Introduction to C++, online slides

What are we doing today?

Today we are going to Live program a full particle simulation for an proton (or antiproton) in a penning trap

Introduce you to compiler explorer

- https://compiler-explorer.com/

Get comfortable with classes / objects

Start to get an eye for optimisation

C++ features

- Compiled language: Fast!
- Object orientated
- Backwards compatible to C
- Things to remember:
  - White space doesn't matter
    - But if you use indentation nicely it will be more readable
  - Because whitespace doesn't matter ';' is used for the end of line

If you get the basics with C++, every other language gets easier

Data types

Today we wont need many data types

Assumptions for today: We are writing code to run on a 64bit PC

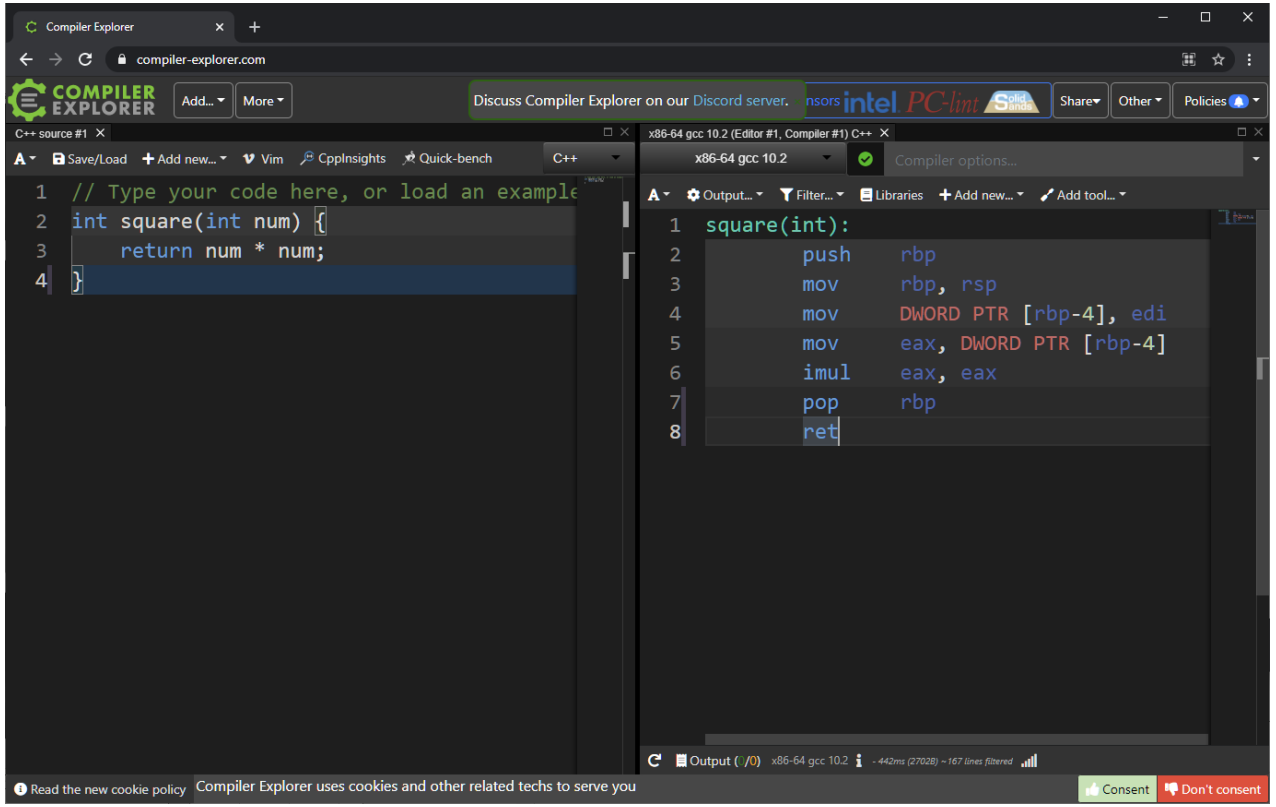
int

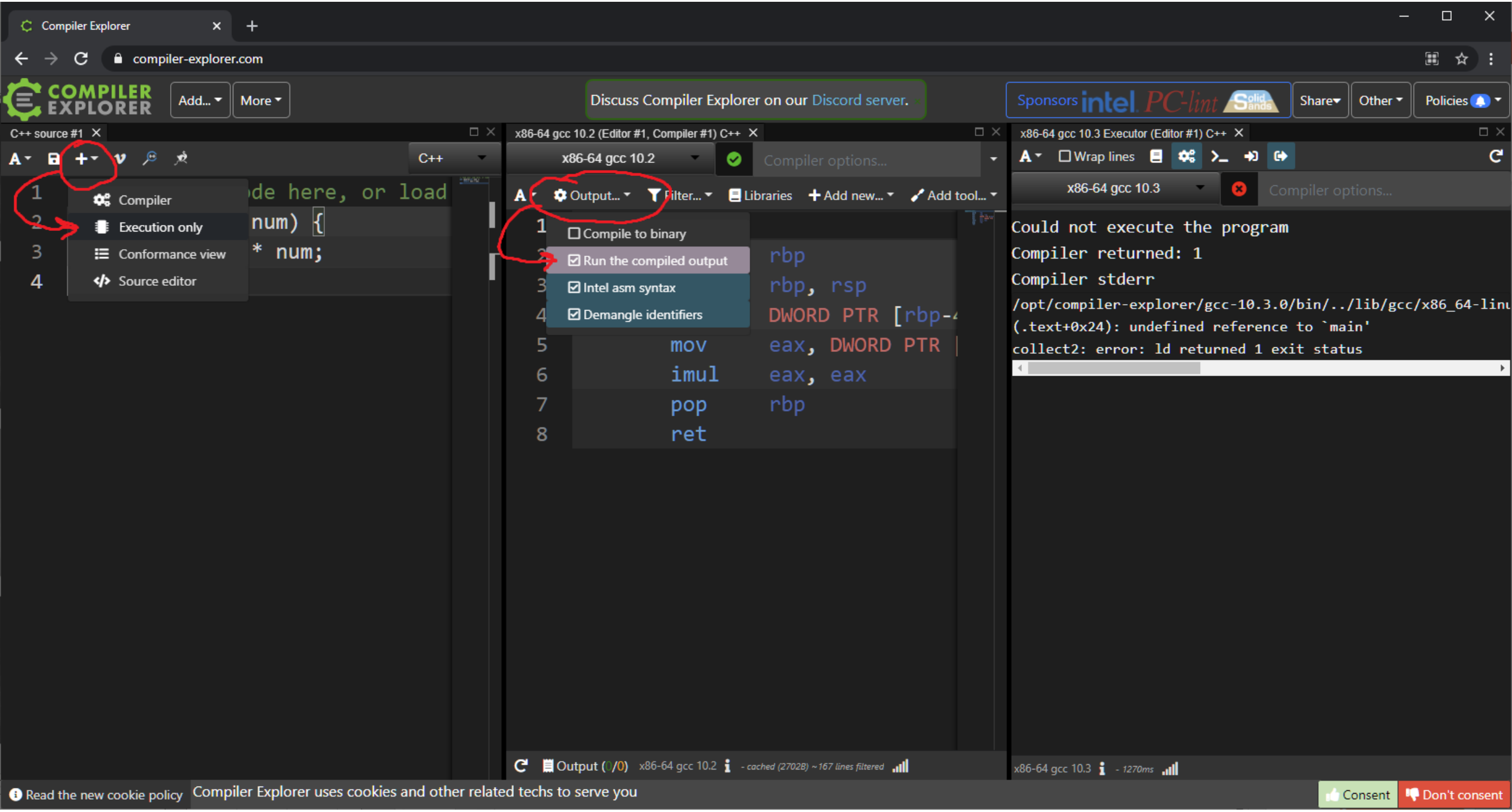
- Typically 32 bit, first bit used for sign
  - Add, subtract math operations are fast!

