

# **C++ and Generic Programming: Rapid Development of New Monte Carlo Simulations**

**Gregory Brown**

*Center for Computational Science*

*Oak Ridge National Laboratory*

and

*School of Computational Science and Information Technology*

*Florida State University*

# C++: more-convenient C

- **On-the-fly declaration**

```
double sum,ave;  
int i;  
sum = 0;  
for(i=0; i<10; i++) sum += array[i];  
ave = sum/10.;
```

```
double sum=0;  
for(int i=0; i<10; i++) sum += array[i];  
double ave = sum/10.;
```

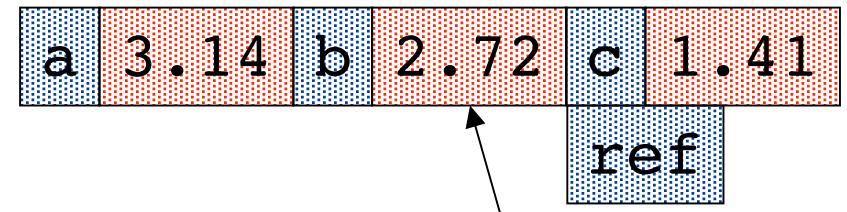
- **Boolean variables**
- **constant values**
- **default arguments**

```
Fill(float array[], int length=10);
```

# Objects, Pointers, References

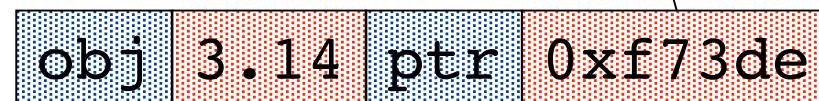
- **object** has its own memory
- **pointer** specifies some other memory
- **reference** is an alias for some other memory

```
function( a, &b, c );
```



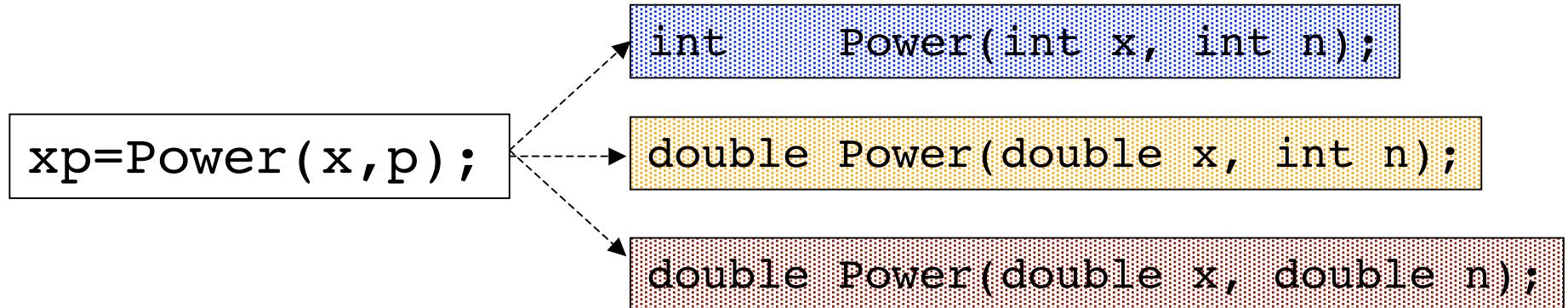
```
function(float obj, float* ptr, float& ref)
```

```
{
```



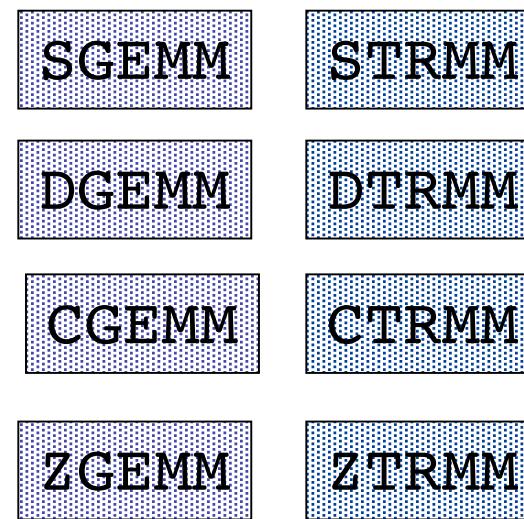
```
}
```

# Overloading: reusing names



Allows focus on semantics, not details

`Multiply(MatA,MatB)`



# Creating data types: classes

```
class Vec3
{
public:
    double x,y,z;
};
```

```
Vec3 u;
u.x = 1;
u.y = 0;
u.z = 1;
```

# Instances and Construction

```
class Vec3
{
public:
    Vec3(double xv=0, double yv=0, double zv=0);
    double x,y,z;
};
```

Each Vec3 object we create is an **instance** of Vec3.  
When it is created, the constructor is called.

```
Vec3 u(1,1,1);
Vec3 v(0,1,0);
```

# Member functions

```
class Vec3
{
public:
    Vec3(double xv=0, double yv=0, double zv=0);
    double norm();
    double data[3];
};
```

```
Vec3 u(1,1,1);
Vec3 v(0,1,0);
double ul = u.norm();
double vl = v.norm();
```

# Operators: symbols for names

```
class Vec3
{
public:
    Vec3(double xv=0, double yv=0, double zv=0);
    Vec3& operator=(const Vec3& rhs);
    double& operator[](int idim);
    double norm();
    double data[3];
};
```

```
Vec3 u(1,1,1);
Vec3 v;
v = u;
v[1]=0;
```

# **private: Hiding the Data**

```
class Vec3
{
public:
    Vec3(double xv=0, double yv=0, double zv=0);
    ~Vec3();
    Vec3& operator=(const Vec3& rhs);
    double& operator[](int idim);
private:
    double* data;
};
```

# Graphical view of objects

interface

Vec3	
Vec3( x=0 , y=0 , z=0 )	double data[ 3 ]
operator=( vec3 )	
operator[ ]( int )	

public

private

# Overloading can be tedious

Box-Müller algorithm for Gaussian random numbers

```
double BoxMuller(MyRNG& rand)
{
    double amp = log( -2.*rand() );
    double ang = 2*M_PI*rand();
    return amp*cos(ang);
}
```

```
double BoxMuller(TheirRNG& rand)
{
    double amp = log( -2.*rand() );
    double ang = 2*M_PI*rand();
    return amp*cos(ang);
}
```

The only thing different is specific type  
of the random number generator!

# Templates: types as parameters

```
template<class Random>
double BoxMuller(Random& rand)
{
    double amp = log( -2.*rand() );
    double ang = 2.*M_PI*rand();
    return amp*cos(ang);
}
```

The right type gets used when function is called just like right object gets used.

Technically, the compiler will create the appropriate function when it sees what types are used. (Can't create object files!)

# What can be templated

- **Functions**

```
template<class B, class P>
B Pow(B base, P power) {...};
```

- **Classes**

```
template<int Dim>
class Vec { ... };
```

- **Member functions**

```
class Brick {
    template<class NVector>
        specify_corner(NVector newCorner);
};
```

# Concepts and Models

```
template<class Random>
double BoxMuller(Random& rand) {...}
```

For this to work MyRNG and TheirRNG must each define member function `double operator()`  
Also need `void seed(long int iseed)`

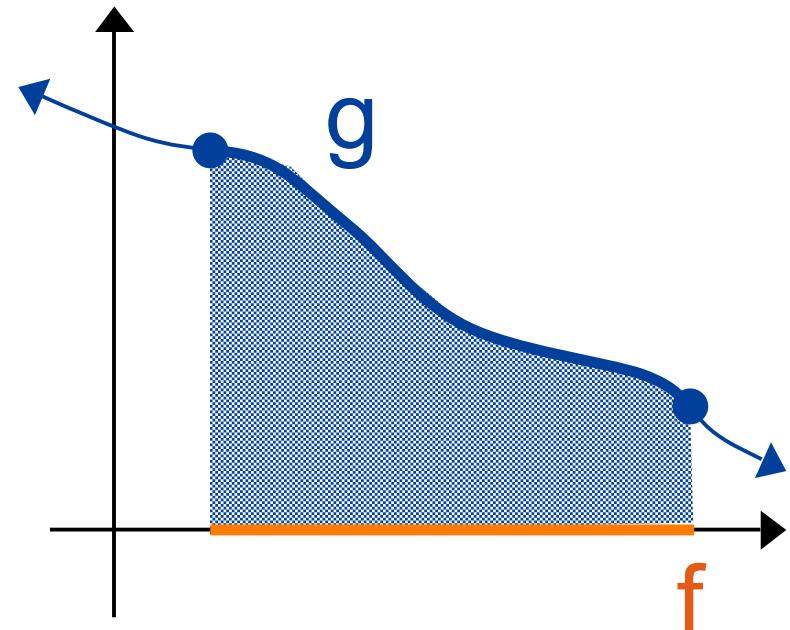
The set of requirements associated with a certain type of thing is called a **concept**.

The classes which implement those requirements are called **models**.

# Monte-Carlo Integration

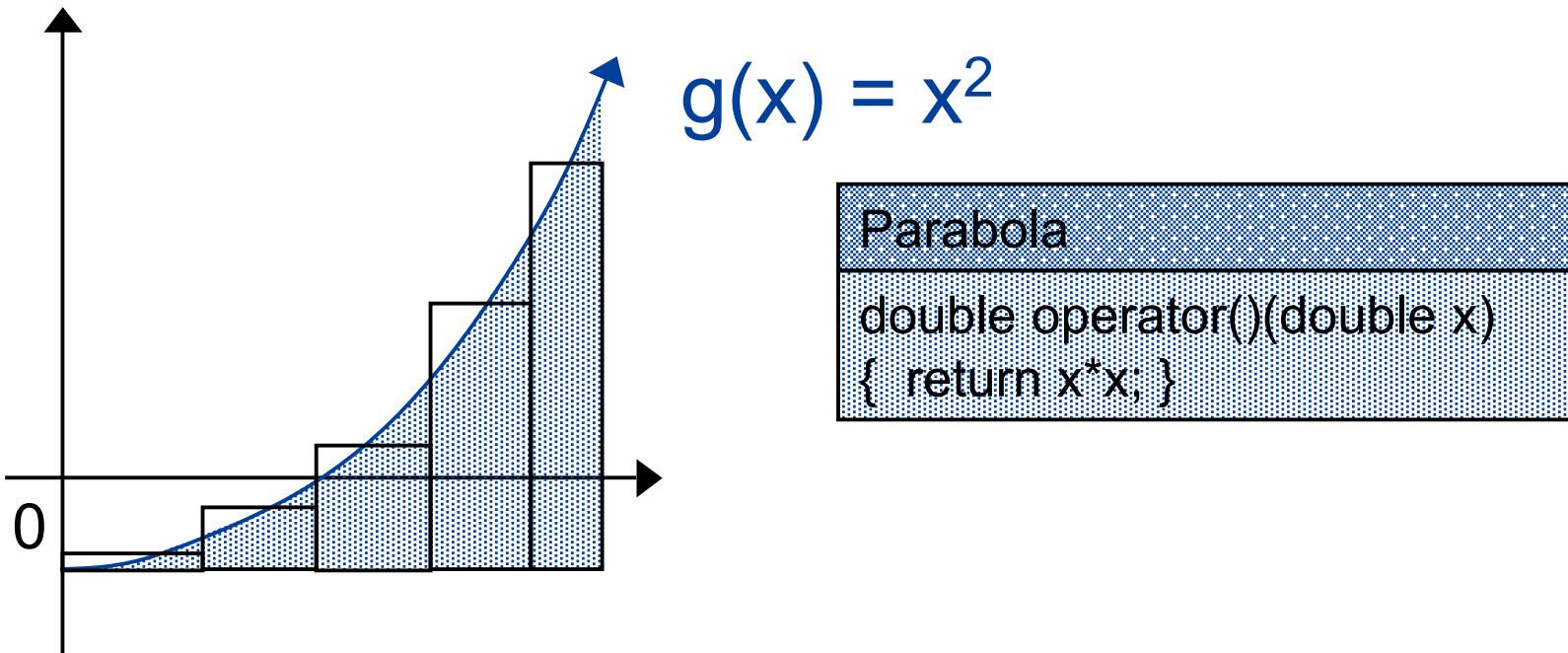
$$I = \int g(x)f(x)dx \approx \frac{1}{N} \sum_i g(x_i)$$

