The Generic Field and Black Box Matrix Models in the LinBox Library

William J Turner
North Carolina State University
www.math.ncsu.edu/˜wjturner

Joint work with members in the LinBox Research Project at LMC/IMAG Grenoble, NCSU, U. Calgary, U. Delaware, and U. Western Ontario.

Abstract

The LinBox library is a generic library with algorithms for black box linear algebra that are generic with respect to the field of matrix entries and the black box matrix-times-vector function objects. The common object interface for these fields and black box matrices, that is, the class members that are required by the library's algorithms, must be defined. LinBox has C++ plugable classes called archetypes to specify this interface by providing patterns for programming fields and black box matrices. They can be used to program new fields and black box matrices or to program new wrapper/adaptors of existing code. Through the use of abstract base classes they can be used to control code bloat. Archetypes are used for all but the most basic objects (floating point numbers, 10 by 10 matrices). An archetype is distinguished from a Java interface by being an explict class.

The Concept of an Archetype

- Three uses:

  1. To define the common object interface, i.e., specify what an explicitly designed class must have.
  2. To distribute compiled code and prototype library components.
  3. To control code bloat.

- Used for all but the most basic objects such as floating point numbers and 10 by 10 matrices.

- Not a Java interface; an archetype is an explicit class.

Field Entry Archetype

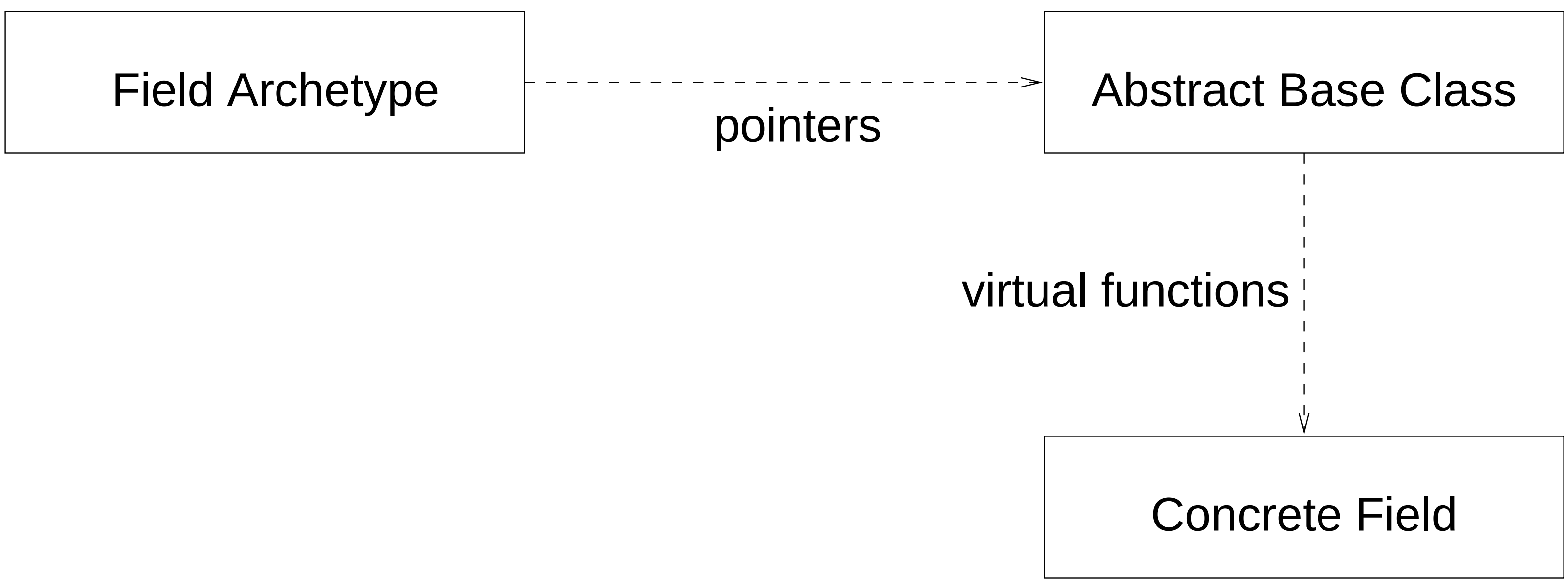
Allow C++ types such as doubles as field elements

