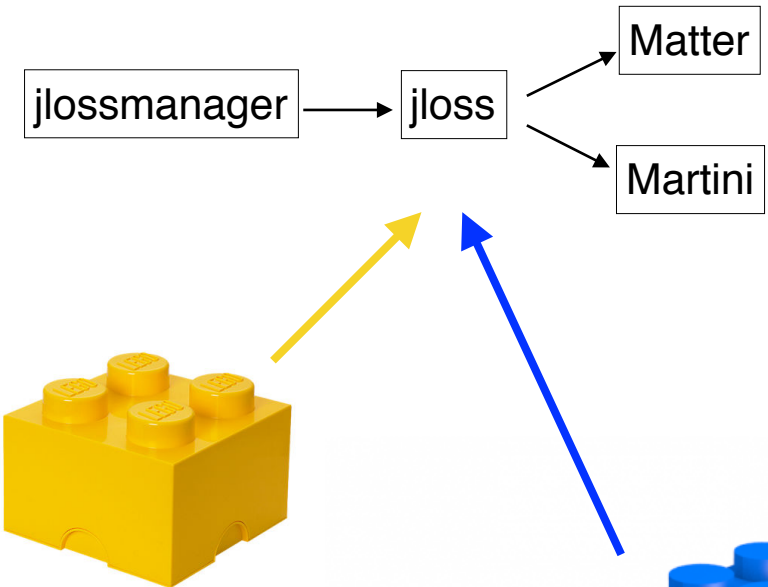


Color Coherence/Antenna Block

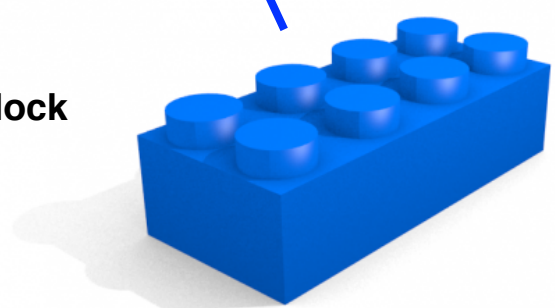


...

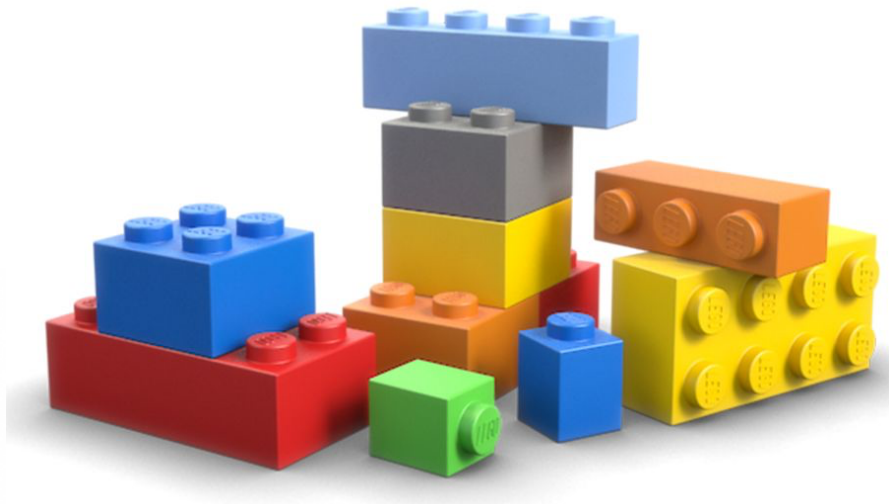
Extension/more modular/including more physics via “JetScape Blocks”



Virtuality Calculation Block



Color Coherence/Antenna Block



...

<u>A potential additional feature wrt to the current framework:</u>

Instead of providing “individual” partons to the jet-energy loss modules provide full shower/history. To be discussed and iterated/your input is needed!

Future: Can we get even more modular (more blocky)?

