MarlinRave

An Alternative for Vertexing and Kinematic Fitting within Marlin

Fabian Moser Wolfgang Waltenberger

Institute for High Energy Physics of the Austrian Academy of Sciences

ECFA Workshop 9-12 June 2008, Warsaw, Poland

The Rave Toolkit

http://projects.hepforge.org/rave/

- Toolkit in form of a shared library providing vertex finding and fitting tools
- Written in C++ with bindings to Java as well
- Has its roots in the CMS vertexing community
- Source code compatibility with CMSSW
- Thoroughly tested code
- Full range of robust vertexing algorithms
- B-Tagging
- Kinematic fitting

ÖAW Austrian Academy of Sciences

The Rave Toolkit

http://projects.hepforge.org/rave/

- Toolkit in form of a shared library providing vertex finding and fitting tools
- Written in C++ with bindings to Java as well
- Has its roots in the CMS vertexing community
- Source code compatibility with CMSSW
- Thoroughly tested code
- Full range of robust vertexing algorithms
- B-Tagging
- Kinematic fitting

ÖAW Austrian Academy of Sciences

The Rave Toolkit

http://projects.hepforge.org/rave/

- Toolkit in form of a shared library providing vertex finding and fitting tools
- Written in C++ with bindings to Java as well
- Has its roots in the CMS vertexing community
- Source code compatibility with CMSSW
- Thoroughly tested code
- Full range of robust vertexing algorithms
- B-Tagging
- Kinematic fitting

The Rave Toolkit

http://projects.hepforge.org/rave/

- Toolkit in form of a shared library providing vertex finding and fitting tools
- Written in C++ with bindings to Java as well
- Has its roots in the CMS vertexing community
- Source code compatibility with CMSSW
- Thoroughly tested code
- Full range of robust vertexing algorithms
- B-Tagging
- Kinematic fitting

The Rave Toolkit

http://projects.hepforge.org/rave/

- Toolkit in form of a shared library providing vertex finding and fitting tools
- Written in C++ with bindings to Java as well
- Has its roots in the CMS vertexing community
- Source code compatibility with CMSSW
- Thoroughly tested code
- Full range of robust vertexing algorithms
- B-Tagging
- Kinematic fitting

The Rave Toolkit

http://projects.hepforge.org/rave/

- Toolkit in form of a shared library providing vertex finding and fitting tools
- Written in C++ with bindings to Java as well
- Has its roots in the CMS vertexing community
- Source code compatibility with CMSSW
- Thoroughly tested code
- Full range of robust vertexing algorithms
- B-Tagging
- Kinematic fitting

The Rave Toolkit

http://projects.hepforge.org/rave/

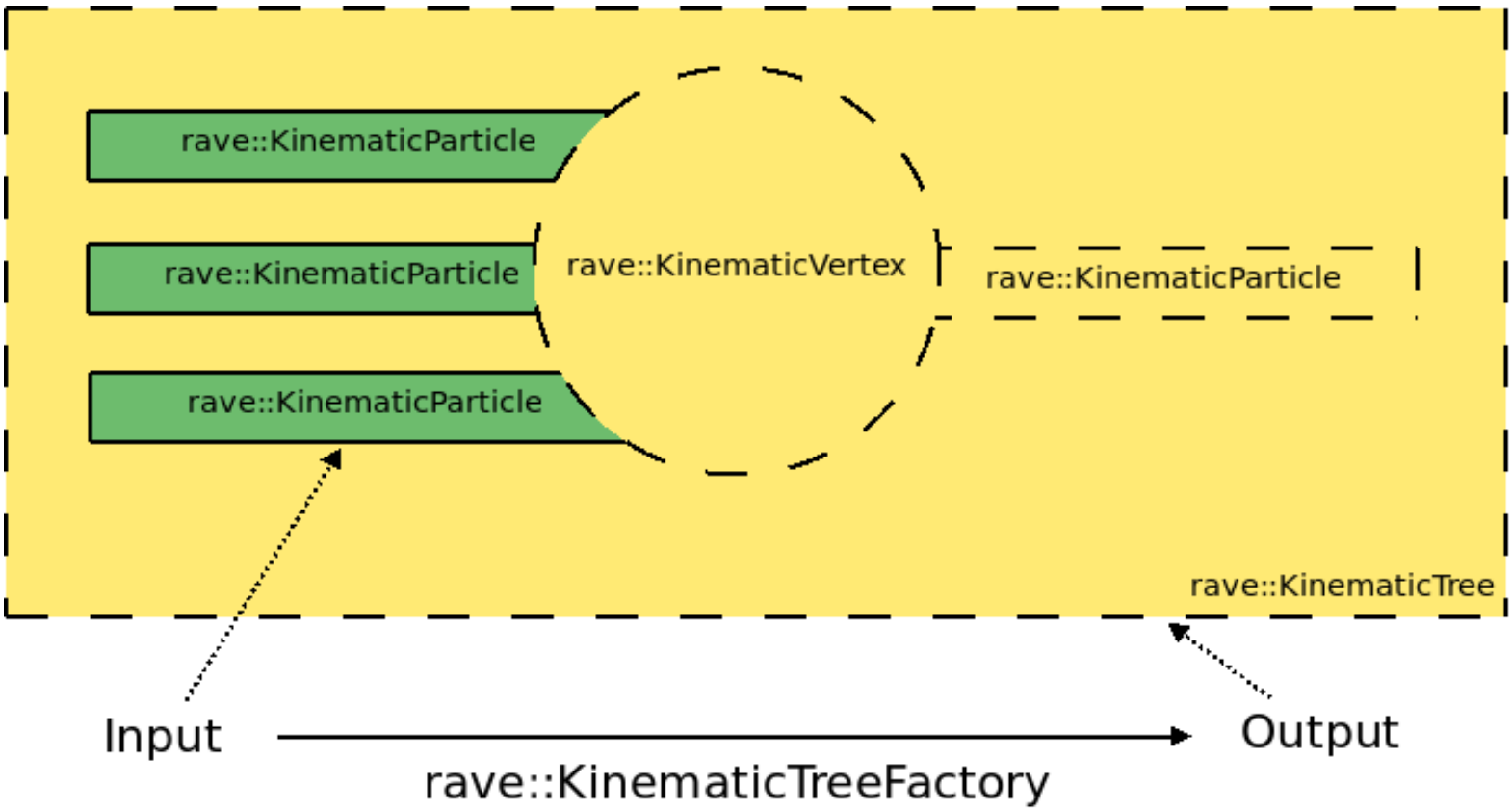
- Toolkit in form of a shared library providing vertex finding and fitting tools
- Written in C++ with bindings to Java as well
- Has its roots in the CMS vertexing community
- Source code compatibility with CMSSW
- Thoroughly tested code
- Full range of robust vertexing algorithms
- B-Tagging
- Kinematic fitting