$\Downarrow$

Field elements must have (non-virtual) constructors

$\Downarrow$

Field archetype $\neq$ Abstract base class



```
class field_archetype
{
public:
  class element // encapsulated element class
  {
  public:
    element(void);                     // element default constructor
    element(const element&);           // element copy constructor
    ~element(void);                    // element destructor
    element& operator=(const element&); // assignment operator
  };

  // object management
  element& init(element& x, const element& y) const;
  ...

  // arithmetic operations
  bool isequal(const element& x, const element& y) const;
  element& add(element& x, const element& y, const element& z) const;
  ...
};
```

Demonstrations of code available upon request.

Black Box Matrix Archetype

No C++ types to be used as black box matrices

$\Downarrow$

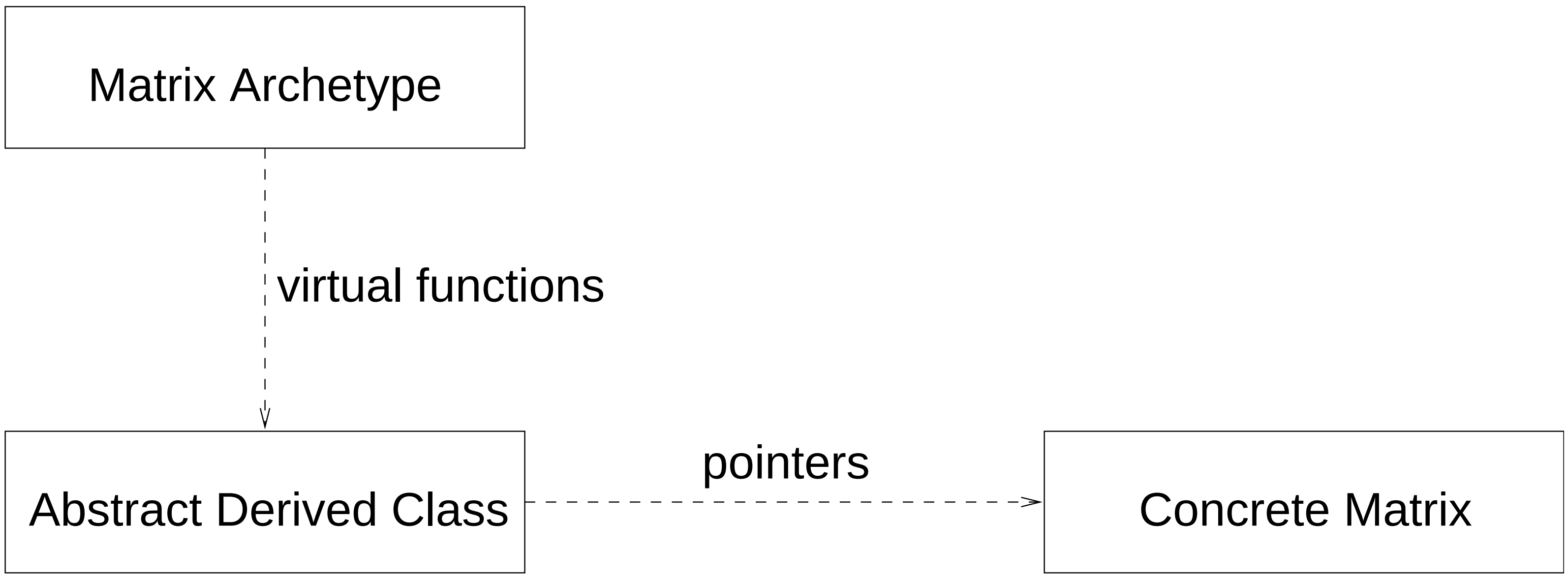
Black box matrices do not need (non-virtual) constructors

$\Downarrow$

Use virtual `clone()` to construct derived classes

$\Downarrow$

Matrix archetype $=$ Abstract base class



```
class blackbox_archetype
{
public:

  virtual ~blackbox_archetype(); // destructor

  virtual blackbox_archetype* clone() const = 0; // virtual constructor

  // Function to apply black box matrix to vector, store output through
  // pointer to another vector, and to return reference to ouput pointer.
  // References to pointers used so that output vector does not need to
  // be initialized before use.
  template <class Vout, class Vin> virtual Vout*&
           apply(Vout*&, const Vin* const&) const = 0;

  // retrieve matrix dimensions
  virtual long get_coldim(void) const = 0;
  virtual long get_rowdim(void) const = 0;
};
```

Demonstrations of code available upon request.

