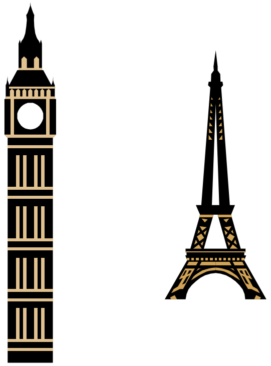
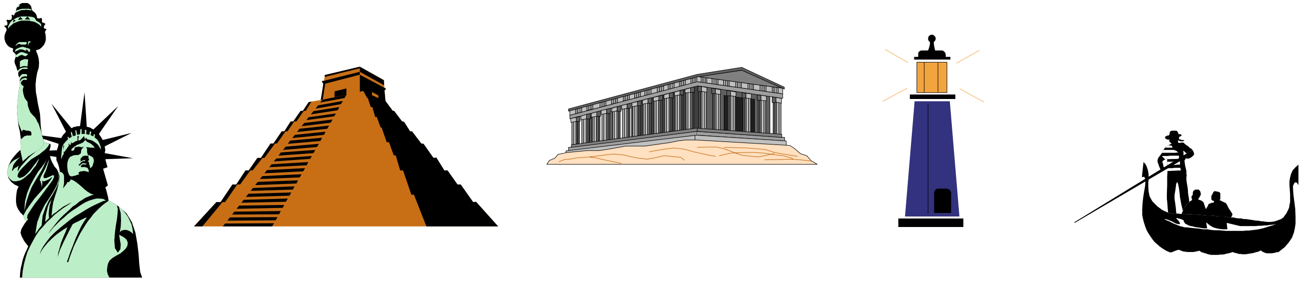
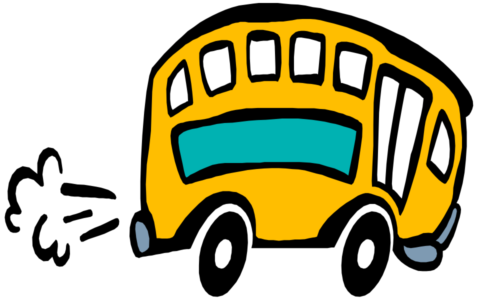




A Scenic tour of C++

Dietrich Liko

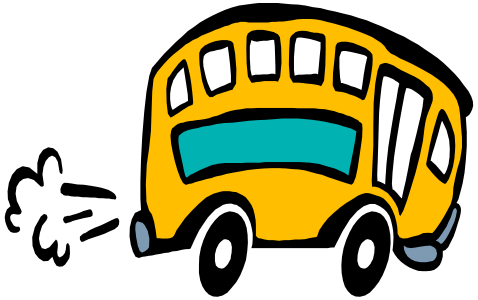
Dietrich Liko



A tour of the world ...

- We will visit many places
- We will stay only short
- You will get an overview
- If you want to know these places better, you will have to visit them yourself afterwards

Dietrich Liko



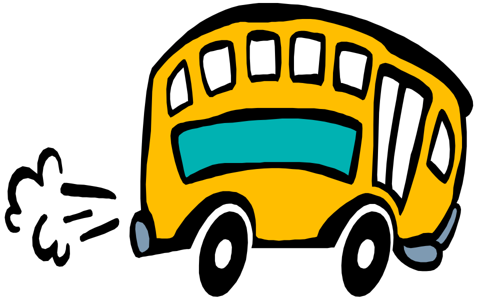
The C++ programming language

- Was created by Bjarne Stroustrup
  - You can visit him on

    `http://www.research.att.com/~bs/homepage.html`
    - You find also an audio file to pronounce his name

- Is better than C
- Supports data-abstraction
- Supports object-oriented programming
- Supports generic programming

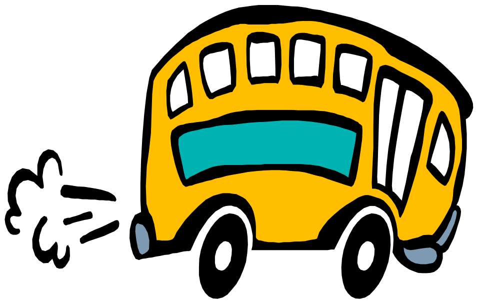
Dietrich Liko



Stop 1 : Hello world

```
#include <iostream>


int main(int argc, const char** argv ) {
    std::cout << “Hello World” << std::endl;
    return 0;
}
```

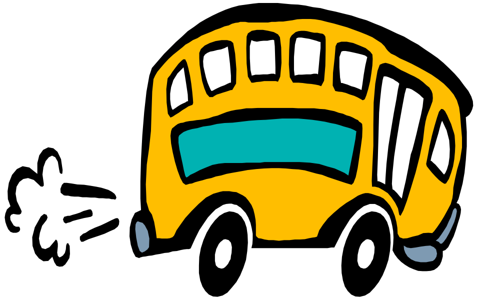
Dietrich Liko



Stop 2: Procedural programming

- This is what we well know from FORTRAN or C

```
#include <cmath>

double do_something(double a) {

    double b = a * 2;
    return std::sqrt(b);

}
```



Build in data types

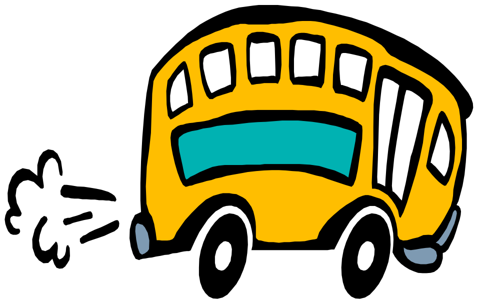
- float
- double
- int
- long
- short
- char

```
int a;
int b = 5;
char c = ‘c’;
char name[] = “Dietrich”;
```

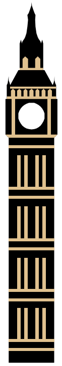
C++ allows to define your own data types. There is a number of prefabricated available like strings and complex numbers

```
string name = “Dietrich”;
```

Better then C



Operators

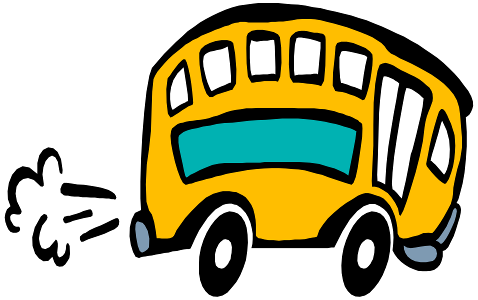


Things in between variables

• assignment	=
• arithmetic	+ - * /
• shortcut	+= -= *= /=
• comparison	== != < > >= <=
• increment	++ --
• more exotic	? & << >>

```
a = a + 1;

++a;
```

```
double a = d > 0 ?  sqrt(d) : sqrt(-d);
```



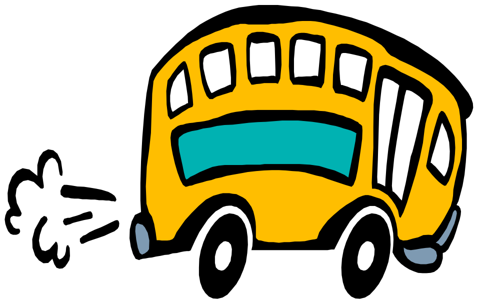
How to pass arguments ?

- Pass by value
  - C

```
void do_it(double b) {
        b = 4.0
}

double a = 5;
do_it(a)
```

```
void do_it(double * b) {
       *b = 4.0
}

double a = 5;
do_it(&a);
```

What happens to “a” ?

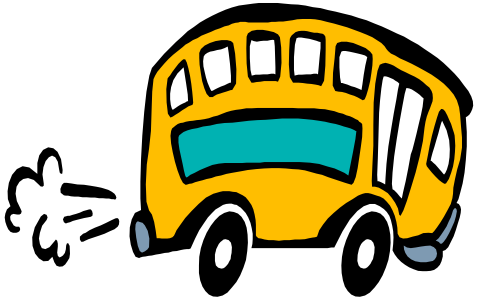


How to pass arguments cont.

- Pass by reference
  - FORTRAN

```
SUBROUTINE DO_IT(B)
REAL B
B = 4.0
 END

PROGRAM TEST
REAL A
A = 5.
CALL DO_IT(A)
 END
```

What happens to “a” ?



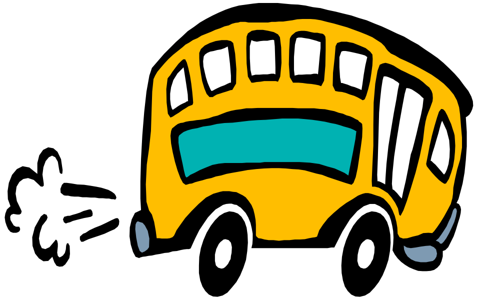
How to pass arguments cont.

- Pass by reference
  - C++

```
void do_it (double b &) {
  b = 4.0;
}
double a = 5;
do_it(a);
```

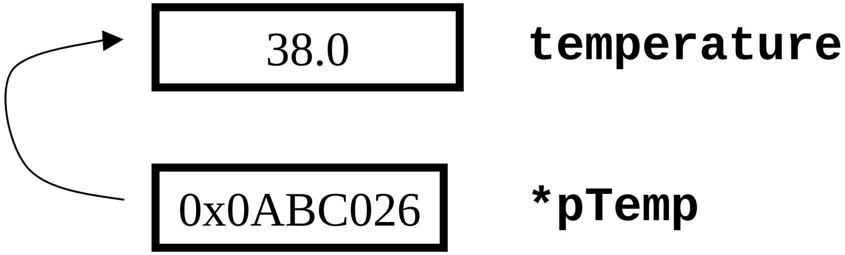
```
void do_it (const double b &) {
  b = 4.0;
}
```

Illegal



Pointers & References

- Pointer to a variable *

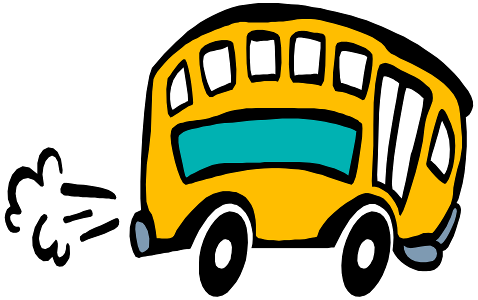


```
double temperature = 38.0;
double * pTemp = &temperature;

*pTemp += 5.0;
```

- Reference to a variable &

```
void do_something(double b &) {
     b += 5.0;
}
```