The Rave Toolkit

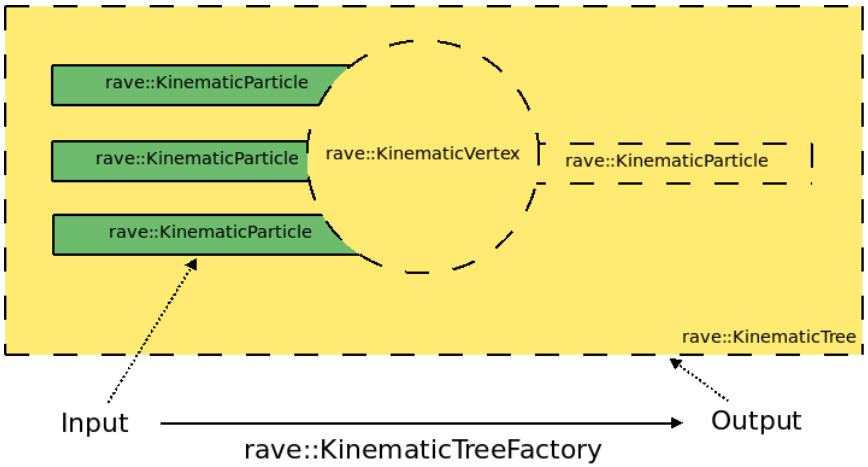
http://projects.hepforge.org/rave/

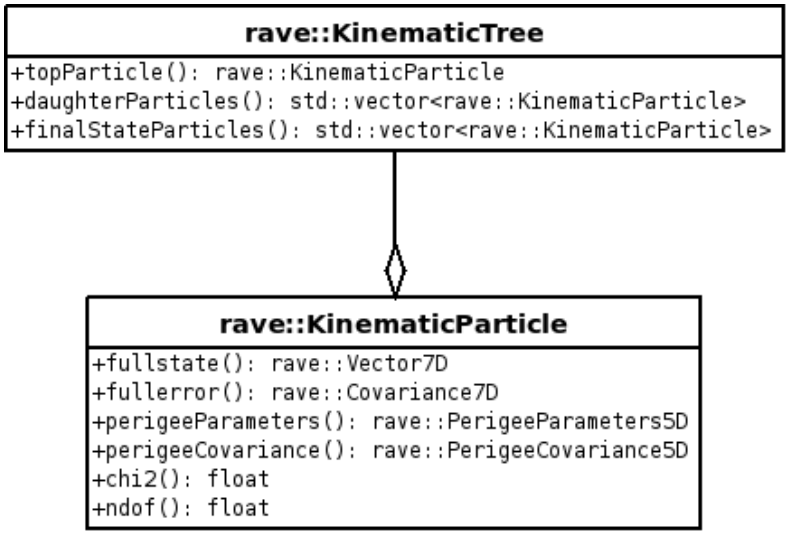
- Toolkit in form of a shared library providing vertex finding and fitting tools
- Written in C++ with bindings to Java as well
- Has its roots in the CMS vertexing community
- Source code compatibility with CMSSW
- Thoroughly tested code
- Full range of robust vertexing algorithms
- B-Tagging
- Kinematic fitting

