Status of the Common PWA-Framework for PANDA

Mathias Michel
Helmholtz Institut Mainz

Amplitude Analysis Techniques

19.09.2013

GSI

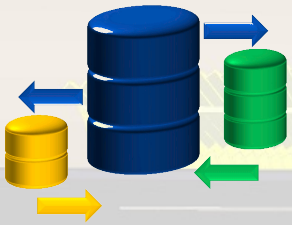
HELMHOLTZ GEMEINSCHAFT
Helmholtz-Institut Mainz

JGU JOHANNES GUTENBERG UNIVERSITÄT MAINZ

Overview

- Framework Idea
- $J/\psi \rightarrow \gamma\pi^0\pi^0$ Sandbox
- FunctionTree
- Geneva
- Outlook

PWA Framework



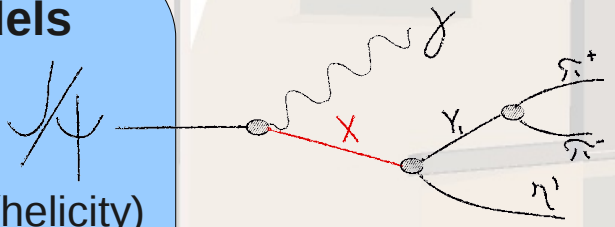
Data/MC

represents measurements

- local and global values
- multiple experiments (at once)
- caching

Physics & Models

calculates amplitudes



- various formalisms (helicity)
- various models (isobar)
- simple ways to add new modules (wrapper)

event

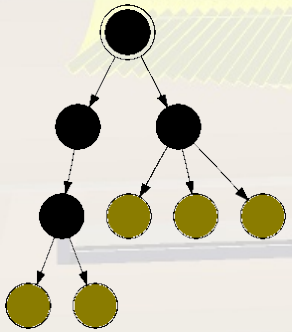
weight

amplitude

likelihood

Controls

user interface & run manager



Estimators

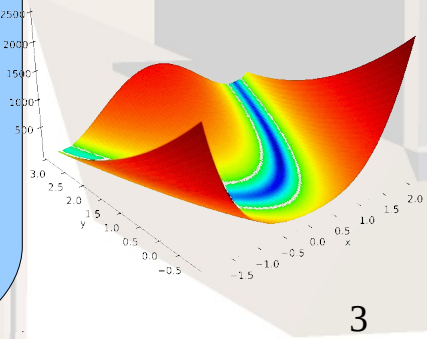
calculates discrepancy from model to data

- function tree
- combined fits and re-fits
- documentation of procedure

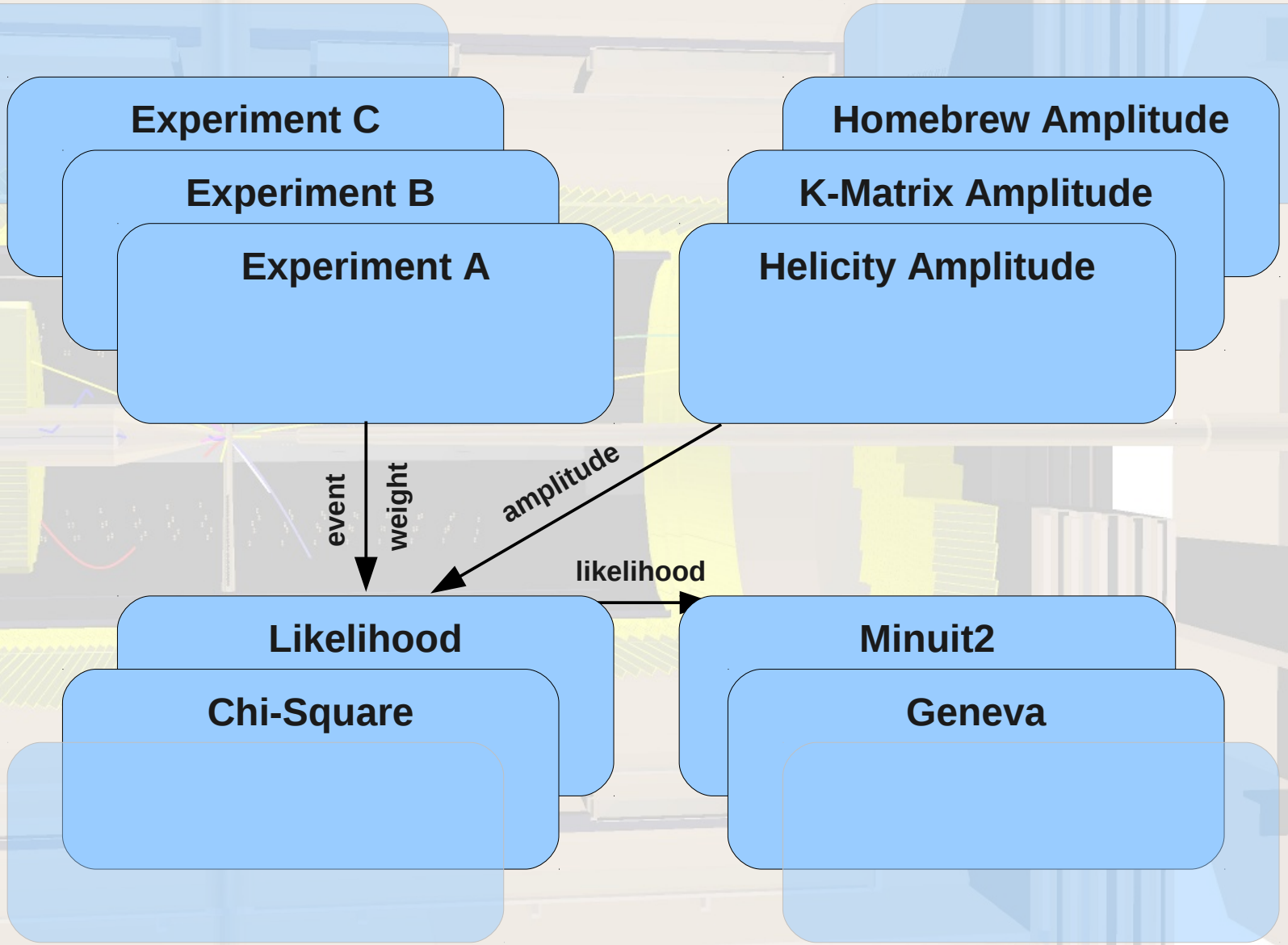
Optimizer

varies model parameter

- interface to external libraries
- various algorithms (gradient decent, genetic, swarm)
- flexible strategies