double

- 64 bit floating point (not a whole number)
  - Very precise (around 17 digits of precision)
  - Maths operations slower than integer, imagine how complex adding two exponential numbers together actually is

Setup compiler-explorer.com




Compiler Explorer
compiler-explorer.com
COMPILER EXPLORER
Add...
More
Discuss Compiler Explorer on our Discord server.
Sponsors intel PC-lint
Share
Other
Policies
C++ source #1
C++
Compiler
Execution only
Conformance view
Source editor
x86-64 gcc 10.2 (Editor #1, Compiler #1) C++
x86-64 gcc 10.2
Compiler options...
Output...
Filter...
Libraries
Add new...
Add tool...
Compile to binary
Run the compiled output
Intel asm syntax
Demangle identifiers
rbp
rbp, rsp
DWORD PTR [rbp-
mov eax, DWORD PTR
imul eax, eax
pop rbp
ret
Output (0/0)
x86-64 gcc 10.2
- cached (2702B) ~167 lines filtered
x86-64 gcc 10.3 Executor (Editor #1) C++
Wrap lines
x86-64 gcc 10.3
Compiler options...
Could not execute the program
Compiler returned: 1
Compiler stderr
/opt/compiler-explorer/gcc-10.3.0/bin/../lib/gcc/x86_64-linu
(.text+0x24): undefined reference to `main'
collect2: error: ld returned 1 exit status
x86-64 gcc 10.3
- 1270ms
Read the new cookie policy
Compiler Explorer uses cookies and other related techs to serve you
Consent
Don't consent

Hello World! https://godbolt.org/z/5bjn4Gncs

```
int main()
{
    int number = 0;
    number = number + 5;
    number*=2;
    for (int i=0; i<10; i++)
    {
        number += i;
    }
    return number;
}
```

Lets make a new data type with a class!

```
class Vector3
{
    public:
    double x;
    double y;
    double z;
    Vector3()
    {
        x = 0;
        y = 0;
        z = 0;
    }
```

Setters and mathematical operators

```
    Vector3(double _x, double _y, double _z)
    {
        x = _x;
        y = _y;
        z = _z;
    }
    Vector3& operator+=(const Vector3& rhs)
    {
        x+=rhs.x;
        y+=rhs.y;
        z+=rhs.z;
        return *this;
    }
};
```

What's this look like if I used python?

```
class Vector3:
    def __init__(self, _x = 0., _y = 0., _z= 0.):
        self.x = _x
        self.y = _y
        self.z = _z
    def __iadd__(self, other):
        self.x += other.x
        self.y += other.y
        self.z += other.z
        return self
```

https://godbolt.org/z/sE3P1nfaq

```
int main()
{
    Vector3 a{1,2,3};
    return sizeof(a);
    Vector3 b{4,5,6};
    b+=a;
    return b.x;
}
```

Print statements

https://godbolt.org/z/YvEPqGdvE

```
int main()
{
    Vector3 v(1.,3.,4.);
    std::cout<< v.x <<"\t"<<v.y<<"\t"<<"\n";
    v+=Vector3(6,7,8);
    std::cout<< v.x <<"\t"<<v.y<<"\t"<<"\n";
    std::cout<<sizeof(v.x) << "\t"<<sizeof(v)<<"\n";
}
```

Lets make a basic particle

```
class Particle
{
public:
    Vector3 position;
    Vector3 velocity;
    double charge;
    double mass;
public:
    Particle(double c, double m):
       charge(c), mass(m)
    {

    }
    void Print()
    {
        std::cout<<"Charge: "<< charge<< " Mass: "<<mass<<"\n";
    }
};
```

What's this look like if I used python?

```
class Particle:
    def __init__(self, c, m):
        self.position = Vector3()
        self.velocity = Vector3()
        self.charge = c
        self.mass = m
    def Print(self):
        print("Charge: "+str(self.charge) + " Mass:
" + str(self.mass))
```

https://godbolt.org/z/cWGndfc6n

```
int main()
{
    Particle electron(0.1,0.5);
    Particle proton(12,34);
    std::cout<<"electron:";
    electron.Print();
    return 0;
}
```

Lets make a parent class

```
class PhysicsProcess
{
    public:
    virtual Vector3 Force(Particle p) = 0;
};
```

Gravity is easy!

```
class Gravity: public PhysicsProcess
{
private:
    double g = -9.81;
public:
    Gravity()
    {

    }
    Vector3 Force(Particle p)
    {
        // F = m * g
        return Vector3( 0, p.mass*g, 0 );
    }
};
```

What's this look like if I used python?

```
class Gravity:
    def __init__(self):
        self.g=-9.81
    def Force(self,p):
        return Vector3(0.0, p.mass* self.g, 0.0)
```

https://godbolt.org/z/s8G4GTqb3

```
//Test gravity force
int main()
{
   Particle electron(-1.60217646E-19, 9.1093837015E-31);
   Particle proton(  +1.60217646E-19, 1.67262192369E-27);
   Gravity g;
   Vector3 F=g.Force(proton);
   std::cout<< F.x <<"\t"<<F.y<<"\t"<<"\n";
   // F = m * a;
   // a = F / m;
   std::cout << F.y / proton.mass << "\n";
}
```