The Kinematic Fitting Interface 1

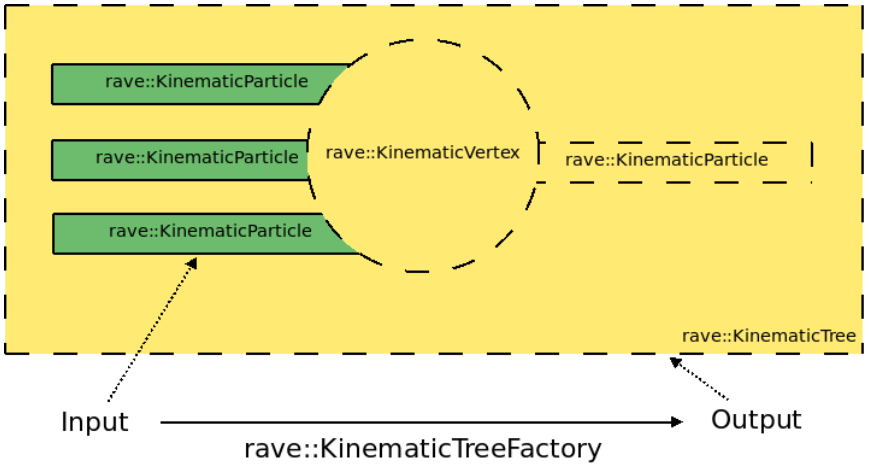


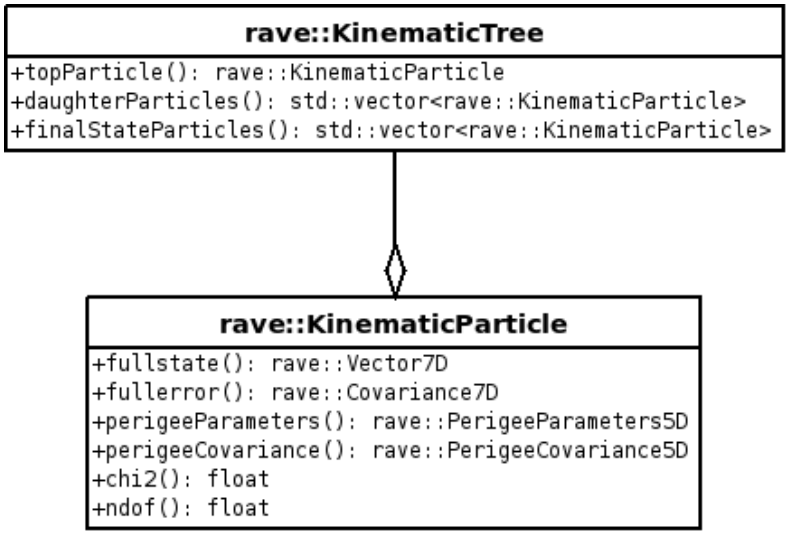
The Kinematic Fitting Interface 1





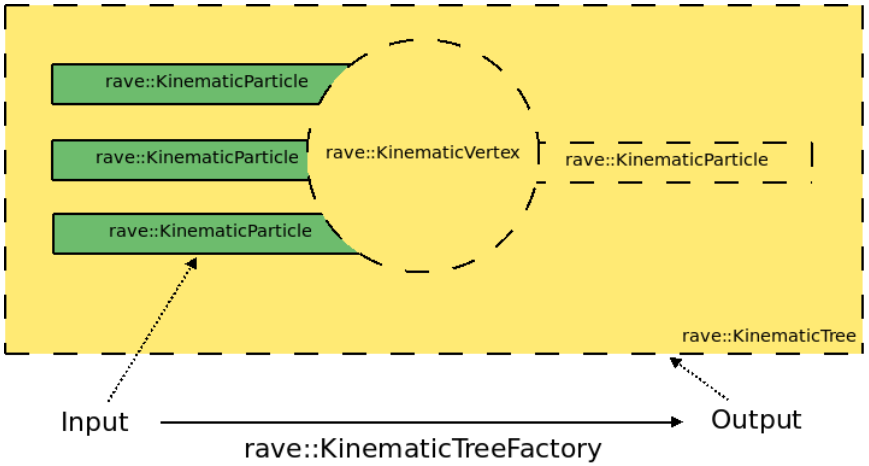
The Kinematic Fitting Interface 1

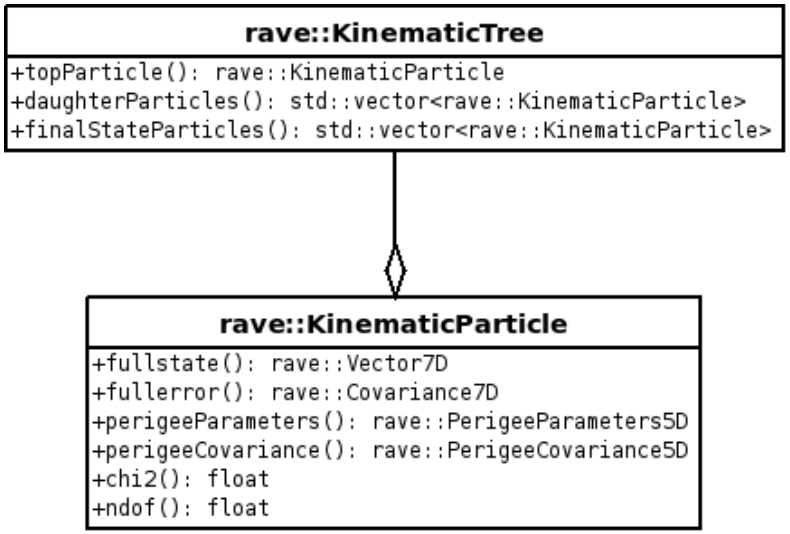





```
void build(const std::vector<rave::KinematicParticle> & particles)
{
  rave::KinematicTreeFactory factory();
  rave::KinematicTree tree = factory.useVertexFitter(particles);
  rave::KinematicParticle mother = tree.topParticle();
  std::cout « "The mass is " « mother.fullstate().m();
  return;
}
```

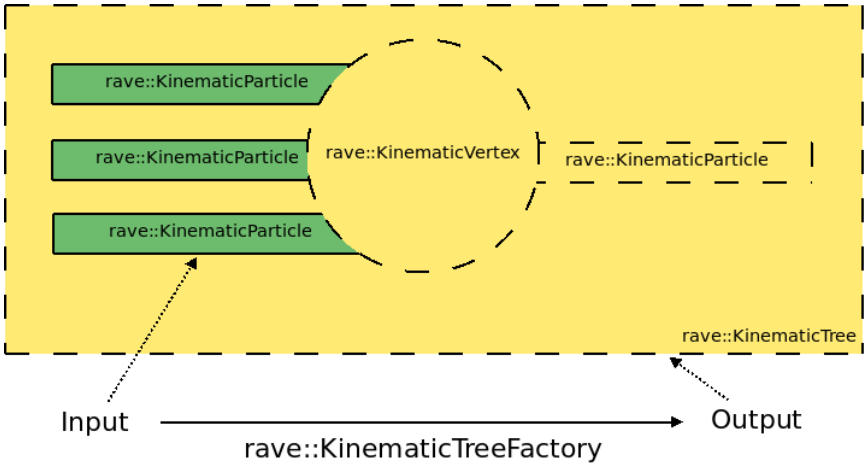
The Kinematic Fitting Interface 1

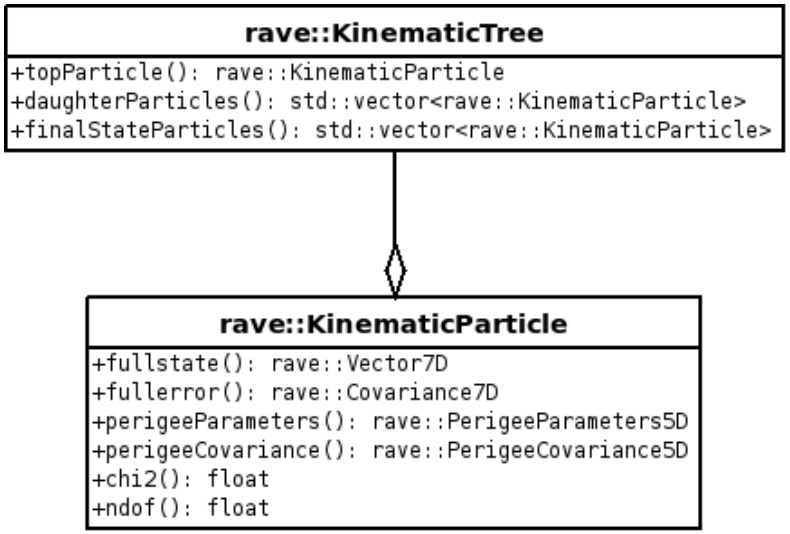





```
void build(const std::vector<rave::KinematicParticle> & particles)
{
  rave::KinematicTreeFactory factory();
  rave::KinematicTree tree = factory.useVertexFitter(particles);
  rave::KinematicParticle mother = tree.topParticle();
  std::cout « "The mass is " « mother.fullstate().m();
  return;
}
```

