Runtime Concepts for the C++ STL

Peter Pirkelbauer[1] Sean Parent[2] Mat Marcus[2]
Bjarne Stroustrup[1]

[1]Texas A&M University

[2]Adobe Systems Inc.

23^{rd} Symposium on Applied Computing 2008 - OOPS Track

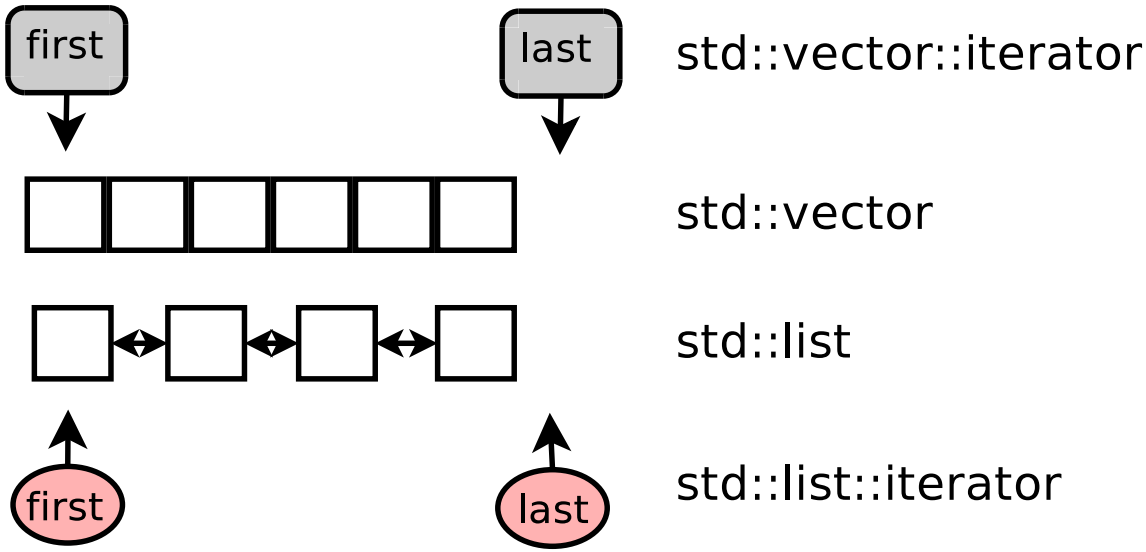
Overview

- Introduction
  - C++ Standard Template Library (STL)
  - Problem Definition
  - The Runtime Concept Idiom
- Contribution
  - The Algorithms Library
  - Runtime Concepts on Container elements
- Conclusion

The C++ Standard Template Library (STL)

- Data structures (vector, list, deque, map, set, etc.)
- Algorithms

- Concept based (will be in C++0x; Gregor et al. [4])
- Regular Semantics (Dehnert and Stepanov [3])
- Iterators



The Problem - Conventional template code

Interface + Implementation

```
template <class Iterator>
Iterator
random_elem(Iterator first, Iterator last)
{
  typename Iterator::difference_type dist = distance(first, last);
  return advance(first, rand() % dist);
}
```

User code

```
// v is a std::vector<int>
int elem = *random_elem(v.begin(), v.end());
```

Our Ideal World

Interface

Abstracts from containers

Abstracts from element types

```
ForwardIter<Proxy>
random_elem(ForwardIter<Proxy> first, ForwardIter<Proxy> last);
```