MCIntegration
operator( <i>g,f,rng</i> )
operator( <i>g,f,rng,minNStep</i> )
accuracy( <i>relative_tolerance</i> )
maximum_number_steps( <i>nstep</i> )
maximum_number_steps()



# Function Object

Defines member function `operator() (Number x)`



```
Parabola g;
Real nineA = gauss5(g,0,3);
Real nineB = romberg(g,0,3);
```

# Sampling Distribution

$f(\text{rng}())$  is a function object  
it sets the limits of integration  
it provides the importance sampling

```
psimag::UniformInCuboid<int DIM>
Vec<Real,DIM> operator()(RNG u)
lower_bound(Vec<Real,DIM> x0)
upper_bound(Vec<Real,DIM> x1)
```

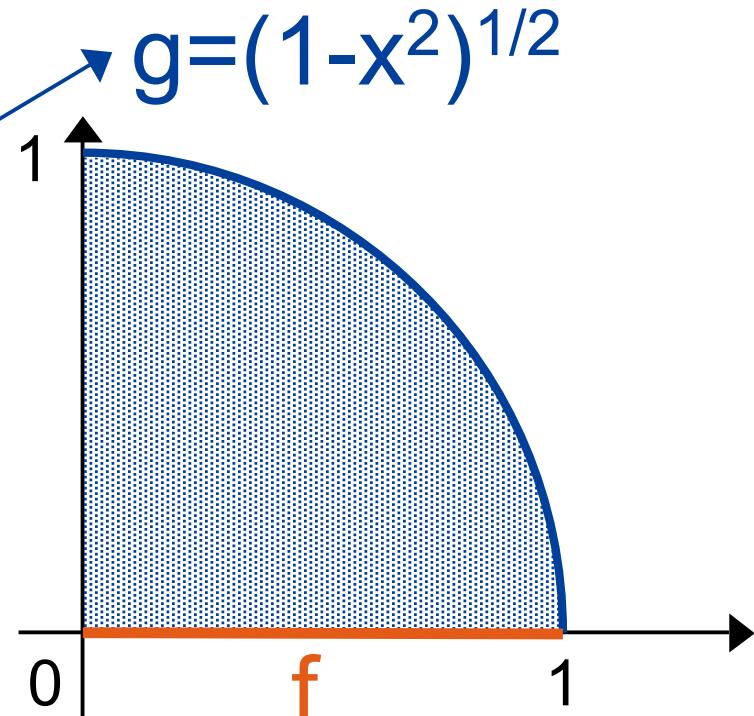
UniformInCuboid<1>

UniformInCuboid<2>

# Estimation of Pi

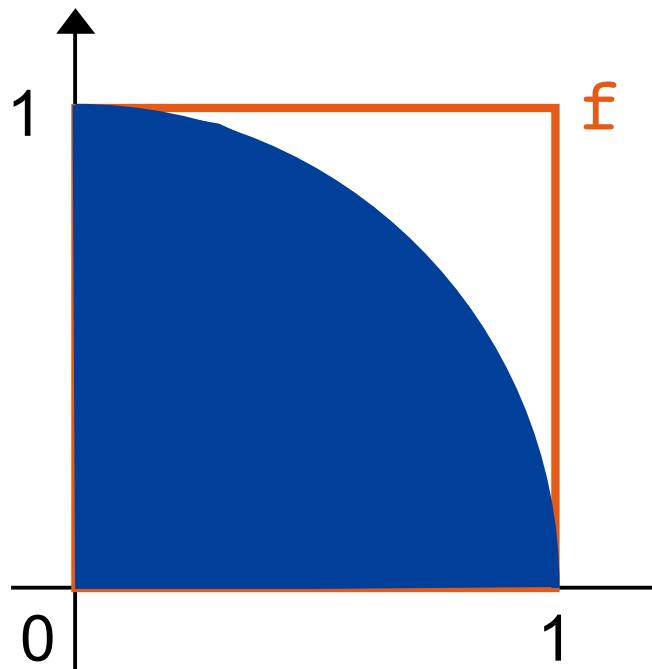
$$4 \int (1-x^2)^{1/2} = \pi$$

```
Arc  
double operator()(double x)  
{ return std::sqrt(1-x*x); }
```



```
Arc arc;  
psimag::UniformInCuboid<1> interval;  
psimag::MCIntegration(arc,interval,rng);
```

# “Hit and Miss” Estimation of Pi



$$g(x,y) = \begin{cases} 1, & x^2+y^2 \leq 1 \\ 0, & x^2+y^2 > 1 \end{cases}$$

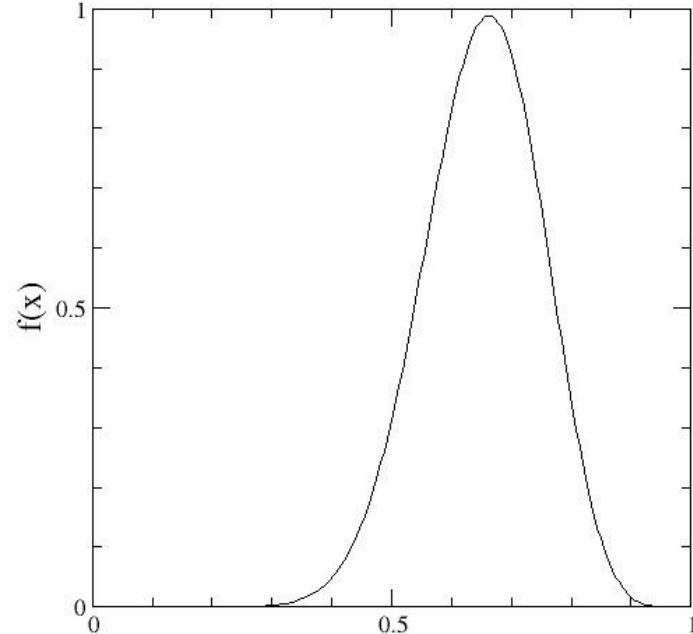
```
SphericalStepFunction  
double operator()(NVEC x) {  
    if( L2Norm(x) <= 0 )  
        return 1;  
    else  
        return 0;
```

```
SphericalStepFunction circ;  
psimag::UniformInCuboid<2> interval;  
psimag::MCIntegration(circ,interval,rng);
```

# Metropolis Sampling

$$I = \int f(x)g(x)dx$$

$$E = \frac{1}{N} \sum_{i=1}^N g(x_i)$$



Detailed Balance

$$A(X|Y)T(X|Y)f(Y) = A(Y|X)T(Y|X)f(X)$$

$$A = \min(q, 1)$$

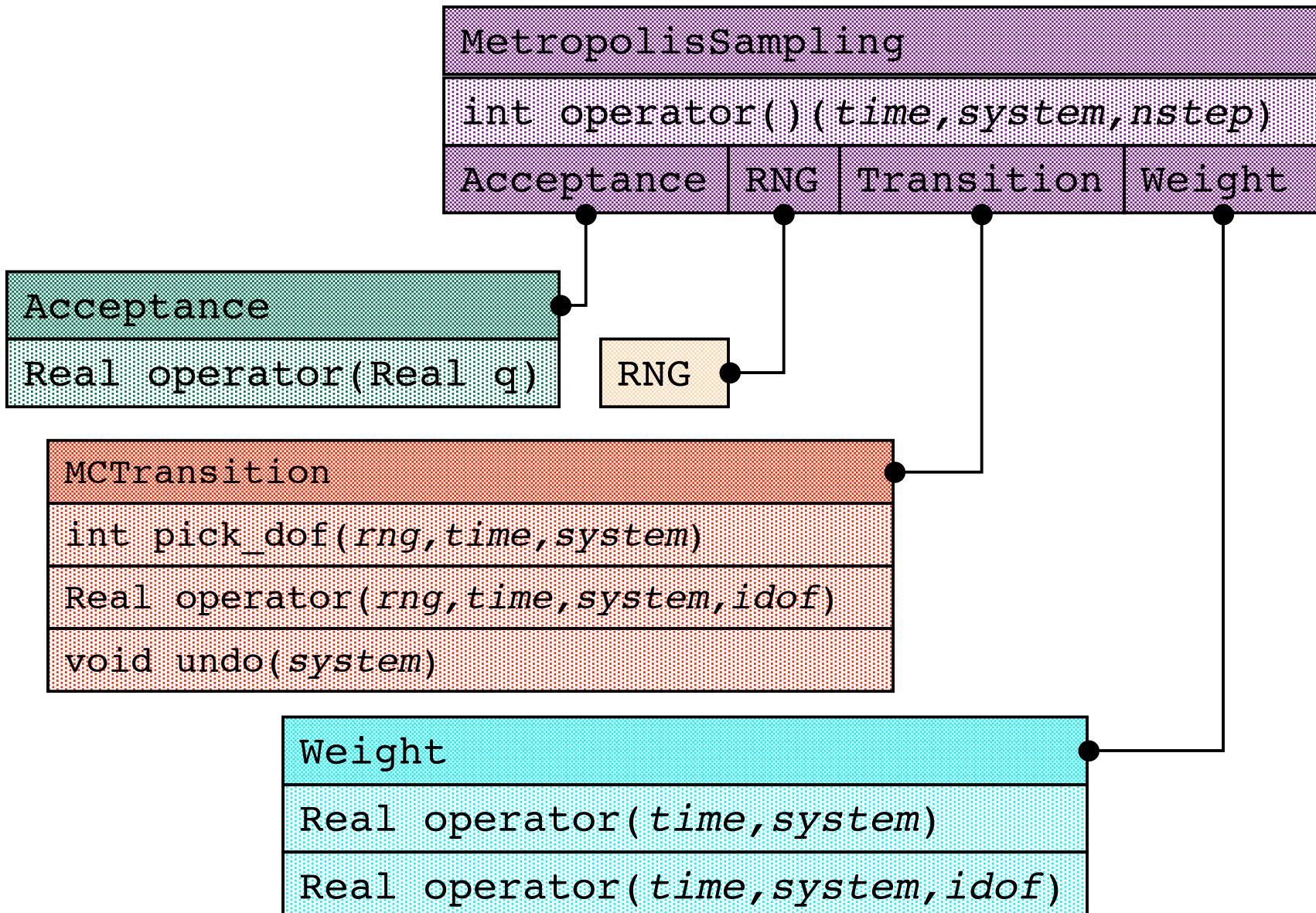
$$A(X|Y) = \frac{T(Y|X)f(X)}{T(X|Y)f(Y)} A(Y|X)$$