PWA Framework

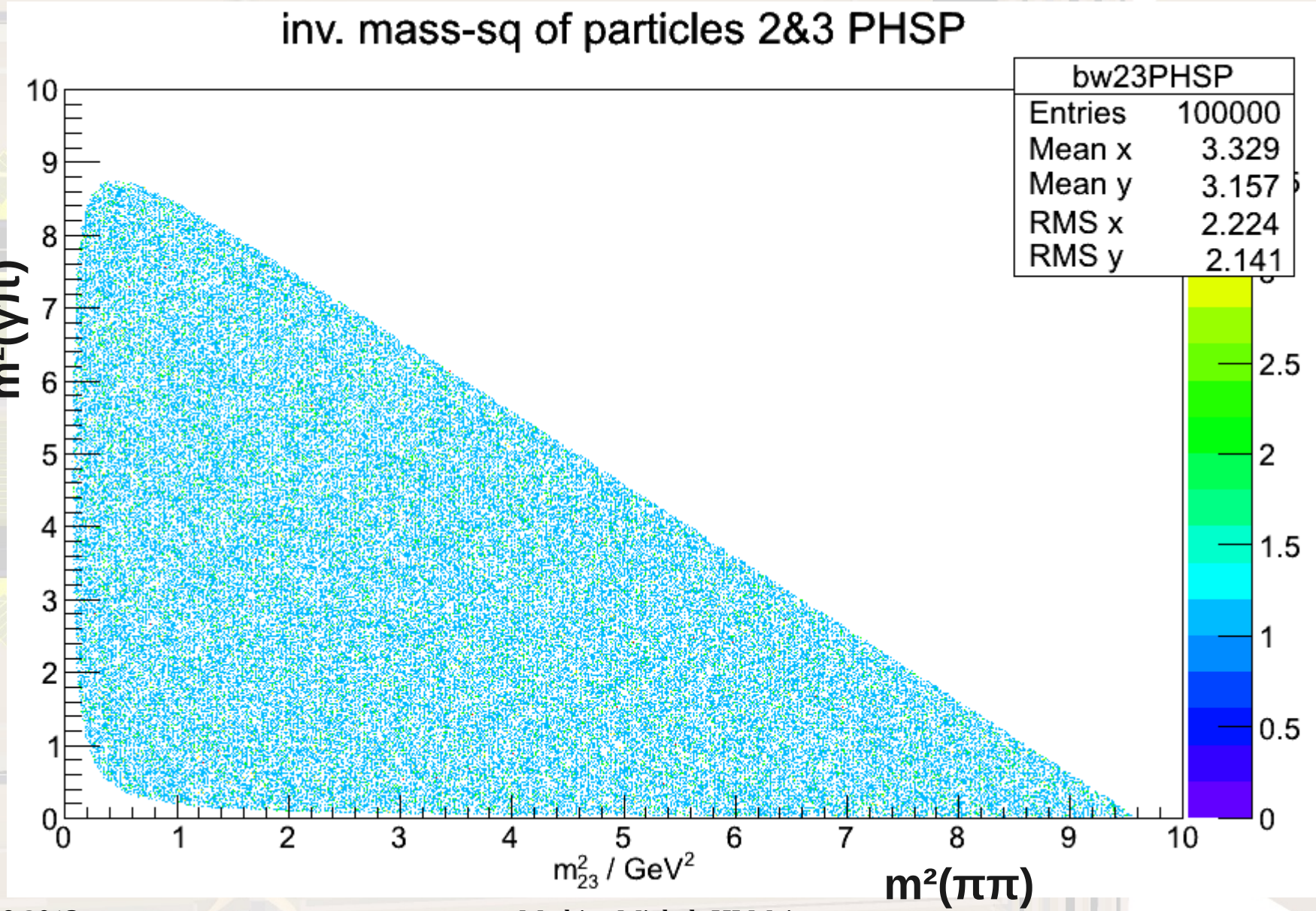


$J/\psi \rightarrow \gamma\pi^0\pi^0$ Model

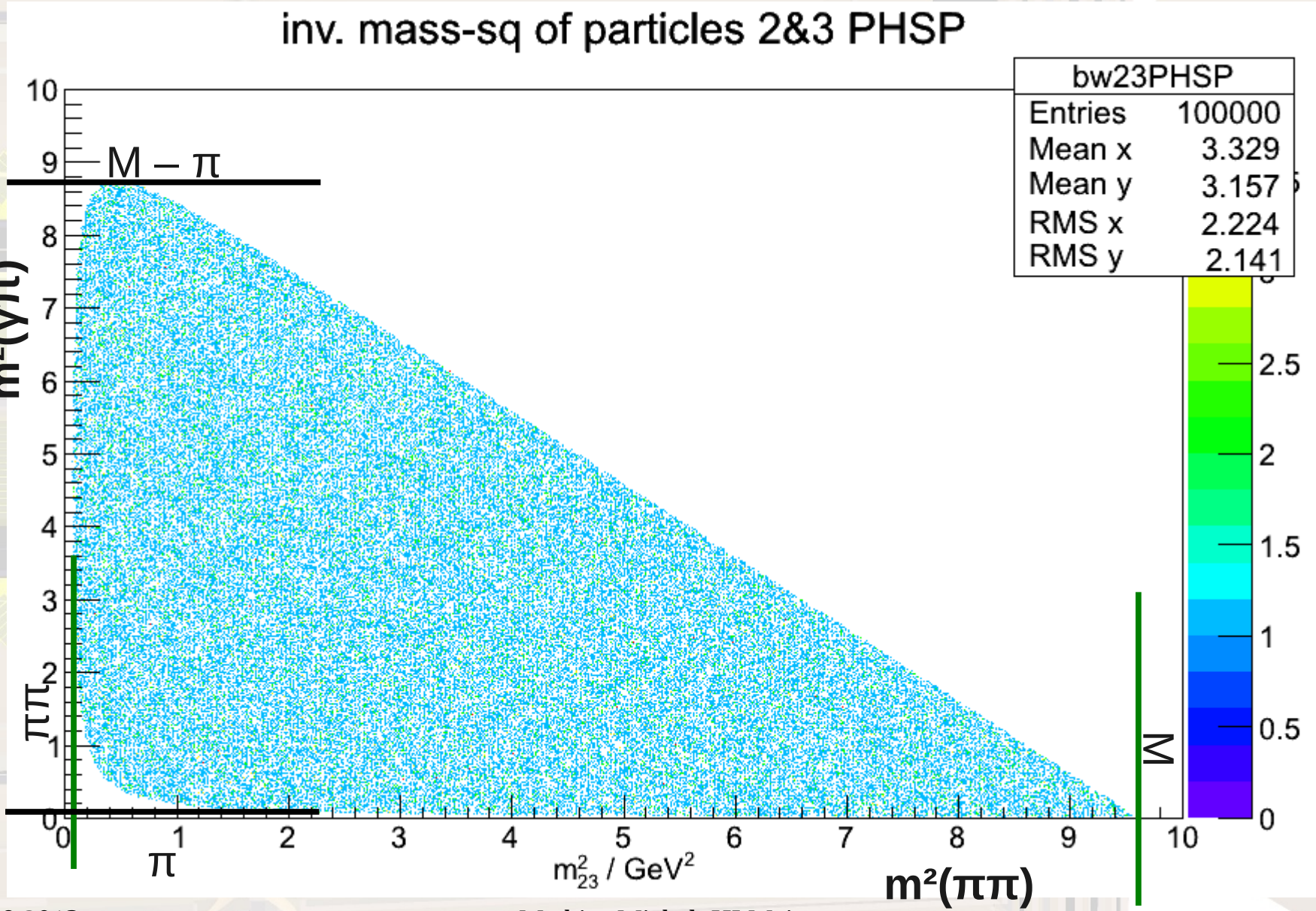
$$I = \left| \sum_n T_n r_n e^{i\varphi_n} D_n \right|^2$$

T = Breit-Wigner Function
D = D-Wigner Function
r = Strength of Resonance
φ = Phase of Resonance

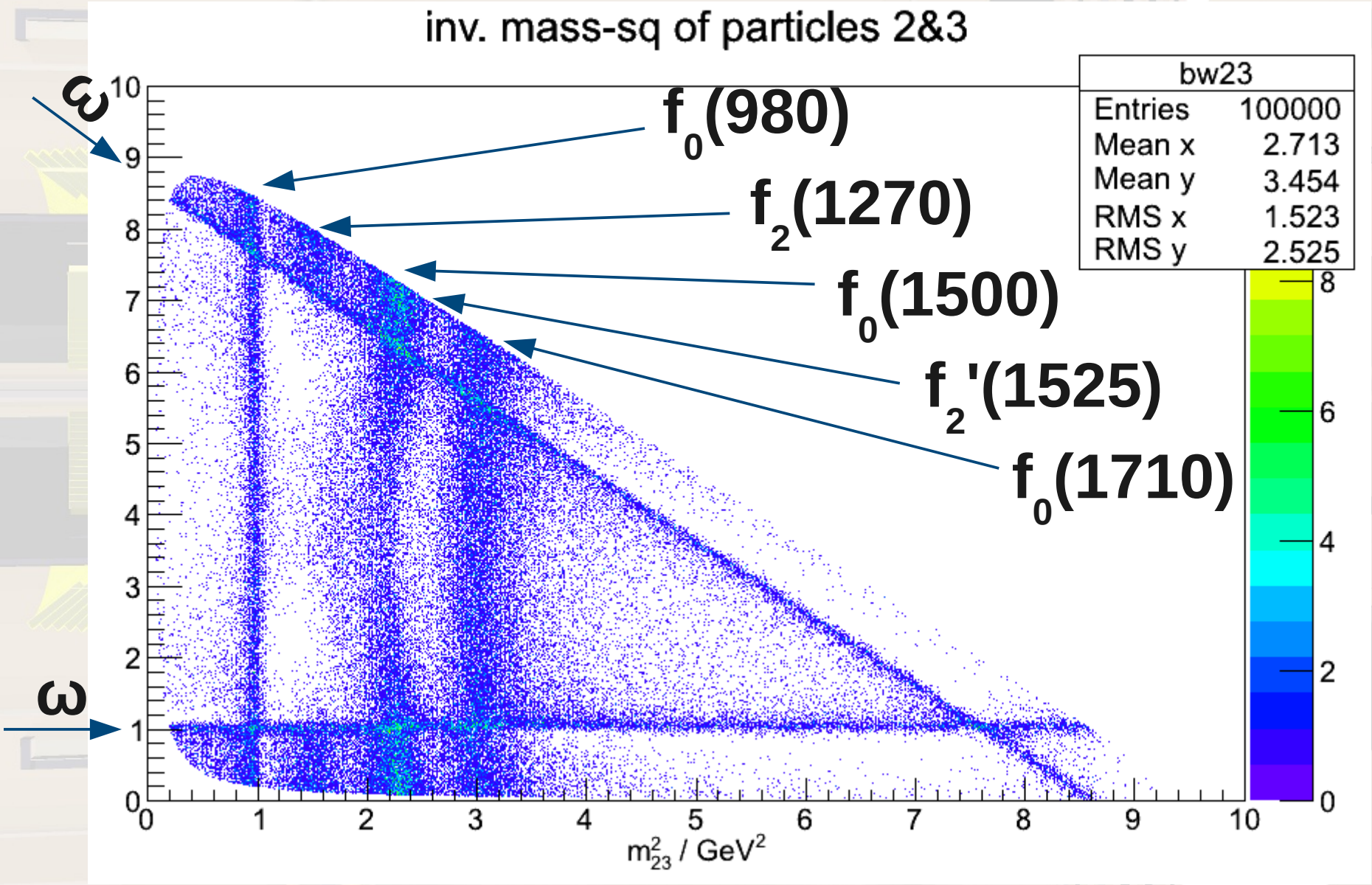
$J/\psi \rightarrow \gamma\pi^0\pi^0$ Phasespace