Implementation

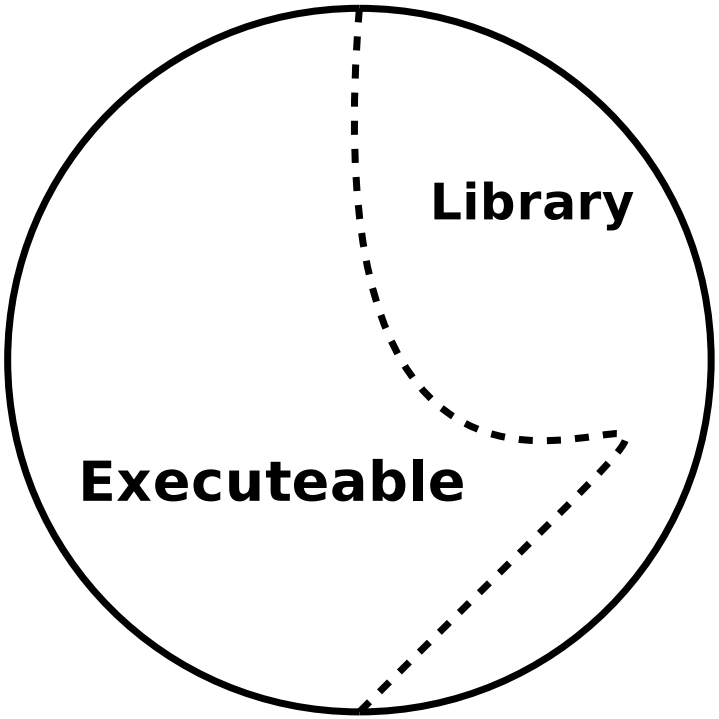
```
ForwardIter<Proxy>
random_elem(ForwardIter<Proxy> first, ForwardIter<Proxy> last)
{
   ForwardIter::difference_type dist = distance(first, last);
   return advance(first, rand() % dist);
}
```

User code

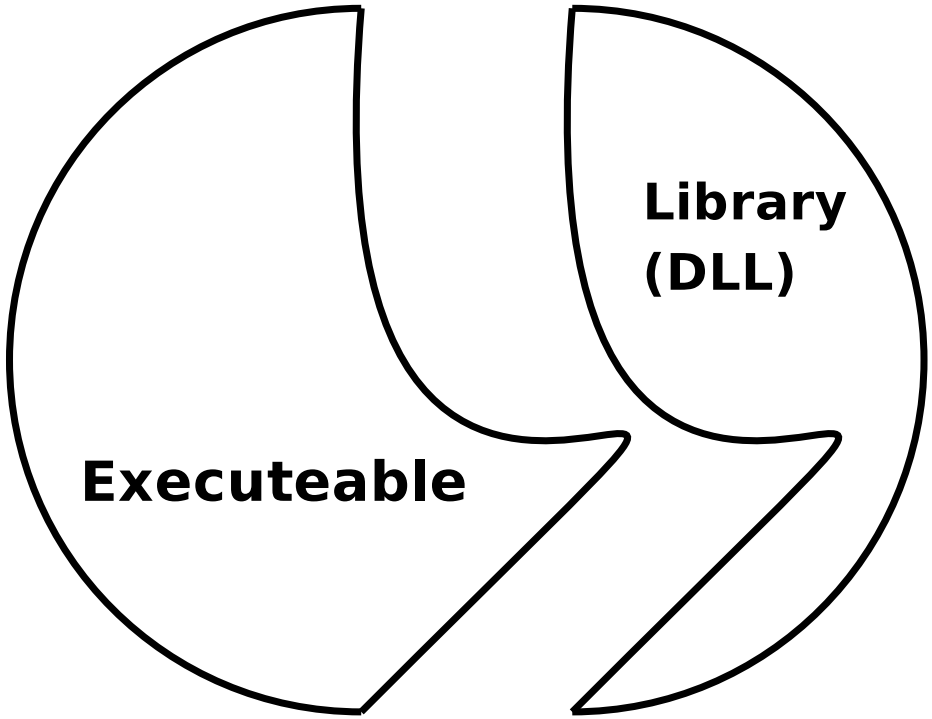
```
// v is a std::vector<int>
int elem = *random_elem(v.begin(), v.end());
```

Performance vs Loose Coupling

Conventional Templates



Runtime Concepts



Polymorphic Iterators

Interface

Abstracts from containers

```
ForwardIter<int>
random_elem(ForwardIter<int> first, ForwardIter<int> last);
```