Time stepping class

```
#include <vector>
class TimeStepper
{
private:
    std::vector<PhysicsProcess*> physics_list;
    Particle p;
    double dt;
public:
    TimeStepper(double time_step_size, double charge, double mass):
       p(charge,mass)
    {
       dt = time_step_size;
    }
    void AddProcess(PhysicsProcess* process)
    {
       physics_list.push_back(process);
    }
    void Setup(Vector3 position, Vector3 Velocity)
    {
        p.position = position;
        p.velocity = Velocity;
    }
```

Making a timestep in the simulation

```
void Step()
{
   Vector3 F(0,0,0);
   for (int i =0 ; i< physics_list.size(); i++)
   {
       F += physics_list.at(i)->Force(p);
   }
   //F = m * a
   //F = m * v * dt
   //dv = F / m
   Vector3 dv(
       dt * F.x / p.mass,
       dt * F.y / p.mass,
       dt * F.z / p.mass
       );
   p.velocity += dv;

   //x = v * dt
   p.position += Vector3(
       p.velocity.x * dt,
       p.velocity.y * dt,
       p.velocity.z * dt
       );
}
```

```
    void Print()
    {
        std::cout<<"[ "<<p.position.x<<" "<<p.positi
on.y<<" "<<p.position.z<<" ]\n";
    }
```

What's this look like if I used python?

```
class TimeStepper:
    def __init__(self,dt,c,m):
        self.physics_list=[]
        self.p=Particle(c,m)
        self.dt=dt
    def AddProcess(self,p):
        self.physics_list.append(p)
    def Setup(self,position,velocity):
        self.p.position.x=position.x
        self.p.position.y=position.y
        self.p.position.z=position.z
        self.p.velocity.x=velocity.x
        self.p.velocity.y=velocity.y
        self.p.velocity.z=velocity.z
    def Step(self):
        f = Vector3(0.,0.,0.)
        for process in self.physics_list:
            f+=process.Force(self.p)
        dv = Vector3(f.x / self.p.mass, f.y / self.p.mass, f.z / self.p.mass)
        self.p.velocity += dv
        self.p.position += Vector3( self.p.velocity.x * self.dt , self.p.velocity.y * self.dt, self.p.velocity.z * self.dt )
    def Print(self):
        print( "[ " + str(self.p.position.x) + " " + str(self.p.position.y) + " " + str(self.p.position.z) + "]")
```

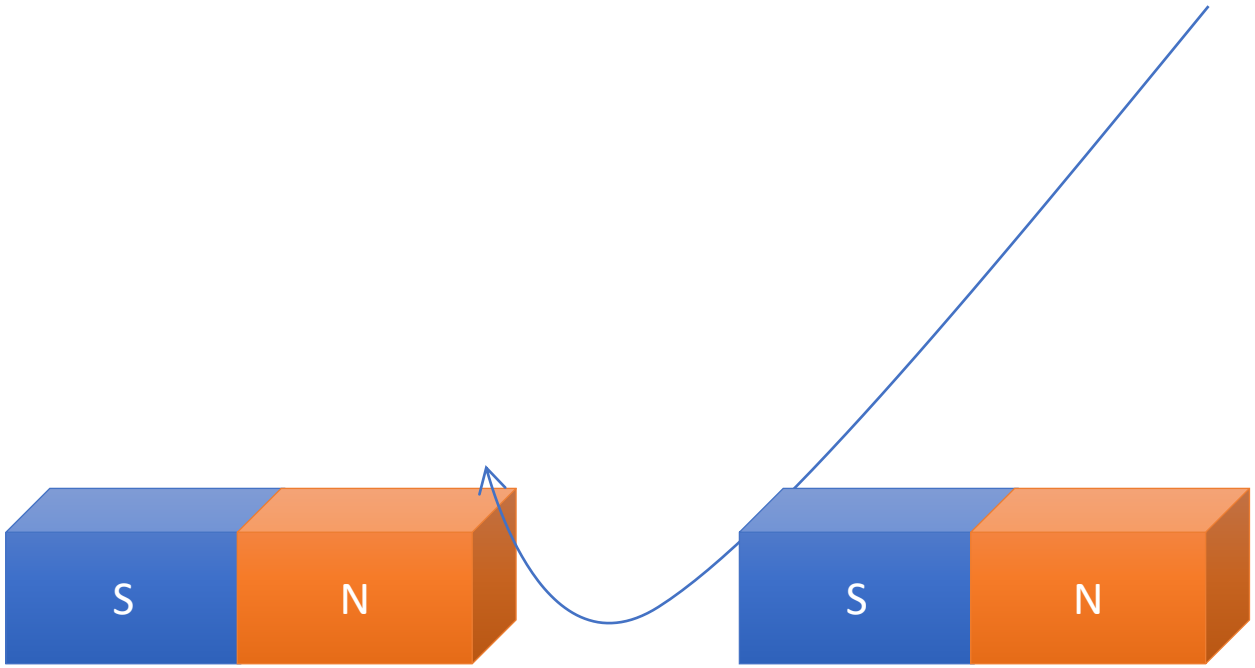
https://godbolt.org/z/xEP6G9qsd

```
int main()
{
   TimeStepper ElectronStepper(0.001,-1.60217646E-19, 9.1093837015E-31);

   ElectronStepper.Setup(Vector3(0,1,2), Vector3(0,4,0));
   //TimeStepper ProtonStepper(0.00001,  +1.60217646E-19, 1.67262192369E-27);
   ElectronStepper.AddProcess(new Gravity());
   for (int i=0; i<100000; i++)
   {
       ElectronStepper.Step();
       if (i%1000 == 0)
          ElectronStepper.Print();
   }
}
```