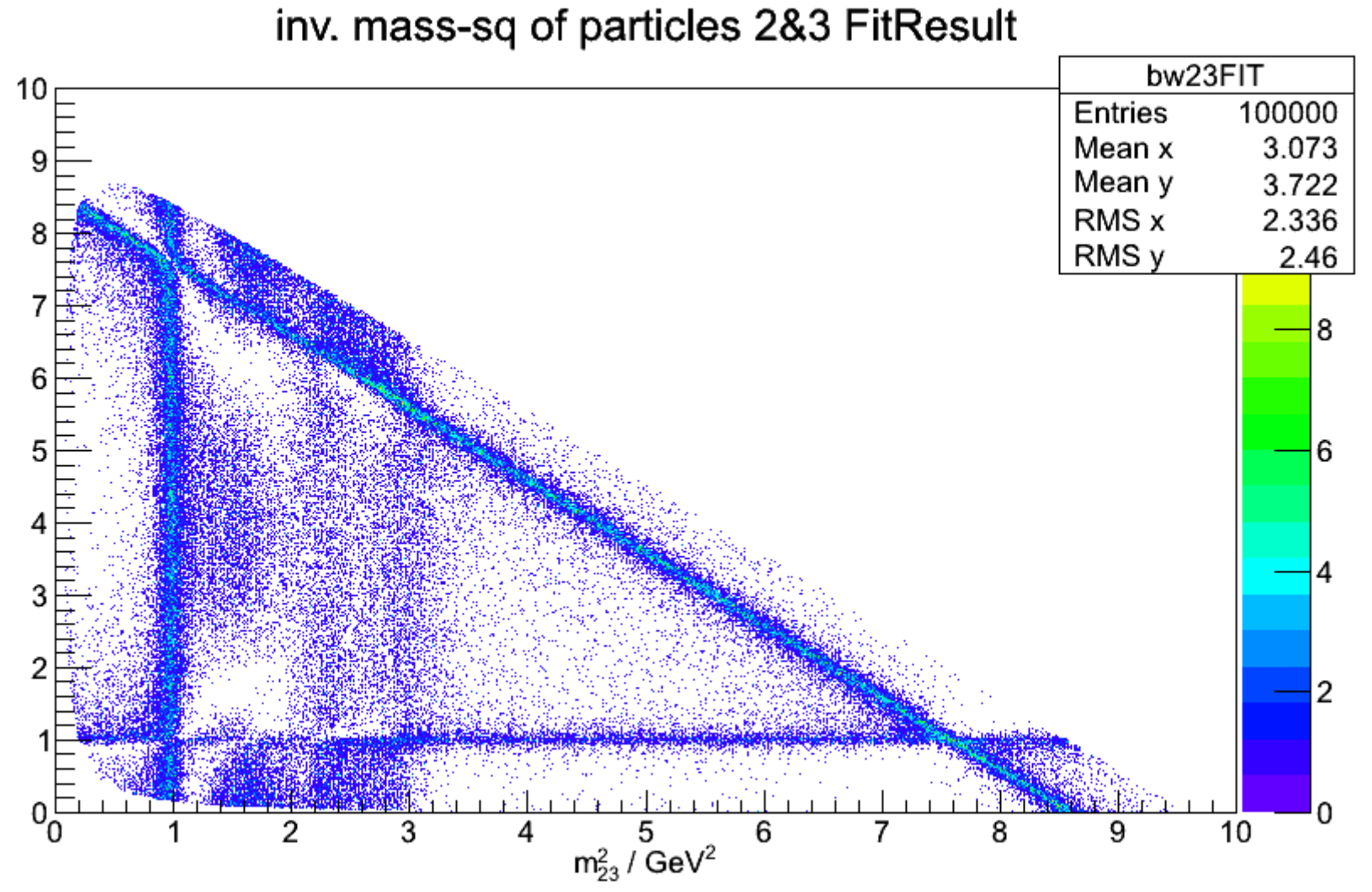
$J/\psi \rightarrow \gamma\pi^0\pi^0$ Phasespace



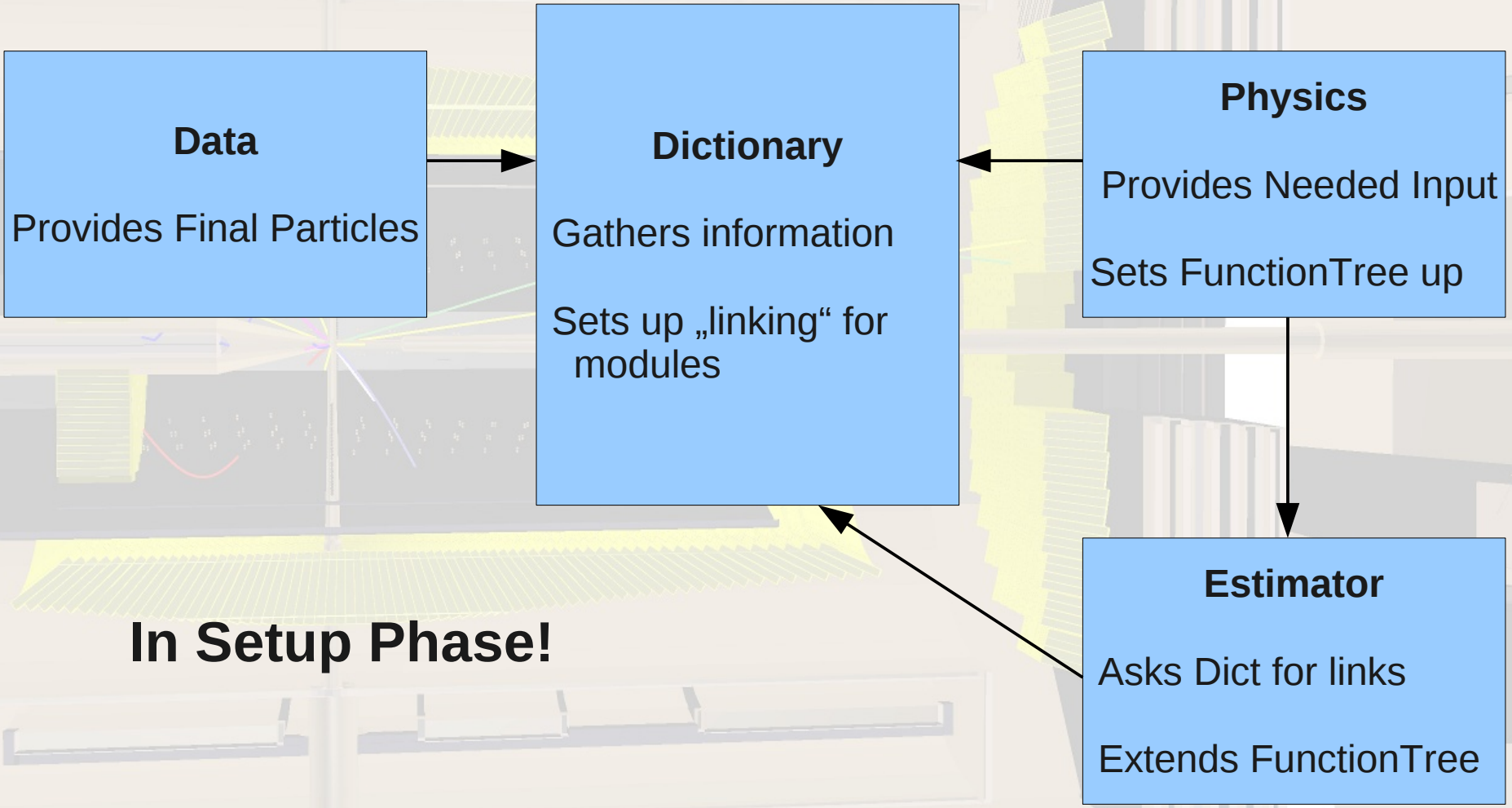
$J/\psi \rightarrow \gamma\pi^0\pi^0$ Generated