Better then C



Control structures

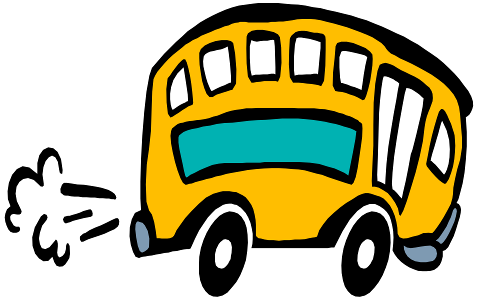
```
if ( a>5 ) {
   ……
} else {
   ……
}
```

```
for (int i=0; i<100; ++i) {
   ……
}
```

```
while( a>5) {
     ……
}
```

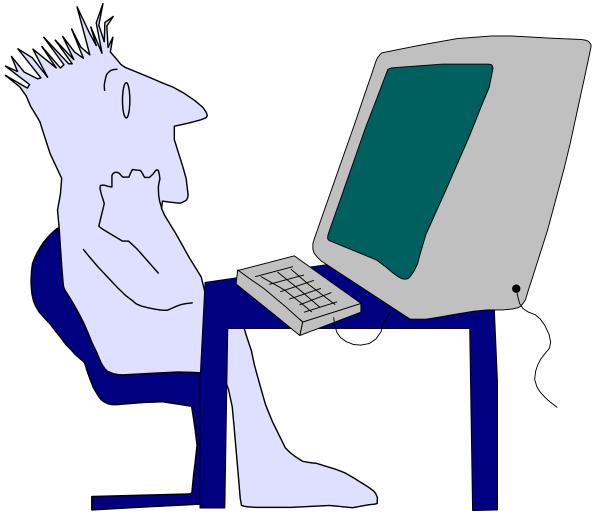
```
switch a {
   case 5 :
      ……
      break;
   case 3 :
      … …
      break;
   default:
      … …
}
```

Dietrich Liko

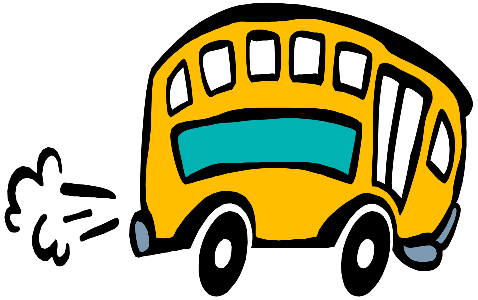


Programming paradigm

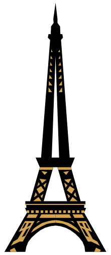
- Decide which procedures you want
- Use the best algorithms you can find



- Functions are used to create order
- Leads to structured programming

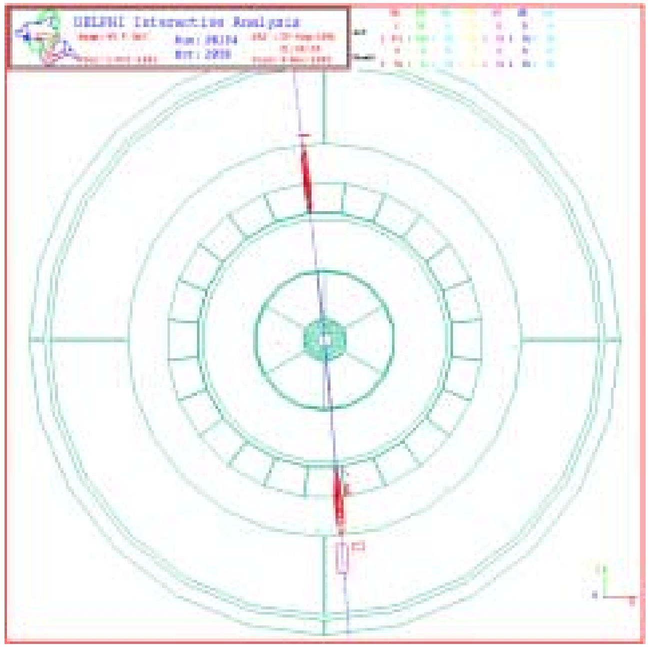


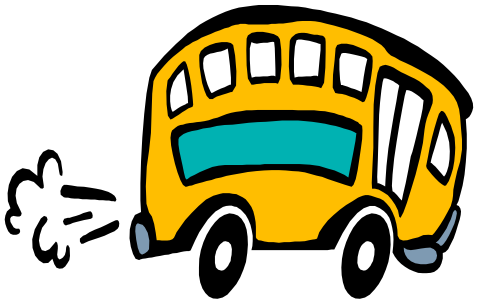
Stop 2: Data abstraction



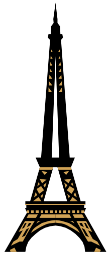
- C++ allows us to make our own data types

```
struct Particle {
        double xpos;
        double ypos;
        double zpos;
        double xmom;
        double ymom;
        double zmom;
        double mass;
}
```





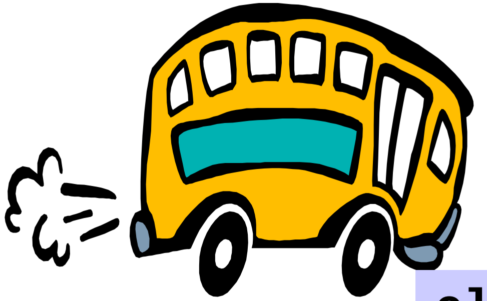
If I want to know the energy ...



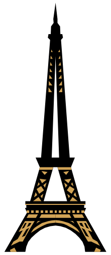
- `energy = sqrt(p.mass*.p.mass + p.xmom*p.xmom + ….)`

But sometimes later ...

```
struct NewParticle {
        double xpos;
        double ypos;
        double zpos;
        double mom;
        double theta;
        double phi;
        double mass;
}
```

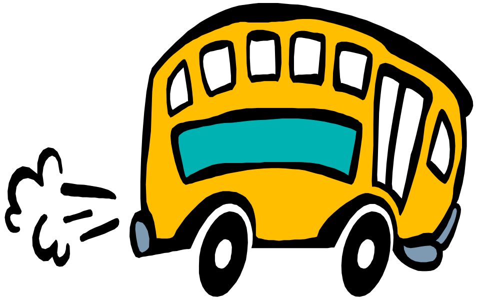


Better use classes

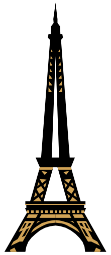


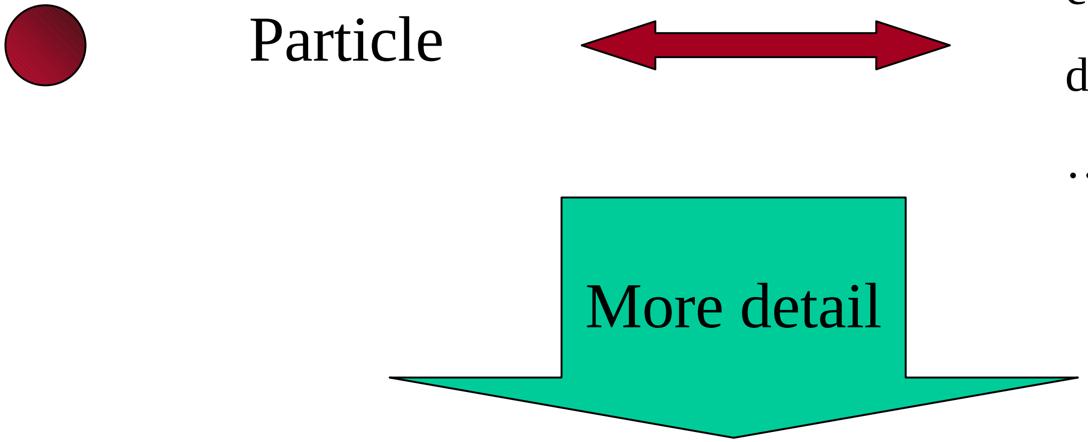
```
class Particle {
public:
       void setMomentum(double x, double y,
                                  double z);
       double energy();
private:
       …whatever I prefer …
}
```

- Only public part visible to outside world
- Obviously Interface design most important aspect
- Dependencies are minimized



Abstraction level - Reality





Particle

boost

collide

decay

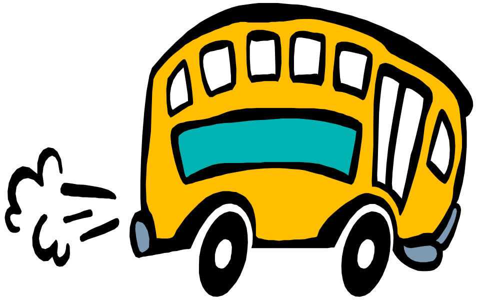
………..