Implementation

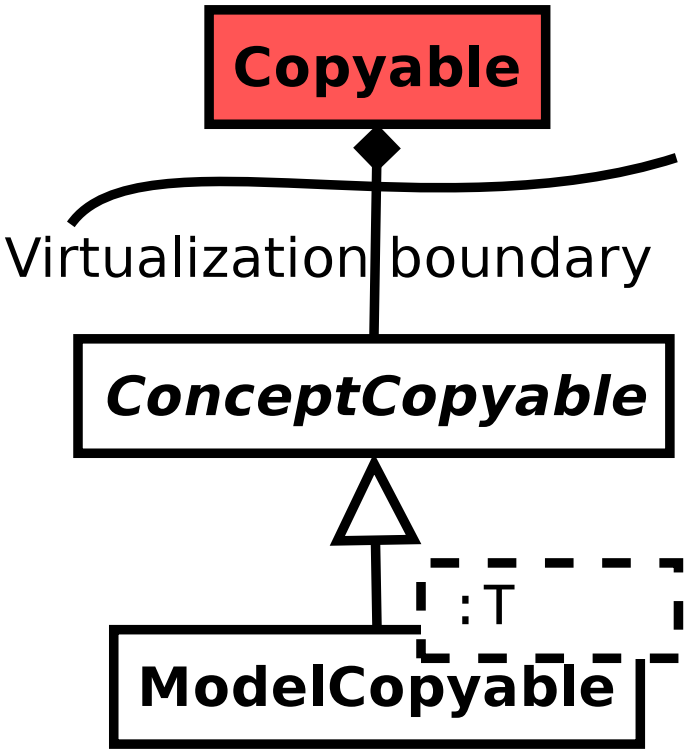
```
ForwardIter<int>
random_elem(ForwardIter<int> first, ForwardIter<int> last)
{
  ForwardIter::difference_type dist = distance(first, last);
  return advance(first, rand() % dist);
}
```

User code

```
// v is a std::vector<int>
int elem = *random_elem(v.begin(), v.end());
```

The Runtime Concept Idiom

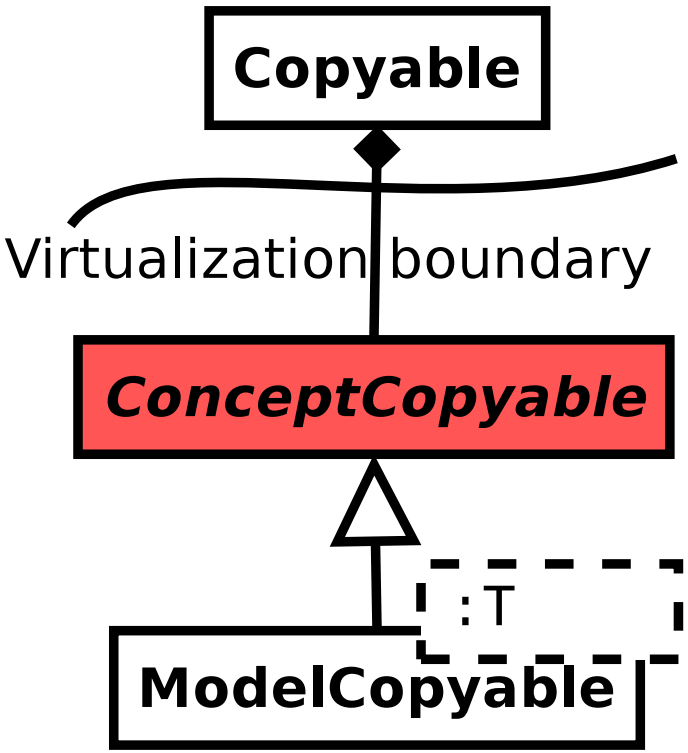
Parent [7], Järvi et al. [5], Marcus et al. [6]




```
struct Copyable {
  ConceptCopyable* c;

  template<typename T>
  Copyable(const T& t)
    : c(new ModelCopyable<T>(t)) {}

  Copyable(const Copyable& obj)
    : c(&obj.c->clone()) {}

  ~Copyable()
  {
    delete c;
  }
};
```