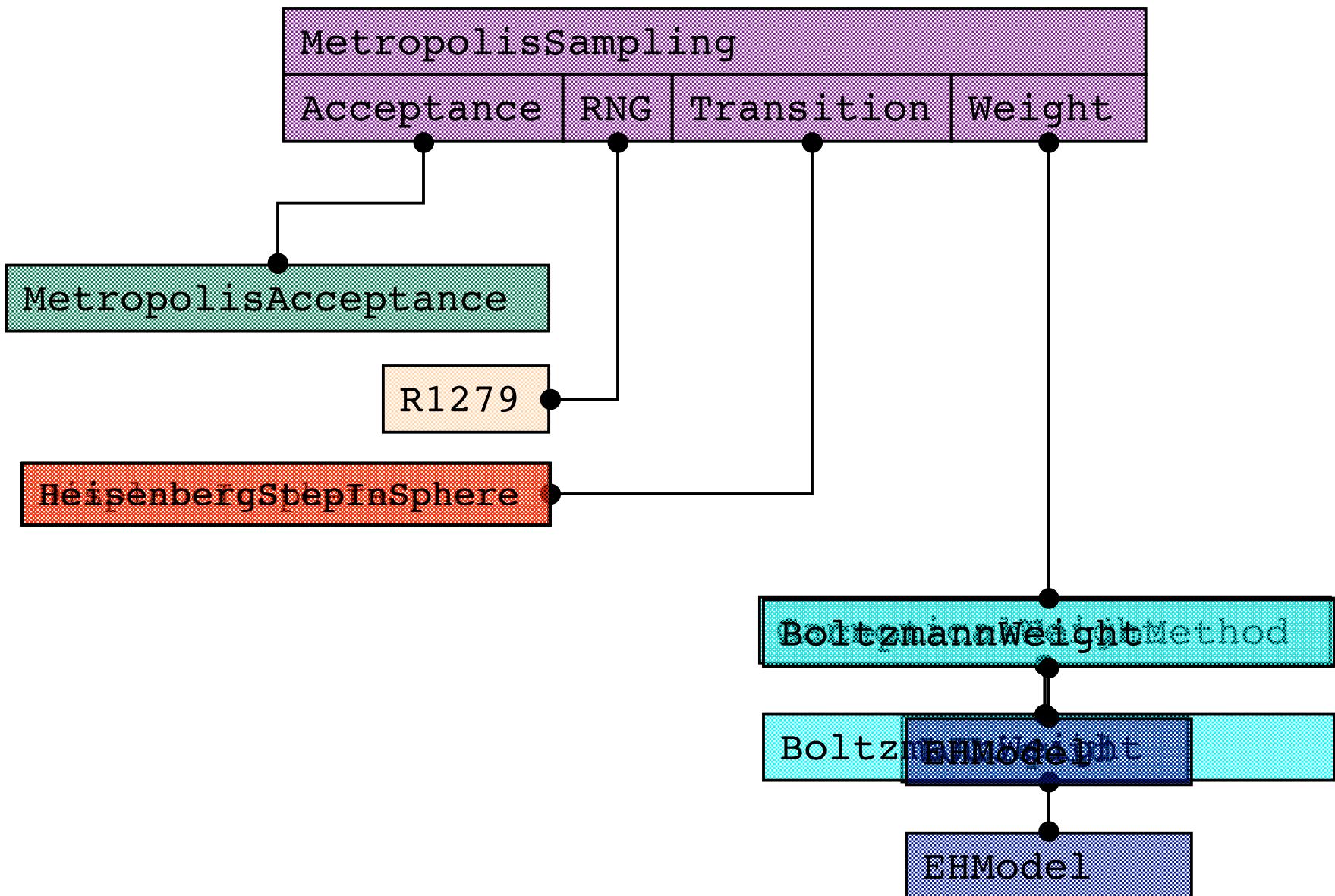
$$A = q / (1+q)$$

$$A(X|Y) = q(X|Y) \cdot A(Y|X)$$

# Metropolis Sampling Advantage

- **High-dimensional state space**
  - Generating independently inefficient
- **Relative weight of states**
  - Boltzmann factors
  - Intuitive
- **Transition from one state to another**
  - ergodicity
  - need to know imbalance in transitions



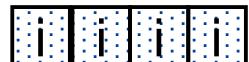


# Different “Systems”

## Ising

- Spin container

`std::vector<int>`



- Field container

`std::vector<double>`



## vs Heisenberg

`std::vector< Vec<Real, 3> >`

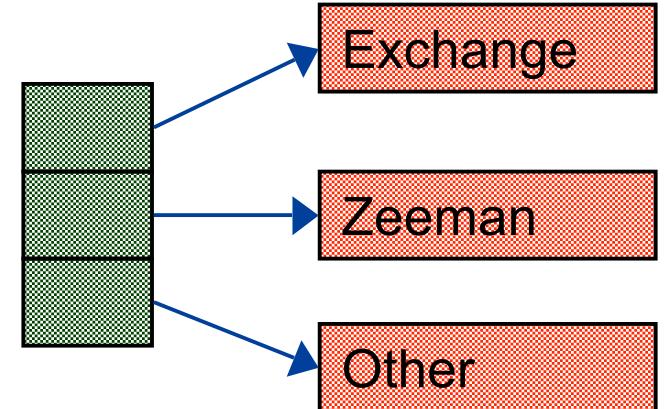
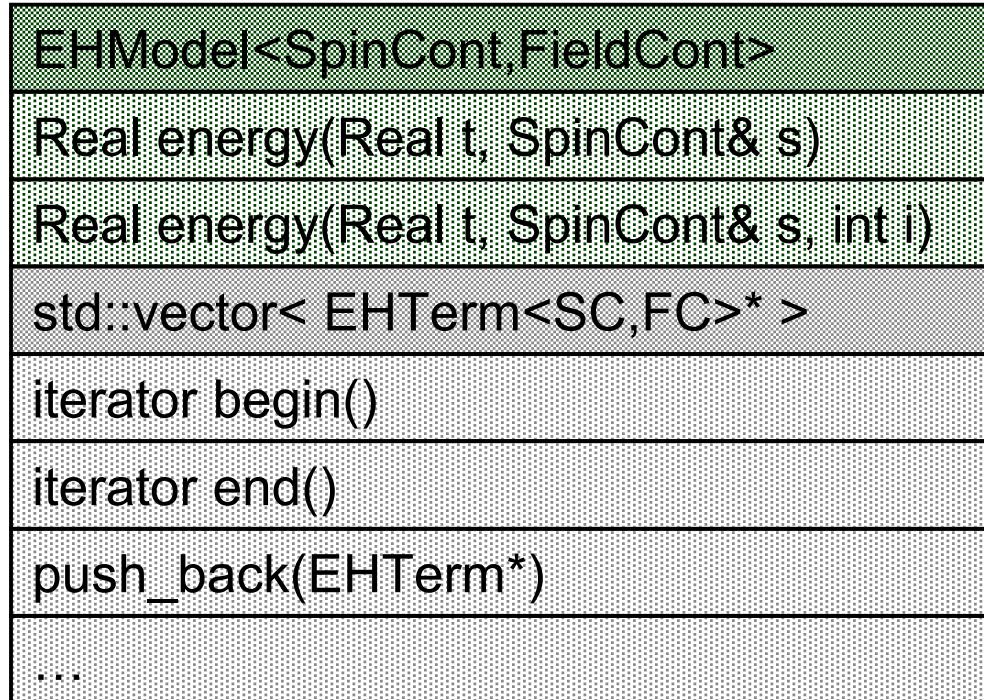


`std::vector< Vec<Real, 3> >`



# Spin-Model Energies: EHModel

derived class

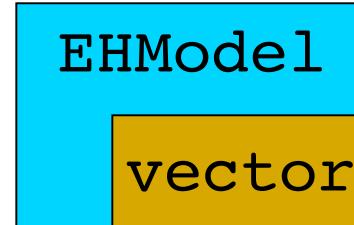
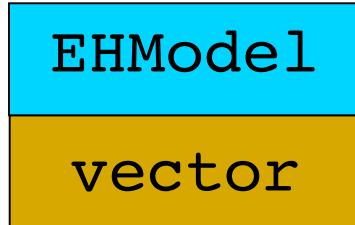


base class

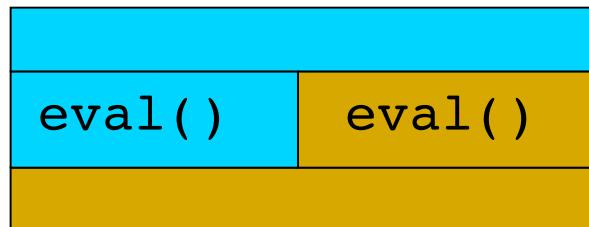
```
class EHModel : public std::vector< T*> {...};
```