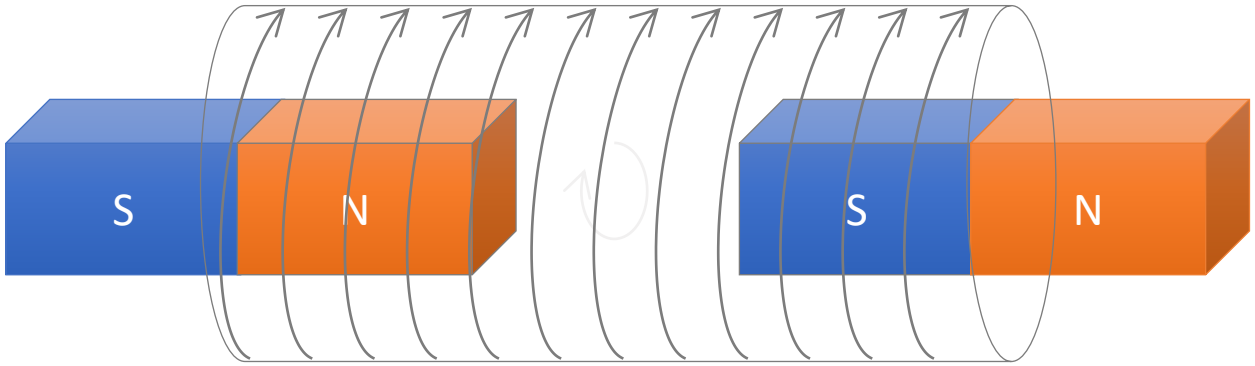
Penning trap

Penning trap

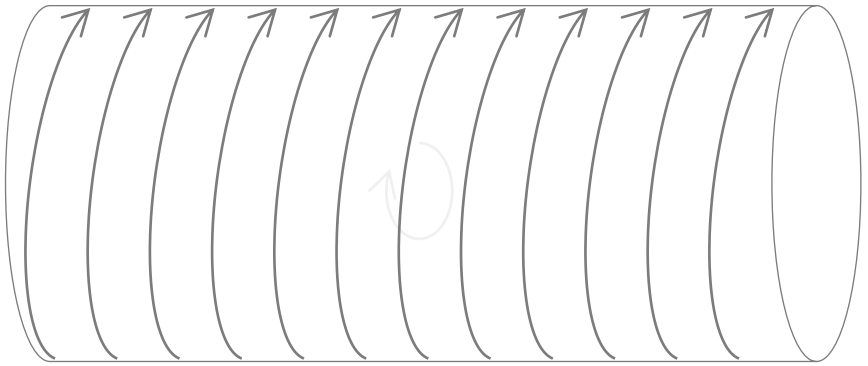


Penning trap

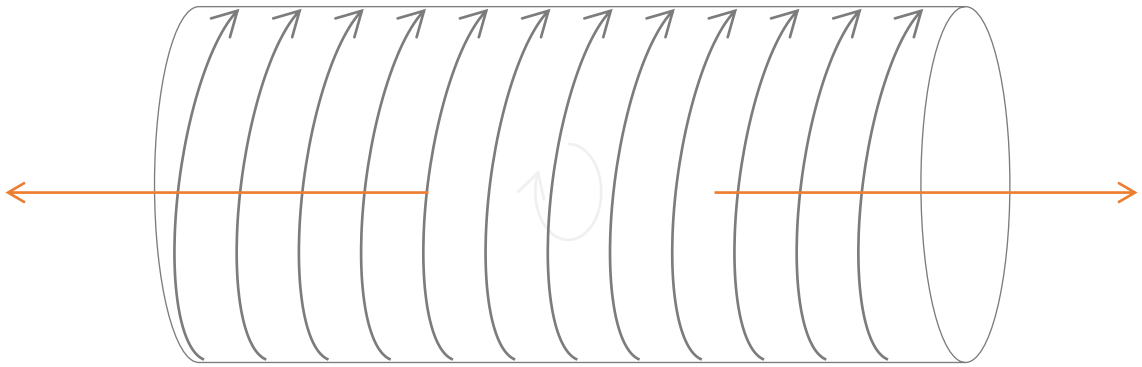
Penning trap



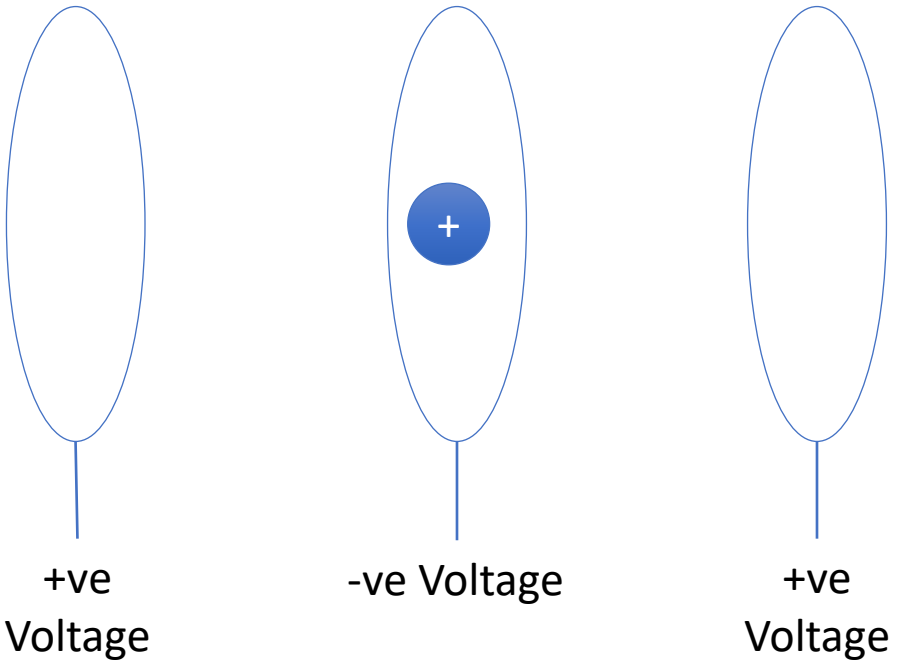
Penning trap



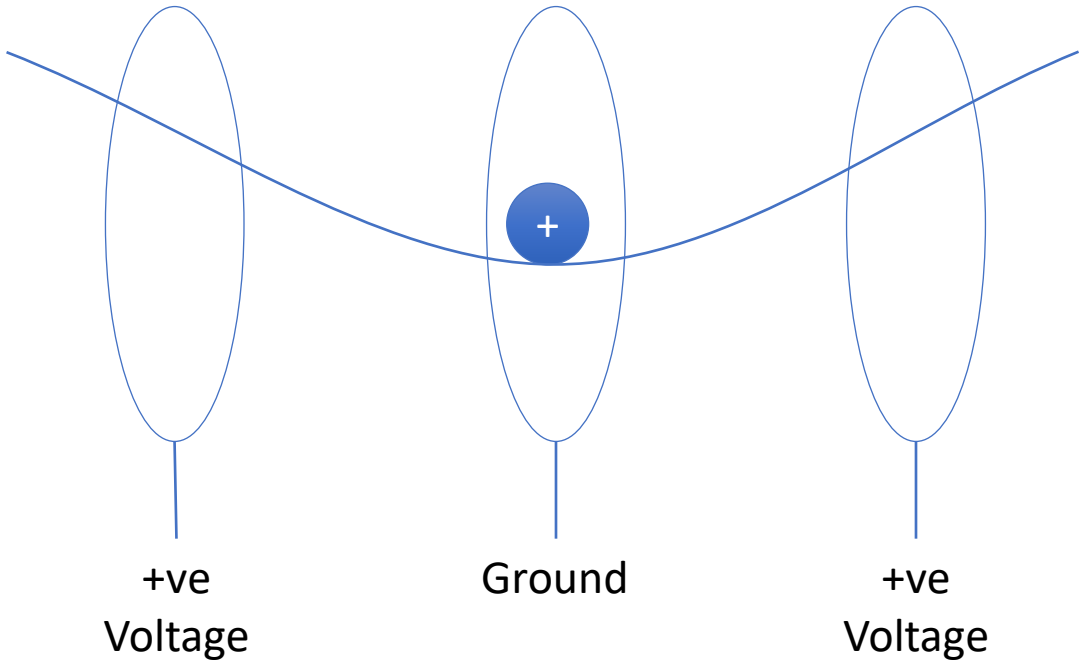
Penning trap



Penning trap



Penning trap



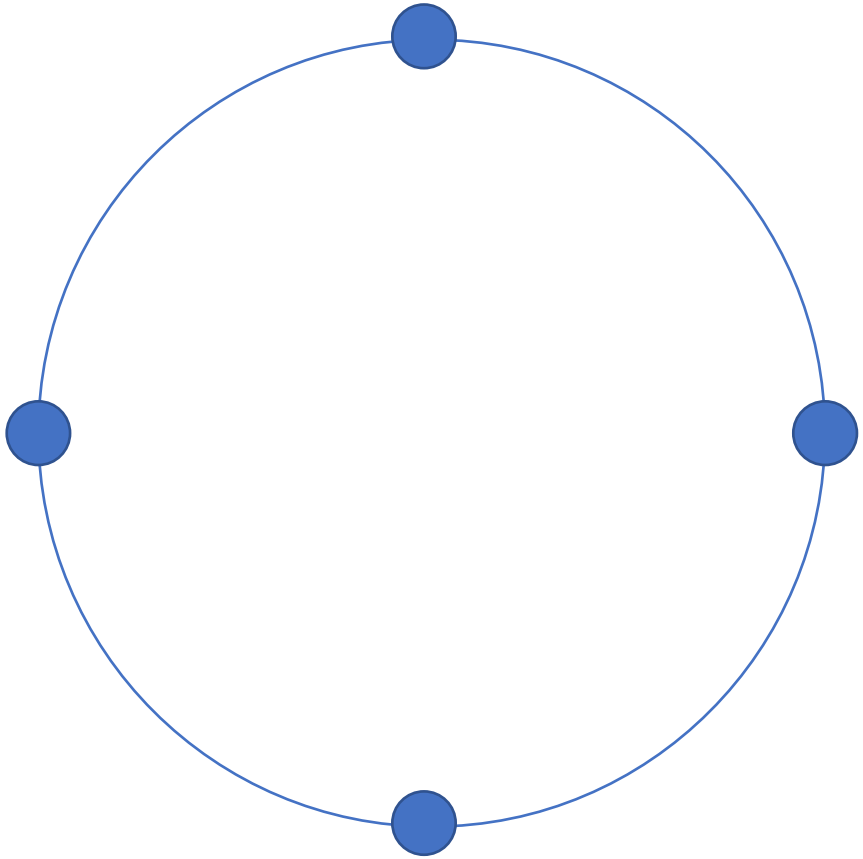
Magnetic field is easy…

```
class UniformB: public PhysicsProcess
{
    private:
    Vector3 B;
    public:
    UniformB(Vector3 Field):
        B( Field.x,
            Field.y,
            Field.z )
    {
    }
    Vector3 Force(Particle p)
    {
        //F = qV x B
        return Vector3(