More detail

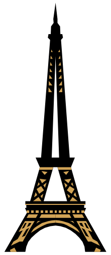
Physical Properties

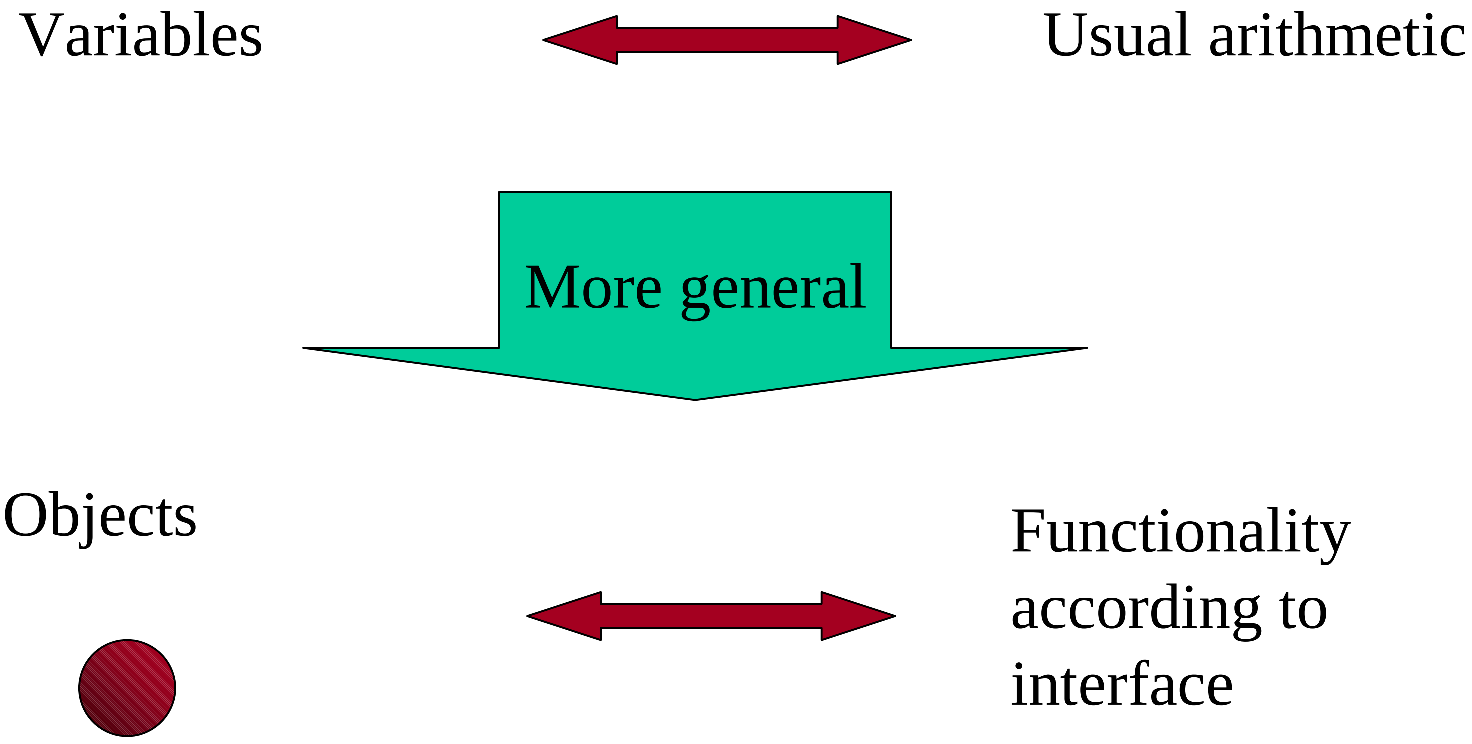
- position
- momentum
- mass

- Apply transformations
- physics laws



Abstraction level - Program





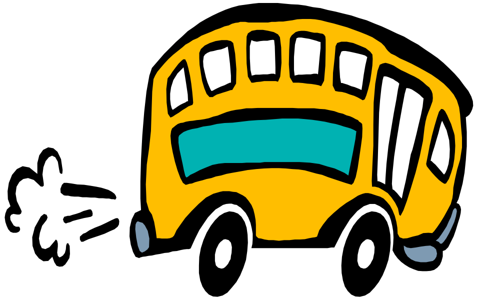
Variables

Usual arithmetic

More general

Objects

Functionality according to interface



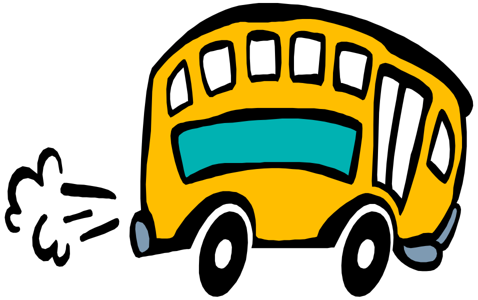
Stop 3: Classes and Objects

- A class is the definition
  - in C++ it is a real data type
- An object is an instance of a class
  - You can create as many instances as you like

```
void do_something(){
      ………
      Particle p;
       ………
}
```

```
Particle  * p = new Particle();
………
delete p;
```

Risk of memory leaks



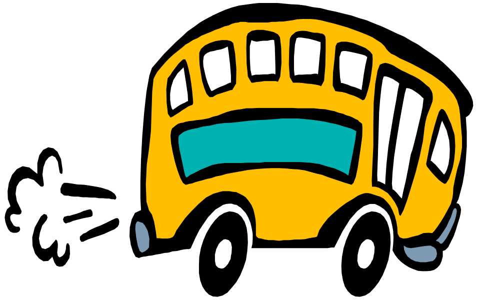
Member Attributes

- Each object has its own set of variables associate to it
- Usually attributes are "private"

Invisible
to the world

- Often a naming convention is used "m_name"
- Attributes define the state of an object
- Static attributes exists only once

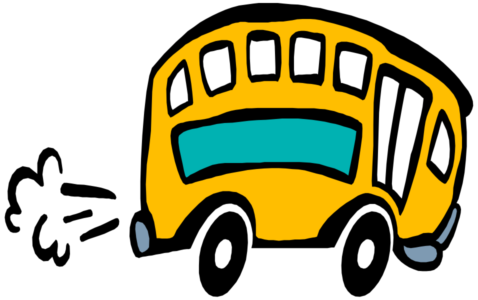
Dietrich Liko



Member Functions

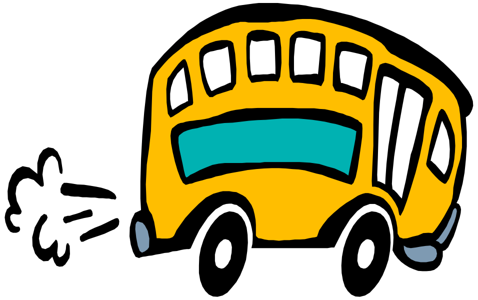
- Also called “Methods”
- Usually public
- Provide functionality
- They act on an instance of a class
- They can change the state of the object

Dietrich Liko



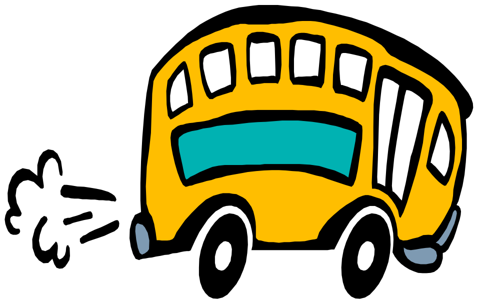
Constructor - Destructor

- Special Member Function
- Particle::Particle (constructor)
  - Called when the object is created
  - defines the initial state
  - allocate resources (open file, open window ...)
  - allocate other objects
- Particle::~Particle (destructor)
  - cleanup
  - delete other objects



More exotic

- Copy constructor
  - Particle newParticle = oldParticle;
- Assignment operator
  - Particle p;
  - p = otherParticle;



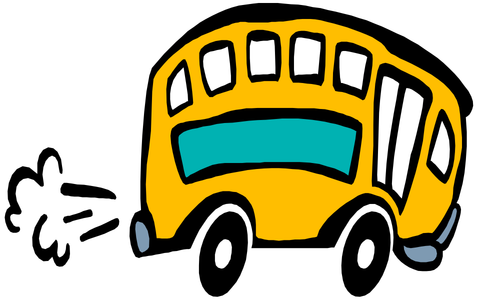
And even more ...

- Complex numbers in C++
- Not build in, but a feature added later on
- Operators can be defined
  - Operator overloading

```
complex a = complex(4.,5.);
complex b = a + 5.;
```

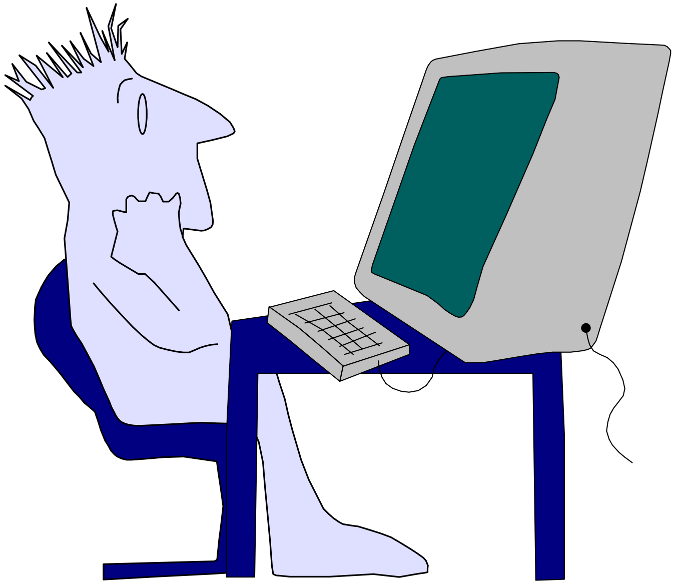
Somewhere it is defined
that this means ...

```
b.re = a.re + 5.;
b.im = a.im;
```



