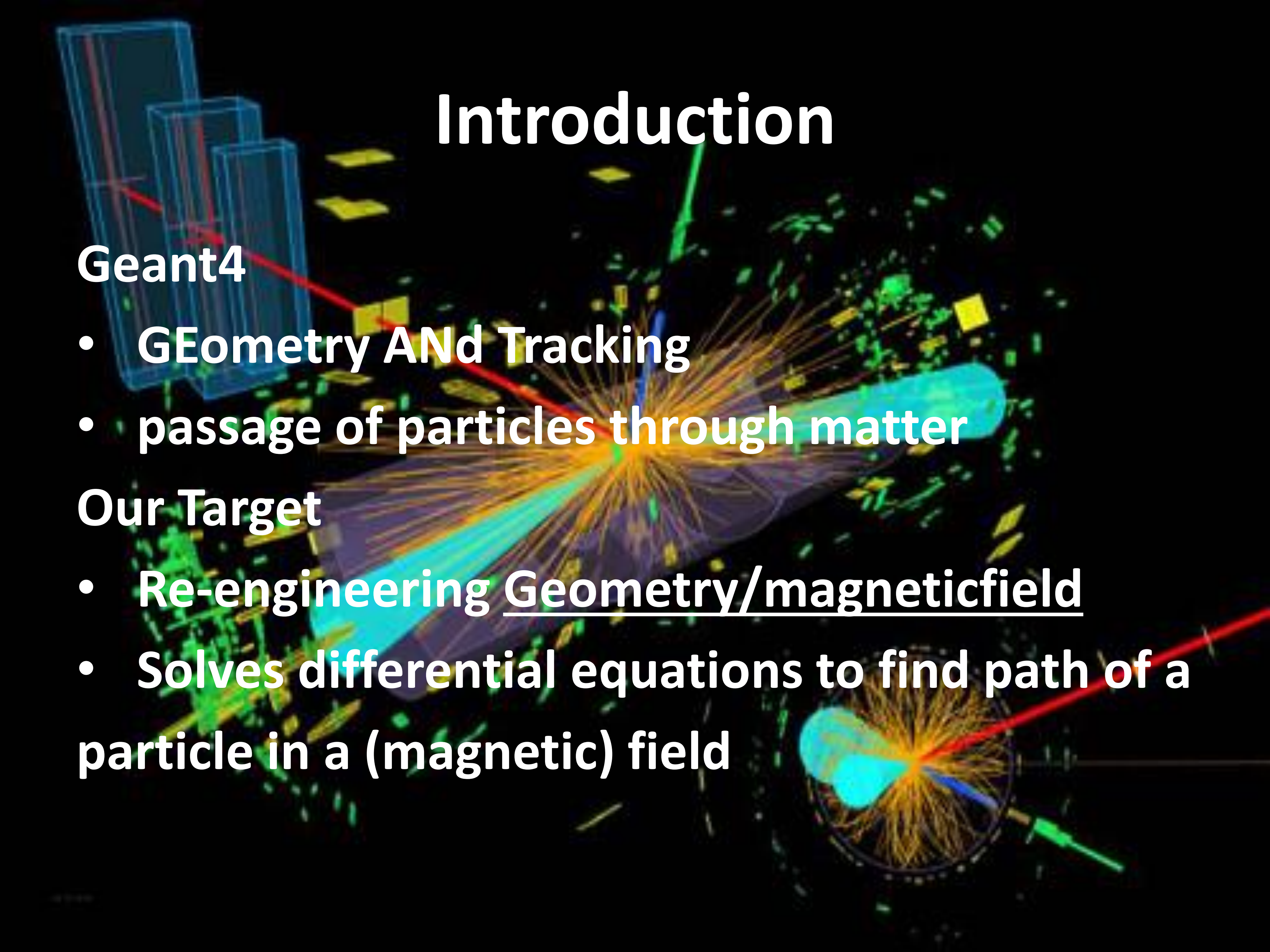
Re-engineer Propagation of Charged Tracks in Electromagnetic Field (for Geant4)

Qiuchen Xie
Mentors: Dr. Sandro Wenzel, Dr. John Apostolakis

Introduction



Geant4

- GEometry ANd Tracking
- passage of particles through matter

Our Target

- Re-engineering Geometry/magneticfield
- Solves differential equations to find path of a particle in a (magnetic) field

Overview

- **Redesign with template polymorphism to get rid of virtual function calls**
  - Template method pattern: CRTP;
  - Benefit without using virtual function: we can inline those functions to further imporve its performance, i.e. there are no function calls left in stepper classes;
- **Vecotorization**
  - version 1.0: revise the code and use compiler's optimizer only (e.g. loop unrolling & vectorization);
  - version 2.0: use a template vectorization library (Blaze-lib) to vectorize the code.