            p.charge*( p.velocity.y*B.z - p.velocity.z*B.y),
            p.charge*( p.velocity.z*B.x - p.velocity.x*B.z),
            p.charge*( p.velocity.x*B.y - p.velocity.y*B.x)
        );
    }
};
```

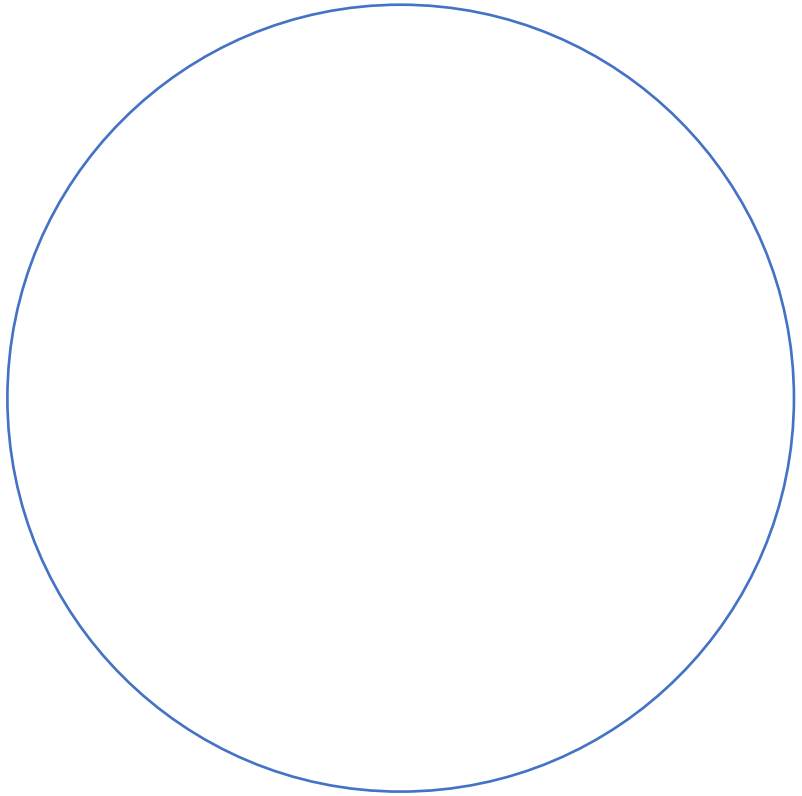
What's this look like if I used python?

```
class UniformB:
    def __init__(self, B):
        self.B=Vector3(B.x,B.y,B.z)
    def Force(self,p):
        return Vector3( \
                p.charge* ( p.velocity.y*self.B.z -
 p.velocity.z*self.B.y),\
                p.charge* ( p.velocity.z*self.B.x -
 p.velocity.x*self.B.z), \
                p.charge* ( p.velocity.x*self.B.y -
 p.velocity.y*self.B.x) )
```