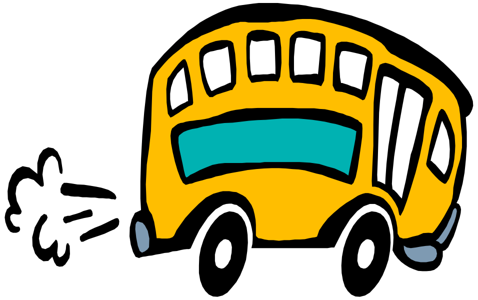
Programming Paradigm

- Decide which types you want
- Provide a full set of operations



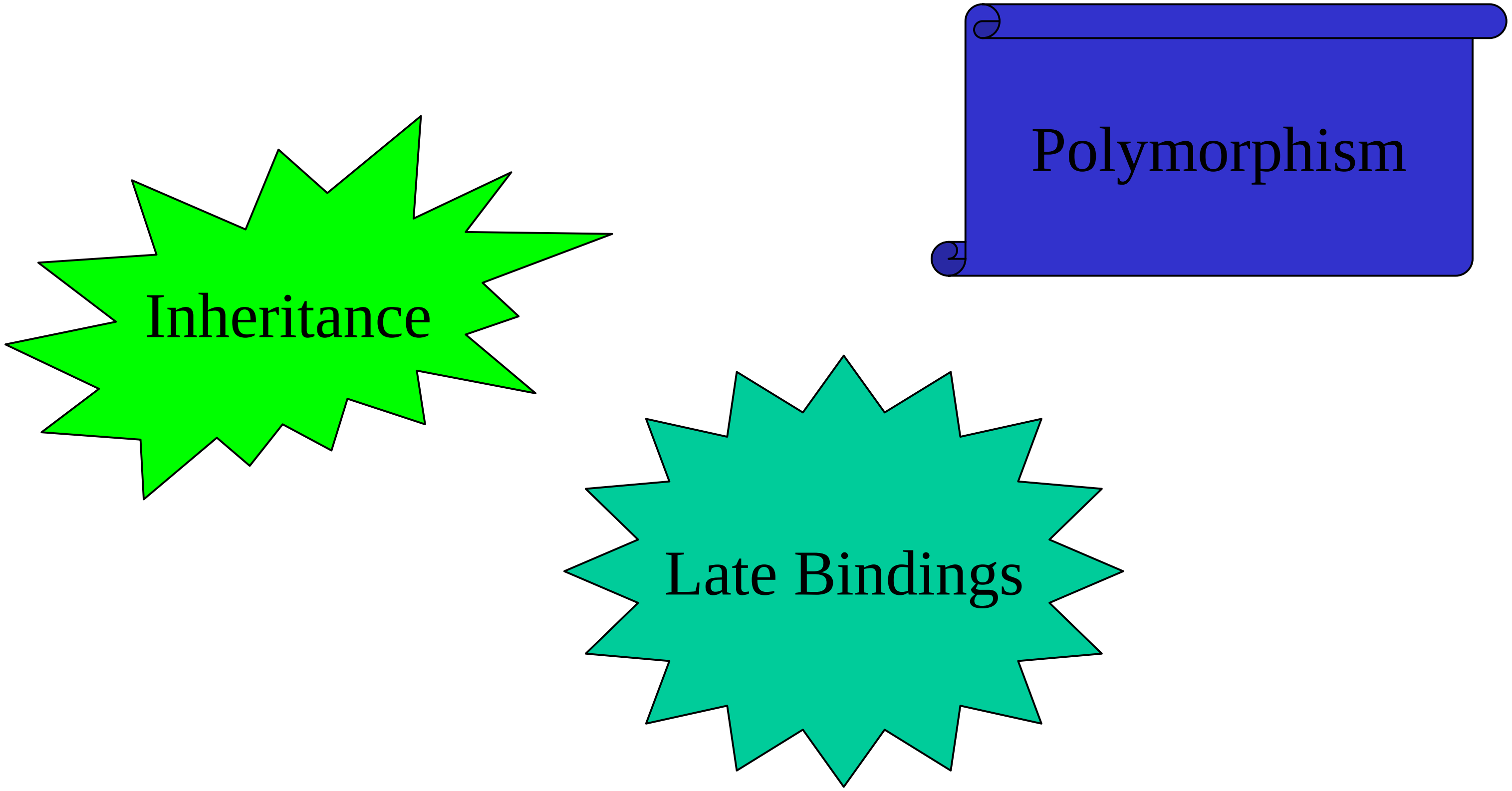
Dietrich Liko

- A step in a new direction for program organization
- But needs a bit more …



Stop 4: Object oriented programming

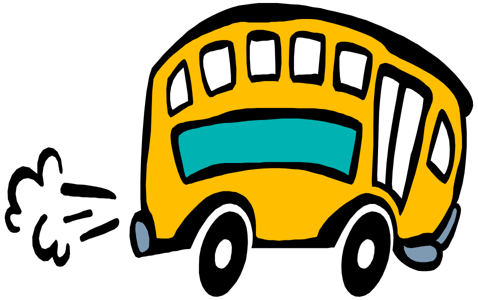
- Add some more ingredients



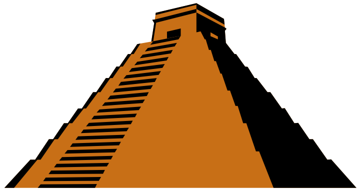
Polymorphism

Inheritance

Late Bindings



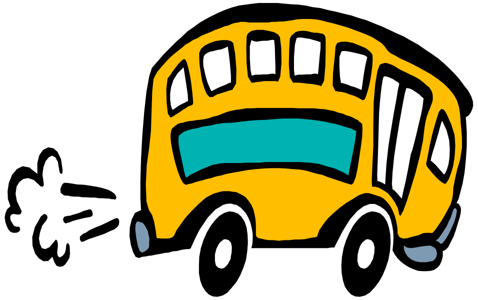
Define an Interface



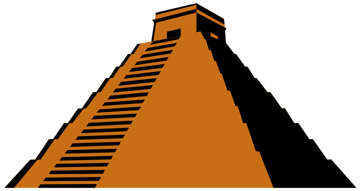
```
class Particle {
        virtual double energy() = 0;
        virtual void boost(double x, double y, double z,
                        double t) = 0;
        ………
}
```

Define what your object should do

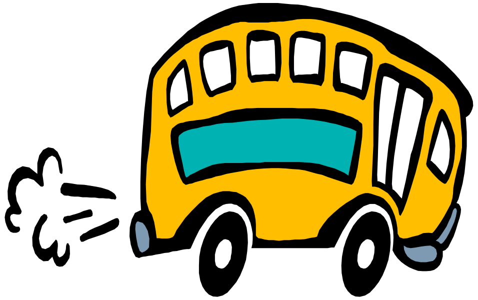
Provide an implementation



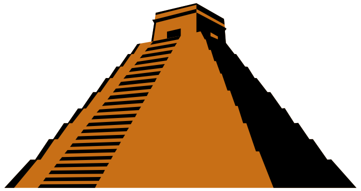
Implementation



```
class MyParticle : public virtual Particle {
      virtual double energy();
      ……
};


double MyParticle::energy() {
      return …..
}
```



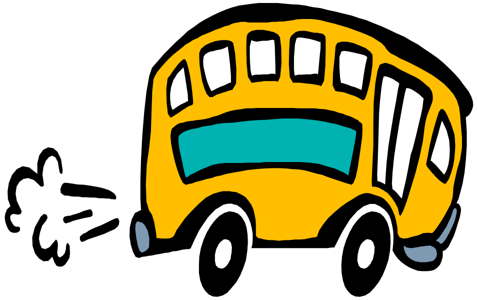
How to use ???



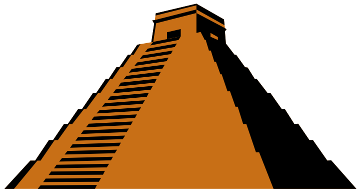
- I can write now a program in terms of “particles” and I do not need to know at all which particles are there

```
Particle * particle = new MyParticle();



…………

double energy = particle->energy();
```

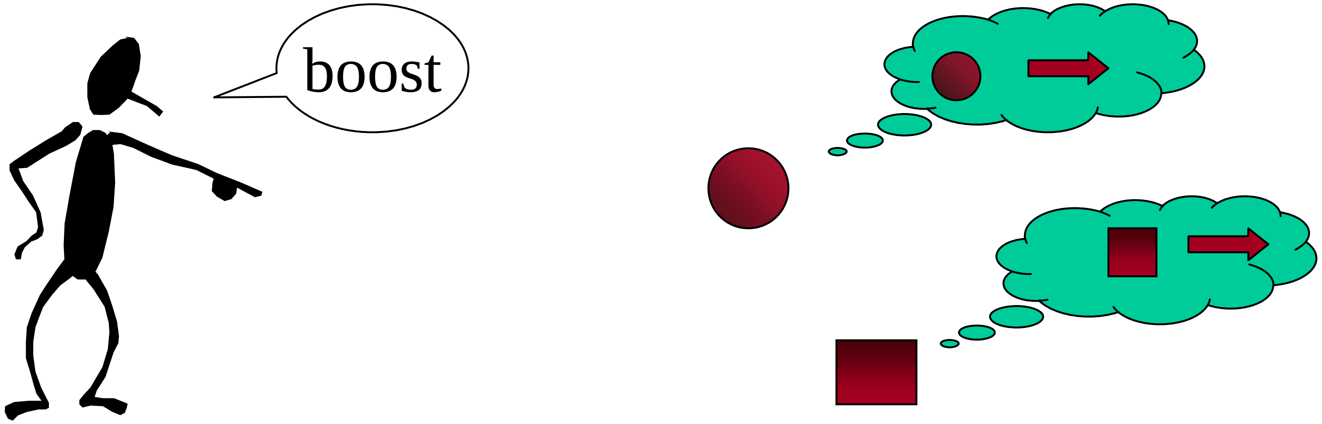
- As a matter of fact I can have a number particles, which are in truth MyParticles and YourParticles at the same time, and I do not know or care what they do to provide the answer



Dynamic Bindings

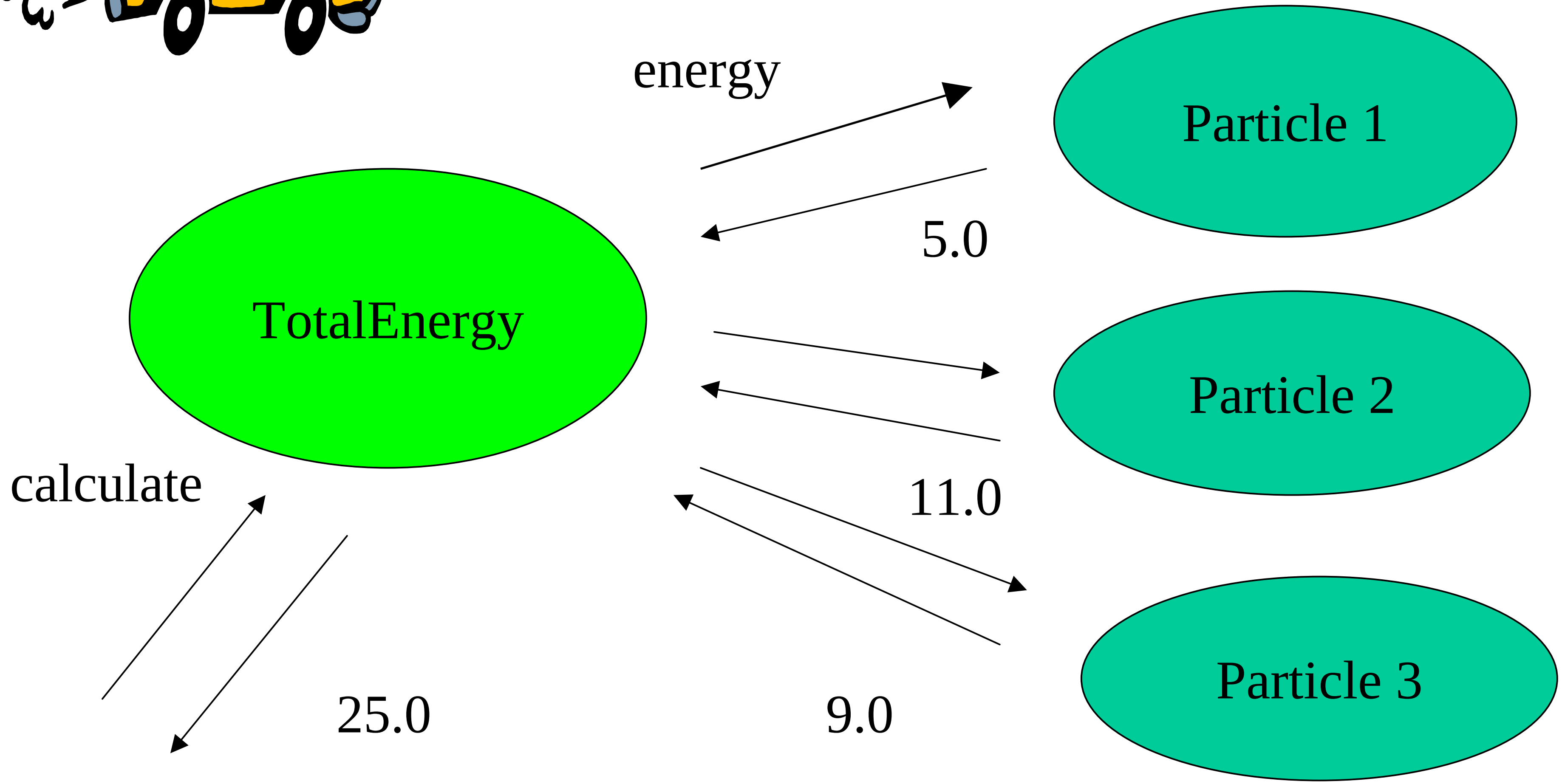


- Other name for virtual methods
- Only in the last moment it is known what is going to happen