# C++ Aside: Inheritance

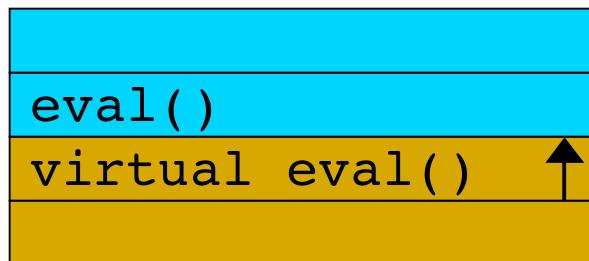
- inheritance can be public or private



- derived can override functions of base

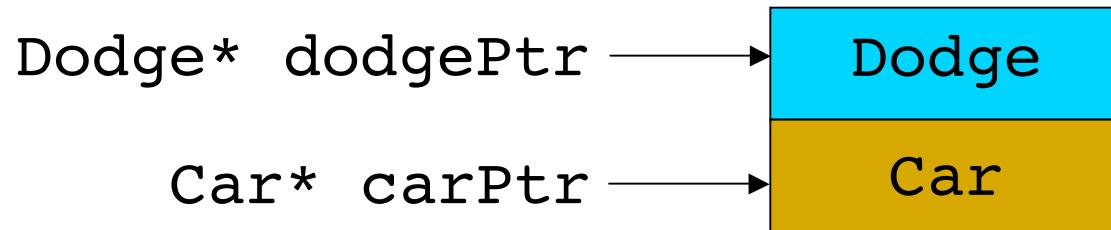


- base function refer up with virtual

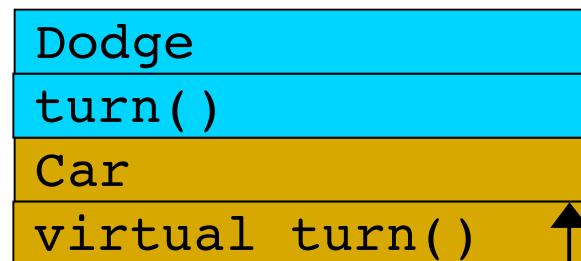


# Pointers and Objects

Base pointers only point to the base part of the object

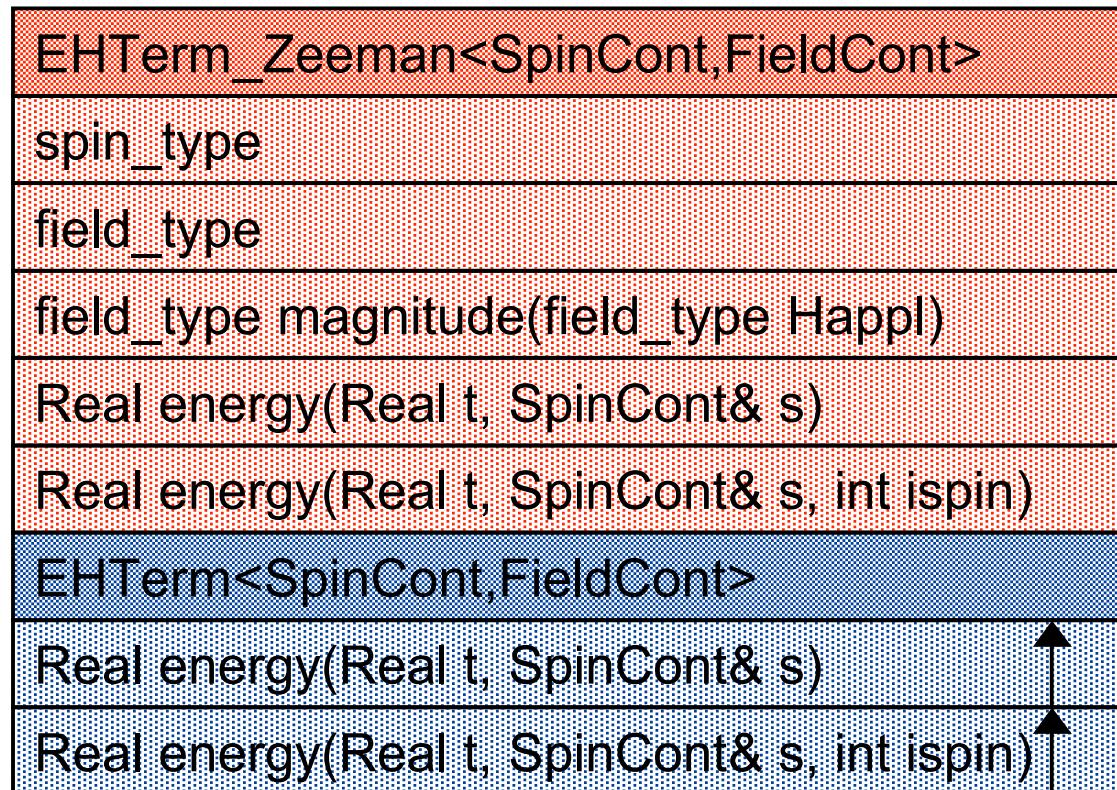


virtual functions redirect calls to derived type

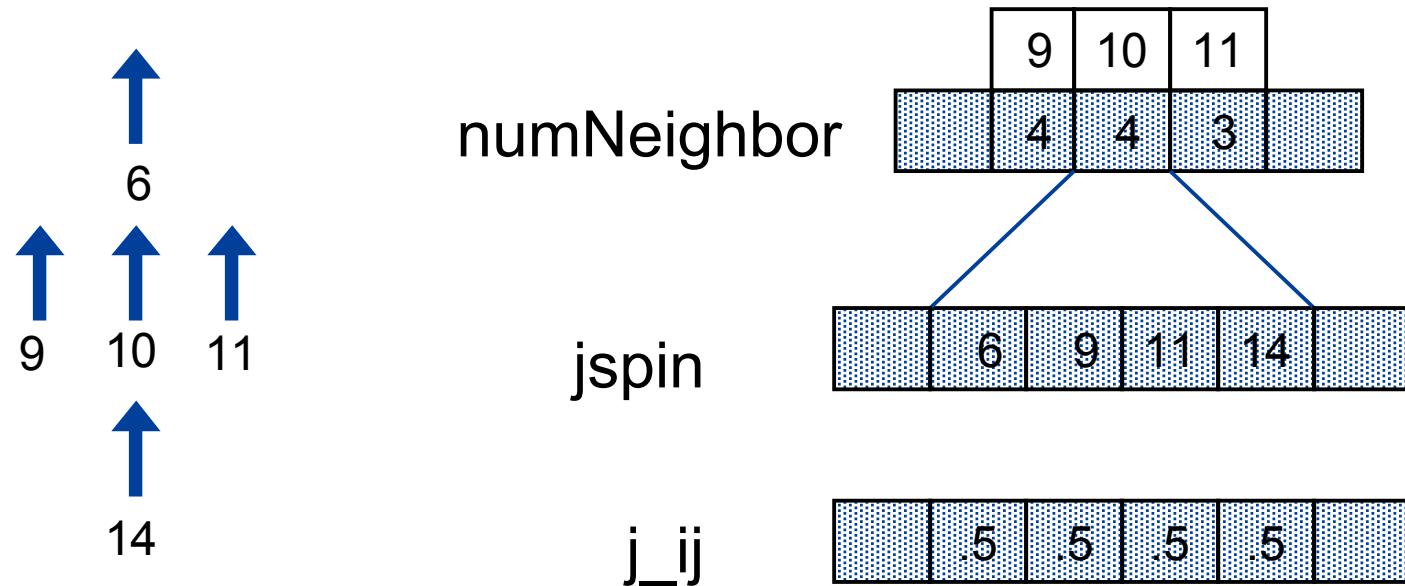


# Zeeman Energy

$$E = - \sum_{\langle i, j \rangle} J_{i,j} s_i s_j - H \sum_i s_i$$



# EHTerm\_ExchangeScalar

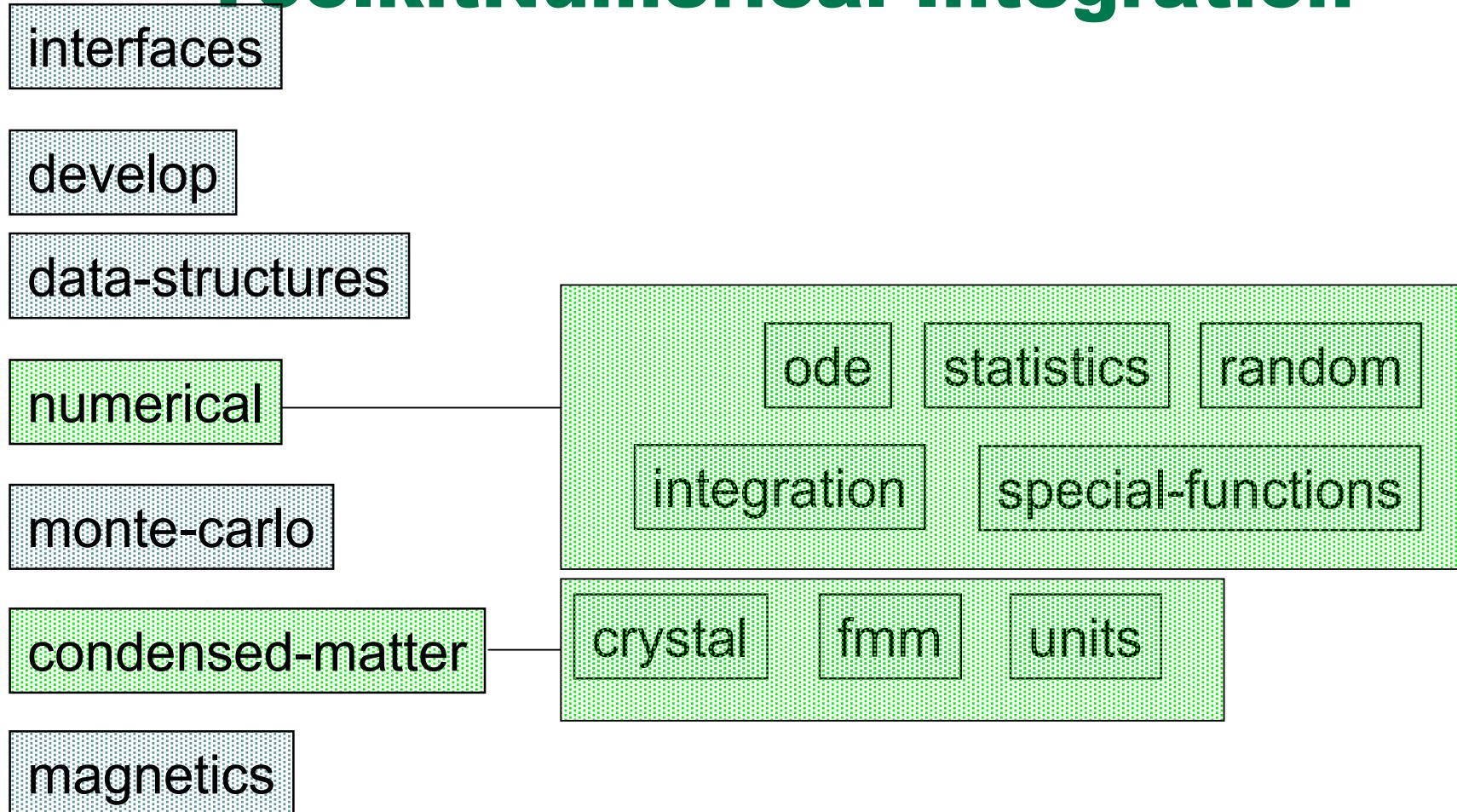


```
psimag::ExchangeScalar<SpinCont,FieldCont>
exchangeTerm(numNeighbor,jspin,j_ij);
```

# **Goals of the $\Psi$ -Mag toolkit**

- Provide tools that can be quickly assembled into new applications
- Utilize generic programming ideas to produce more flexible tools
- Foster a community approach to algorithm development and implementation