The Runtime Concept Idiom

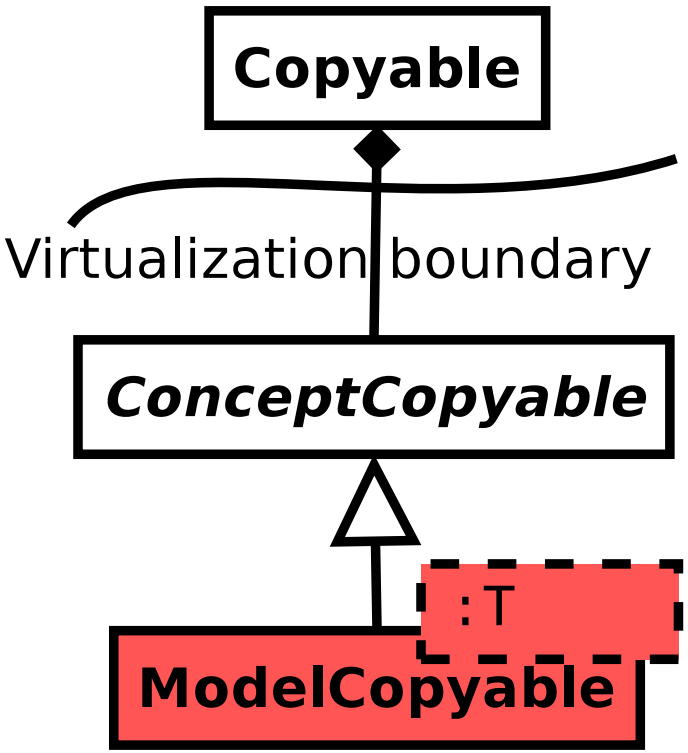
Parent [7], Järvi et al. [5], Marcus et al. [6]




```
struct ConceptCopyable {
   virtual ConceptCopyable& clone() const = 0;
};
```

The Runtime Concept Idiom

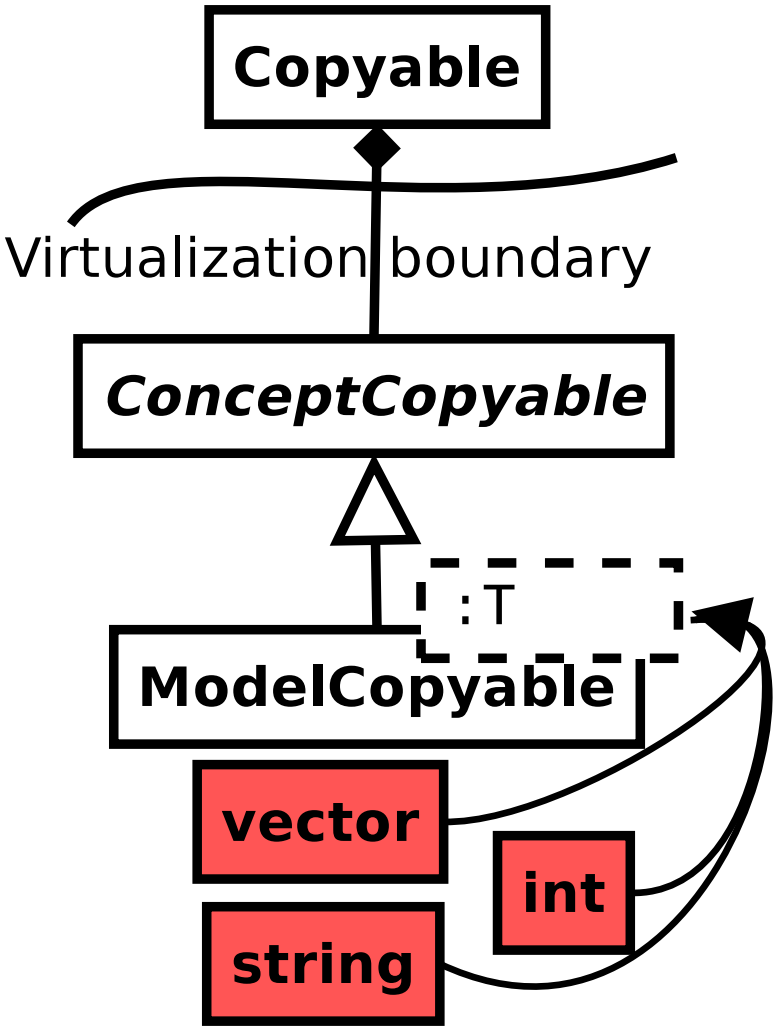
Parent [7], Järvi et al. [5], Marcus et al. [6]




```
template <typename T>
struct ModelCopyable : ConceptCopyable {
  T t;

  ModelCopyable(const T& val)
    : t(val) {}

  ModelCopyable& clone() const
  {
    return *new ModelCopyable(t);
  }
};
```