$J/\psi \rightarrow \gamma\pi^0\pi^0$ First Fit



Dictionary & FunctionTree

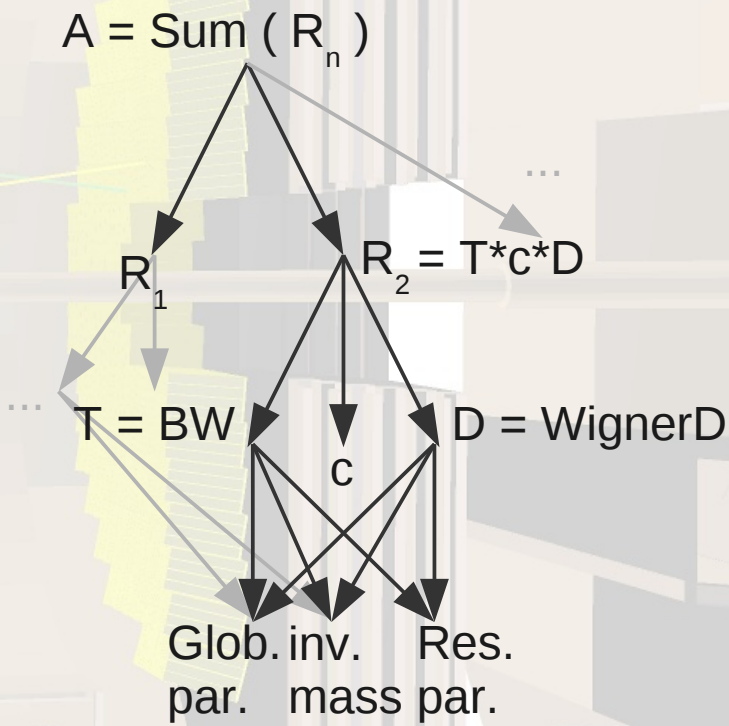


In Setup Phase!

FunctionTree of Sandbox

$$I = \left| \sum_n T_n r_n e^{i\varphi_n} D_n \right|^2$$

T = Breit-Wigner Function
D = D-Wigner Function
r = Strength of Resonance
φ = Phase of Resonance



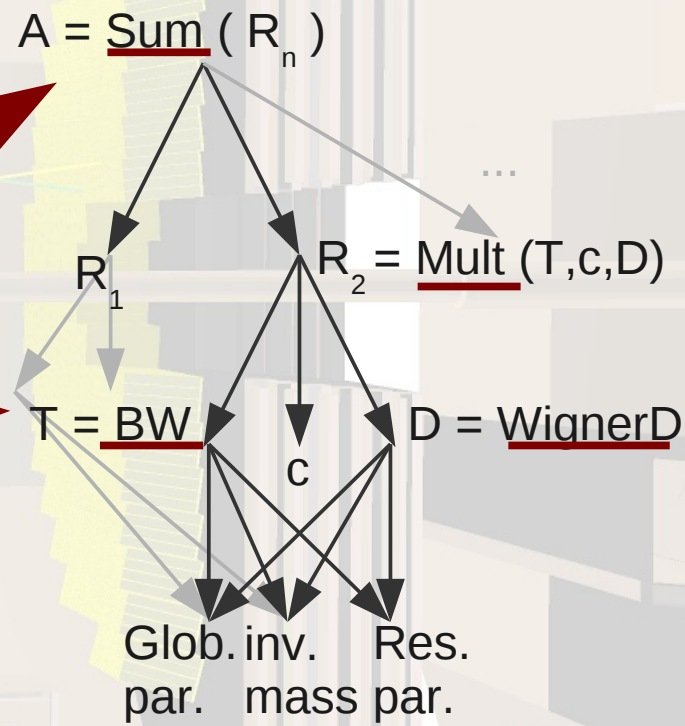
FunctionTree: Strategies

Node

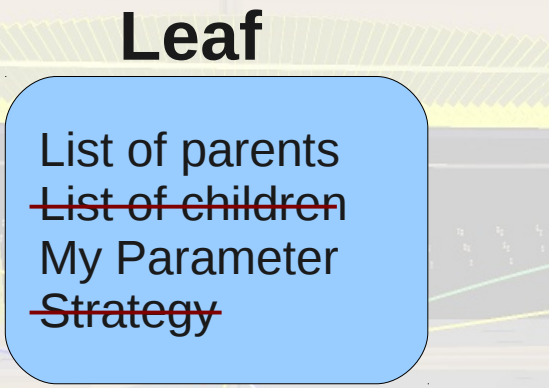
List of parents
List of children
My Parameter
Strategy

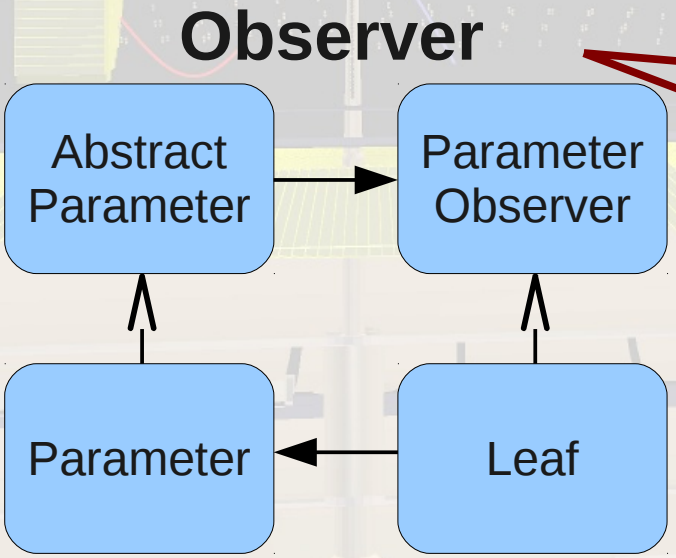
Strategy

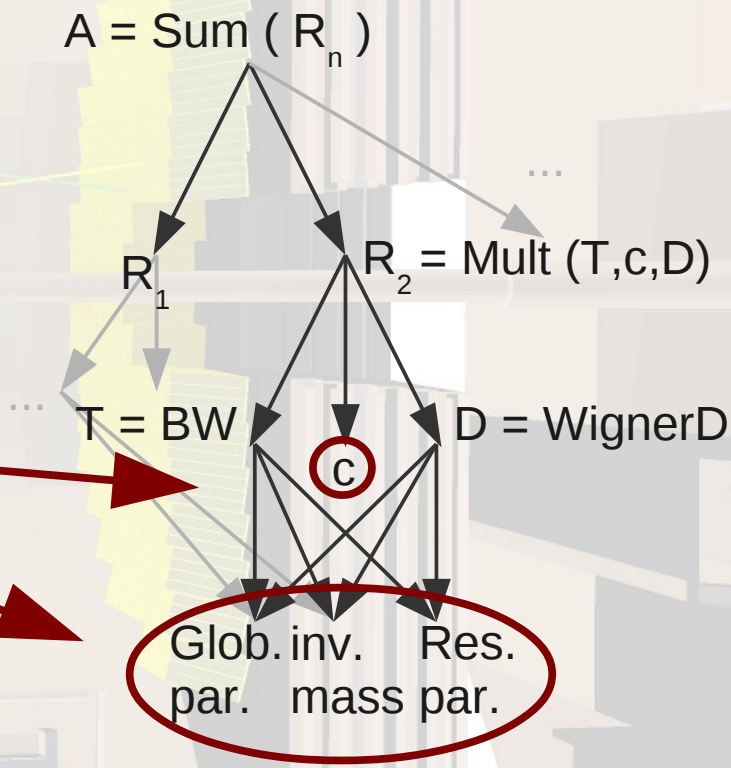
execute function
input: list of para-
meter (childr.)
output: parameter