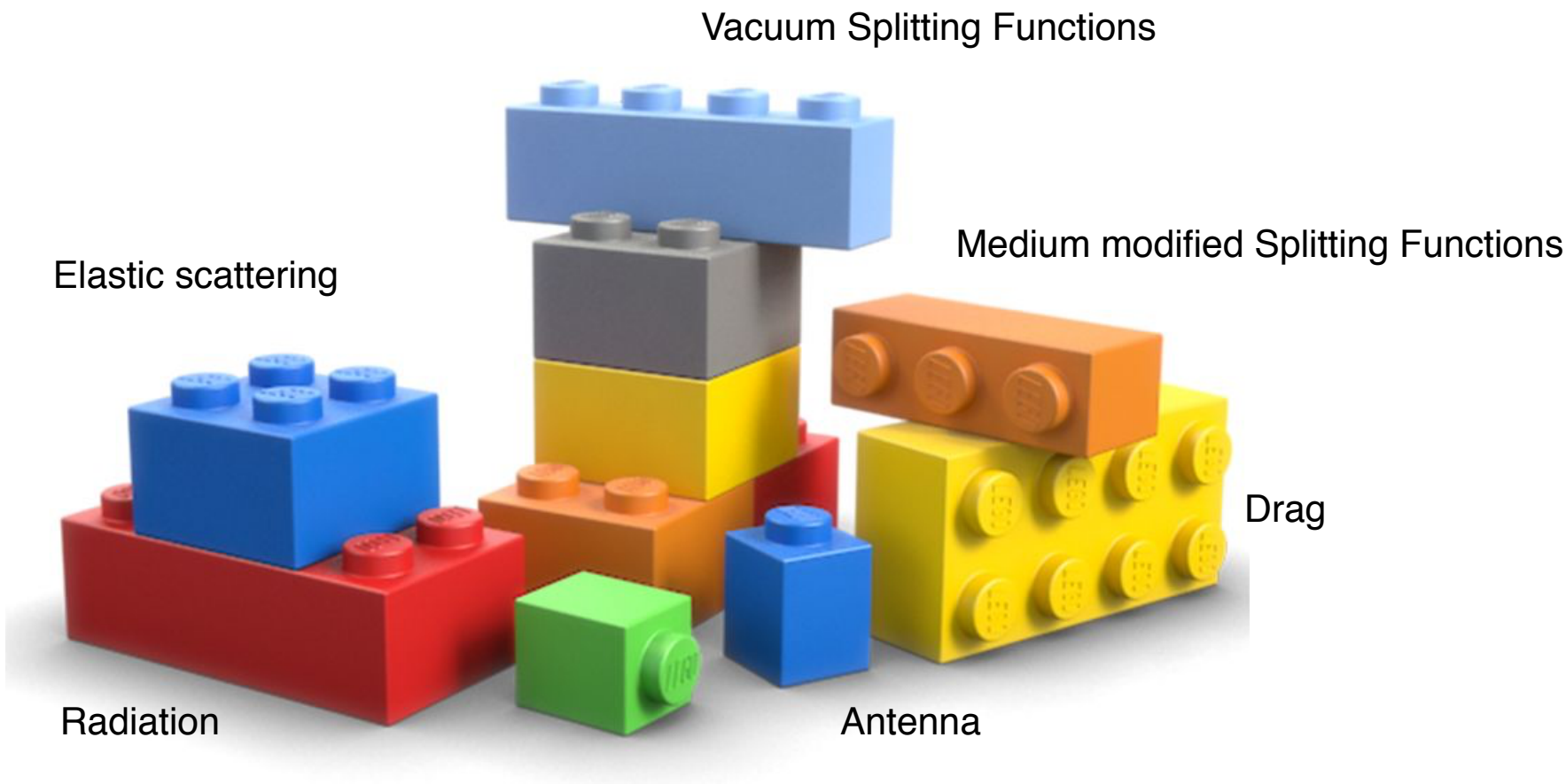
Why would this be desirable?

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

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

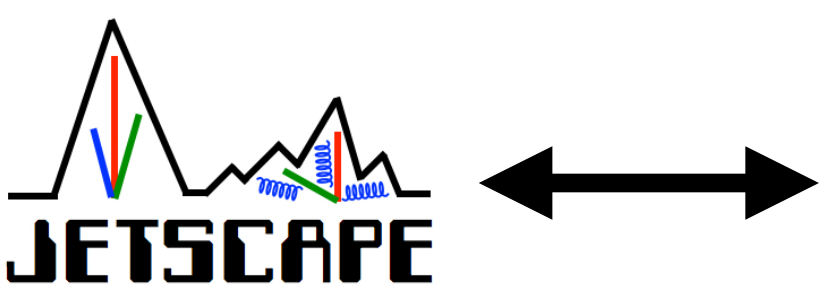
Easier to develop and test new mechanisms/ideas!