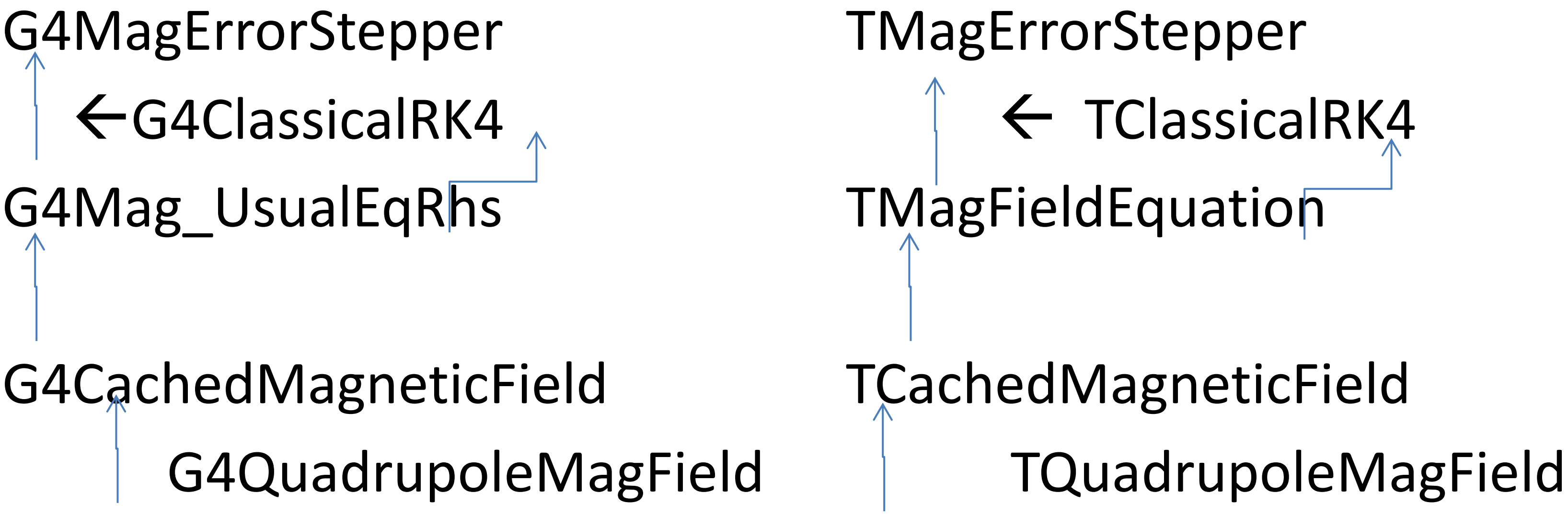
Classes Structure

- There are total 15 different steppers that users can use. 3 of them are embedded steppers
- Stepper returns dy/dx and error based upon a fixed step size h
- RightHandSide(x) from equation class will be called by stepper
- Equation class will need values from GetFieldValue() from Field Classes
- User can customize the field classes as needed