# Topical Structure of ToolkitNumerical Integration

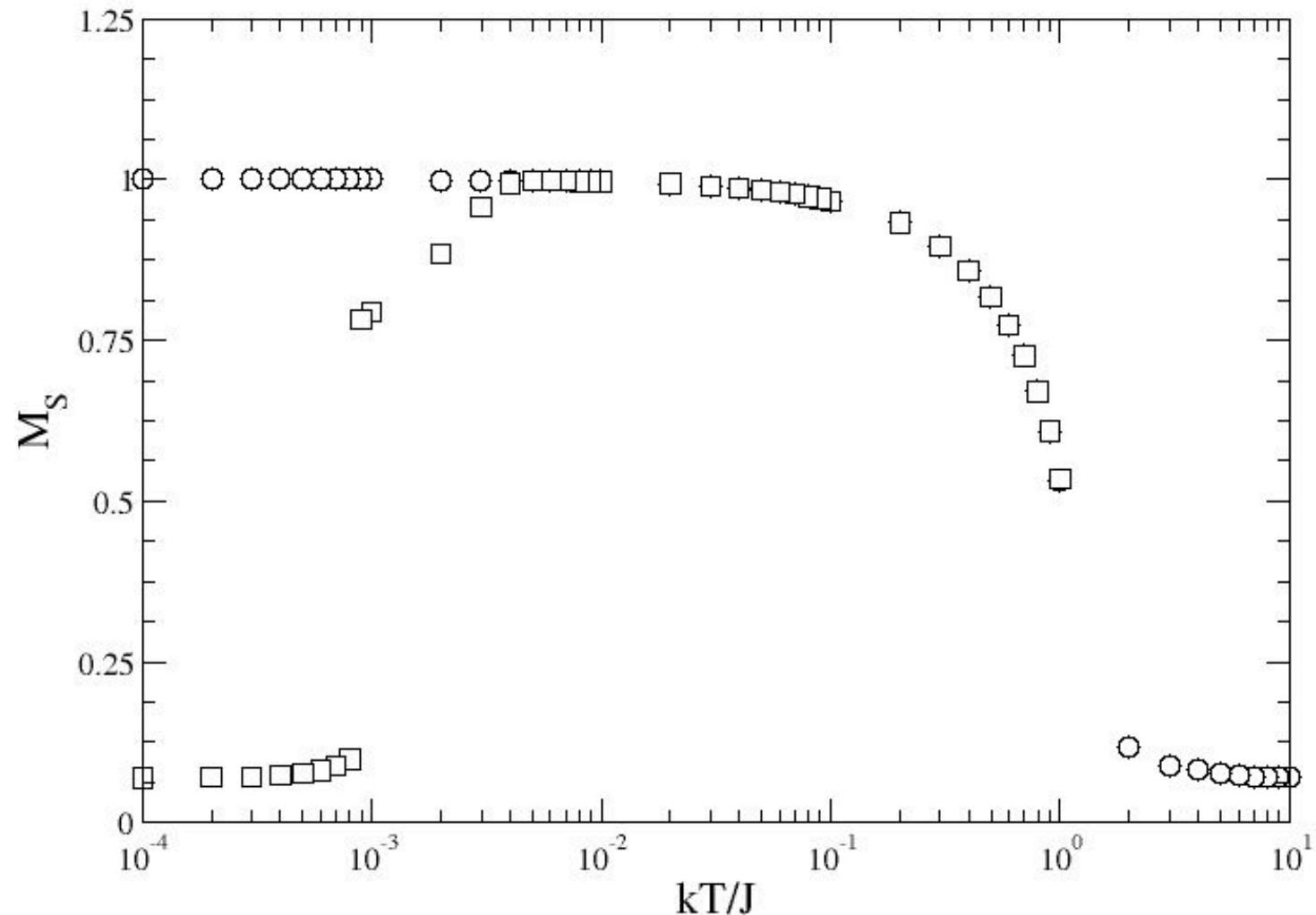


# **Case Study:**

## **Heisenberg Antiferromagnet**

- Recently needed to evaluate in a hurry
- Assembled, debugged in 24 hours
  - From Ising code used in CNMS workshop
  - Mostly dealing with underflow
  - Getting output of data just right
- Success of Generic Programming Approach