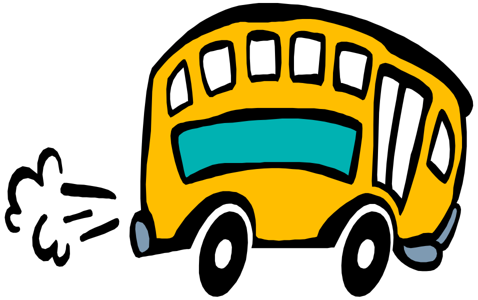


- The programmer tells the object to do something
- The object does it in its own specific way

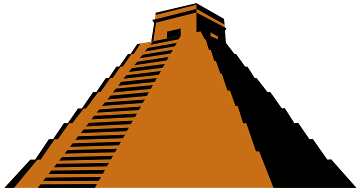
Messages between objects



Dietrich Liko

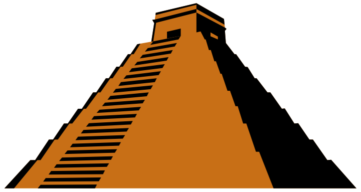


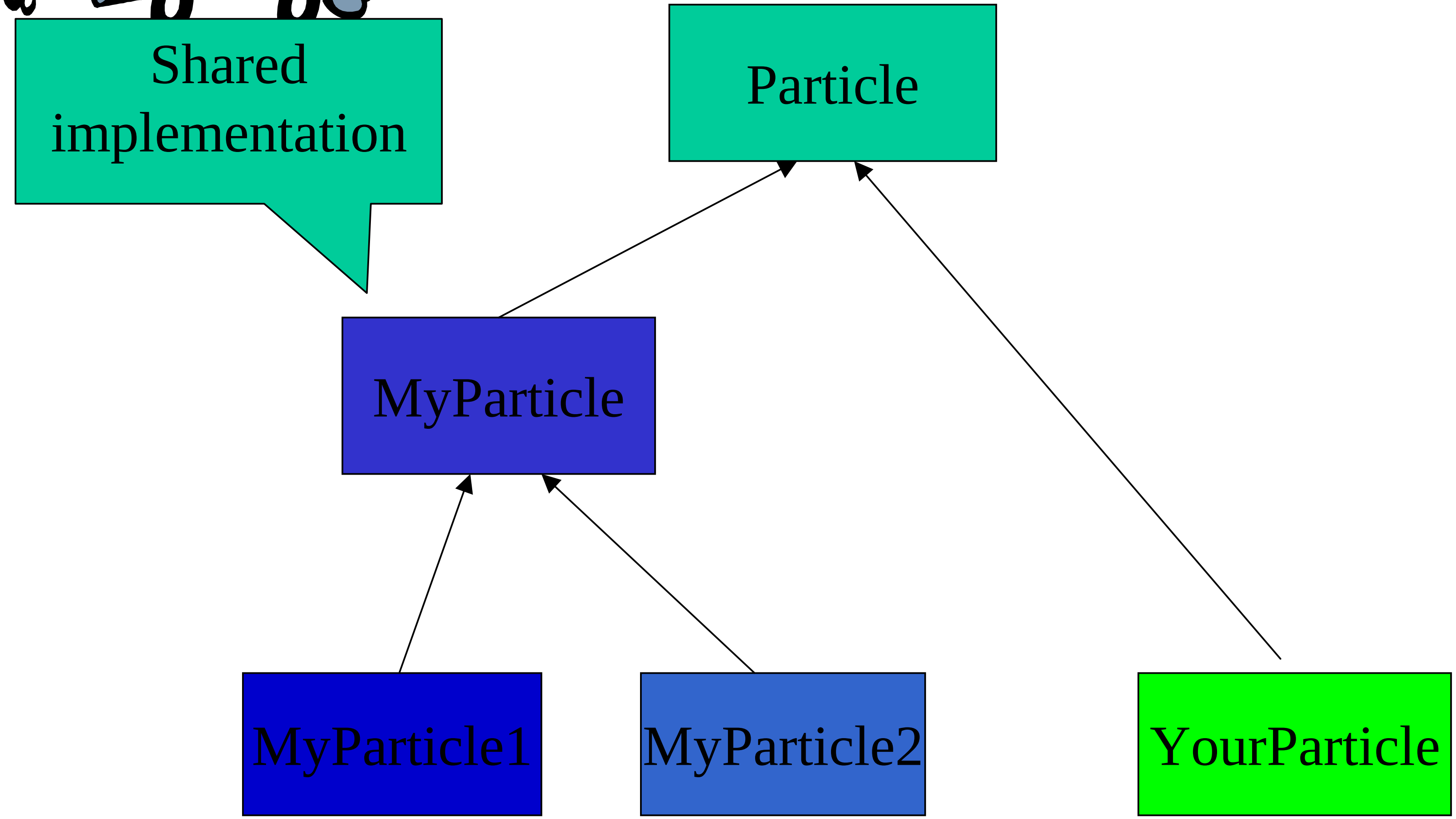
Just a different point of view



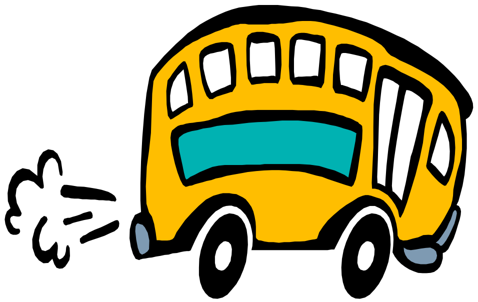
```
double TotalEnergy::calculate() {

        double sum = 0;
        for(int i=0;i<3;++i) {
                sum += particle[i].energy();
        }

        return sum;
}
```