Vector Archetype

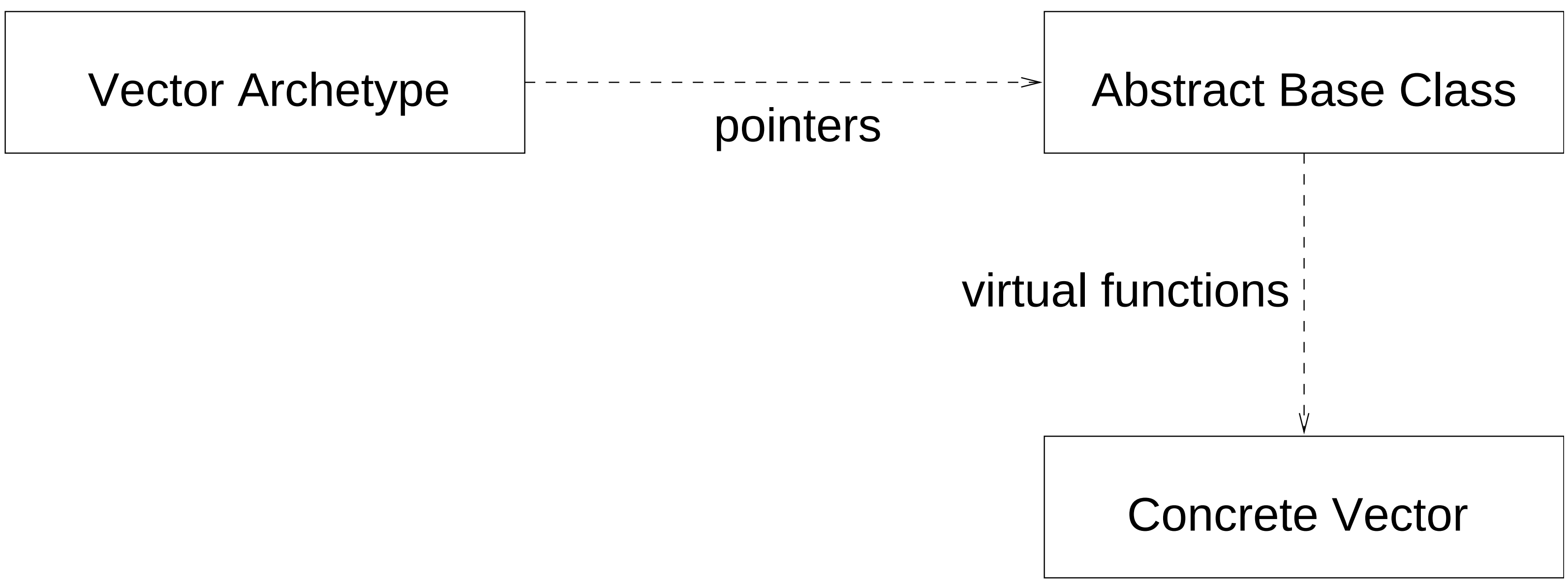
Allow STL vectors as LinBox vectors

$$\Downarrow$$

Vectors must have (non-virtual) constructors

$$\Downarrow$$

Vector archetype $\neq$ Abstract base class



```
template <class Element>
class vector_archetype
{
public:

  vector_archetype(void); // default constructor

  vector_archetype(const vector_archetype<Element>&); // copy constructor

  ~vector_archetype(void); // destructor

  vector_archetype<Element>& operator=(const vector_archetype<Element>&);
    // assignment operator

  Element& operator[](long); // access to vector element
  const Element& operator[](long) const; // constant acess to vector element
};
```

Demonstrations of code available upon request.