The Runtime Concept Idiom

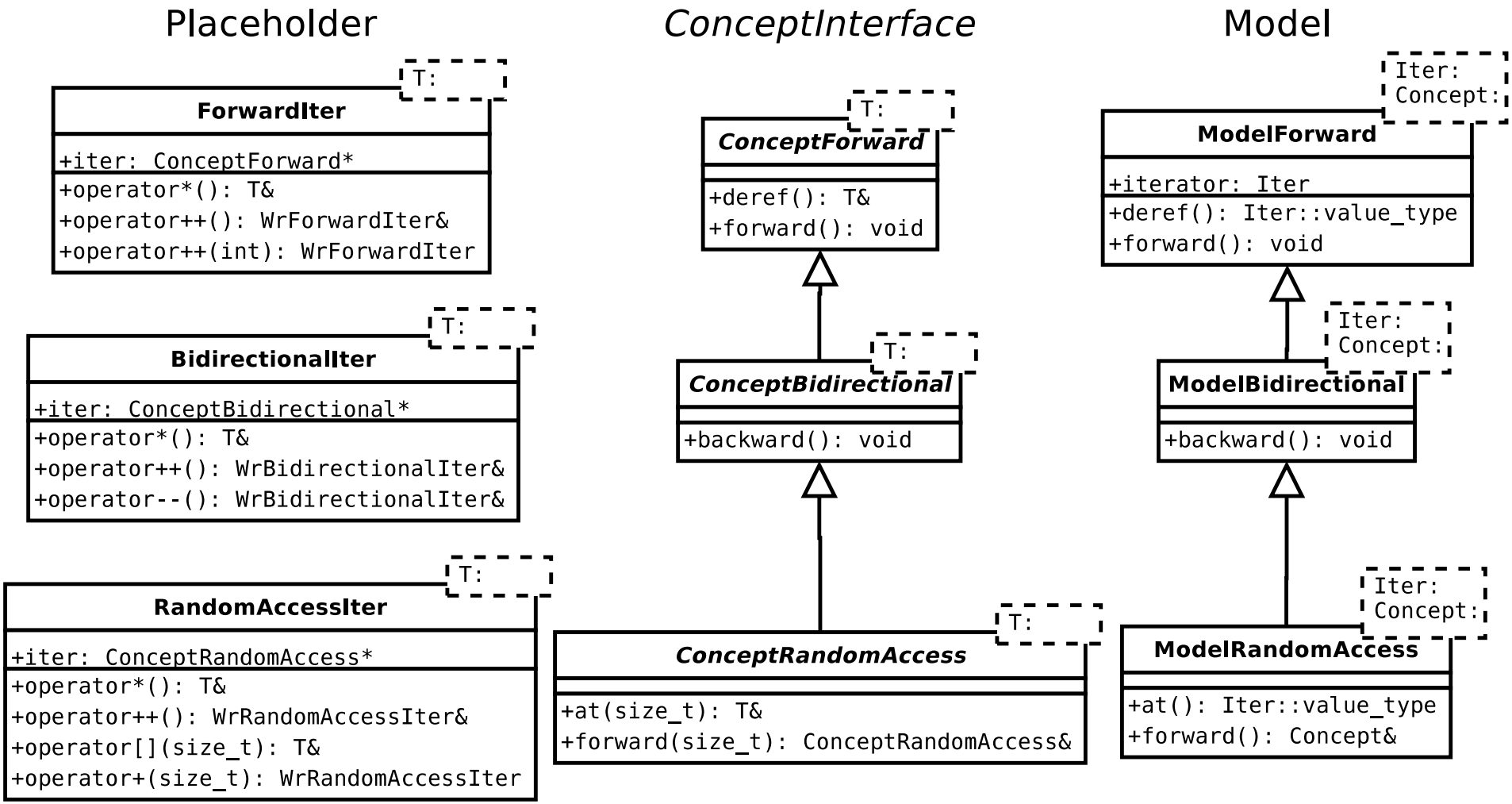
Parent [7], Järvi et al. [5], Marcus et al. [6]




```
void accept_copyables(Copyable obj)
{
   // invoke copy constructor
   Copyable cpy(obj);
}

// vector
std::vector<int> vec;
accept_copyables(vec);

// integer
accept_copyables(7);

// strings
std::string str("Hello World");
accept_copyables(str);
```