Electrodes don’t have such a simple field (outside of cylindrical coordinates)



Electrodes don’t have such a simple field (outside of cylindrical coordinates)



Lets add some electrostatics!

```
class PointCharge: public PhysicsProcess
{
    private:
    double Q;
    Vector3 origin;
    public:
    PointCharge(int Charge, Vector3 position )
    {
        //Q = C*V
        Q = 1.60217646E-19*Charge;
        origin.x = position.x;
        origin.y = position.y;
        origin.z = position.z;
    }
    Vector3 Force(Particle p)
    {
        // F = (Ke * Q * q / r^3) * R
        Vector3 R(
            p.position.x - origin.x,
            p.position.y - origin.y,
            p.position.z - origin.z
            );
        double r =
            std::sqrt(R.x*R.x + R.y*R.y + R.z*R.z
);
        double r3 = r*r*r;
        double F = -8.988E9*Q*p.charge/r3;
        return Vector3(F*R.x,F*R.y,F*R.z);
    }
};
```

What's this look like if I used python?

```
import math
class PointCharge:
    def __init__(self, charge, position):
        self.Q = 1.60217646E-19*charge
        self.origin=Vector3(position.x, position.y, position.z)
    def Force(self, p):
        R = Vector3( p.position.x - self.origin.x, p.position.y -
 self.origin.y, p.position.z - self.origin.z )
        r = math.sqrt(R.x*R.x + R.y*R.y + R.z*R.z)
        r3 = r*r*r
        F = -8.988E9*self.Q*p.charge/r3
        return Vector3(F*R.x, F*R.y, F*R.z)
```

From point charge to electrode

```
class electrodePair: public PhysicsProcess
{
    private:
    std::vector<PointCharge> points;
    public:
    electrodePair(int Charge, double radius, double z)
    {
        for (double dz: {-z, z})
        {
            for (int phi = 0; phi< 360; phi+=10 )
            {
                double dx=radius*std::sin((double)phi * M_PI / 180);
                double dy=radius*std::cos((double)phi * M_PI / 180);
                //std::cout<<dx<<"\t"<<dy<<std::endl;
                points.push_back(
                    PointCharge(
                        Charge,
                        Vector3(dx,dy,dz)
                        )
                );
            }
        }
    }
    Vector3 Force(Particle p)
    {
        Vector3 Force = {0.,0.,0.};
        for (PointCharge& point: points)
        {
            Force += point.Force(p);
        }
        //std::cout<<"F"<< Force.x<<"\t"<<Force.y<<std::endl;
        return Force;
    }
};
```

What's this look like if I used python?

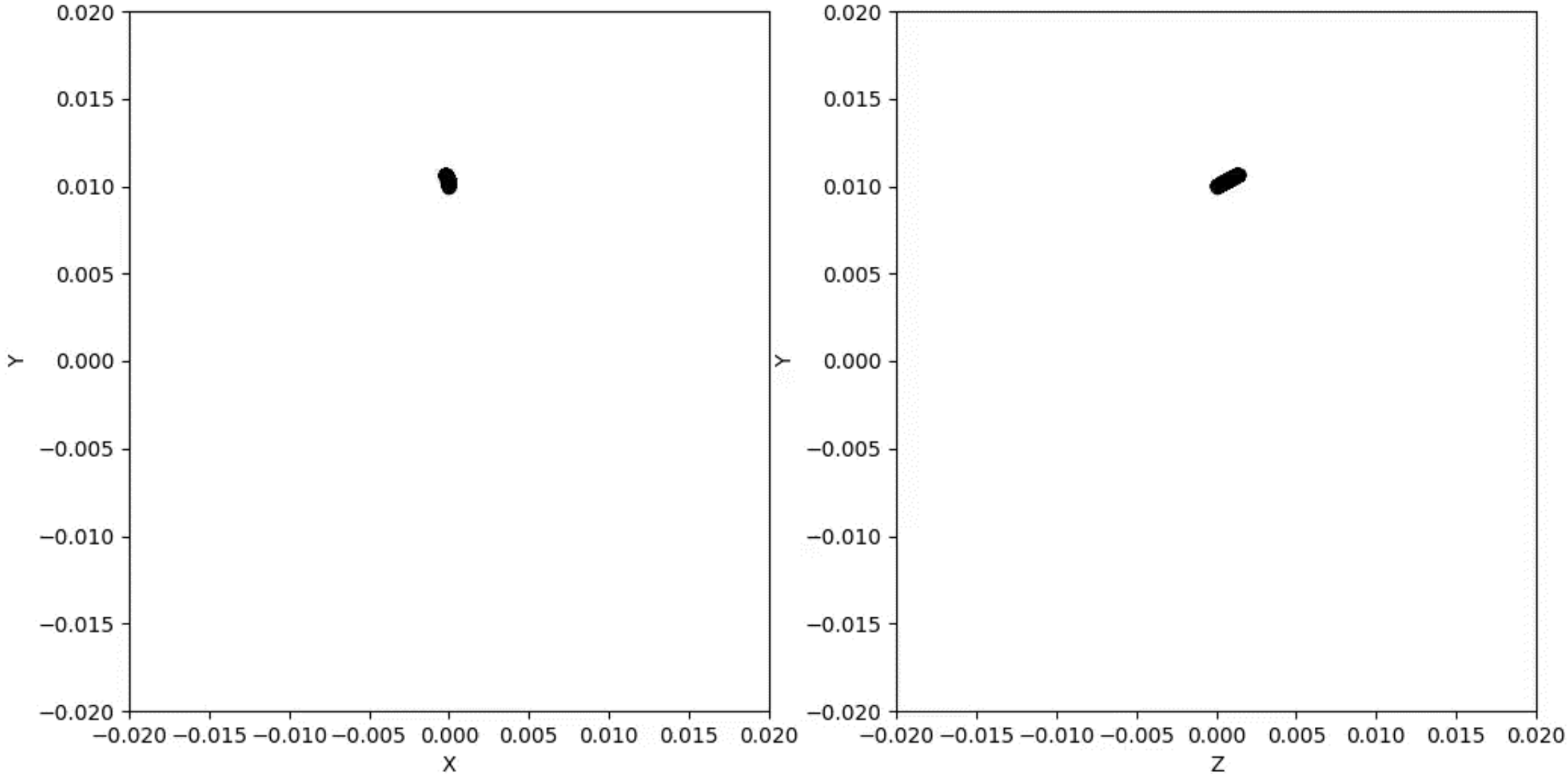
```
class electrodePair:
    def __init__(self, charge, radius, z):
        self.points=[]
        for dz in {-z, z}:
            for phi in range(0,360,10):
                dx = radius* math.sin(phi * math.pi / 180)
                dy = radius* math.cos(phi * math.pi / 180)
                self.points.append(PointCharge(charge, Vector3(dx,dy,dz)))
    def Force(self,p):
        F = Vector3()
        for point in self.points:
            F += point.Force(p)
        return F
```