Section 1- master branch

TEMPLATE STEPPER, EQUATION, AND FIELD CLASSES

Plug and play- A comparsion



Current version

G4MagErrorStepper

←G4ClassicalRK4

G4Mag_UsualEqRhs

G4CachedMagneticField

G4QuadrupoleMagField

Templated version

TMagErrorStepper

← TClassicalRK4

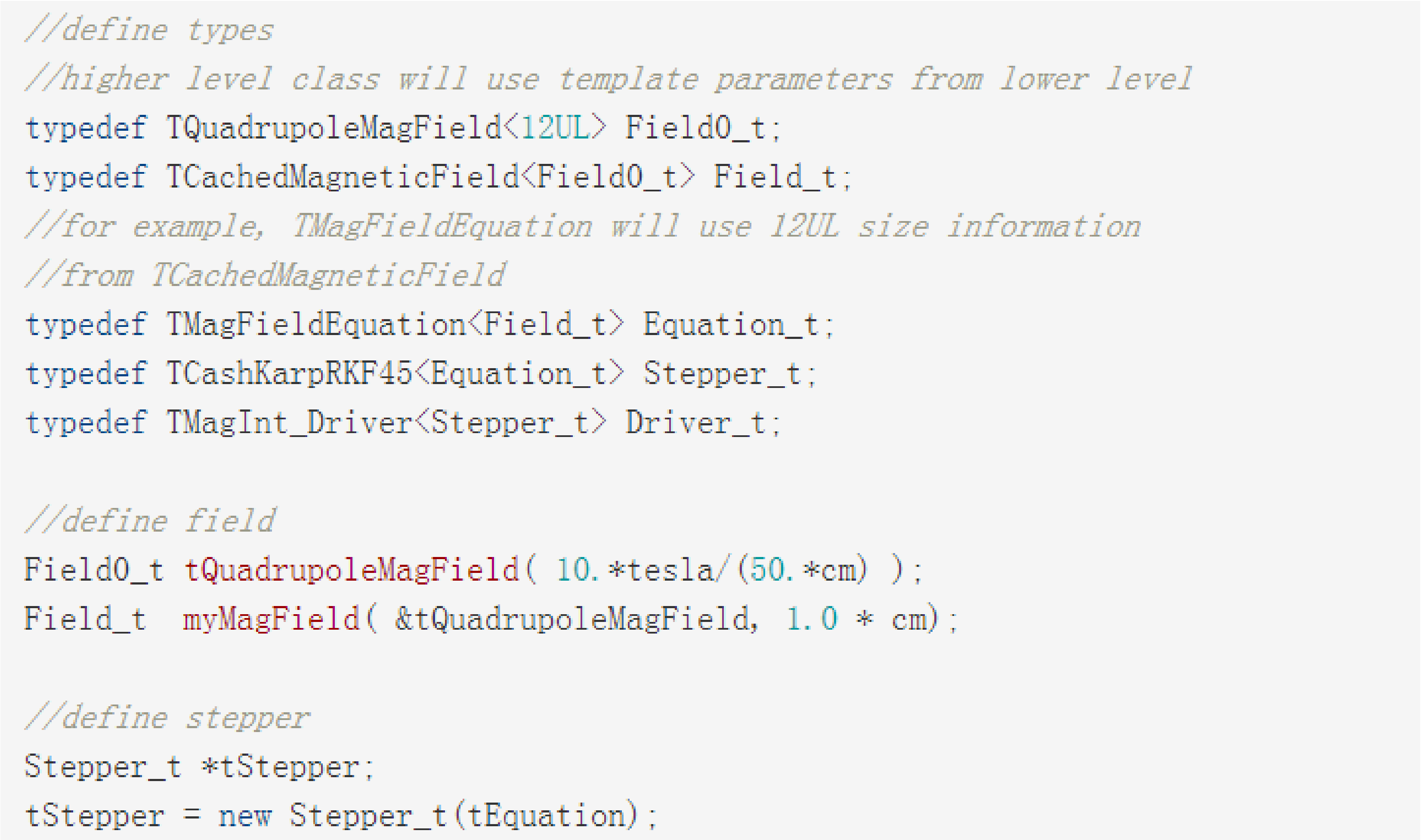
TMagFieldEquation

TCachedMagneticField

TQuadrupoleMagField

As shown, the template classes follow the **same inheritance structures**; we can **easily plug into** the new templated version to replace the current version.

New Classes



```
//define types
//higher level class will use template parameters from lower level
typedef TQuadrupoleMagField<12UL> Field0_t;
typedef TCachedMagneticField<Field0_t> Field_t;
//for example, TMagFieldEquation will use 12UL size information
//from TCachedMagneticField
typedef TMagFieldEquation<Field_t> Equation_t;
typedef TCashKarpRKF45<Equation_t> Stepper_t;
typedef TMagInt_Driver<Stepper_t> Driver_t;

//define field
Field0_t tQuadrupoleMagField( 10.*tesla/(50.*cm) );
Field_t  myMagField( &tQuadrupoleMagField, 1.0 * cm);

//define stepper
Stepper_t *tStepper;
tStepper = new Stepper_t(tEquation);
```

Field Propagation

```
template <class T_Equation>
class TClassicalRK4 : public  TMagErrorStepper
                      <TClassicalRK4<T_Equation>>
{
//example1-function call with known namespace
fEquation_Rhs->T_Equation::RightHandSide(y, dydx);

//example2- take variable size as template parameters
static const size_t Nvar = Equation::Nvar;
//build static arrays better for vectorization
private:
G4double dydxm[Nvar];
}

template <class T_Field>
class TMagFieldEquation : public G4Mag_UsualEqRhs
{
//example1-function call
itsField->T_Field::GetFieldValue(Point, Field);

//example2-we apply G4Pow and vdt::fast_isqrt_general in all classes to improve their
//performance
   G4double inv_momentum_magnitude =
                    vdt::fast_isqrt_general( momentum_mag_square, 4);
}
```