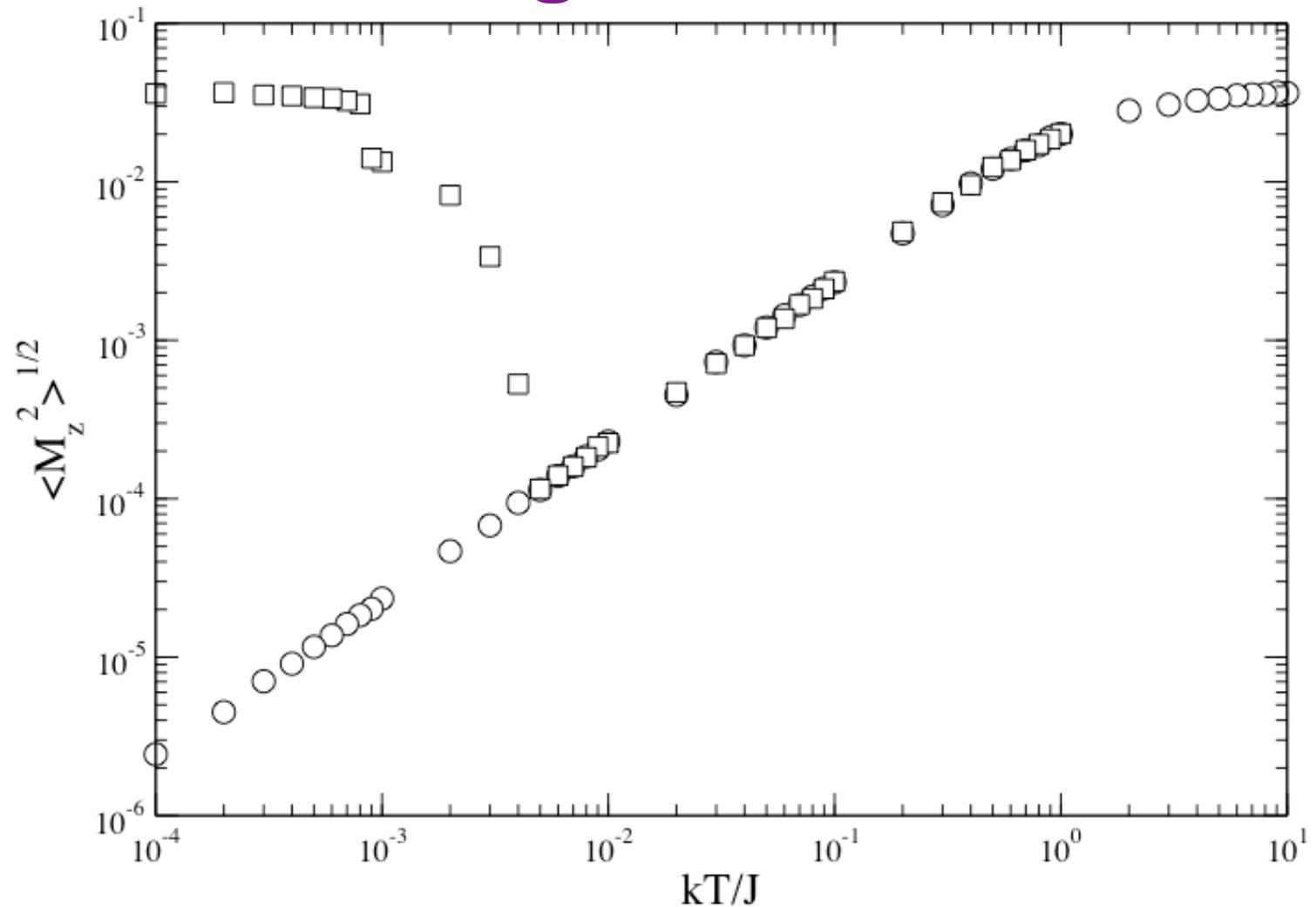
# Staggered Magnetization



OAK RIDGE NATIONAL LABORATORY  
U. S. DEPARTMENT OF ENERGY

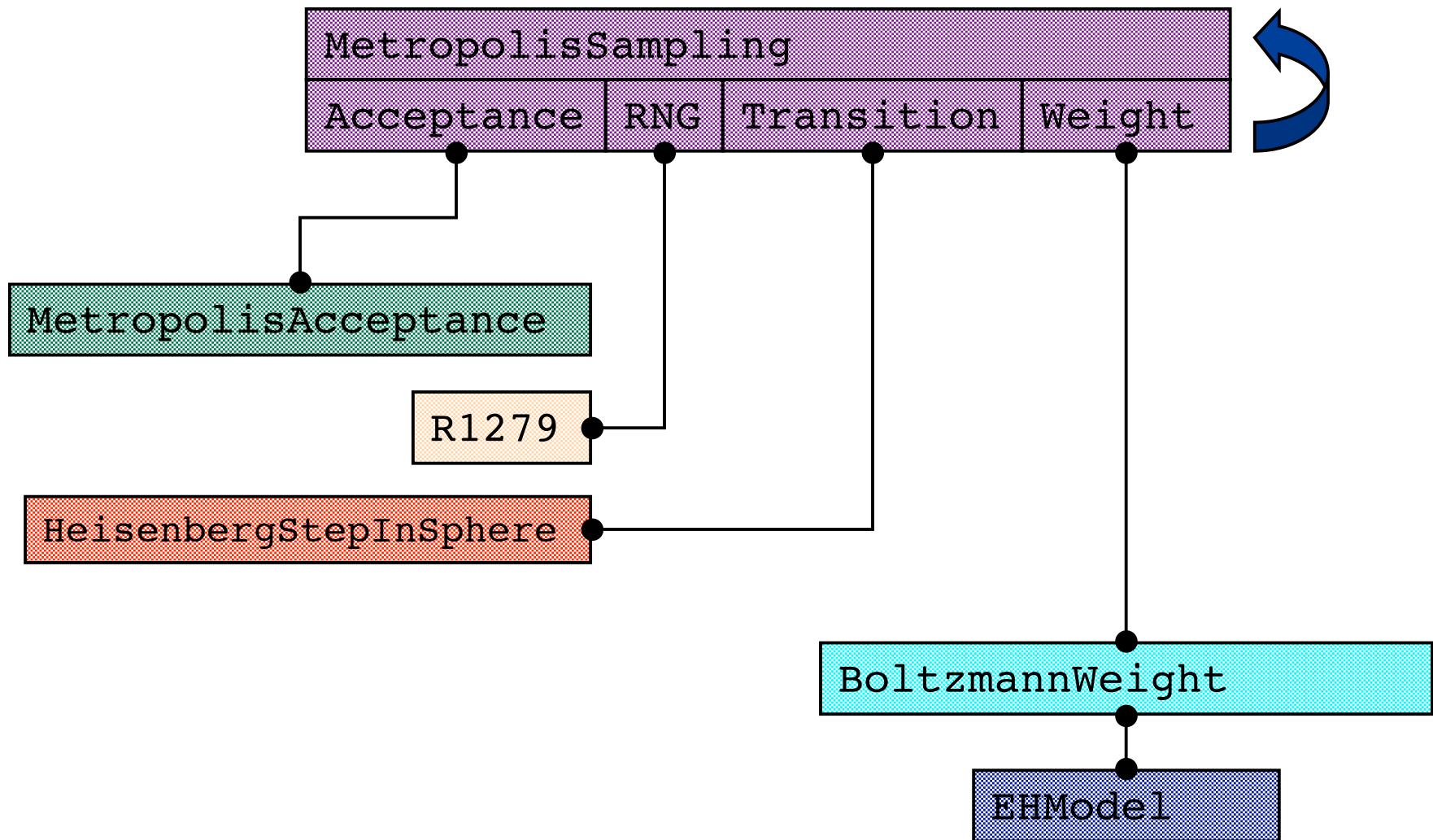


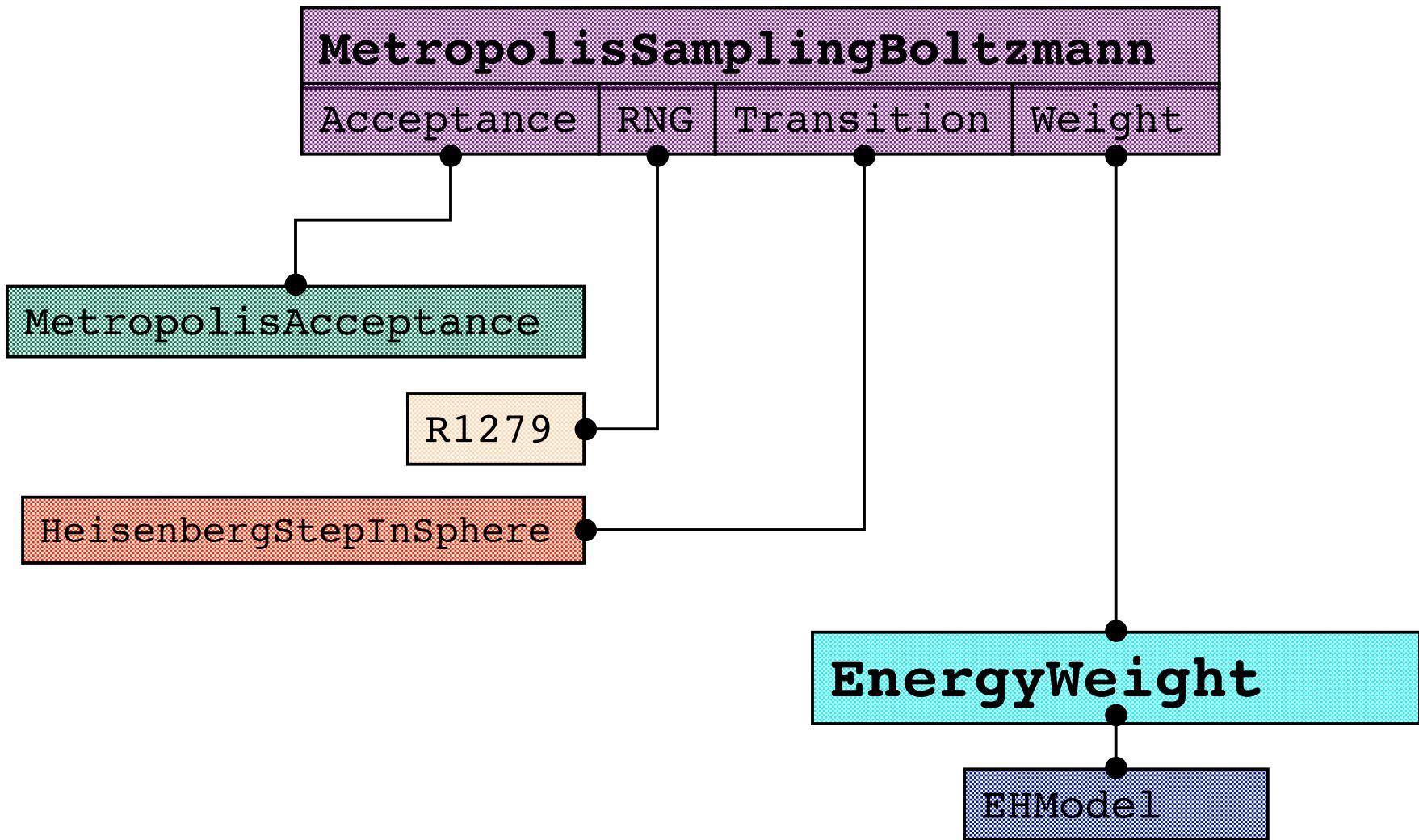
# Magnetization



OAK RIDGE NATIONAL LABORATORY  
U. S. DEPARTMENT OF ENERGY





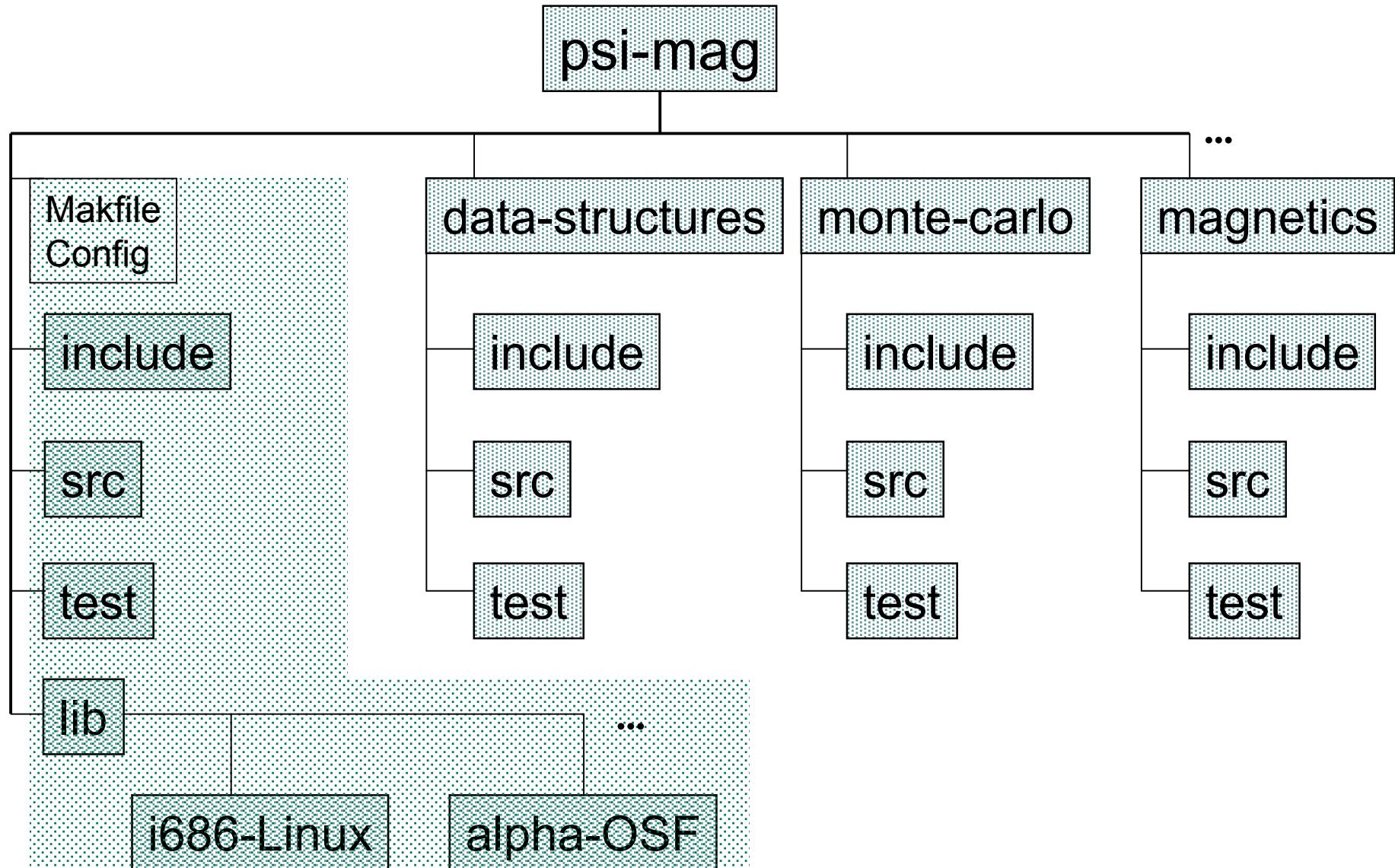


# The End

OAK RIDGE NATIONAL LABORATORY  
U. S. DEPARTMENT OF ENERGY



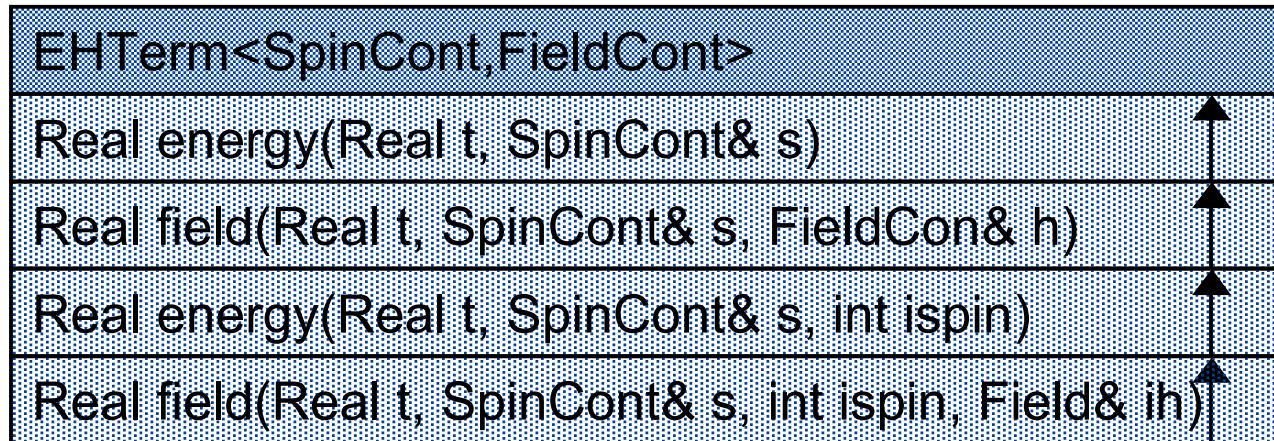
# Logical Structure of Toolkit



OAK RIDGE NATIONAL LABORATORY  
U. S. DEPARTMENT OF ENERGY

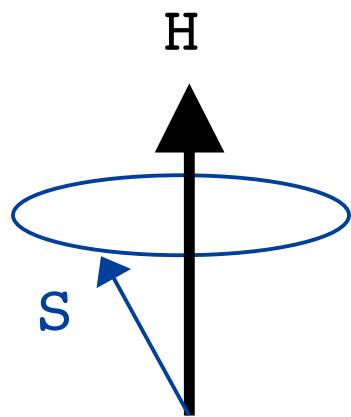


## EHModel also calculates Fields



With generic programming we have one class  
for Ising and Heisenberg models  
usable with Monte Carlo and Spin Dynamics  
Interface that can be mimicked for other models

# Spin Dynamics



$$\frac{dS}{dt} = -\beta S \times H$$

```
SuzukiTrotter<OpBase,Order>
```

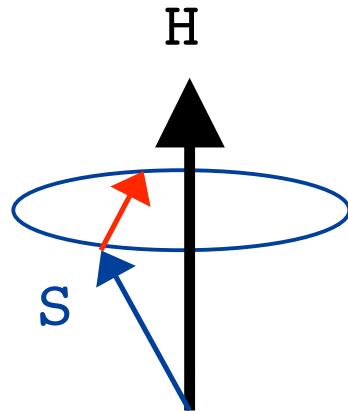
```
SuzukiTrotter(operators)
```

```
operator(system,dt)
```

There are **operators** for exchange, Zeemanfield, and anisotropy that can wrap EHTerm objects.