Inheritance tree

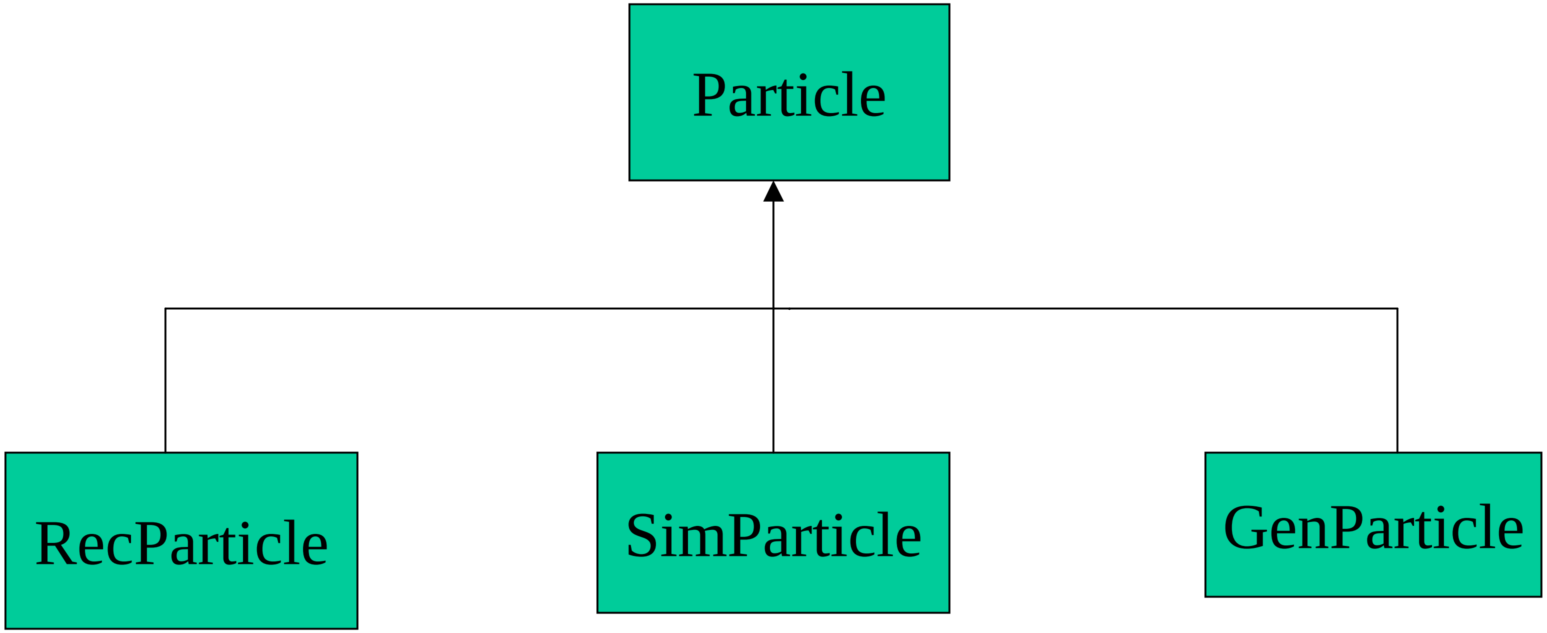




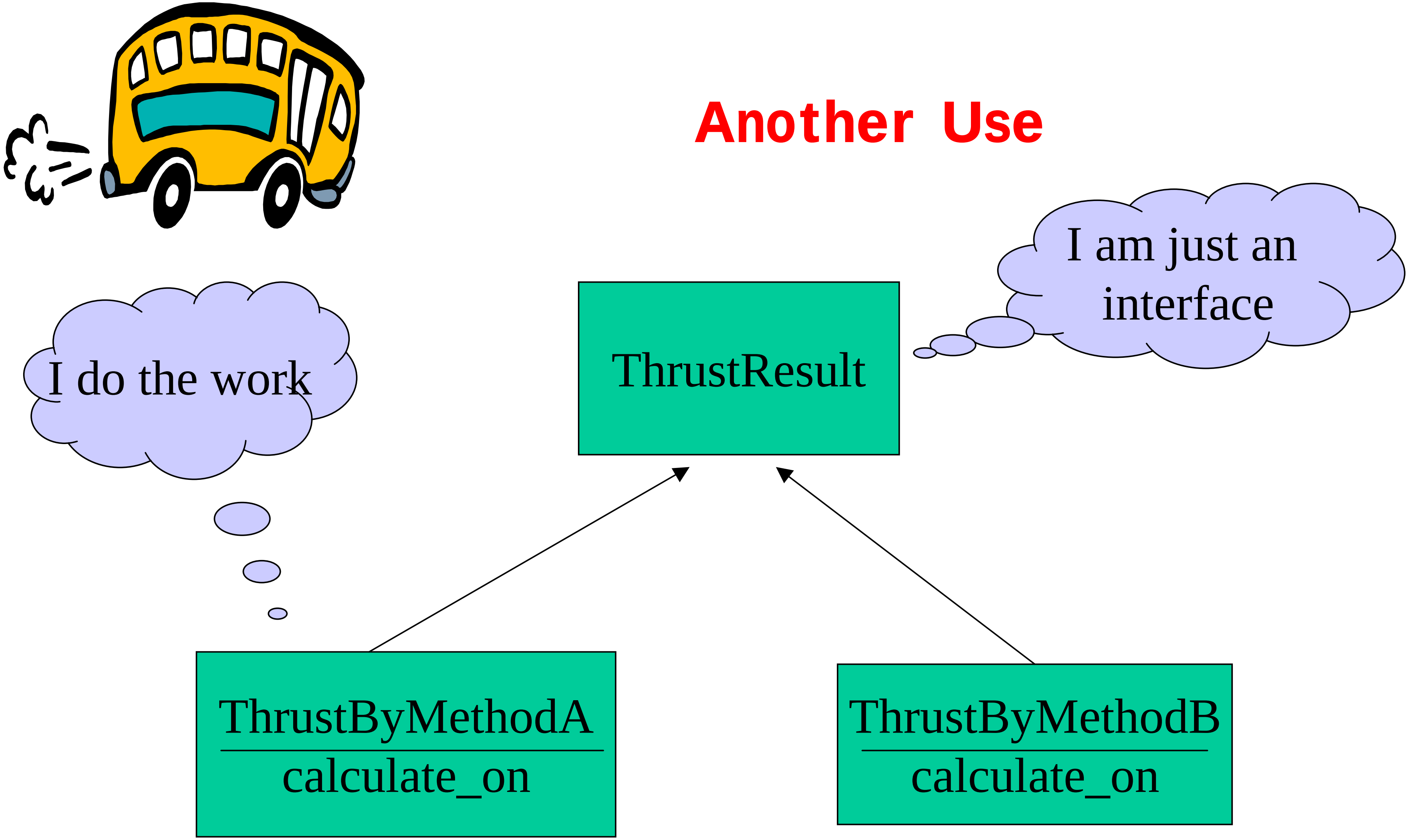
Shared implementation



In our case ...



Another Use



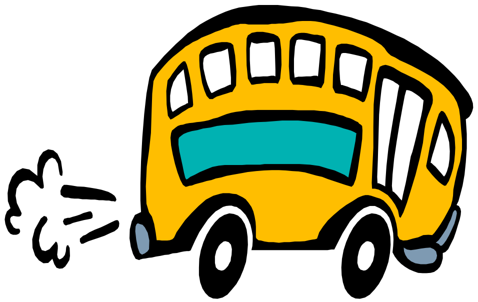
I do the work

I am just an interface

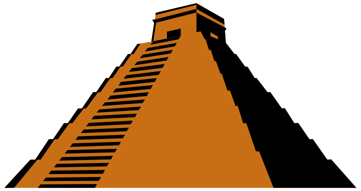
ThrustResult

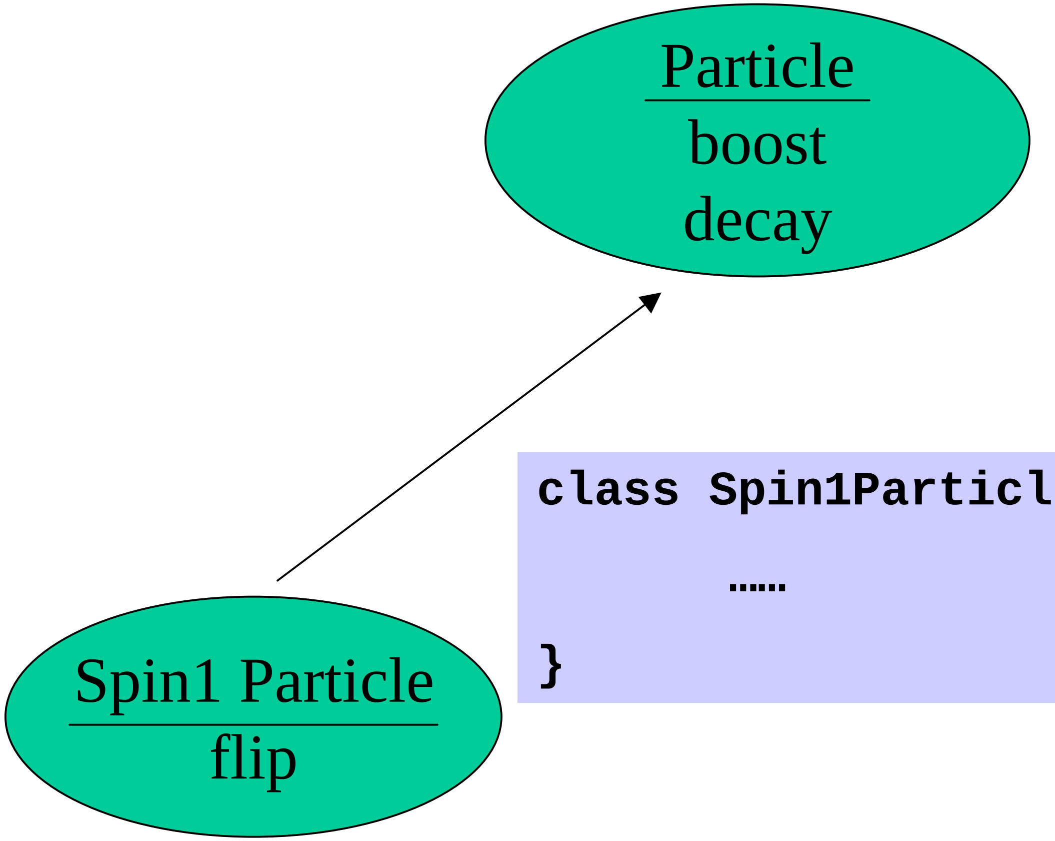
ThrustByMethodA
calculate_on

ThrustByMethodB
calculate_on

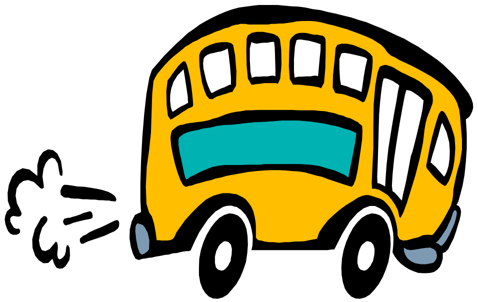


Is-a Relation

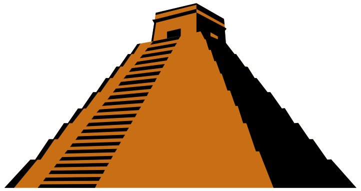


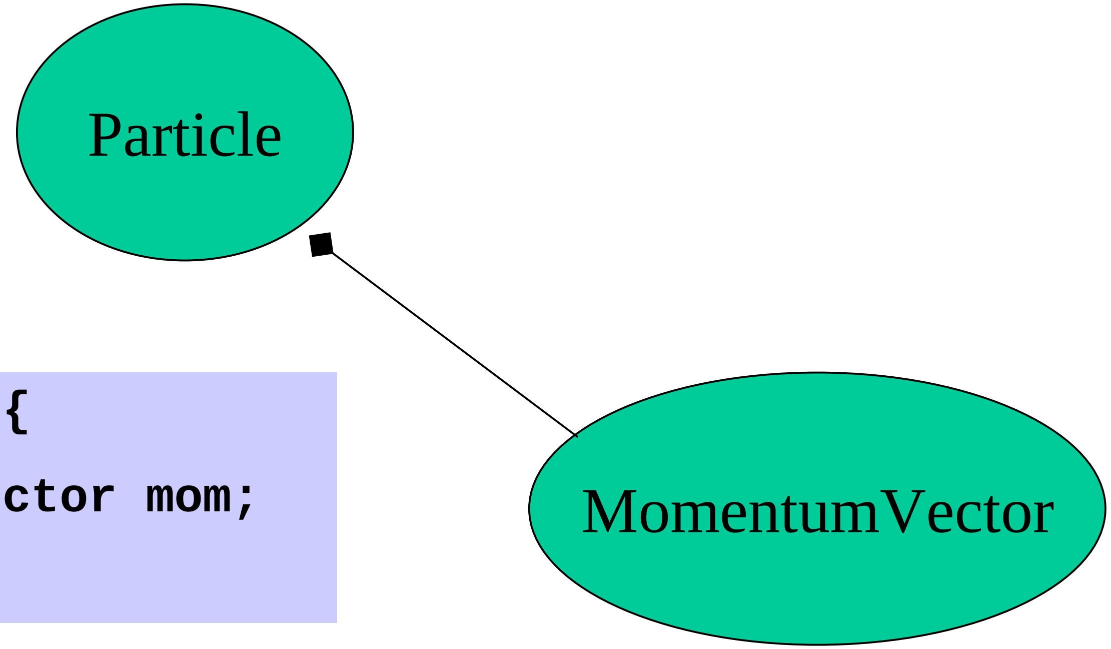


```
class Spin1Particle : public Particle {

        ……

}
```

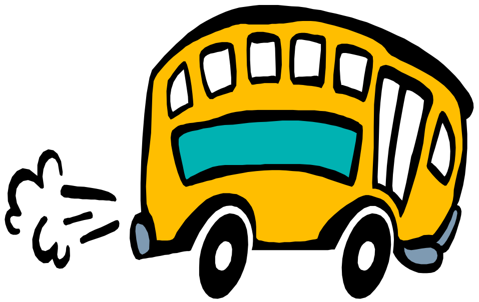


Has-a Relation

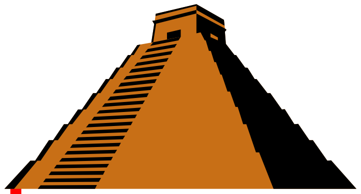




```
class Particle {
     MomentumVector mom;
}
```



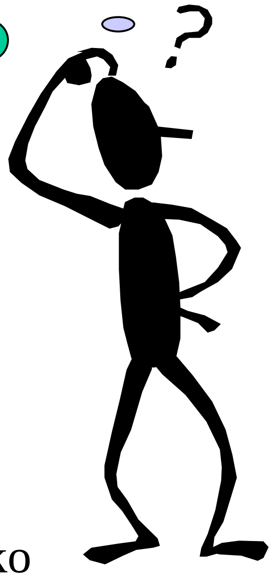
Often difficult to decide



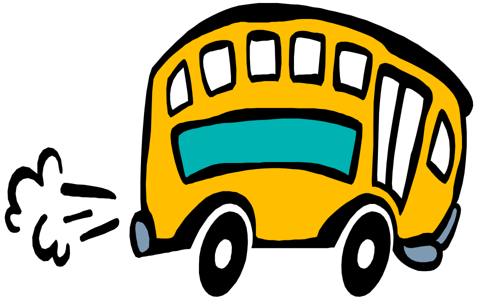
Has-it or Is-it ?