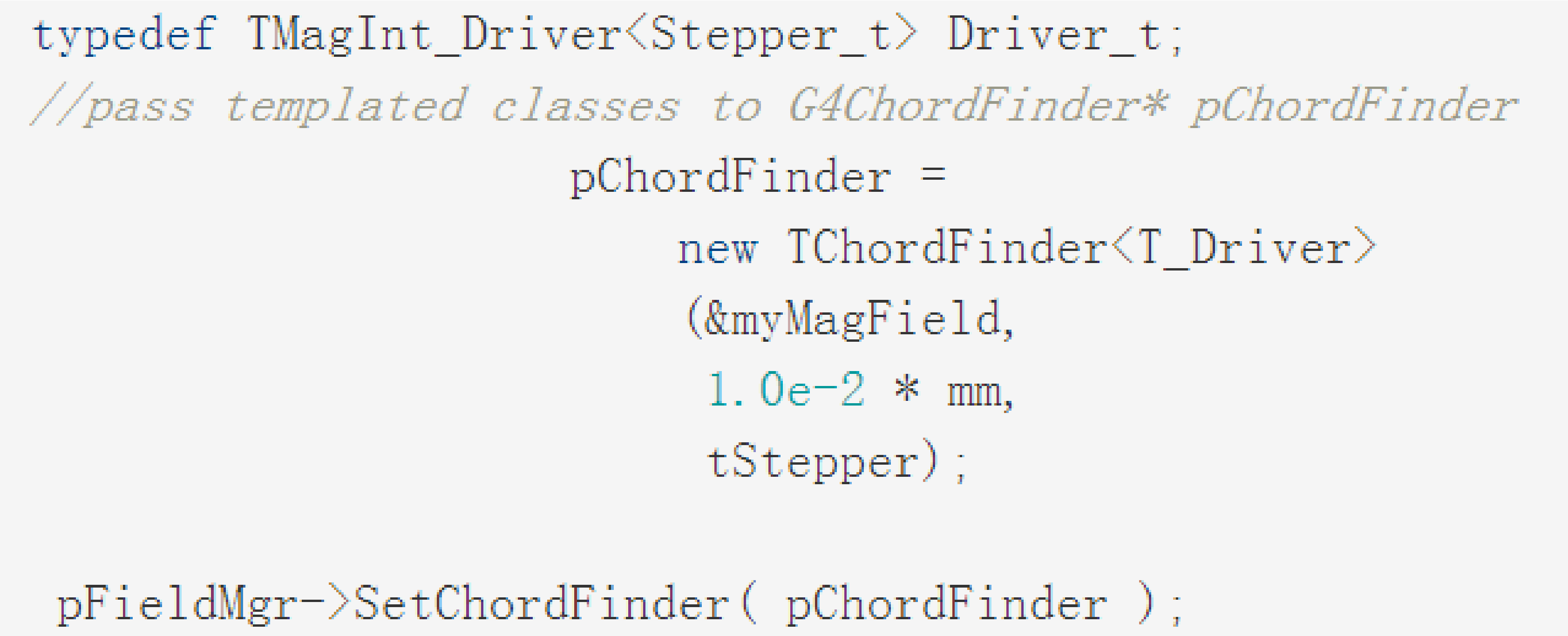
Section 2-branch: “t_driver”

TEMPLATED DRIVER AND CHORDFINDER CLASSES

Overview

- We prepared the master branch that called by G4ChordFinder; so user can test their performance without worrying changing any code
- In this branch, we templated higher level classes; thus we can get rid of virtual function calls completely
- We also use G4Pow to replace std::pow, providing a significantly faster pow method