# Micromagnetics

$$\frac{dS}{dt} = -\beta S \times \{H - \alpha \operatorname{sgn}(\beta) S \times H\}$$

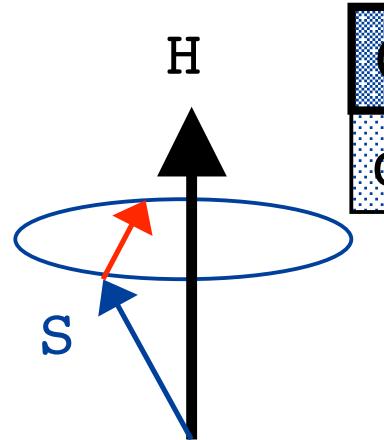


LLGOperator<Model>
LLGOperator( <i>model, alpha, beta, eps</i> )
operator()( <i>t, y, dy</i> )
operator()( <i>t, y, dy, dn</i> )

LLGOperator is passed as argument to deterministic or stochastic ODESolver

# ODESolvers

$$\frac{dy_i}{dt} = f(t, \{y_i\})$$



**ODESolvers**  
operator()(f,t<sub>0</sub>,t<sub>1</sub>,y)

EulerODE  
RungeKutta45  
RotateEuler

**StochODESolvers**  
operator()(f,rng,t<sub>0</sub>,t<sub>1</sub>,y)

RotateEuler

LLGOperator is passed as argument to deterministic or stochastic ODESolver

# **StatisticalMoments.h**

## **StatisticalMomentsUtil.h**

$$S_{iMom} = \sum_j (x_j)^{iMom}$$

```
psimag::StatisticalMoments<T>
```

```
StatisticalMoments(int maxMom)
```

```
operator+=(T newVal)
```

```
T moment(int iMom)
```

```
std::vector<T>
```

```
int operator[](int iSum)
```

$$\bar{x} = S_1 / S_0$$

psimag::Mean( mag )

$$\text{var}(x) = \frac{S_2 - S_1^2 / S_0}{S_0}$$

psimag::Variance( mag )

# StatisticalMoments< Vec<T,D> >

$$S_{iComp,iMom} = \sum_j (x_{iComp,j})^{iMom}$$
$$S_{D,iMom} = \sum_j |x_j|^{iMom}$$

```
psimag::StatisticalMoments<T>
```

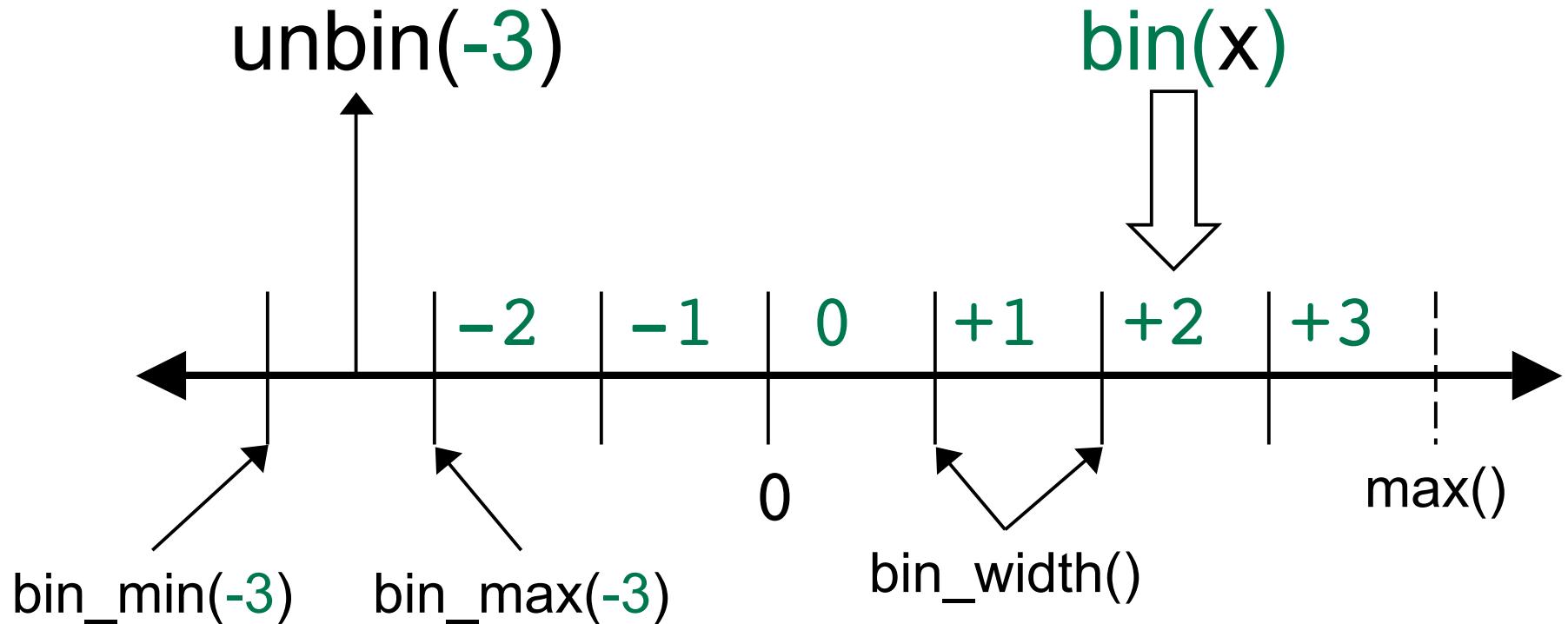
```
operator+=(T newVal)
```

```
T moment(int iComp, int iMom)
```

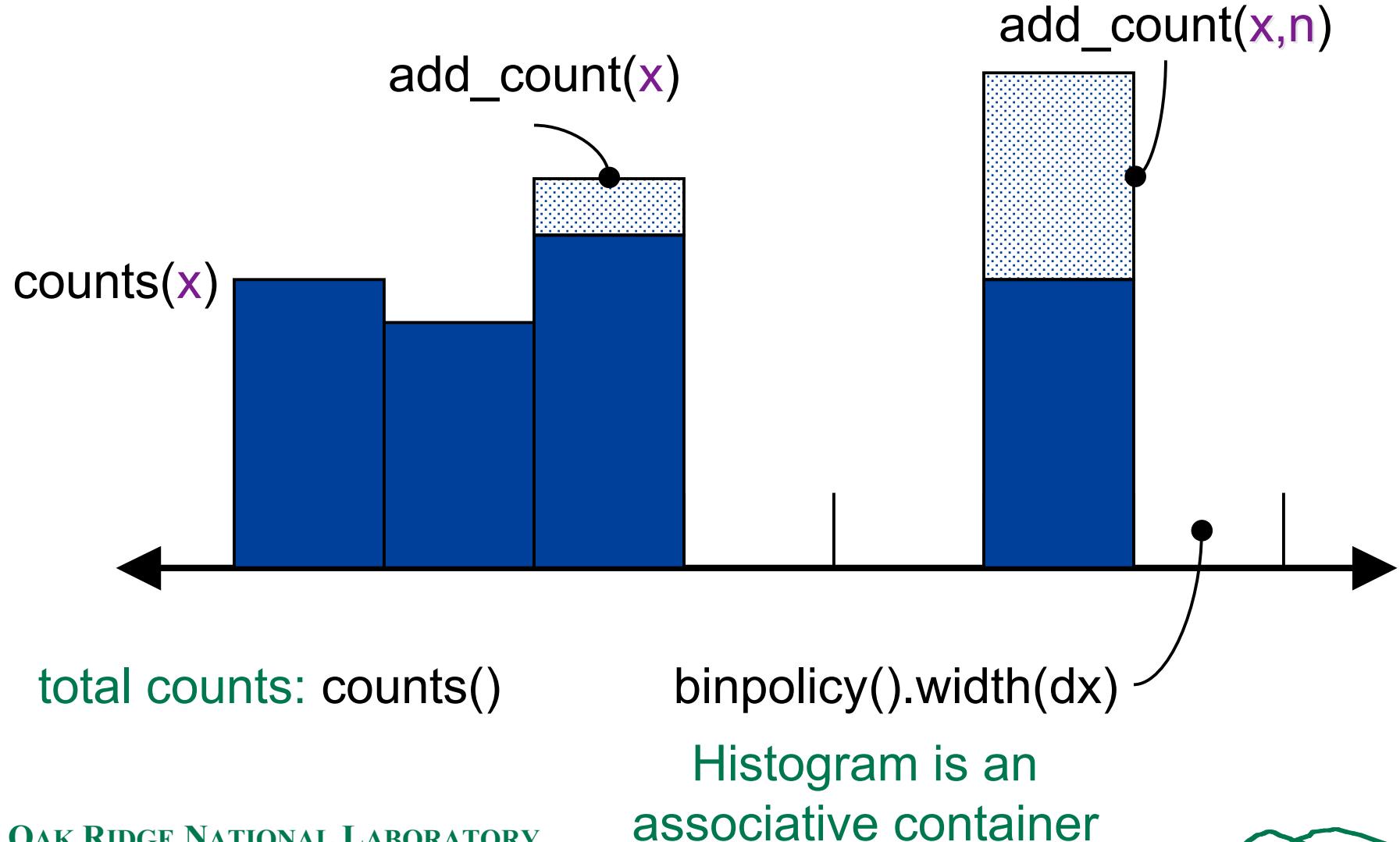
```
StatisticalMoments<T>
    moment(int iComp)
```

Mean( a.moment(3) )

# **BinUniform<DomainType,BinType>**

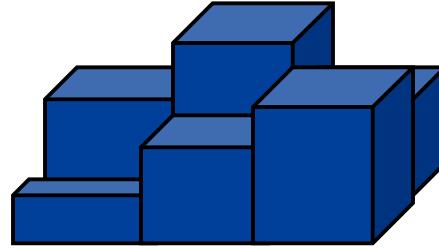


# Histogram<BP,DType,RTYPE>



# Multi-dimensional Histograms

Histogram<BinUniform,Vec<Real,3>,int>



Histogram<BinUniform,map<string,Real>,int>

```
sample[“energy”] = ecalc;  
sample[“pressure”] = pssr;  
sample[“density”] = den;  
histObj.add_counts( sample );
```