Applied to Iterators

Refinements (Marcus et al. [6]), Type Erasure and Iterators (Becker [1])



The Algorithms Library

Interface

```
ForwardIter<int>
random_elem(ForwardIter<int> first, ForwardIter<int> last);
```

Implementation

```
ForwardIter<int>
random_elem(ForwardIter<int> first, ForwardIter<int> last)
{
  ForwardIter::difference_type dist = distance(first, last);
  return advance(first, rand() % dist);
}
```

User code

```
// v is a std::vector<int>
int elem = *random_elem(v.begin(), v.end());
```

The Algorithms Library - Design Principles

- Algorithm selection
  - considers refinement and type information
- Customizeable - without modification of library code
  - cannot assume to know all container types
- Implementable in current C++
  - does not use any language extension
- Minimal performance overhead

The Algorithms Library - Implementation

- Default implementation
  - defined based on more general concept in STL
- Algorithm selection
  - finds best match according to a dedicated argument
- Class hierarchy traversal
  - Query base class of a given type
- Accessor function
  - defined in the library's iterator namespace

The Algorithms Library - Customization

Add generic implementation for randomaccess iterators

```
algolib::add_generic<
    algolib::advance<int>, // library name
    RandomaccessIter<int> // iterator−type
  >();
```

Unchanged library code

```
ForwardIter<int>
random_elem(ForwardIter<int> first, ForwardIter<int> last)
{
  ForwardIter<int>::difference_type dist = distance(first, last);
  return advance(first, rand() % dist);
}
```

The Algorithms Library - Customization

Add specific implementation for std::list<int>

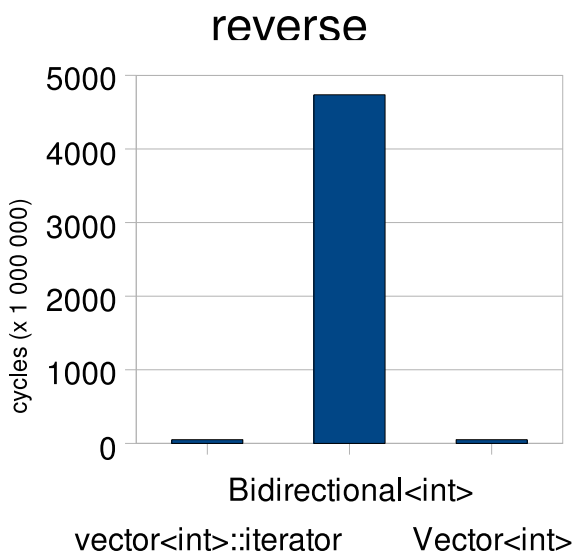
```
algolib::add_specific<
    algolib::advance<int>, // library name
    std::list<int>::iterator // iterator−type
  >();
```