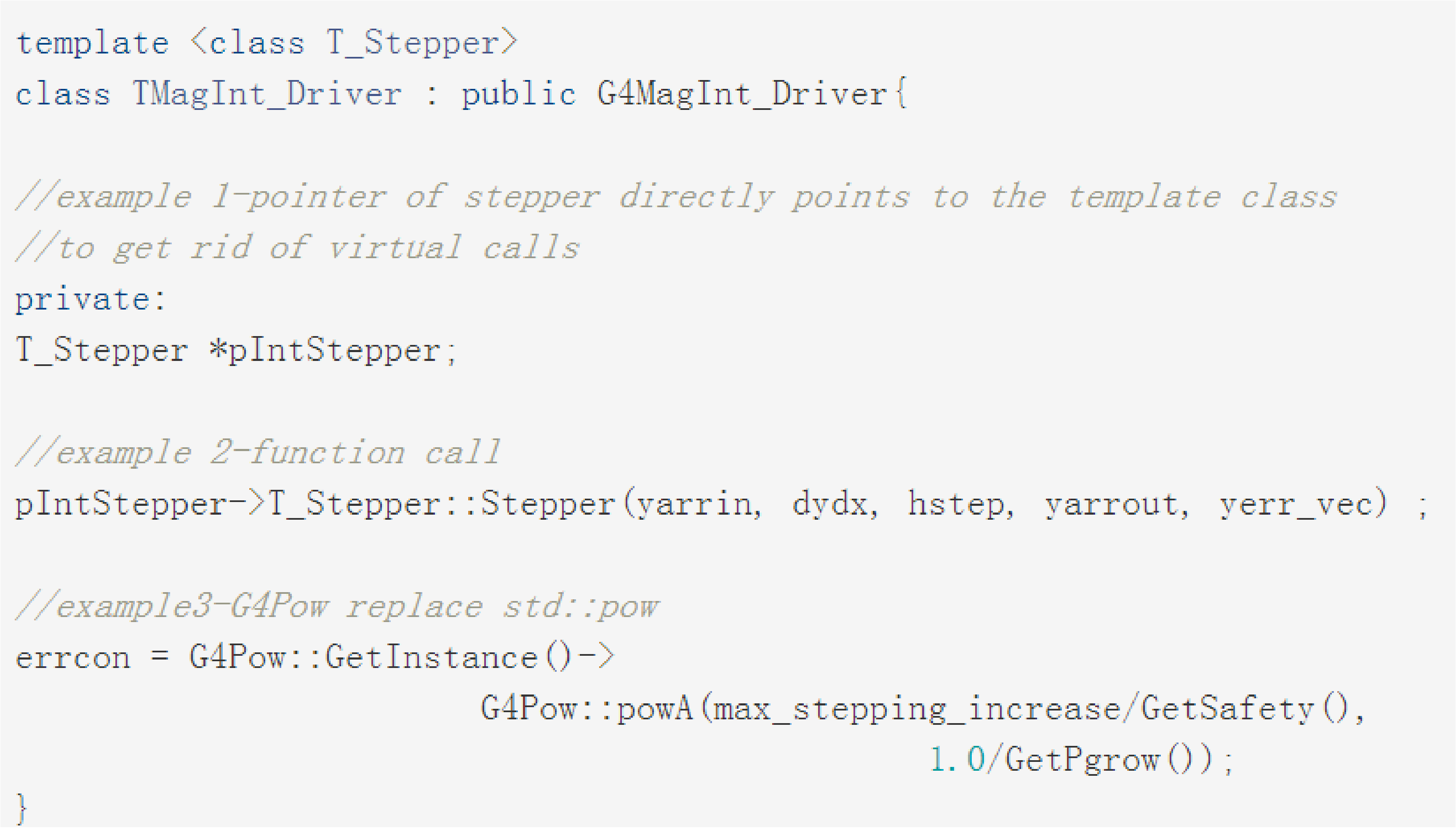
New Classes



```
typedef TMagInt_Driver<Stepper_t> Driver_t;
//pass templated classes to G4ChordFinder* pChordFinder
                    pChordFinder =
                        new TChordFinder<T_Driver>
                        (&myMagField,
                         1.0e-2 * mm,
                         tStepper);

 pFieldMgr->SetChordFinder( pChordFinder );
```

Templated Driver Class



```
template <class T_Stepper>
class TMagInt_Driver : public G4MagInt_Driver{

//example 1-pointer of stepper directly points to the template class
//to get rid of virtual calls
private:
T_Stepper *pIntStepper;

//example 2-function call
pIntStepper->T_Stepper::Stepper(yarrin, dydx, hstep, yarrout, yerr_vec) ;

//example3-G4Pow replace std::pow
errcon = G4Pow::GetInstance()->
                        G4Pow::powA(max_stepping_increase/GetSafety(),
                                               1.0/GetPgrow());
}
```