FunctionTree: Parameters







FunctionTree: Usage

```
//------------Setup Tree----------------------
myTree = std::shared_ptr<FunctionTree>(new FunctionTree());

//----Strategies needed
rbwStrat = std::shared_ptr<BreitWignerStrategy>(new BreitWignerStrategy());
angdStrat = std::shared_ptr<WignerDStrategy>(new WignerDStrategy());
multStrat = std::shared_ptr<MultAll>(new MultAll());
addStrat = std::shared_ptr<AddAll>(new AddAll());

//----Add Nodes
myTree->createHead("Amplitude", addStrat); //A=Sum{Resos}

//----Parameters needed
//unsigned int numReso = ini.getResonances().size();
for(std::vector<Resonance>::iterator reso=ini.getResonances().begin(); reso!=ini.getResonances().end(); reso++){
  Resonance tmp = (*reso);
  //setup RooVars
  mr.push_back( std::shared_ptr<DoubleParameter> (new DoubleParameter(("m_{"+tmp.m_name+"}"), tmp.m_mass, tmp.m_mass_min,
tmp.m_mass_max) ) );
                ⋮


  //----Add Nodes
    unsigned int last = mr.size()-1;
  myTree->createNode("Reso_"+tmp.m_name, multStrat, "Amplitude"); //Reso=BW*c*AD
  myTree->createNode("RelBW_"+tmp.m_name, rbwStrat, "Reso_"+tmp.m_name); //BW
  myTree->createLeaf("Intens_"+tmp.m_name, rr[last], "Reso_"+tmp.m_name); //c
  myTree->createNode("AngD_"+tmp.m_name, angdStrat, "Reso_"+tmp.m_name); //AD
  //BW Par
  myTree->createLeaf("m0_"+tmp.m_name, mr[last], "RelBW_"+tmp.m_name); //m0
                ⋮
```

Geneva
(Grid Enabled Evolutionary Algorithms)

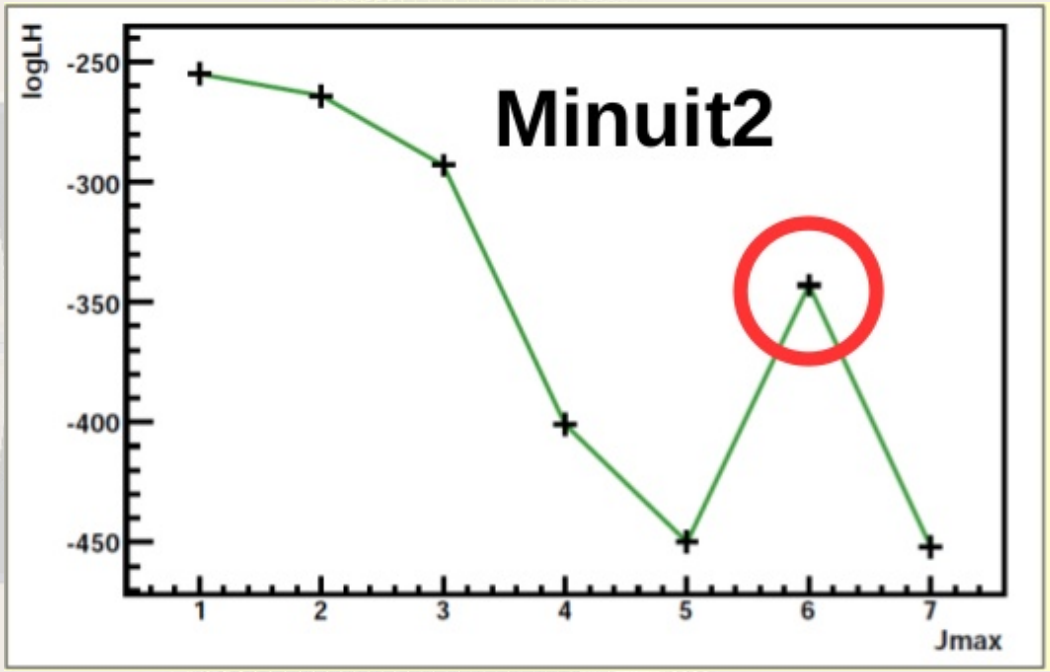
- Evolutionary & Swarm Algorithms
- Gradient Descent
- Simulated Annealing

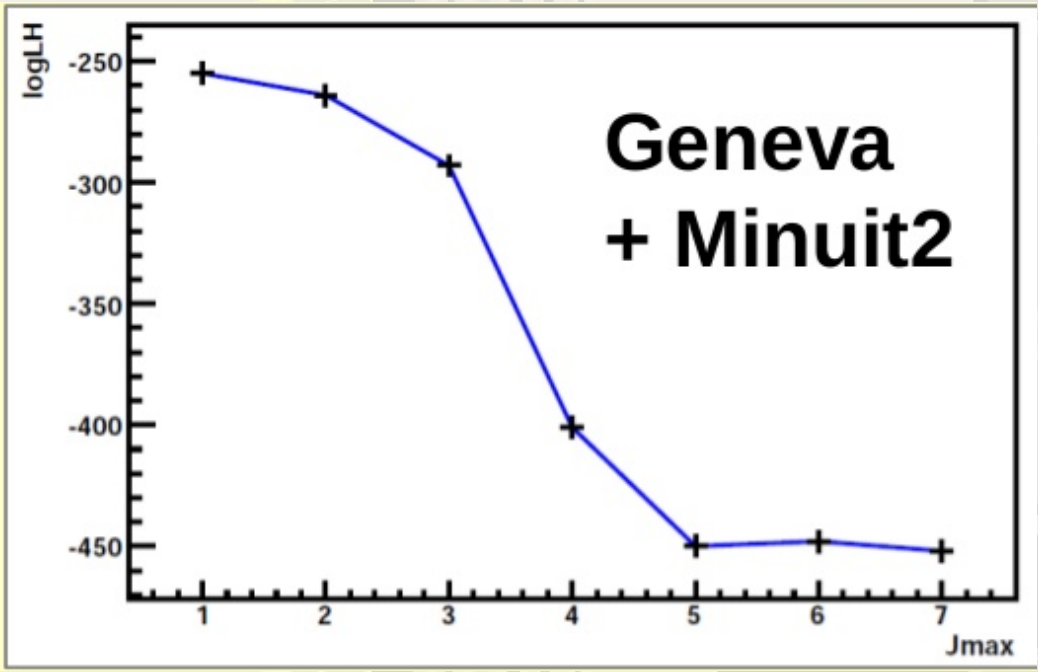
- Single-Thread, Multi-Thread & Networked Mode

- Simple UI, still everything accessable

  (e.g. algorithms, parameter modification, ...)

Geneva
(Grid Enabled Evolutionary Algorithms)





Summary

Modules:

- Ascii & Root Data Reader
- 2 Particle Breit-Wigner Amplitude
- 3 Particle Breit-Wigner Sum Amplitude
- X^2 & logLH Estimator
- Minuit2 & Geneva Optimizer Interfaces

First Test Enviroments:

- Two Particle Breit-Wigner Full Fit
- Three Particle Breit-Wigner Sum Full Fit

Dictionary and FunctionTree:

- FunctionTree ready, needs testing
- Dictionary work in progress

ToDo:

- Documentation of fit
- Control-Module, configuration
- License
- ...

Thank you!