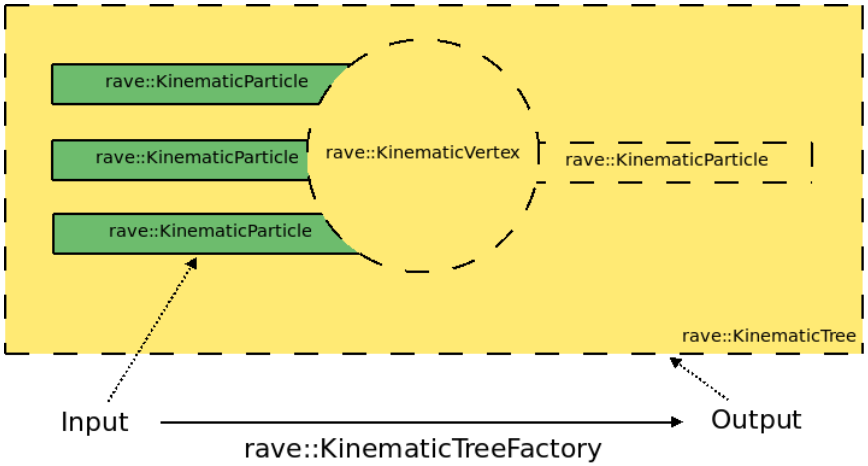
The Kinematic Fitting Interface 1

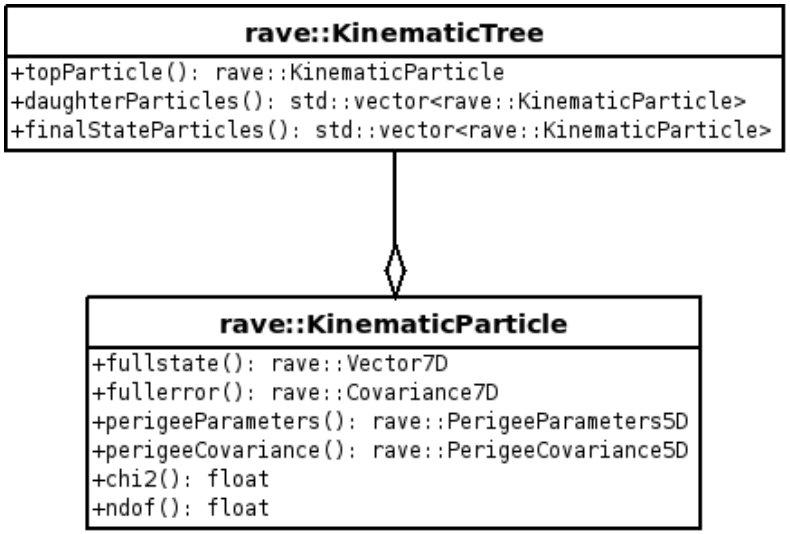





```
void build(const std::vector<rave::KinematicParticle> & particles)
{
  rave::KinematicTreeFactory factory();
  rave::KinematicTree tree = factory.useVertexFitter(particles);
  rave::KinematicParticle mother = tree.topParticle();
  std::cout « "The mass is " « mother.fullstate().m();
  return;
}
```

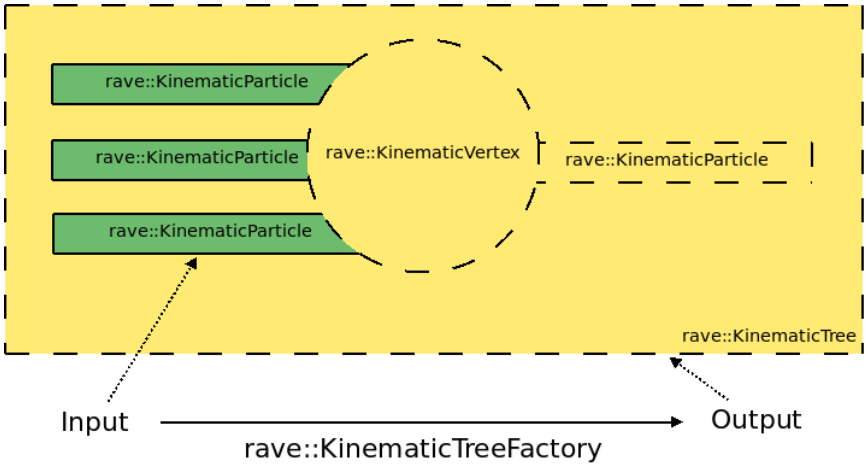
The Kinematic Fitting Interface 1

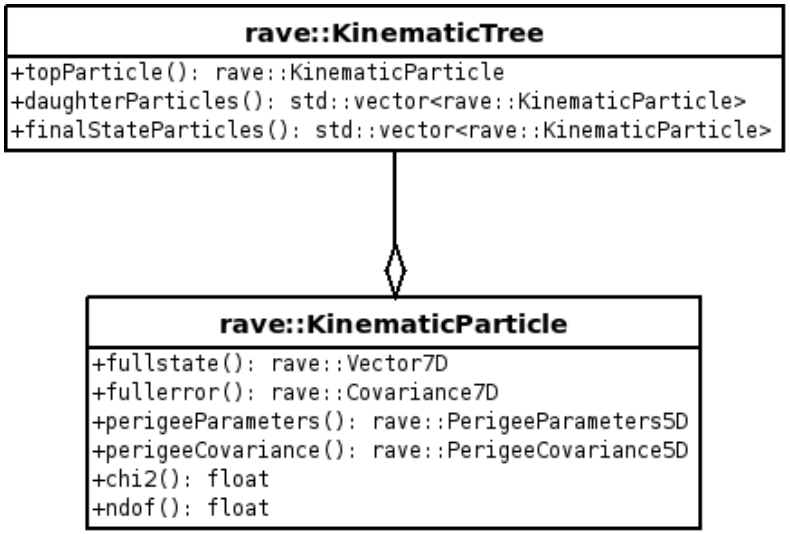





```
void build(const std::vector<rave::KinematicParticle> & particles)
{
  rave::KinematicTreeFactory factory();
  rave::KinematicTree tree = factory.useVertexFitter(particles);
  rave::KinematicParticle mother = tree.topParticle();
  std::cout « "The mass is " « mother.fullstate().m();
  return;
}
```

The Kinematic Fitting Interface 1






```
void build(const std::vector<rave::KinematicParticle> & particles)
{
  rave::KinematicTreeFactory factory();
  rave::KinematicTree tree = factory.useVertexFitter(particles);
  rave::KinematicParticle mother = tree.topParticle();
  std::cout « "The mass is " « mother.fullstate().m();
  return;
}
```