# Input/Output classes

C++ thinks of I/O in terms of streams

## istream

```
std::cin >> myInt >> myFloat >> myWholeSystem;  
std::ifstream myFile("InputFile")  
std::istringstream fakeFile(stringOfInput);
```

## ostream

```
std::cout << myInt << " " << myWholeSystem;  
std::ofstream myFile("OutputFile")  
std::ostringstream fakeFile;
```

# Formatting Output

C++ will always output the data even if it looks messy

- **std::endl** end of line
- **std::flush** flush the buffer
- << “ ” << and << “\t” <<
- **std::setw(NN)** minimum width of next piece

```
std::cout << std::setw(14) << std::numSpin  
      << std::endl
```
- **std::setprecision(NN)** floating point digits

```
std::cout << std::setprecision(9) << mag << std::endl
```

# Namespace: grouping code

What if I want to make a function called `sqrt`?

```
namespace Scientist {  
  
    double sqrt  
    {  
        ...  
    };  
  
}
```

```
namespace std {  
  
    double sqrt  
    {  
        ...  
    };  
  
}
```

`Scientist::sqrt(3.);`

`std::sqrt(3.);`

`<iostream>` and `<cmath>` in  
namespace std

# Include files: declaring what

RNG.h

```
#ifndef MyRNG_H_
#define MyRNG_H_

class MyRNG
{
public:
    MyRNG(long int seedVal = 1);
    void operator=(const RNG& other)
    void seed(long int seedVal);
    double operator()();
    double operator()(double max);
private:
    long int ix;
};

#endif // MyRNG_H_
```

# Source code: defining how

## RNG.cc

```
#include "RNG.h"

MyRNG::MyRNG(long int SeedVal)
{ this->seed(seedVal); }

void MyRNG::seed(long int seedVal)
{ ix = seedVal; }

void MyRNG::operator=(const RNG& orig)
{ this->ix = orig->ix; }

double MyRNG::operator()()
{
    const int m = 101; const int a = 97; const int c = 11;
    ix = ( a*ix + c ) % m;
    return static_cast<double>(ix)/static_cast<double>(m);
}

double MyRNG::operator()(double scale)
{ return scale * operator()(); }
```

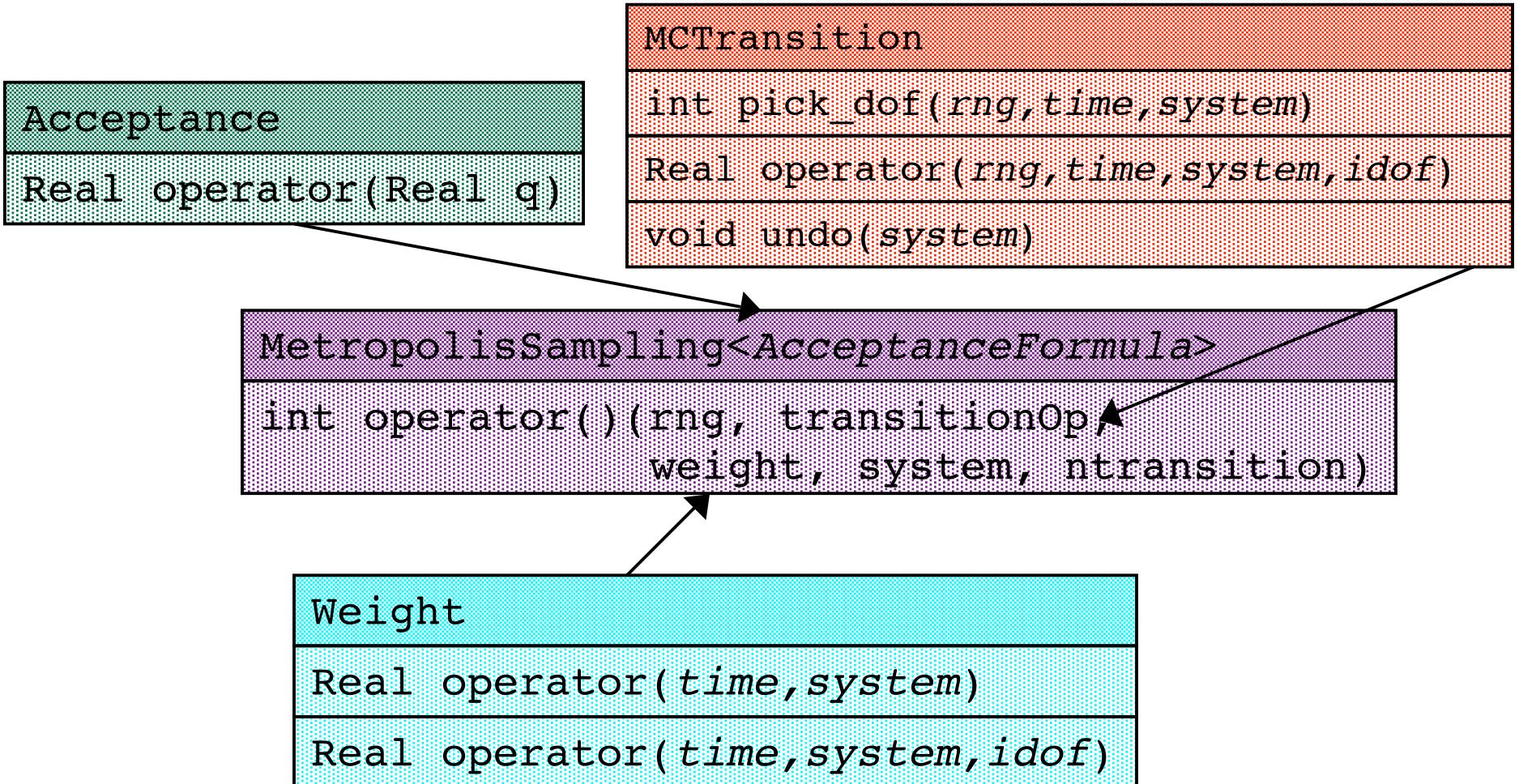
# main: tying it together

main.cc

```
#include "RNG.h"
#include <iostream>

int main(int argc, char* argv[])
{
    MyRNG rand;
    const int nout = 10;
    for(int i=0; i<nout; i++)
        std::cout << rand() << std::endl;
    return 0;
}
```

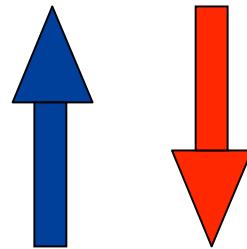
# MetropolisSampling



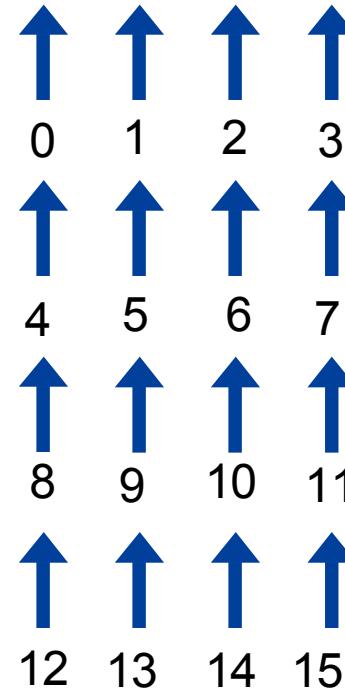
# System

## Describing Ising Models

States



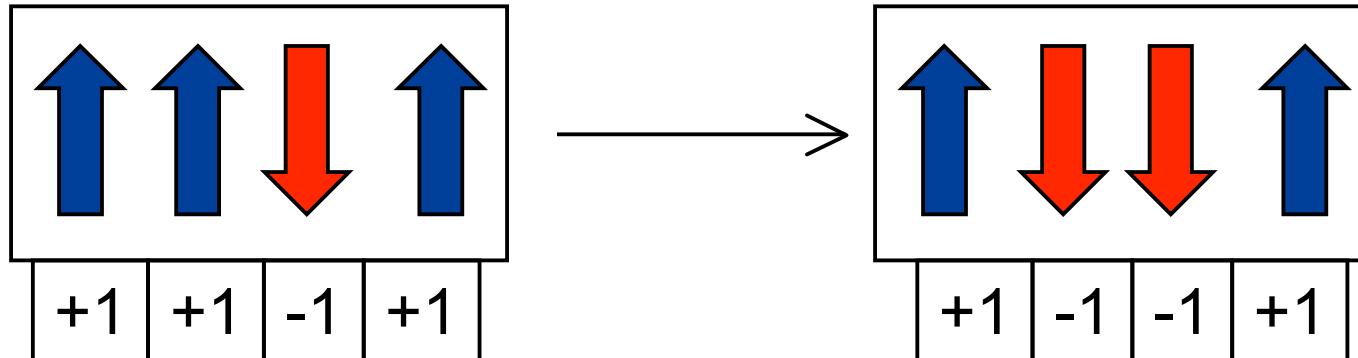
+1   -1



```
typedef std::vector<int> SpinCont;  
SpinCont s(16);
```

+1	+1	+1	+1	+1	+1	+1	+1	+1	+1	+1	+1	+1	+1	+1	+1	+1
----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----

# Transitions



MCTransition

```
int pick_dof(rng, time, system, idof)
```

```
Real operator(rng, time, system, idof)
```

```
void undo(system)
```

`psimag::IsingFlip`

`psimag::HeisenbergStepOnSphere`

`psimag::HeisenbergStepInSphere`

`psimag::HeisenbergStepAlongAxis`

`psimag::OverRelaxation`

T is prob.  $X \Rightarrow Y$   
returns  $T(Y|X)/T(X|Y)$

# Weights

Weight
Real operator( <i>time, system</i> )
Real operator( <i>time, system, idof</i> )

## Boltzmann Weight

$$f(x) = \exp(-E(x)/k_B T)$$

psimag::BoltzmanWeight< <i>Energy</i> >	
BoltzmanWeight( <i>EnergyObj</i> )	<b>Energy</b>
Real operator( <i>time, system</i> )	Real energy( <i>time, system</i> )
Real operator( <i>time, system, idof</i> )	Real energy( <i>time, system, idof</i> )
Real temperature( $k_B T$ )	

# Acknowledgements

- Sponsored by the Mathematical, Information, and Computational Sciences Division; Office of Advanced Scientific Computing Research; U.S. Department of Energy
- Sponsored by the Division of Materials Sciences and Engineering; Office of Basic Energy Sciences; U.S Department of Energy
- Sponsored by the Laboratory Directed Research and Development Program of Oak Ridge National Laboratory
- This research used resources of the Center for Computational Sciences at Oak Ridge National Laboratory
- Work performed at the Oak Ridge National Laboratory, which is managed by UT-Battelle, LLC under Contract No. DE-AC05-00OR22725
- This document described research activities performed under contract number DE-AC0500OR22750 between the U.S. Department of Energy and Oak Ridge Associated Universities.
- All opinions expressed in this report are the authors' and do not necessarily reflect policies and views of the U.S. Department of Energy or the Oak Ridge Institute for Science and Education.