Ensures modern/flexible and safe code!



JetScape Event Generator version 1.0

Implemented a modern, general, modular and extendable framework which allows users, as well as developers, to easily use and extend the very complex generation of heavy-ion collisions by providing a consistent user interface between the various physics modules.




```
[Info] *------------------------------------------------------------*
[Info] |                                                            |
[Info] |                    /\                                      |
[Info] |                   /  \                                     |
[Info] |                  / |  \                                    |
[Info] |                 /  |   \             /\                   |
[Info] |                /   |    \         /\/  \                  |
[Info] |               /  \  |  /  \       /\/   |  \/\            |
[Info] |              /    \ | /    \     /     %  |  %   \         |
[Info] |           __/      \|/      \__/       %|%      \/\__     |
[Info] |                                                            |
[Info] |                  JETSCAPE release 1.0                      |
[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] *------------------------------------------------------------*
```

JetScape Framework (technical and physics) Goals for version 2.0

Concurrent Hydro/Jet-Energy loss running.
Synchronization problem to be solved.
—> first tests look promising

Hadronization (Recombination) and Cascade after-burner

Inclusion of more physics via *JetScape Blocks* (color coherence …)

Extension of hydro to more generic *medium* to allow different medium evolutions, like PHSD …

JetScape Framework (technical and physics) Goals for version 2.0

Concurrent Hydro/Jet-Energy loss running.
Synchronization problem to be solved.
—> first tests look promising

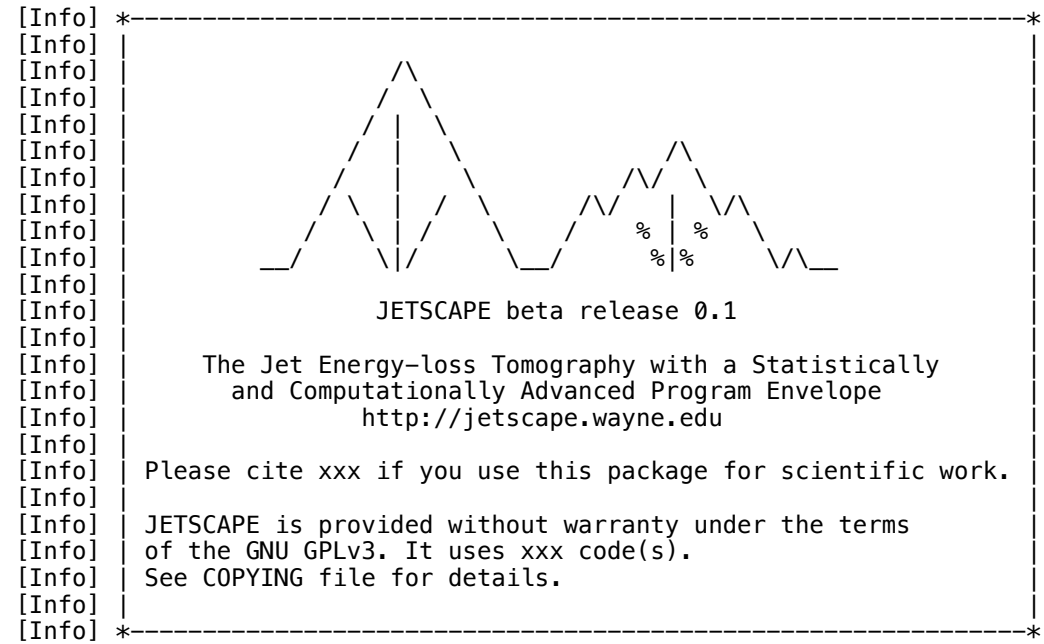
Hadronization (Recombination) and Cascade after-burner

Inclusion of more physics via *JetScape Blocks* (color coherence …)

Extension of hydro to more generic *medium* to allow different medium evolutions, like PHSD …

<u>For more:</u> Your input and feedback is needed!

DEMO(s)…



Play around and do it yourself:

If you are a Mac OS X user, you can download from the indico page: Demo (OSX) …
(or: https://www.dropbox.com/s/hy1dfna2ah3nx7z/JSDemo_v0.2.zip?dl=0)

Unzip and you should see Application icon with the JetScape Logo.
Open for the first with control+right click and accept (we are not authorized Apple developers ;-)). Then you should see a normal GUI. Lets have a look …