The Kinematic Fitting Interface 2

rave::KinematicConstraintBuilder
+createFourMomentumKinematicConstraint(momentum:rave::Vector4D, deviation:rave::Vector4D): rave::KinematicConstraint
+createMassKinematicConstraint(mass:const float, sigma:const float): rave::KinematicConstraint
+createMultipleKinematicConstraint(): rave::KinematicConstraint
+createEqualMassKinematicConstraint(): rave::KinematicConstraint

```
void build(const std::vector<rave::KinematicParticle> & particles)
{
  rave::KinematicTreeFactory factory();
  rave::KinematicConstraintBuilder builder();
  rave::KinematicConstraint constraint =
      builder.createFourMomentumKinematicConstraint(
          rave::Vector4D(0.,0.,0.,500.),
          rave::Vector4D(0.,0.,0.,0.), true);
  std::vector< rave::KinematicParticle > refitted =
      factory.useParticleFitter(particles, constraint, "ppf:leppf");
  rave::KinematicTree tree = factory.useVertexFitter(refitted);
  rave::KinematicParticle mother = tree.topParticle();
  std::cout « "The mass is " « mother.fullstate().m();
  return;
}
```

The Kinematic Fitting Interface 2

rave::KinematicConstraintBuilder
+createFourMomentumKinematicConstraint(momentum:rave::Vector4D, deviation:rave::Vector4D): rave::KinematicConstraint
+createMassKinematicConstraint(mass:const float, sigma:const float): rave::KinematicConstraint
+createMultipleKinematicConstraint(): rave::KinematicConstraint
+createEqualMassKinematicConstraint(): rave::KinematicConstraint

```
void build(const std::vector<rave::KinematicParticle> & particles)
{
  rave::KinematicTreeFactory factory();
  rave::KinematicConstraintBuilder builder();
  rave::KinematicConstraint constraint =
      builder.createFourMomentumKinematicConstraint(
          rave::Vector4D(0.,0.,0.,500.),
          rave::Vector4D(0.,0.,0.,0.), true);
  std::vector< rave::KinematicParticle > refitted =
      factory.useParticleFitter(particles, constraint, "ppf:leppf");
  rave::KinematicTree tree = factory.useVertexFitter(refitted);
  rave::KinematicParticle mother = tree.topParticle();
  std::cout « "The mass is " « mother.fullstate().m();
  return;
}
```

The Kinematic Fitting Interface 2

rave::KinematicConstraintBuilder
+createFourMomentumKinematicConstraint(momentum:rave::Vector4D, deviation:rave::Vector4D): rave::KinematicConstraint
+createMassKinematicConstraint(mass:const float, sigma:const float): rave::KinematicConstraint
+createMultipleKinematicConstraint(): rave::KinematicConstraint
+createEqualMassKinematicConstraint(): rave::KinematicConstraint

```
void build(const std::vector<rave::KinematicParticle> & particles)
{
  rave::KinematicTreeFactory factory();
  rave::KinematicConstraintBuilder builder();
  rave::KinematicConstraint constraint =
      builder.createFourMomentumKinematicConstraint(
          rave::Vector4D(0.,0.,0.,500.),
          rave::Vector4D(0.,0.,0.,0.), true);
  std::vector< rave::KinematicParticle > refitted =
      factory.useParticleFitter(particles, constraint, "ppf:leppf");
  rave::KinematicTree tree = factory.useVertexFitter(refitted);
  rave::KinematicParticle mother = tree.topParticle();
  std::cout « "The mass is " « mother.fullstate().m();
  return;
}
```

The Kinematic Fitting Interface 2

rave::KinematicConstraintBuilder
+createFourMomentumKinematicConstraint(momentum:rave::Vector4D, deviation:rave::Vector4D): rave::KinematicConstraint
+createMassKinematicConstraint(mass:const float, sigma:const float): rave::KinematicConstraint
+createMultipleKinematicConstraint(): rave::KinematicConstraint
+createEqualMassKinematicConstraint(): rave::KinematicConstraint

```
void build(const std::vector<rave::KinematicParticle> & particles)
{
  rave::KinematicTreeFactory factory();
  rave::KinematicConstraintBuilder builder();
  rave::KinematicConstraint constraint =
      builder.createFourMomentumKinematicConstraint(
          rave::Vector4D(0.,0.,0.,500.),
          rave::Vector4D(0.,0.,0.,0.), true);
  std::vector< rave::KinematicParticle > refitted =
      factory.useParticleFitter(particles, constraint, "ppf:leppf");
  rave::KinematicTree tree = factory.useVertexFitter(refitted);
  rave::KinematicParticle mother = tree.topParticle();
  std::cout « "The mass is " « mother.fullstate().m();
  return;
}
```

The Kinematic Fitting Interface 2

rave::KinematicConstraintBuilder
+createFourMomentumKinematicConstraint(momentum:rave::Vector4D, deviation:rave::Vector4D): rave::KinematicConstraint
+createMassKinematicConstraint(mass:const float, sigma:const float): rave::KinematicConstraint
+createMultipleKinematicConstraint(): rave::KinematicConstraint
+createEqualMassKinematicConstraint(): rave::KinematicConstraint

```
void build(const std::vector<rave::KinematicParticle> & particles)
{
  rave::KinematicTreeFactory factory();
  rave::KinematicConstraintBuilder builder();
  rave::KinematicConstraint constraint =
      builder.createFourMomentumKinematicConstraint(
          rave::Vector4D(0.,0.,0.,500.),
          rave::Vector4D(0.,0.,0.,0.), true);
  std::vector< rave::KinematicParticle > refitted =
      factory.useParticleFitter(particles, constraint, "ppf:leppf");
  rave::KinematicTree tree = factory.useVertexFitter(refitted);
  rave::KinematicParticle mother = tree.topParticle();
  std::cout « "The mass is " « mother.fullstate().m();
  return;
}
```

The Marlin Processors

RaveVertexing Takes a `LCIO::Track` collection and produces a refitted `LCIO::Track` collection and a `EVENT::Vertex` collection.

RaveKinematics Takes a `LCIO::ReconstructedParticle` collection and produces a collection of new and refitted `LCIO::ReconstructedParticles` representing the edges of the `rave::KinematicTree` togehter with a `LCIO::Vertex` collection holding its nodes.

OAW Austrian Academy of Sciences

The Marlin Processors

RaveVertexing Takes a `LCIO::Track` collection and produces a refitted `LCIO::Track` collection and a `EVENT::Vertex` collection.

RaveKinematics Takes a `LCIO::ReconstructedParticle` collection and produces a collection of new and refitted `LCIO::ReconstructedParticle`s representing the edges of the `rave::KinematicTree` togehter with a `LCIO::Vertex` collection holding its nodes.