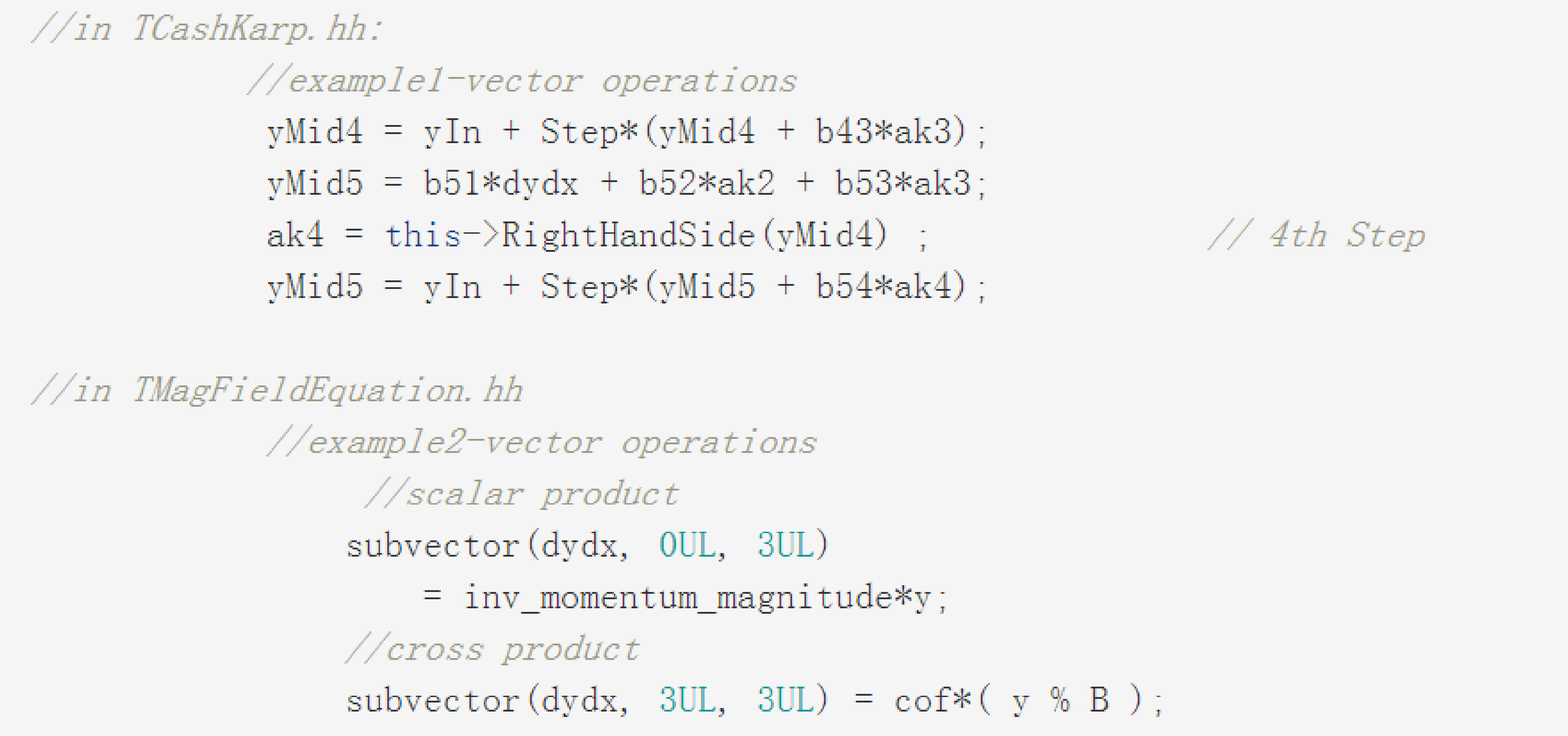
Section 3- branch: “vector_interface”

VECTORIZE WITH BLAZE-LIB

Overview

- Vector to replace arrays
  - Cleaner code: no loops; dot product: (v1,v2); cross product: v1%v2; array copy: v1 = v2; etc.
  - Vectorize more thoroughly with compiler optimizer
- Vector-type signatures
  - Instead of passing by reference, we return result storing in vectors
- Interface to high level classes
  - TChordFinder only requires a few changes to adapt to this method; no higher level class above TChordFinder is required to revise