ThreeVector

Hamlet, Prince of Denmark

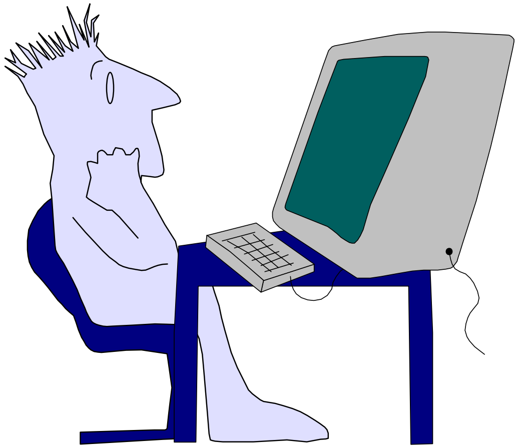


FourVector



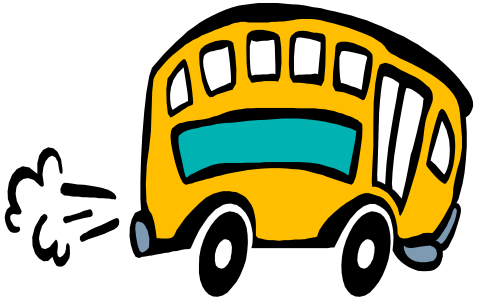
Programming paradigm

- Decide which classes you want
- Provide a full set of operation
- Avoid being dependent on implementation



Program will be organized as interaction objects

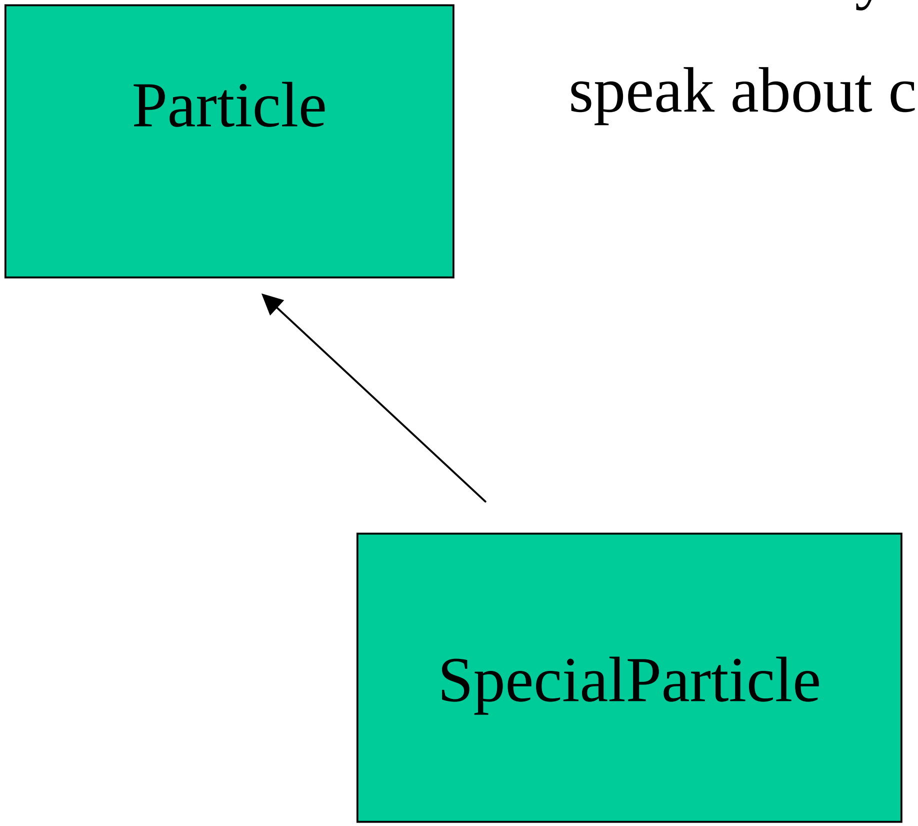
Dietrich Liko



UML

Current way to

speak about classes

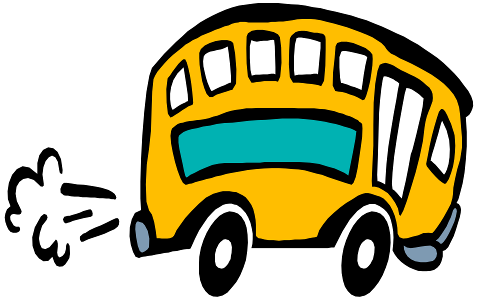


Booch

Rumbaugh

Jacbosen

Dietrich Liko



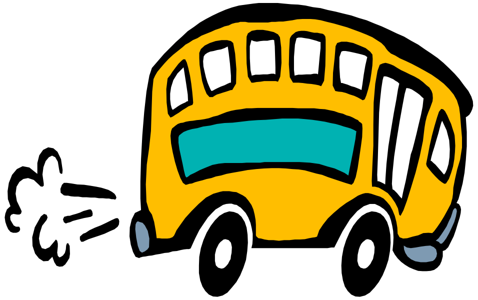
Design patterns

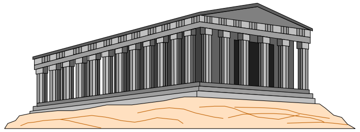
- Object Solutions
- very useful
- Solves the problems you did not have before you used objects
- You will use them every day

Gang of Four

- Gamma
- Helm
- Johnson
- Vlissides

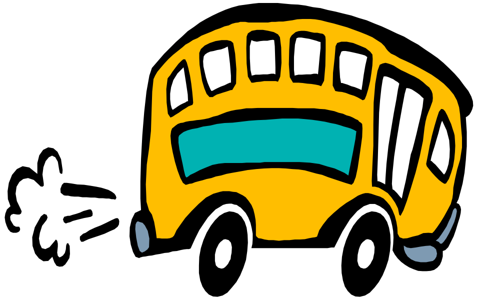
Dietrich Liko





Stop 5: Generic Programming

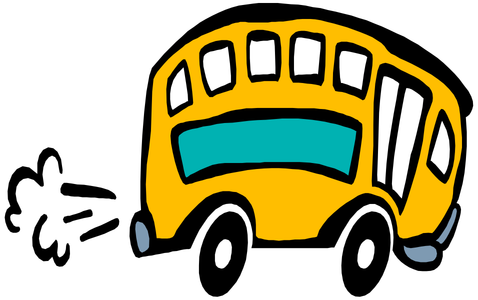
- A further idea ...
- Often a similar operation can be applied to different data types
- An example:
  - Lets try to implement complex numbers



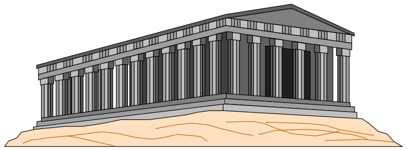
Example complex numbers

```
class complex {
public:
       float real() const;
       float imaginary() const;
       float mag() const;
private :
     float re;
     float im;
}
```

- Polymorphism does not help,
- A way how to automatically generate the source



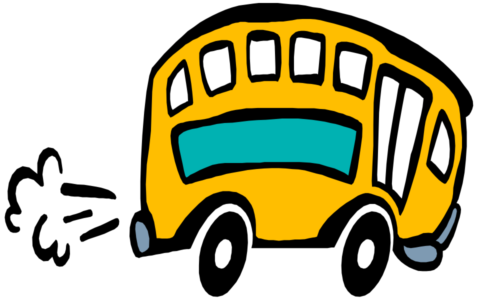
Complex numbers (generic)

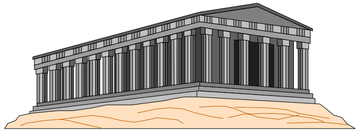


```
templace <class T>
class complex<T> {

private :
     T re;
     T im;
}
```

```
complex<double> a;


complex<float> b;
```

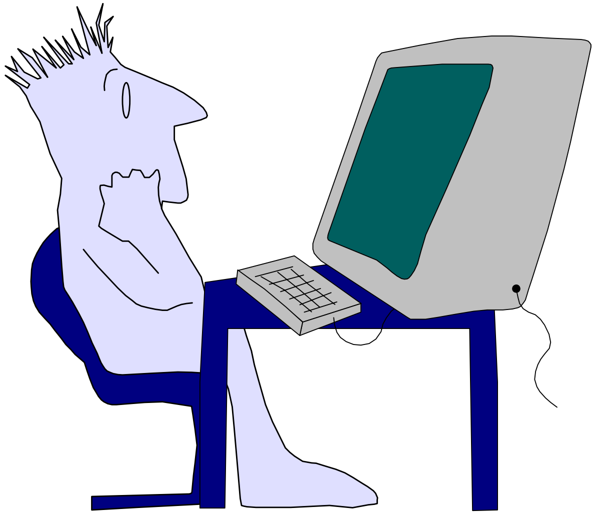
- Practical
- Used for many libraries





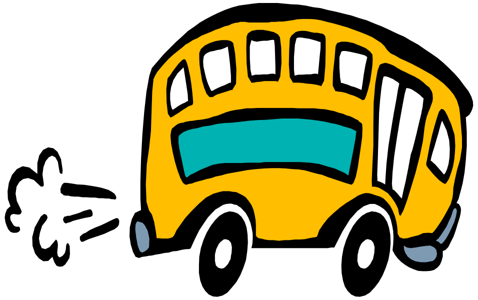
Programming Paradigm

- Decide which algorithm you want
- Parameterize them so that they work with a variety of data types