The RaveVertexing Processor

Configuration

```
<processor name="MyRaveVertexing" type="RaveVertexing">
  <parameter name="Tracks" type="string">Tracks </parameter>
  <parameter name="Method" type="string">default </parameter>
  <parameter name="Verbose" type="int">0 </parameter>
  <parameter name="Vertices" type="string">Vertices </parameter>
  <parameter name="RefittedTracks" type="string">RefittedTracks </parameter>
</processor>
```

Available Methods:

- default Standard Kalman filter
- tkf Trimmed Kalman vertex finder
- avf Adaptive vertex finder/fitter
- mvf Multi vertex finder/fitter

All methods accept optional tuning parameters.

Performance has been shown in Berlin 2008 and Hamburg in May/June 2007

The RaveVertexing Processor

Configuration

```
<processor name="MyRaveVertexing" type="RaveVertexing">
  <parameter name="Tracks" type="string">Tracks </parameter>
  <parameter name="Method" type="string">default </parameter>
  <parameter name="Verbose" type="int">0 </parameter>
  <parameter name="Vertices" type="string">Vertices </parameter>
  <parameter name="RefittedTracks" type="string">RefittedTracks </parameter>
</processor>
```

Available Methods:

- default Standard Kalman filter
- tkf Trimmed Kalman vertex finder
- avf Adaptive vertex finder/fitter
- mvf Multi vertex finder/fitter

All methods accept optional tuning parameters.

Performance has been shown in Berlin 2008 and Hamburg in May/June 2007

The RaveVertexing Processor

Configuration

```
<processor name="MyRaveVertexing" type="RaveVertexing">
  <parameter name="Tracks" type="string">Tracks </parameter>
  <parameter name="Method" type="string">default </parameter>
  <parameter name="Verbose" type="int">0 </parameter>
  <parameter name="Vertices" type="string">Vertices </parameter>
  <parameter name="RefittedTracks" type="string">RefittedTracks </parameter>
</processor>
```

Available Methods:

- default Standard Kalman filter
- tkf Trimmed Kalman vertex finder
- avf Adaptive vertex finder/fitter
- mvf Multi vertex finder/fitter

All methods accept optional tuning parameters.

Performance has been shown in Berlin 2008 and Hamburg in May/June 2007

The RaveVertexing Processor

Configuration

```
<processor name="MyRaveVertexing" type="RaveVertexing">
  <parameter name="Tracks" type="string">Tracks </parameter>
  <parameter name="Method" type="string">default </parameter>
  <parameter name="Verbose" type="int">0 </parameter>
  <parameter name="Vertices" type="string">Vertices </parameter>
  <parameter name="RefittedTracks" type="string">RefittedTracks </parameter>
</processor>
```

Available Methods:

- default Standard Kalman filter
- tkf Trimmed Kalman vertex finder
- avf Adaptive vertex finder/fitter
- mvf Multi vertex finder/fitter

All methods accept optional tuning parameters.

Performance has been shown in Berlin 2008 and Hamburg in May/June 2007

The RaveVertexing Processor

Configuration

```
<processor name="MyRaveVertexing" type="RaveVertexing">
  <parameter name="Tracks" type="string">Tracks </parameter>
  <parameter name="Method" type="string">default </parameter>
  <parameter name="Verbose" type="int">0 </parameter>
  <parameter name="Vertices" type="string">Vertices </parameter>
  <parameter name="RefittedTracks" type="string">RefittedTracks </parameter>
</processor>
```

Available Methods:

- **default** Standard Kalman filter
- **tkf** Trimmed Kalman vertex finder
- **avf** Adaptive vertex finder/fitter
- **mvf** Multi vertex finder/fitter

All methods accept optional tuning parameters.

Performance has been shown in Berlin 2008 and Hamburg in May/June 2007

The RaveVertexing Processor

Configuration

```
<processor name="MyRaveVertexing" type="RaveVertexing">
  <parameter name="Tracks" type="string">Tracks </parameter>
  <parameter name="Method" type="string">default </parameter>
  <parameter name="Verbose" type="int">0 </parameter>
  <parameter name="Vertices" type="string">Vertices </parameter>
  <parameter name="RefittedTracks" type="string">RefittedTracks </parameter>
</processor>
```

Available Methods:

default	Standard Kalman filter
tkf	Trimmed Kalman vertex finder
avf	Adaptive vertex finder/fitter
mvf	Multi vertex finder/fitter

All methods accept optional tuning parameters.

Performance has been shown in Berlin 2008 and Hamburg in May/June 2007

The RaveKinematics Processor 1

Configuration

```
<processor name="MyRaveKinematics" type="RaveKinematics">
  <parameter name="Particles" type="string">Particles </parameter>
  <parameter name="Topology" type="string">SingleVertex </parameter>
  <parameter name="Parameters" type="string"> </parameter>
  <parameter name="Verbose" type="int">0 </parameter>
  <parameter name="KinematicParticles" type="string">KinematicParticles </parameter>
  <parameter name="KinematicVertices" type="string">KinematicVertices </parameter>
</processor>
```

Currently available topologies:

SingleVertex Fits all given particles to one vertex

TwoTrackMass Assumes an invariant mass of the first two given particles

WW4Jet Takes four jets and pairs them to two Ws histogramming the resulting W masses

New topologies are easily introduced by pasting them into the "topologies" directory of the MarlinRave processor

The RaveKinematics Processor 1

Configuration

```
<processor name="MyRaveKinematics" type="RaveKinematics">
  <parameter name="Particles" type="string">Particles </parameter>
  <parameter name="Topology" type="string">SingleVertex </parameter>
  <parameter name="Parameters" type="string"> </parameter>
  <parameter name="Verbose" type="int">0 </parameter>
  <parameter name="KinematicParticles" type="string">KinematicParticles </parameter>
  <parameter name="KinematicVertices" type="string">KinematicVertices </parameter>
</processor>
```

Currently available topologies:

SingleVertex Fits all given particles to one vertex

TwoTrackMass Assumes an invariant mass of the first two given particles

WW4Jet Takes four jets and pairs them to two Ws histogramming the resulting W masses

New topologies are easily introduced by pasting them into the “topologies” directory of the MarlinRave processor.

The RaveKinematics Processor 1

Configuration

```
<processor name="MyRaveKinematics" type="RaveKinematics">
  <parameter name="Particles" type="string">Particles </parameter>
  <parameter name="Topology" type="string">SingleVertex </parameter>
  <parameter name="Parameters" type="string"> </parameter>
  <parameter name="Verbose" type="int">0 </parameter>
  <parameter name="KinematicParticles" type="string">KinematicParticles </parameter>
  <parameter name="KinematicVertices" type="string">KinematicVertices </parameter>
</processor>
```

Currently available topologies:

SingleVertex Fits all given particles to one vertex

TwoTrackMass Assumes an invariant mass of the first two given particles

WW4Jet Takes four jets and pairs them to two Ws histogramming the resulting W masses

New topologies are easily introduced by pasting them into the "topologies" directory of the MarlinRave processor.