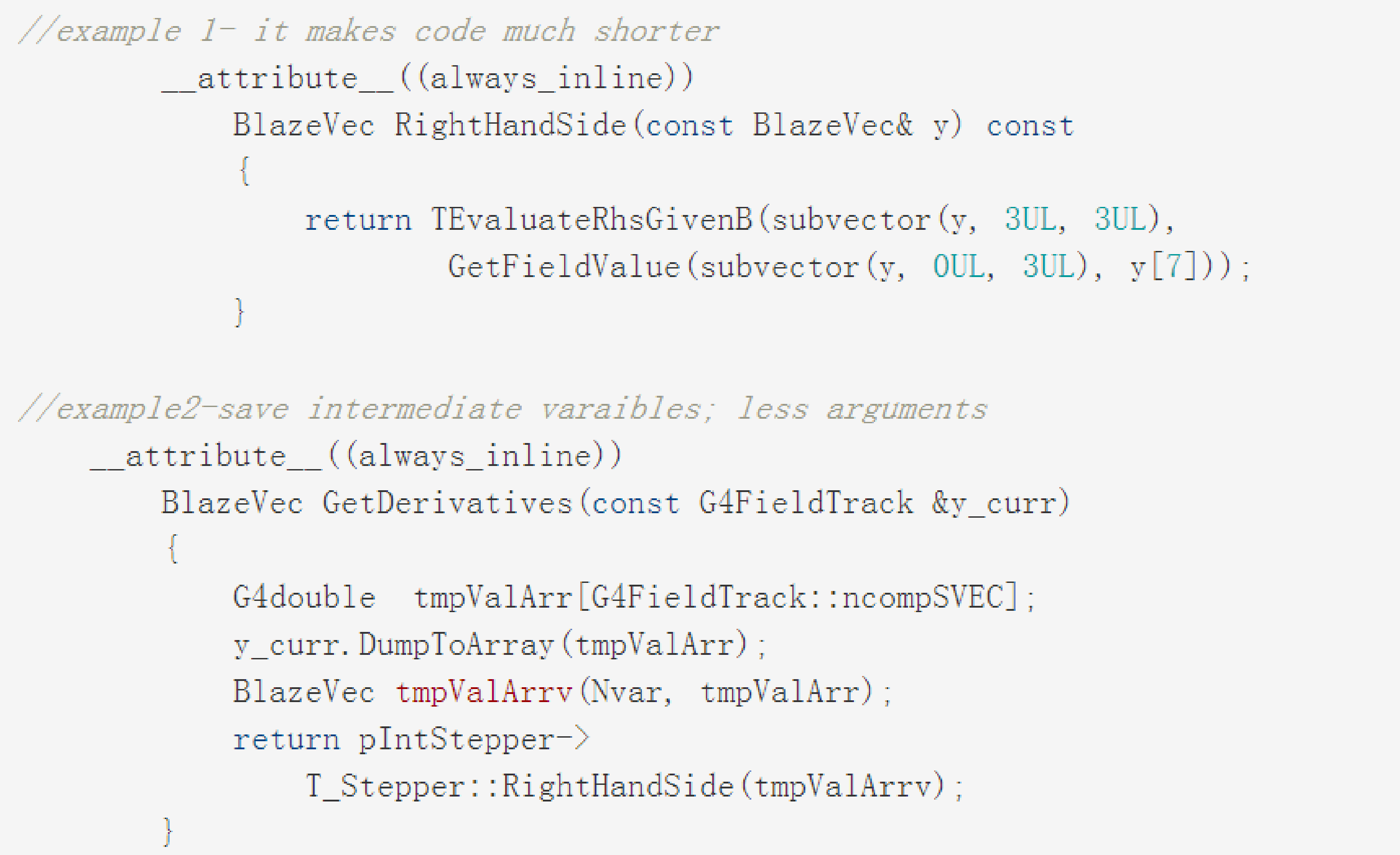
Vector Operators



```
//in TCashKarp.hh:
           //example1-vector operations
            yMid4 = yIn + Step*(yMid4 + b43*ak3);
            yMid5 = b51*dydx + b52*ak2 + b53*ak3;
            ak4 = this->RightHandSide(yMid4) ;              // 4th Step
            yMid5 = yIn + Step*(yMid5 + b54*ak4);

//in TMagFieldEquation.hh
            //example2-vector operations
                 //scalar product
                subvector(dydx, 0UL, 3UL)
                    = inv_momentum_magnitude*y;
                //cross product
                subvector(dydx, 3UL, 3UL) = cof*( y % B );
```

Vector Return Type



```
//example 1- it makes code much shorter
        __attribute__((always_inline))
            BlazeVec RightHandSide(const BlazeVec& y) const
            {
                return TEvaluateRhsGivenB(subvector(y, 3UL, 3UL),
                        GetFieldValue(subvector(y, 0UL, 3UL), y[7]));
            }

//example2-save intermediate varaibles; less arguments
    __attribute__((always_inline))
        BlazeVec GetDerivatives(const G4FieldTrack &y_curr)
        {
            G4double  tmpValArr[G4FieldTrack::ncompSVEC];
            y_curr.DumpToArray(tmpValArr);
            BlazeVec tmpValArrv(Nvar, tmpValArr);
            return pIntStepper->
                T_Stepper::RightHandSide(tmpValArrv);
        }
```

Results

- 5-7% speed up (just template polymorphism) on tdriver branch; No observable speed changes on Blaze branch
- Applied vectorization and improved about 20% of speed
- Obtained a cleaner look of code (blaze version)
- Used G4Pow() and fast_inverse_sqrt() function (much faster in speed)-- provided overall 200% speed boost for tabulated field benchmark case
- Got rid of some virtual function calls and inlined those functions