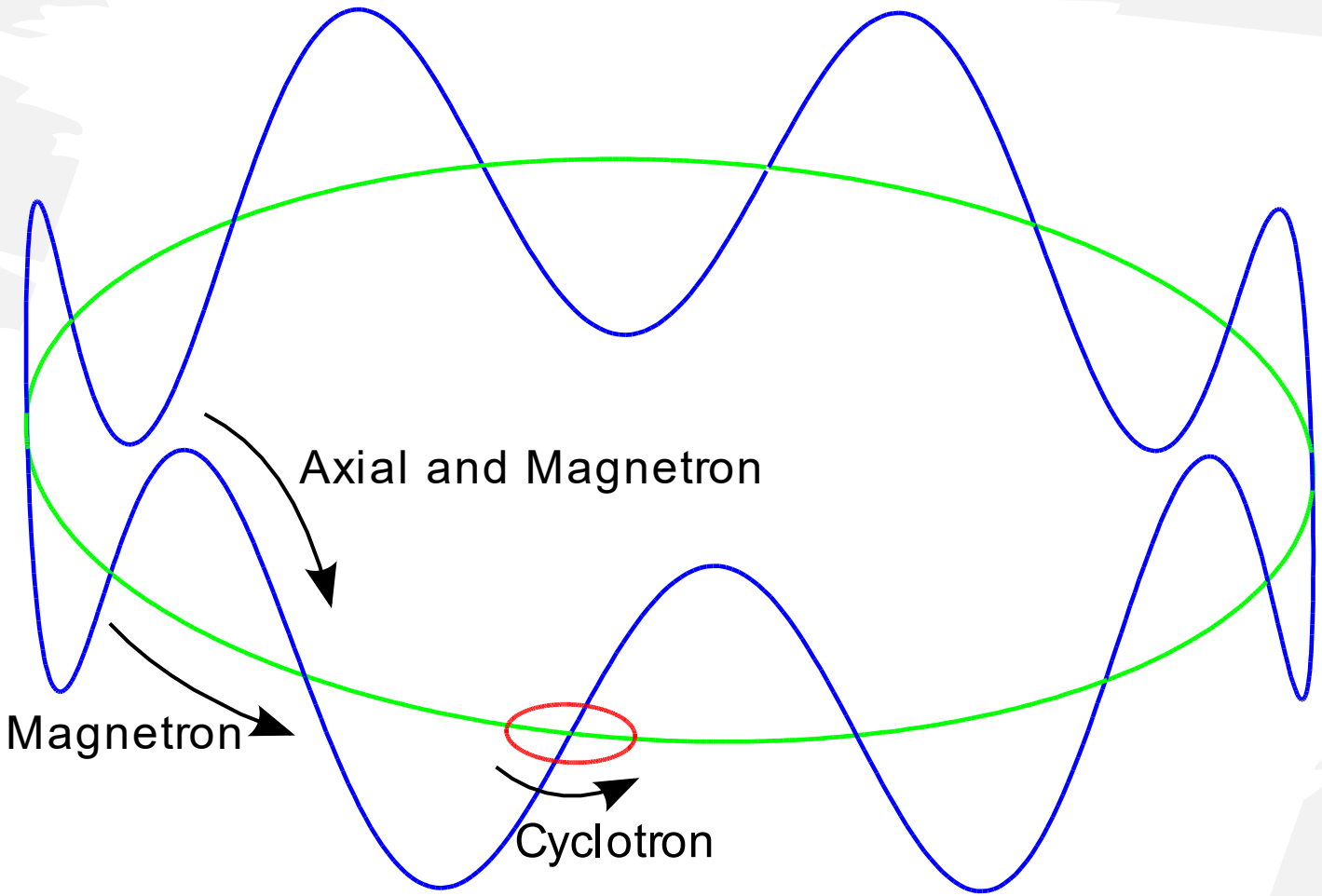
https://godbolt.org/z/5M4oroPeb

```
int main()
{
   TimeStepper ProtonStepper(1E-9,-1.60217646E-19, 1.67262192369E-27);
   ProtonStepper.Setup(Vector3(0,0.01,0), Vector3(0,10,20));
   ProtonStepper.AddProcess(new Gravity());
   ProtonStepper.AddProcess(new electrodePair(10,0.02,0.02));
   ProtonStepper.AddProcess(new UniformB(Vector3(0,0,0.0001)));
   std::cout<<"[ x y z ]"<<"\n";
   for (int i=0; i<1E6; i++)
   {
       ProtonStepper.Step();
       if (i%20000 == 0)
          ProtonStepper.Print();
   }
   ProtonStepper.Print();
}
```

What's this look like if I used python?

```
ProtonStepper = TimeStepper(1E-9,-1.60217646E-
19, 1.67262192369E-27)
ProtonStepper.Setup(Vector3(0.0,0.01,0.0), Vector3(0.0,10.0,2
0.0))
ProtonStepper.AddProcess(Gravity())
ProtonStepper.AddProcess(electrodePair(10.0,0.02,0.02))
ProtonStepper.AddProcess(UniformB(Vector3(0.0,0.0,0.0001)))
for i in range(int(5E7)):
    ProtonStepper.Step()
    if (i%700 == 0):
        ProtonStepper.Print()
```


0.020
0.015
0.010
0.005
0.000
−0.005
−0.010
−0.015
−0.020
Y
−0.020 −0.015 −0.010 −0.005 0.000 0.005 0.010 0.015 0.020
X
0.020
0.015
0.010
0.005
0.000
−0.005
−0.010
−0.015
−0.020
Y
−0.020 −0.015 −0.010 −0.005 0.000 0.005 0.010 0.015 0.020
Z