The RaveKinematics Processor 1

Configuration

```
<processor name="MyRaveKinematics" type="RaveKinematics">
  <parameter name="Particles" type="string">Particles </parameter>
  <parameter name="Topology" type="string">SingleVertex </parameter>
  <parameter name="Parameters" type="string"> </parameter>
  <parameter name="Verbose" type="int">0 </parameter>
  <parameter name="KinematicParticles" type="string">KinematicParticles </parameter>
  <parameter name="KinematicVertices" type="string">KinematicVertices </parameter>
</processor>
```

Currently available topologies:

SingleVertex Fits all given particles to one vertex

TwoTrackMass Assumes an invariant mass of the first two given particles

WW4Jet Takes four jets and pairs them to two Ws histogramming the resulting W masses

New topologies are easily introduced by pasting them into the "topologies" directory of the MarlinRave processor.

The RaveKinematics Processor 1

Configuration

```
<processor name="MyRaveKinematics" type="RaveKinematics">
  <parameter name="Particles" type="string">Particles </parameter>
  <parameter name="Topology" type="string">SingleVertex </parameter>
  <parameter name="Parameters" type="string"> </parameter>
  <parameter name="Verbose" type="int">0 </parameter>
  <parameter name="KinematicParticles" type="string">KinematicParticles </parameter>
  <parameter name="KinematicVertices" type="string">KinematicVertices </parameter>
</processor>
```

Currently available topologies:

SingleVertex Fits all given particles to one vertex

TwoTrackMass Assumes an invariant mass of the first two given particles

WW4Jet Takes four jets and pairs them to two Ws histogramming the resulting W masses

New topologies are easily introduced by pasting them into the "topologies" directory of the MarlinRave processor.

The RaveKinematics Processor 2

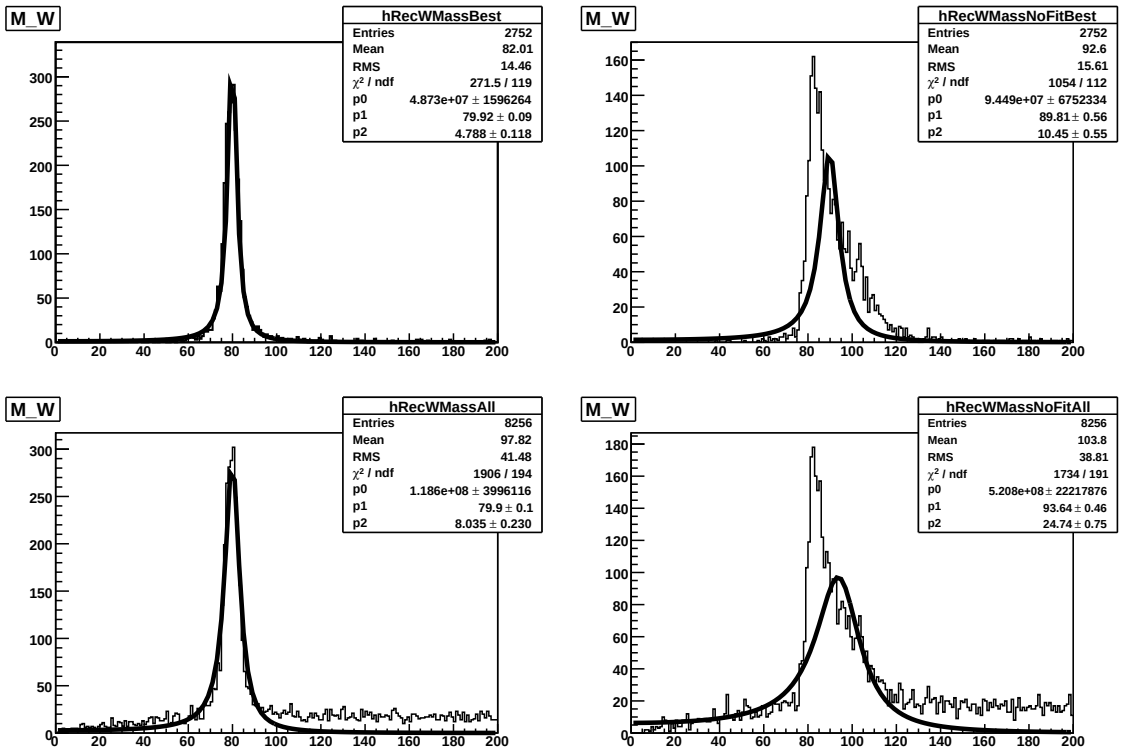
The SingleVertex topology

```
#include "KinematicTopology.h"
#include <rave/KinematicTreeFactory.h>

class TopologySingleVertex :
  public KinematicTopology
{
  public:
    std::string describe() const {
      return "Reconstructs a mother particle from all given daughters.";
    };

    rave::KinematicTree build(
        const rave::KinematicTreeFactory & factory,
        const std::vector< rave::KinematicParticle > & particles,
        const int verbose = 0) const {
      return factory.useVertexFitter( particles );
    };

    bool valid() const { return true; };
};

#include "TopologyBuilder.h"
namespace {
  TopologyBuilder<TopologySingleVertex> t( "SingleVertex", "Only vertex" );
}
```

The RaveKinematics Processor 3

Reconstruction of $WW \longrightarrow 4jet$ input events

Constraint to 500 GeV CME; Best association between jets and Ws.



$30\%/\sqrt{E}$ energy resolution and angular resolution of 10 mrad both on ϕ and θ.