<name>f0_980</name>
<mass>0.99</mass>
<width>0.05</width>
<strength>1.</strength>
<phase>0.</phase>
<spin>0</spin>
<m>0</m>
<n>0</n>
<daugtherA>2</daugtherA>
<daugtherB>3</daugtherB>

<name>f0_1500</name>
<mass>1.505</mass>
<width>0.109</width>
<strength>1.</strength>
<phase>0.</phase>
<spin>0</spin>
<m>0</m>
<n>0</n>
<daugtherA>2</daugtherA>
<daugtherB>3</daugtherB>

<name>f0_1710</name>
<mass>1.72</mass>
<width>0.135</width>
<strength>1.</strength>
<phase>0.</phase>
<spin>0</spin>
<m>0</m>
<n>0</n>
<daugtherA>2</daugtherA>
<daugtherB>3</daugtherB>

<name>f2_1270</name>
<mass>1.274</mass>
<width>0.185</width>
<strength>1.</strength>
<phase>0.</phase>
<spin>2</spin>
<m>0</m>
<n>0</n>
<daugtherA>2</daugtherA>
<daugtherB>3</daugtherB>

<name>f2_1525</name>
<mass>1.525</mass>
<width>0.073</width>
<strength>1.</strength>
<phase>0.</phase>
<spin>2</spin>
<m>0</m>
<n>0</n>
<daugtherA>2</daugtherA>
<daugtherB>3</daugtherB>

<name>omega</name>
<mass>1.</mass>
<width>0.05</width>
<strength>1.</strength>
<phase>0.</phase>
<spin>0</spin>
<m>0</m>
<n>0</n>
<daugtherA>1</daugtherA>
<daugtherB>2</daugtherB>

<name>omega</name>
<mass>1.</mass>
<width>0.05</width>
<strength>1.</strength>
<phase>0.</phase>
<spin>0</spin>
<m>0</m>
<n>0</n>
<daugtherA>1</daugtherA>
<daugtherB>3</daugtherB>

Dalitz-Fit

Load Modules
completed setup
LH mit optimalen intensitäten: 2.42839
LH mit folgenden intensitäten: 6288.42
Parameter 0 = 300
Parameter 1 = 150
Parameter 2 = 100
Parameter 3 = 75
Parameter 4 = 60
Parameter 5 = 50
Parameter 6 = 42.8571
Start Fit
start migrad
current minimized value: 6288.42
current minimized value: 6297.51
current minimized value: 6279.35
current minimized value: 6292.26
current minimized value: 6284.59
current minimized value: 6303.8
.
.
.

.
.
.
current minimized value: 1.63311
current minimized value: 1.63311
current minimized value: 1.63311
current minimized value: 1.63311
current minimized value: 1.63311
current minimized value: 1.63311
current minimized value: 1.63311
current minimized value: 1.63311
current minimized value: 1.63311
Final LH = 1.63311
Optimierte intensitäten: 1.63311
Parameter 0 = 3.0838
Parameter 1 = 0.986076
Parameter 2 = -0.839206
Parameter 3 = 2.28031
Parameter 4 = -0.279455
Parameter 5 = -1.78778
Parameter 6 = -3.77931

Geneva

- New Interface (Go2): “Housekeeping” of parallelization methods
- Direct access of algorithms now via factories
- Individuals (problem description) mainly the same
- Modes: Single-Threaded, Multi-Threaded (shared memory), Networked (multi process)

```cpp
int main(int argc, char **argv){
  bool iamserver=false;
  if( argc>1 ){
    iamserver=true;
    std::cout << "I am the Geneva Server:" << std::endl;
  }else{
    std::cout << "I am a Geneva Client:" << std::endl;
  }
  double p0=-10., p1=10., p2=1., p3=-0.01, sigma_smear=3;
  // Generate data distribution
  std::shared_ptr<ControlParameter> myFit = PolyFit::createInstance(p0, p1, p2, p3, sigma_smear);

  //--------------------------Minimizer IF ----------------------------------------------------------
  std::shared_ptr<GenevaIF> myMinimizer(std::shared_ptr<GenevaIF> (new GenevaIF(myFit)));
  if(iamserver)
    myMinimizer->setServerMode();
  else
    myMinimizer->setClientMode();

  ParameterList par;
  par.AddParameter(DoubleParameter("p0",-50,-100,-5,50));
  par.AddParameter(DoubleParameter("p1",50,0,100,50));
  par.AddParameter(DoubleParameter("p2",10,-20,20,10));
  par.AddParameter(DoubleParameter("p3",-0.1,-0.2,0,0.05));

  if(iamserver){
    std::cout << "Starting Parameters:" << std::endl;
    for(unsigned int i=0; i<par.GetNDouble(); i++)
      std::cout << "Parameter "<< i << ":\t" << par.GetParameterValue(i) << std::endl;
    std::cout << std::endl << std::endl << std::endl;
  }

  double genResult = myMinimizer->exec(par);

  if(iamserver){
    std::cout << "Geneva final par :\t" << genResult << std::endl;
    for(unsigned int i=0; i<par.GetNDouble(); i++)
      std::cout << "final par "<< i << ":\t" << par.GetParameterValue(i) << std::endl;
  }
  std::cout << "Done ..." << std::endl << std::endl;
```

```
const double GenevaIF::exec(ParameterList& par) {
        Go2::init();
        //Go2 go(argc, argv, configFile);
        Go2 go( clientMode, serMode, ip, port,
            (configFileDir+"Go2.json"), parallelizationMode, GO2_DEF_DEFAULTVERBOSE);

        //--------------------------------------------------------------------
        // Initialize a client, if requested

        if(go.clientMode()) {
          std::cout << "Geneva Client waiting for action!" << std::endl;
          return go.clientRun();
        }

        //--------------------------------------------------------------------
        // Add individuals and algorithms and perform the actual optimization cycle

        //Provide Parameter in Geneva-Style
        unsigned int NPar = par.GetNDouble(); //just doubles up to now, TODO
        double val[NPar], min[NPar], max[NPar], err[NPar];
        for(unsigned int i=0; i<NPar; i++){
          val[i] = par.GetDoubleParameter(i).GetValue();
          min[i] = par.GetDoubleParameter(i).GetMinValue();
          max[i] = par.GetDoubleParameter(i).GetMaxValue();
          err[i] = par.GetDoubleParameter(i).GetError();
        }

        // Make an individual known to the optimizer
        boost::shared_ptr<GStartIndividual> p(new GStartIndividual(_myData, NPar, val, min, max, err));
        go.push_back(p);

        // Add an evolutionary algorithm to the Go2 class.
        GEvolutionaryAlgorithmFactory ea((configFileDir+"GEvolutionaryAlgorithm.json"), parallelizationMode
        go & ea();

        // Perform the actual optimization
        boost::shared_ptr<GStartIndividual>
                bestIndividual_ptr = go.optimize<GStartIndividual>();
```

Geneva Algorithms

- Evolutionary Algorithms
- Swarm Algorithms
- Gradient Descent
- Simulated Annealing
- Error estimation with gradient descent?

Geneva User Interface

- Configuration: json files
- Go2 (connection settings)
- Algorithm (e.g. EA: genetic modification setup)

```
//-----------------------------------------------------------------
// This configuration file was automatically created by GParserBuilder
// File creation date: 2011-Oct-08 20:06:11
//-----------------------------------------------------------------

{
    "nEvaluationThreads":
    {
        "comment": "Determines the number of threads simultaneously running",
        "comment": "evaluations in multi-threaded mode. 0 means \"automatic\"",
        "default": "0",
        "value": "2"
    },
    "firstTimeOut":
    {
        "comment": "The timeout for the retrieval of an",
        "comment": "iteration's first timeout",
        "default": "00:00:00",
        "value": "00:00:00"
    },
    "boundlessWait":
    {
        "comment": "Indicates that the broker connector should wait endlessly",
        "comment": "for further arrivals of individuals in an iteration",
        "default": "false",
```