Axial and Magnetron
Magnetron
Cyclotron

Checking performance

https://www.quick-bench.com/q/fY3ACStJ3Argjvj5sGheSsKmAYU

```
static void RunTimeStepper(benchmark::State& state) {
  TimeStepper ProtonStepper(1E-9,-1.60217646E-19, 1.67262192369E-27);
  ProtonStepper.Setup(Vector3(0,0.01,0), Vector3(0,10,20));
  ProtonStepper.AddProcess(new Gravity());
  //ProtonStepper.AddProcess(new electrodePair(10,0.02,0.02));
  ProtonStepper.AddProcess(new UniformB(Vector3(0,0,0.0001)));
  for (auto _ : state) {
     ProtonStepper.Step();
  }
  ProtonStepper.Print();
}
// Register the function as a benchmark
BENCHMARK(RunTimeStepper);
```

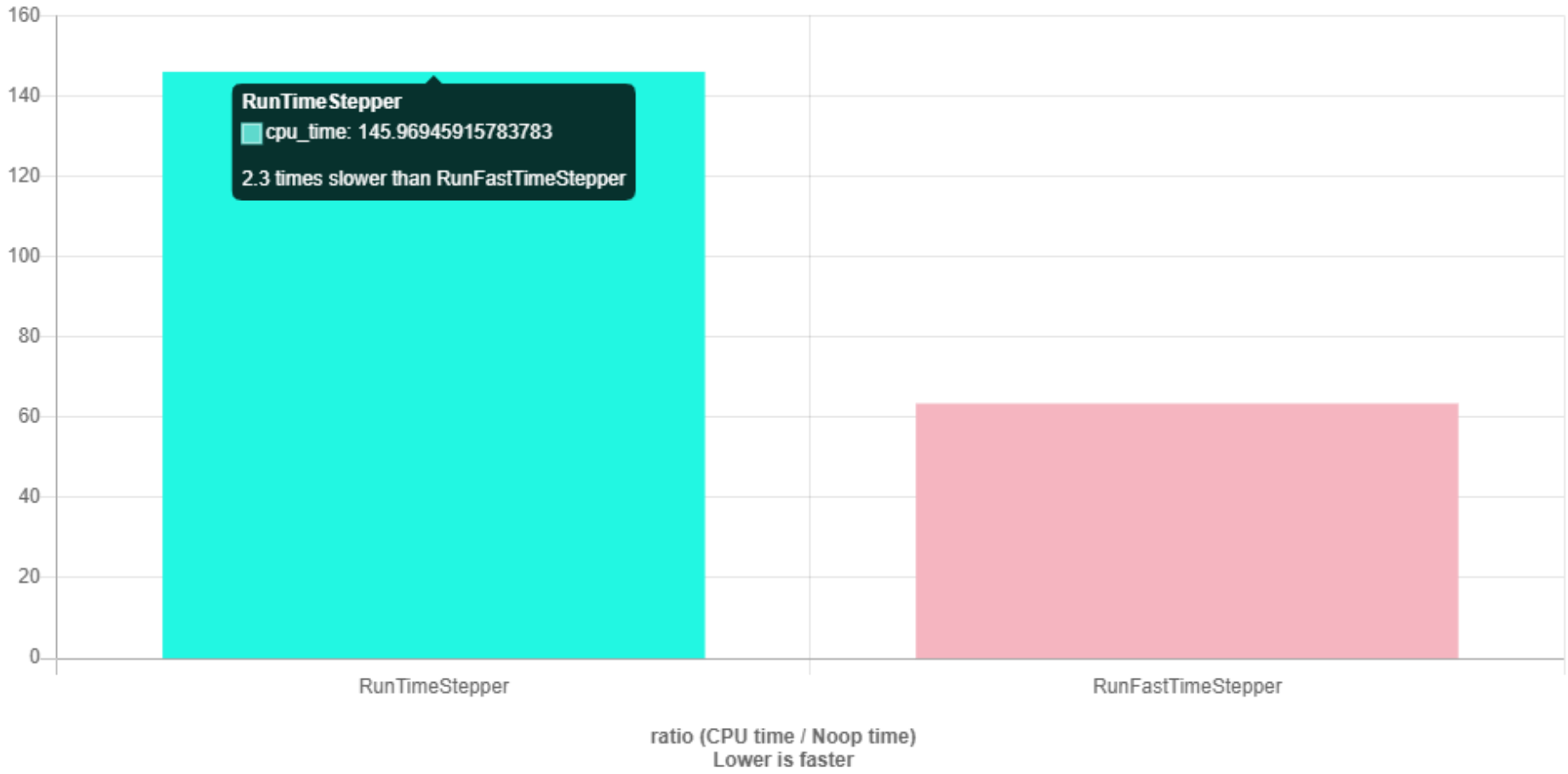
```cpp
#include <vector>
class FastTimeStepper: public Gravity,  public UniformB
{
private:
    Particle p;
    double dt;
public:
    FastTimeStepper(double time_step_size, double charge, double mass):
       p(charge,mass),  UniformB(Vector3(0,0,0.0001))
    {
       dt = time_step_size;
    }
    void Setup(Vector3 position, Vector3 Velocity)
    {
        p.position = position;
        p.velocity = Velocity;
    }
    void Step()
    {
       Vector3 F(0,0,0);
       F += Gravity::Force(p);
       F += UniformB::Force(p);
       //F = m * a
       //F = m * v * dt
       //dv = F / m
       Vector3 dv(
           dt * F.x / p.mass,
           dt * F.y / p.mass,
           dt * F.z / p.mass
           );
       p.velocity += dv;
       //x = v * dt
       p.position += Vector3(
           p.velocity.x * dt,
           p.velocity.y * dt,
           p.velocity.z * dt
           );
    }
    void Print()
    {
        std::cout<<"[ "<<p.position.x<<" "<<p.position.y<<" "<<p.position.z<<" ]\n";
    }
};
```

https://www.quick-bench.com/q/lm154f6_gx0mOWvdPQ44mA6m6mw

```
static void RunFastTimeStepper(benchmark::State& state) {
  FastTimeStepper ProtonStepper(1E-9,-1.60217646E-19, 1.67262192369E-27);
  ProtonStepper.Setup(Vector3(0,0.01,0), Vector3(0,10,20));

  for (auto _ : state) {
     ProtonStepper.Step();
  }
  ProtonStepper.Print();
}
// Register the function as a benchmark
BENCHMARK(RunFastTimeStepper);
```

Inheritance is wonderful!



Conclusion

- Learning programming is just about the practice
- Find a fun hobby project or a problem you want to solve and play
- Google is your best friend

- There is no such thing as a good programmer or a bad programmer, there is just a programmer that know what to google