MarlinRave Installation

- The MarlinRave build system is CMake which should faciliate integration into the existing Marlin toolchain.
- The Rave library is fully integrated, automatically built along with MarlinRave and statically linked to it. (Presence of Autotools is required for Rave to build automatically)
- For detailed instructions please visit the MarlinRave wiki page at http://projects.hepforge.org/rave/trac/wiki/MarlinRave
- If accepted, integration into the ilcinstall setup scripts should be easliy realized

OAW Austrian Academy of Sciences

MarlinRave Installation

- The MarlinRave build system is CMake which should faciliate integration into the existing Marlin toolchain.
- The Rave library is fully integrated, automatically built along with MarlinRave and statically linked to it. (Presence of Autotools is required for Rave to build automatically)
- For detailed instructions please visit the MarlinRave wiki page at http://projects.hepforge.org/rave/trac/wiki/MarlinRave
- If accepted, integration into the ilcinstall setup scripts should be easliy realized

OAW Austrian Academy of Sciences

MarlinRave Installation

- The MarlinRave build system is CMake which should faciliate integration into the existing Marlin toolchain.
- The Rave library is fully integrated, automatically built along with MarlinRave and statically linked to it. (Presence of Autotools is required for Rave to build automatically)
- For detailed instructions please visit the MarlinRave wiki page at http://projects.hepforge.org/rave/trac/wiki/MarlinRave
- If accepted, integration into the ilcinstall setup scripts should be easliy realized

OAW Austrian Academy of Sciences

MarlinRave Installation

- The MarlinRave build system is CMake which should faciliate integration into the existing Marlin toolchain.
- The Rave library is fully integrated, automatically built along with MarlinRave and statically linked to it. (Presence of Autotools is required for Rave to build automatically)
- For detailed instructions please visit the MarlinRave wiki page at http://projects.hepforge.org/rave/trac/wiki/MarlinRave
- If accepted, integration into the ilcinstall setup scripts should be easliy realized

OAW Austrian Academy of Sciences

Summary

- Rave provides full CMSSW vertex finding and fitting facilities and parts of its kinematic fitting algorithms
- MarlinRave provides Marlin with those capabilities through two new Marlin processors called RaveVertexing and RaveKinematics
- First physics analysis tests of those processors have been run with success
- MarlinRave features easy integration in existing Marlin setups

Summary

- Rave provides full CMSSW vertex finding and fitting facilities and parts of its kinematic fitting algorithms
- MarlinRave provides Marlin with those capabilities through two new Marlin processors called RaveVertexing and RaveKinematics
- First physics analysis tests of those processors have been run with success
- MarlinRave features easy integration in existing Marlin setups

Summary

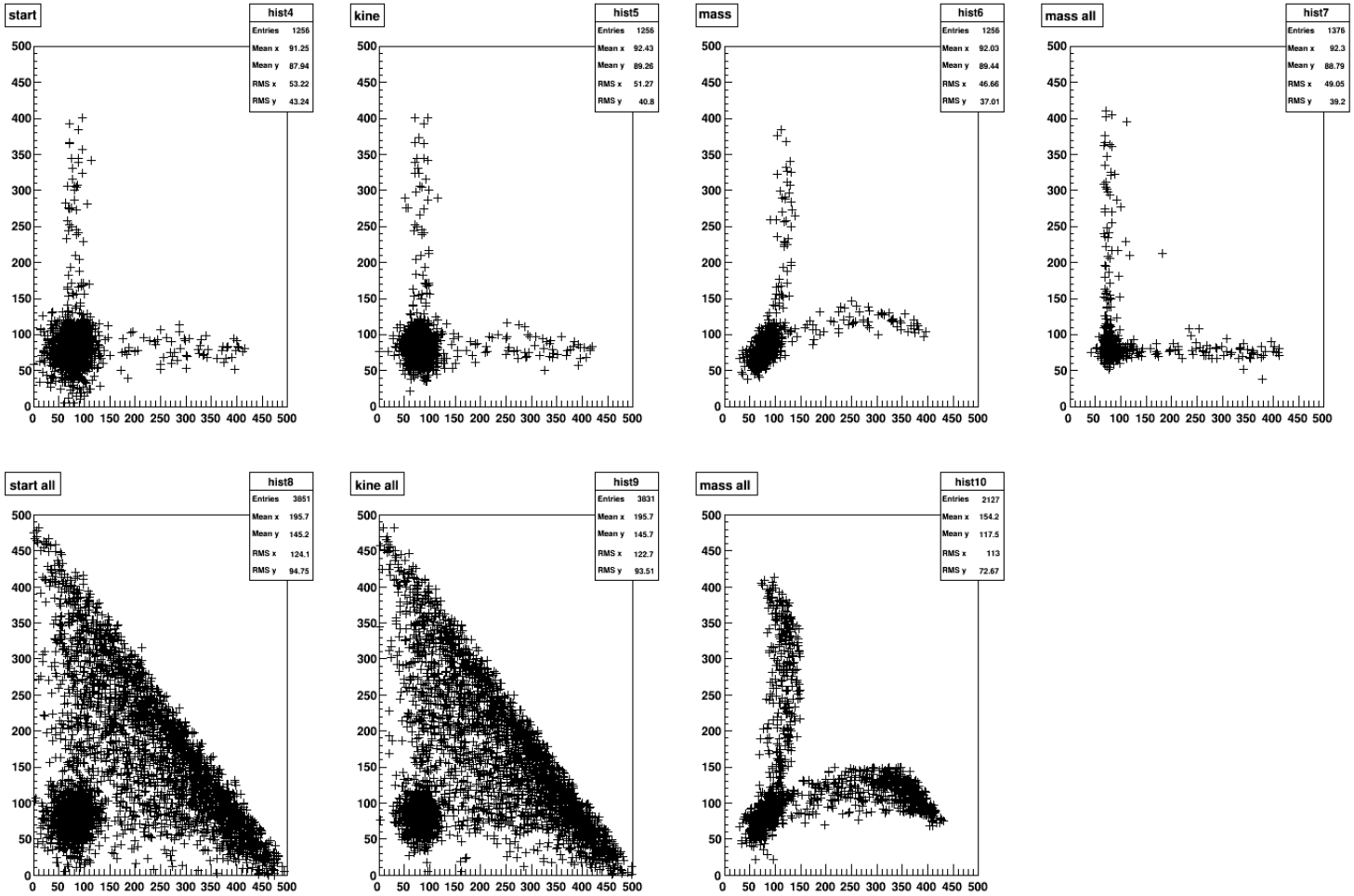
- Rave provides full CMSSW vertex finding and fitting facilities and parts of its kinematic fitting algorithms
- MarlinRave provides Marlin with those capabilities through two new Marlin processors called RaveVertexing and RaveKinematics
- First physics analysis tests of those processors have been run with success
- MarlinRave features easy integration in existing Marlin setups

Summary

- Rave provides full CMSSW vertex finding and fitting facilities and parts of its kinematic fitting algorithms
- MarlinRave provides Marlin with those capabilities through two new Marlin processors called RaveVertexing and RaveKinematics
- First physics analysis tests of those processors have been run with success
- MarlinRave features easy integration in existing Marlin setups

Why no mass constraint?

PiPlots W masses against each other



Why no mass constraint?

1D Hist of W masses