Geneva Networked

- Two kind of binaries: master and slave

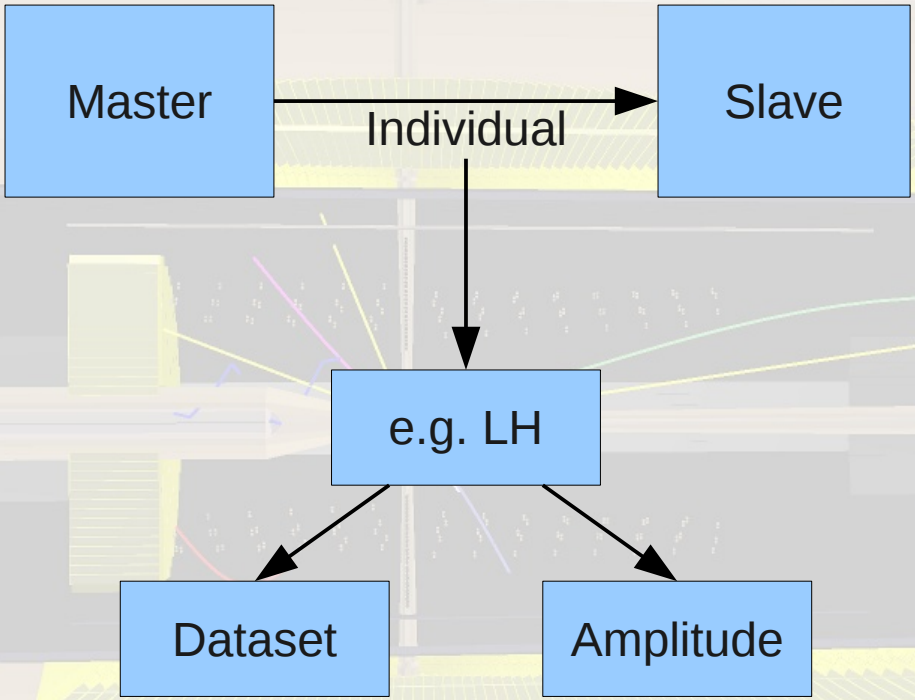
1. Master sets problem up, waits for slaves
2. Slave ask master for problem-set, is registered
3. Slave performs optimization, sends result back
4. Master gathers results, sends new problems
5. Repeat point 3-4 until master solved problem

- Slaves need ip & port of master for communication and established tcp/ip connection

Geneva Individual

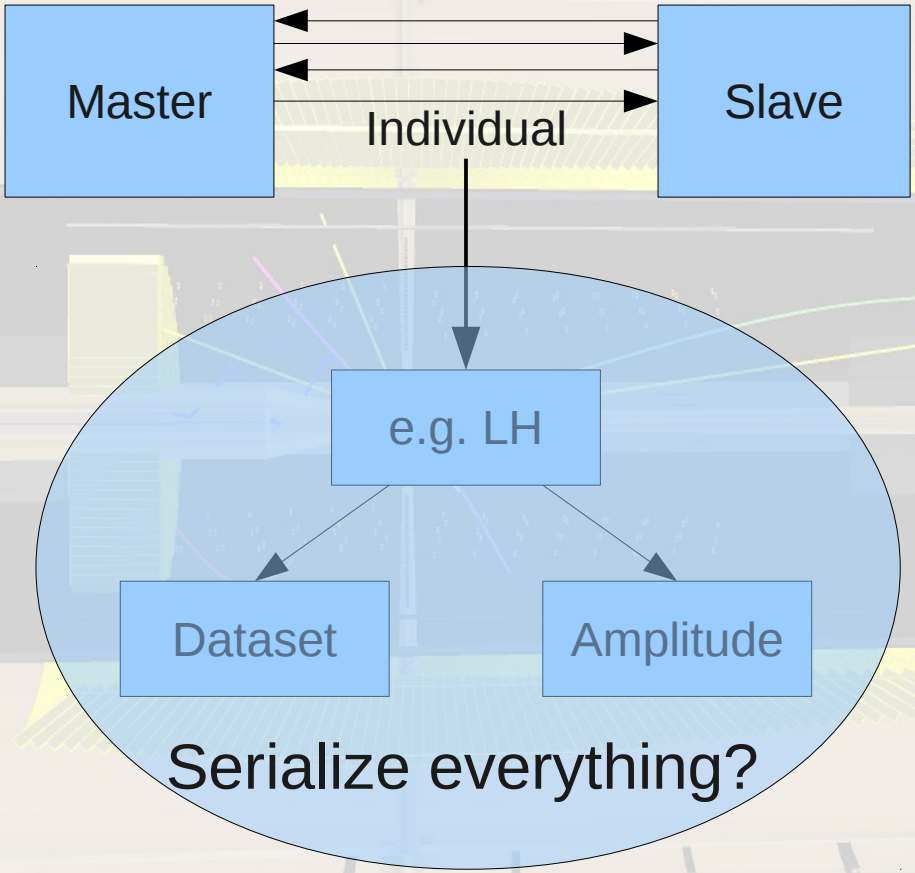
- Problem description interface
- Used in GenevaIF
- Minuit2: MinuitFCN equals Geneva's GIndividual::controlParameter()
- Calls e.g. LH-Estimator

Geneva Individual Networked Mode



- Master sets up Individual as usual
- Master sends Individual to slaves (serialized)

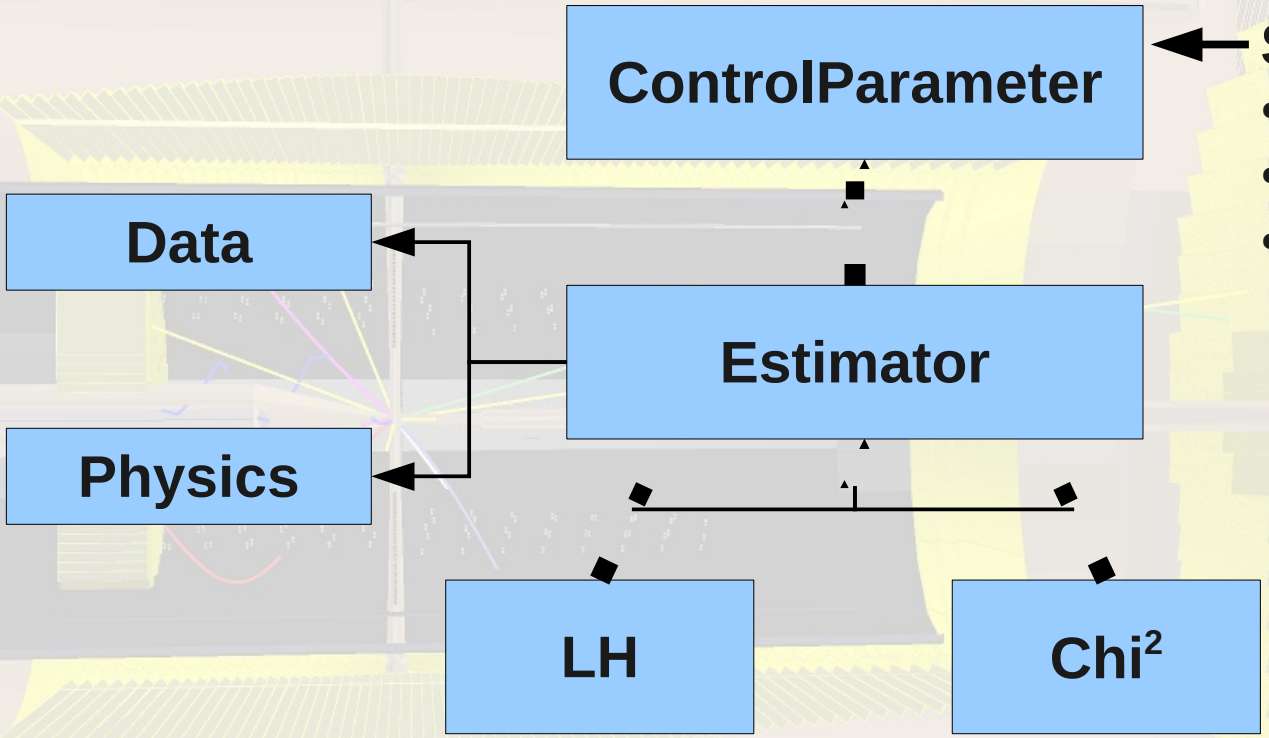
Geneva Individual Networked Mode



- Master sets up Individual **as usual**
- Master sends Individual to slaves (serialized)
- Slaves send best Individual back
- Repeat

=> Provide control Parameter (LH) constantly in each process, new Individuals get LH not from serialized information but from static object