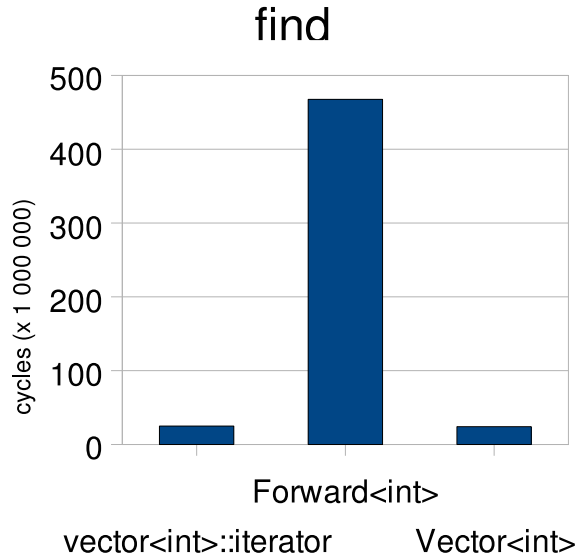
Unchanged library code

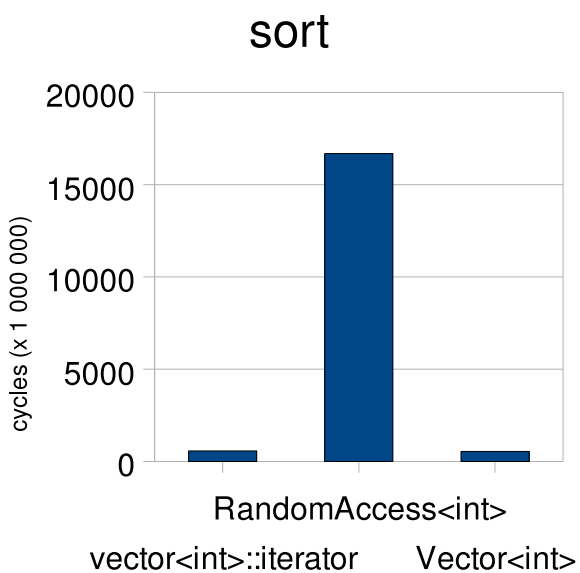
```
ForwardIter<int>
random_elem(ForwardIter<int> first, ForwardIter<int> last)
{
  ForwardIter<int>::difference_type dist = distance(first, last);
  return advance(first, rand() % dist);
}
```

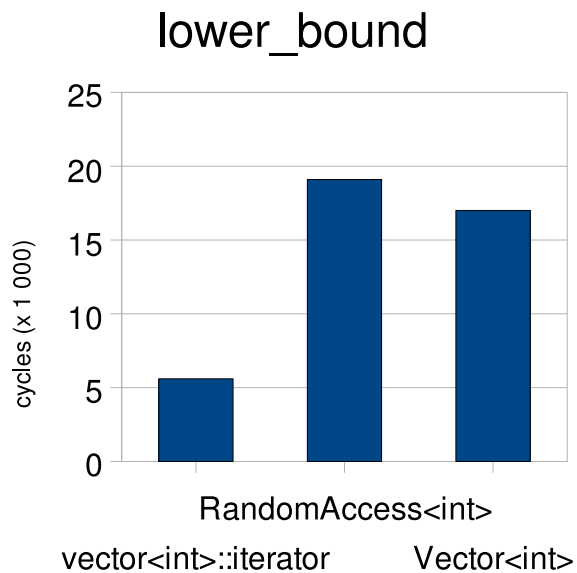
The Algorithms Library - Results

- Pentium-D
- GCC 4.2 -O3 -mach=prescott
- vector<int>
  - reverse
  - find
  - sort
  - lower_bound