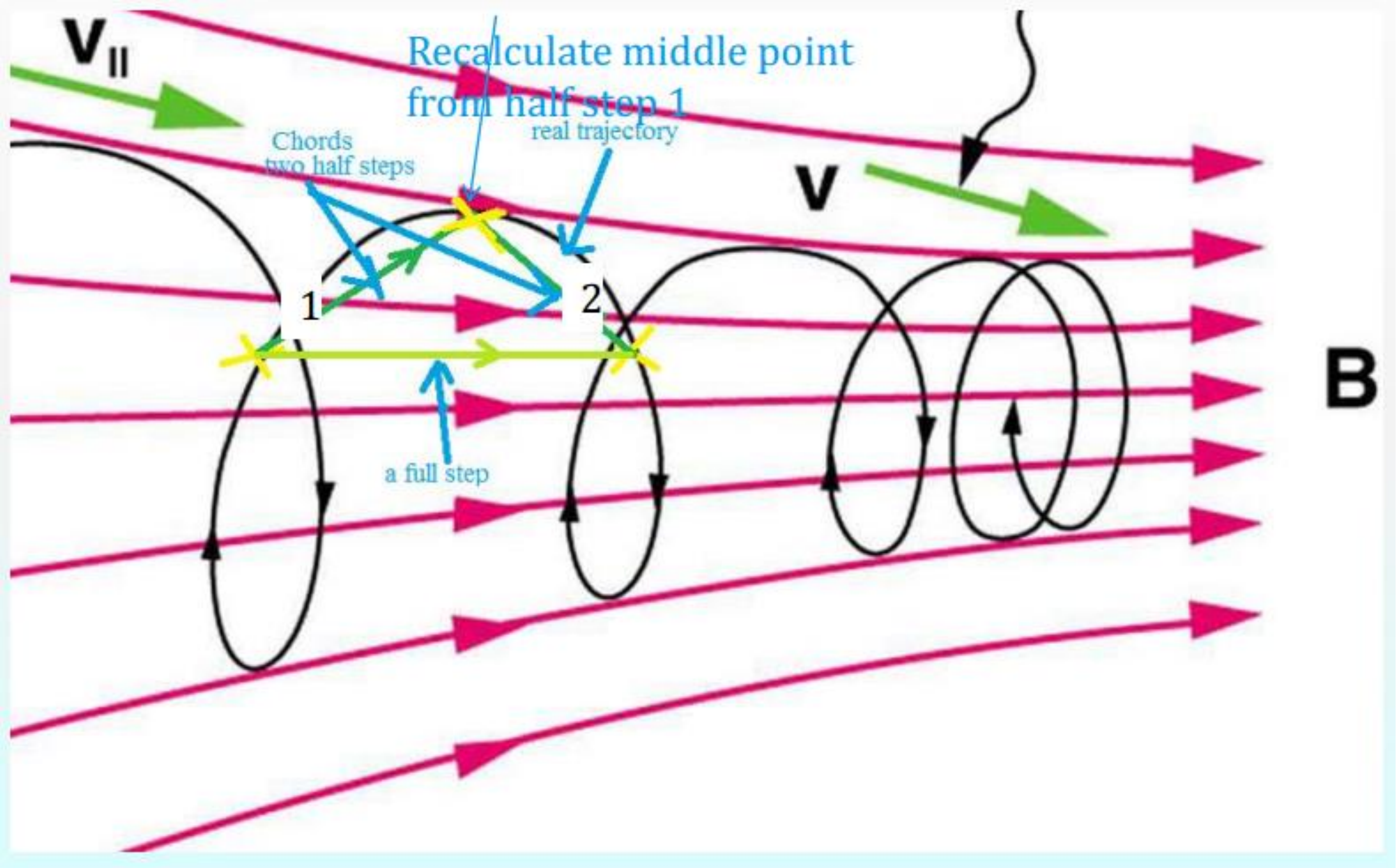
**benchmarked with NTST a drift chamber Geant4 application*

Thank you!

QUESTIONS?

Acknowledgement:
Google Summer of Code 2014
CERN

Particle tracks & numerical methods



Recalculate middle point from half step 1

Why does a virtual function call slow down the execution of the program - and how does the CPU handle it ?

- Constructor of an object that contain virtual function must initialize the vptr table
- run-time method binding: results few extra instructions every time virtual method is called as compared to non-virtual method
- Virtual function can not be inlined
- Current C++ compile are unable to optimize virtual function call(it prevents instruction scheduling, data flow analysis, etc)

Why does a virtual function call slow down the execution of the program - and how does the CPU handle it ?

- Virtual function dispatch can cause an unpreditable branch (branch cache can avoid this problem). Modern microprocessors tend to have long pipelines so that the misprediction delay is between 10 and 20 clock cycles.