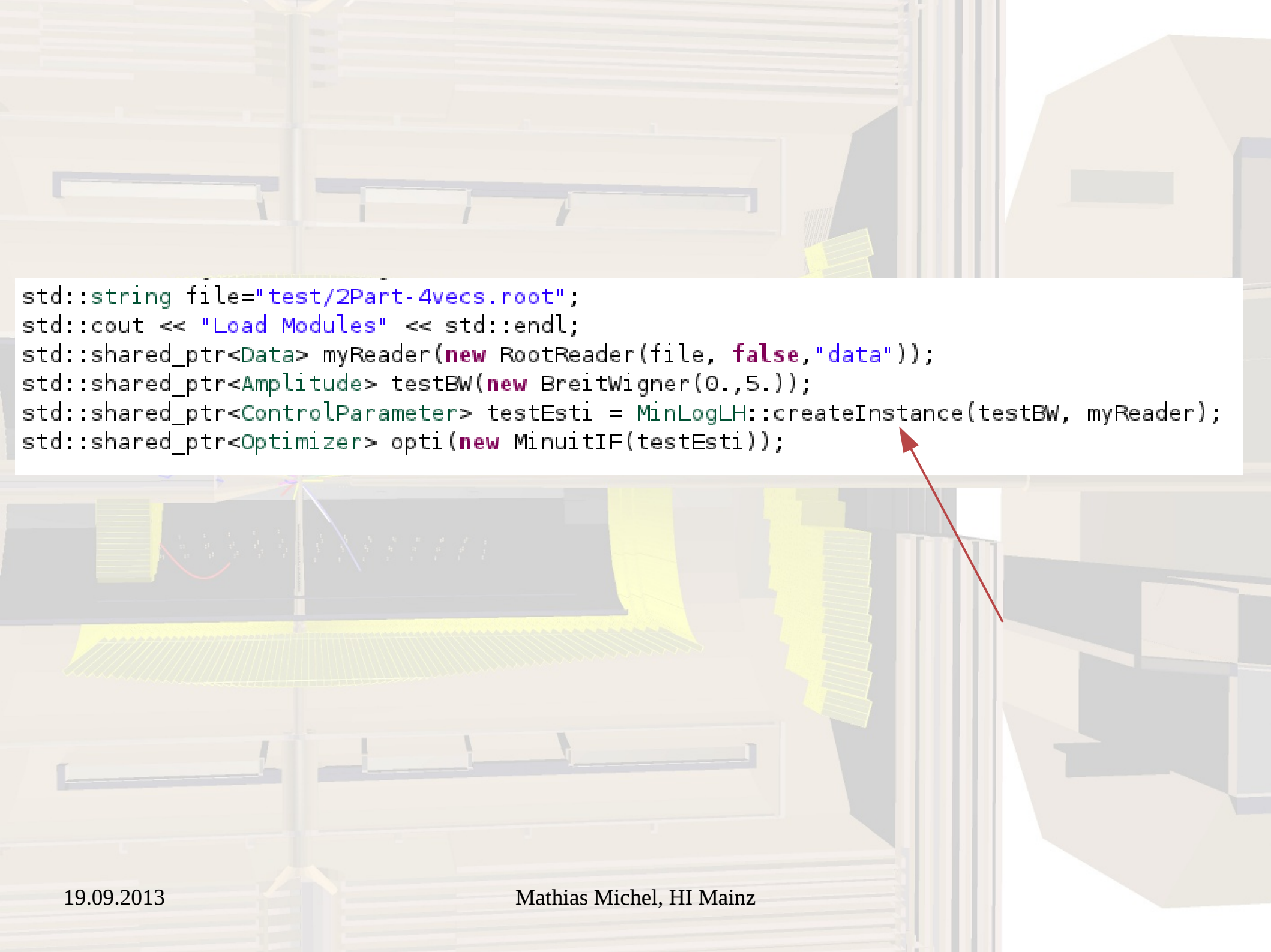
Impact on ComPWA



Singleton Pattern:
- **static pointer to itself**
- **private constructor**
- **pure virtual createObject function**

Effects:
- **ControlParameter instances (e.g. LH) statically accessible from fit**
- **Only one ControlParameter of whatever kind existing per fit (More than one needed: replace singleton with factory)**

```
std::string file="test/2Part-4vecs.root";
std::cout << "Load Modules" << std::endl;
std::shared_ptr<Data> myReader(new RootReader(file, false,"data"));
std::shared_ptr<Amplitude> testBW(new BreitWigner(0.,5.));
std::shared_ptr<ControlParameter> testEsti = MinLogLH::createInstance(testBW, myReader);
std::shared_ptr<Optimizer> opti(new MinuitIF(testEsti));
```



```
//------------SetUp some operations for Amp = a * ( b + c)-----------
std::shared_ptr<Strategy> add = std::shared_ptr<Strategy>(new AddAll());
std::shared_ptr<Strategy> mult = std::shared_ptr<Strategy>(new MultAll());

//------------SetUp some nodes for Amp = a * ( b + c)---------------
std::shared_ptr<TreeNode> Amplitude =
    std::shared_ptr<TreeNode>(new TreeNode(0, "Amplitude", mult, NULL));
std::shared_ptr<TreeNode> A =
    std::shared_ptr<TreeNode>(new TreeNode(5, "a", NULL, Amplitude));
std::shared_ptr<TreeNode> BC =
    std::shared_ptr<TreeNode>(new TreeNode(0, "bc", add, Amplitude));
Amplitude->children.push_back(A);
Amplitude->children.push_back(BC);
std::shared_ptr<TreeNode> B =
    std::shared_ptr<TreeNode>(new TreeNode(2, "b", NULL, BC));
std::shared_ptr<TreeNode> C =
    std::shared_ptr<TreeNode>(new TreeNode(3, "c", NULL, BC));
BC->children.push_back(B);
BC->children.push_back(C);

std::cout << Amplitude << std::endl;

//------------Trigger Calculation---------------
C->changeVal(3.);

std::cout << std::endl << Amplitude << std::endl;

std::cout << "Calculated 5*(2+3) = " << Amplitude->value << std::endl;

A->changeVal(1.);

std::cout << "Changed a from 5 to 1 " << std::endl;

std::cout << std::endl << Amplitude << std::endl;
```

```
Amplitude = 0 with 2 children
 -> a = 5
 -> bc = 0 with 2 children
 ->  -> b = 2
 ->  -> c = 3


Amplitude = 25 with 2 children
 -> a = 5
 -> bc = 5 with 2 children
 ->  -> b = 2
 ->  -> c = 3

Calculated 5*(2+3) = 25
Changed a from 5 to 1

Amplitude = 5 with 2 children
 -> a = 1
 -> bc = 5 with 2 children
 ->  -> b = 2
 ->  -> c = 3
```

```
struct TreeNode{
  TreeNode(double inValue, std::string inName, std::shared_ptr<Strategy> strat, std::shared_ptr<TreeNode> parent)
    :value(inValue),name(inName),myStrat(strat){
    if(parent){
        parents.push_back(parent);
        //parent->children.push_back(shared_from_this());
    }
  };

  void inline changeVal(double newVal){
    value=newVal;
    for(unsigned int i=0; i<parents.size(); i++)
      parents[i]->update();
  };

  void update(){ //darf nur von kindern aufgerufen werden!
    std::vector<double> newVals;
    for(unsigned int i=0; i<children.size(); i++){
        newVals.push_back(children[i]->value);
    }  //end children-loop
    changeVal(myStrat->execute(newVals));
  }; //end update()

  std::string to_str(std::string beginning = ""){
    std::stringstream oss;
    oss << beginning << name << " = " << value ;
    if(children.size())
      oss << " with " << children.size() << " children" << std::endl;
    else
      oss << std::endl;

    for(unsigned int i=0; i<children.size(); i++){
      //oss << " -> ";
      oss << beginning << children[i]->to_str(" -> ");
    }
    return oss.str();
  };

  friend std::ostream & operator<<(std::ostream &os, std::shared_ptr<TreeNode> p);

  std::vector<std::shared_ptr<TreeNode> > parents;
  std::vector<std::shared_ptr<TreeNode> > children;

  double value;
  std::string name;
```

```
class Strategy
{
public:
  Strategy(){
  };

  virtual double execute(const std::vector<double>& paras) = 0;
};

class AddAll : public Strategy
{
public:
  AddAll(){
  };

  virtual double execute(const std::vector<double>& paras){
    double result = 0;
    for(unsigned int i=0; i<paras.size(); i++)
      result+=paras[i];
    return result;
  };
};

class MultAll : public Strategy
{
public:
  MultAll(){
  };

  virtual double execute(const std::vector<double>& paras){
    double result = 1.;
    for(unsigned int i=0; i<paras.size(); i++)
```