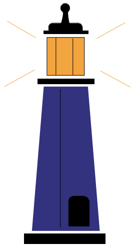


- C++ libraries are written in that style
- Probably you will simply use the features provided for you for some time …

Dietrich Liko

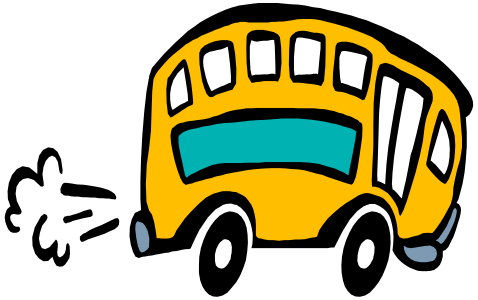


Stop 6: Standard Library



- Manly based generic programming
- example
  - complex
  - string
  - streams
  - containers

Dietrich Liko

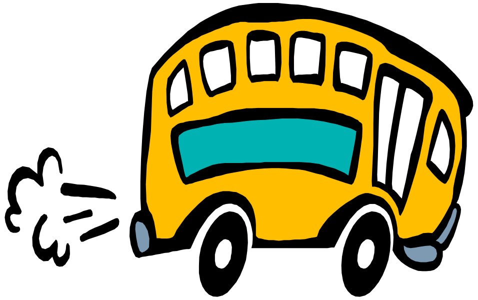


Naming Conventions

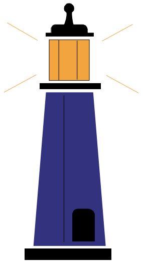
- There are many (for each experiment different)
- Taligent (used by root)

Class	Test
Method	doSomething or do_something
Member attribute	m_momentum
Cont	MAX

- Other Rules: C++ FAQ’s, Books by Scott Mayer



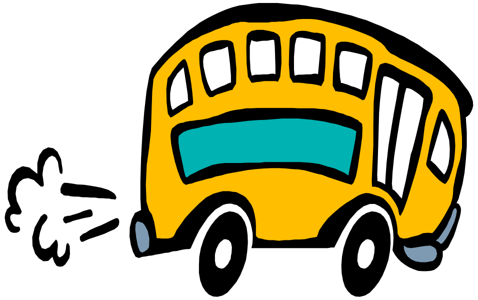
Strings



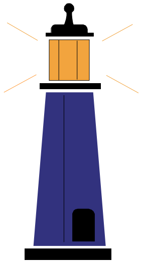
- Truly dynamic strings
- template usually hidden
- Safer then c strings
- Better then c strings
- Many member functions

```
std::string name;


int length = name.size();
```

- No reason why still to use c strings

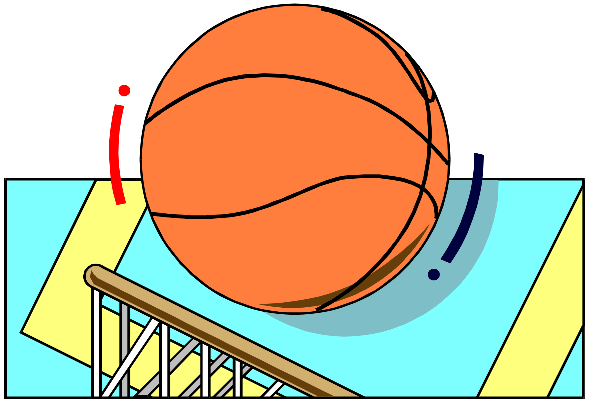


I/O Streams



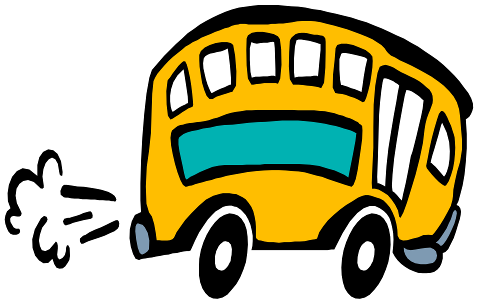
- New I/O syntax
- Modeled after UNIX pipes

```
cout << “Hello World” << endl;

Particle p;

cout << p;
```

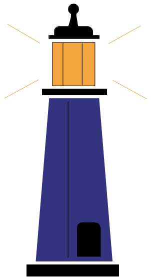


cout Basket

- Fast, easy
- But hard to make “nice” output

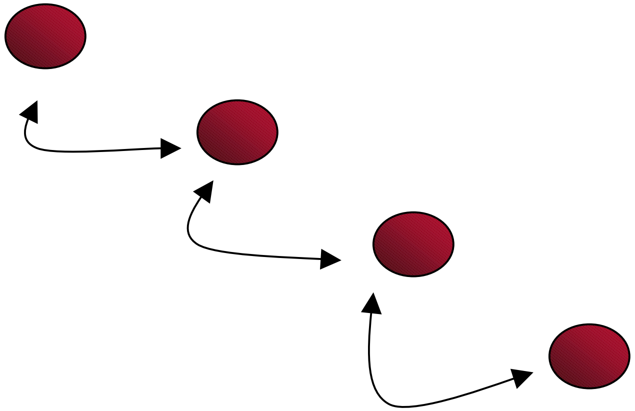


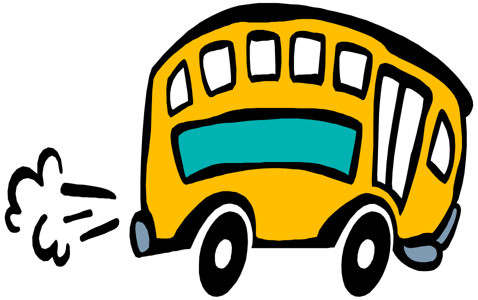
STL Containers



Better then C

- Replacement of C arrays
  - Safer
- vector<double>
- list<particles>





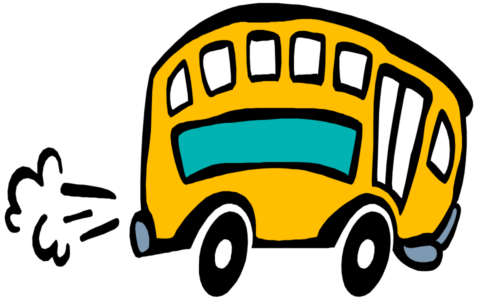
Iterator

- A possible way

```
vector<int> array;

int sum = 0;
for(int i = 0;i<array.size();++i) {
        sum += array[i];
}
```

- Another way (if [] is very expensive!)

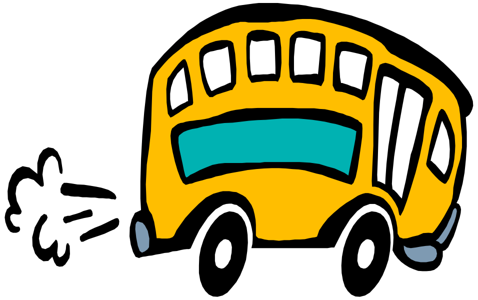
```
int sum = 0;
for (vector<int>::iterator elem = array.begin();
      elem != array.end();
      ++elem ) {
      sum += *elem;
}
```



Stop 7: Other Libraries

- CLHEP for Physics Quantities
  - Vectors, LorentzVectors
  - Geometry & Transformations
  - SI Units
  - Random Numbers
    - many distributions
  - Obsolete packages
    - strings, list etc.

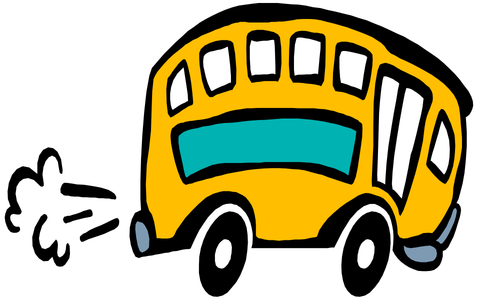
This you will have to learn in any case



GEANT4

- Detector simulation
  - geometry
  - particles
  - physics process

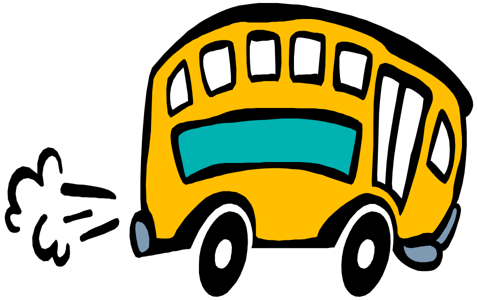
- very large toolkit
- experiments are starting to use it
  - Hard to start (no CERN software!)
  - But quite easy to use !



PYTHIA 7

- Still in development
- Replacement for PHYTHIA 6 event generator

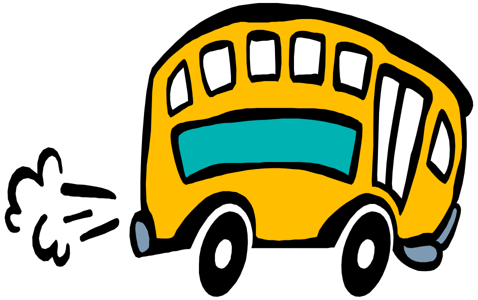
Dietrich Liko



Anaphe

- A number of packages defining the computing at CERN

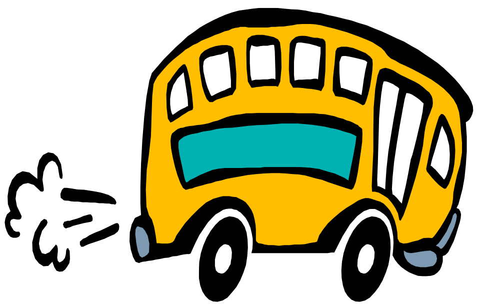
- CLHEP (foundation)
- GEANT4 (detector simulation)
- HTL (Histograms)
- Gemini (Fitting)
- OpenInventor/OpenGL (Graphics)
- Objectivity/DB (Persistency)



How to continue the journey

- Get started
  - Follow the Training session (this afternoon)
  - Get a good book (it’s a great Xmas present!)
  - Attend the CERN C++ course (in few weeks)
  - Try a smaller project

- Get an expert
  - Trough it away and program it again (several times)
  - Get other books (FAQ, Efficient C++) (Easter presents?)
  - Study how other person solve the problem
  - Attend the OO Design course (in some months)



A Final Warning

The difference between C and C++

C lets you shoot yourself in the foot rather easily.

C++ allows you to blow your whole leg off.

Have fun!