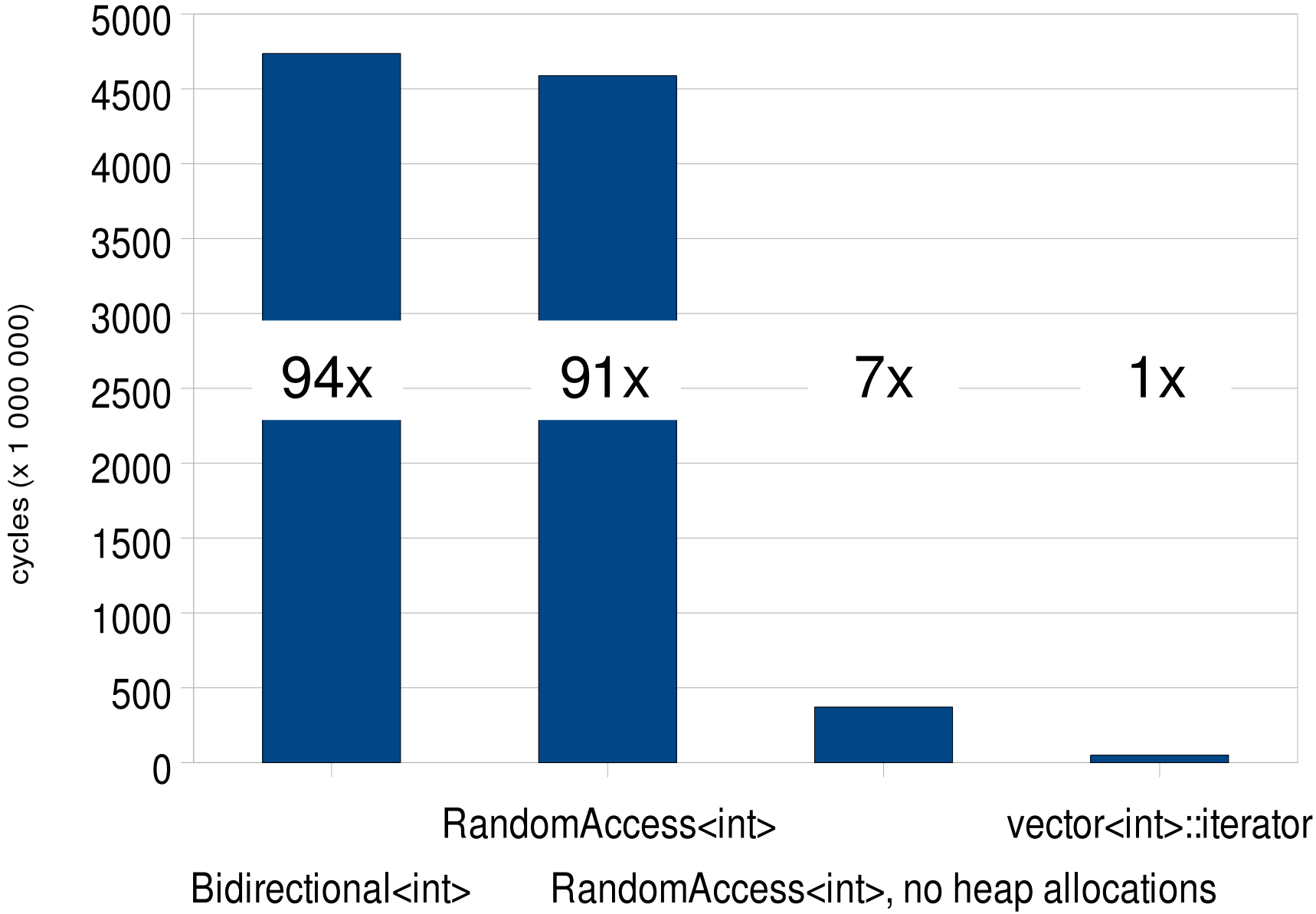








The Algorithms Library - Reverse



The Algorithms Library - Alternatives

- Algorithms as virtual functions
- GIL's dispatch mechanism (Bourdev and Järvi [2])
- Open Multimethods (Smith [9], Pirkelbauer et al [8])

Retroactive Runtime Concepts

Interface

Abstracts from element types

```
ForwardIter<Proxy>
random_elem(ForwardIter<Proxy> first, ForwardIter<Proxy> last);
```

Implementation

```
ForwardIter<Proxy>
random_elem(ForwardIter<Proxy> first, ForwardIter<Proxy> last)
{
  ForwardIter::difference_type dist = distance(first, last);
  return advance(first, rand() % dist);
}
```

User code

```
// v is a std::vector<int>
int elem = *random_elem(v.begin(), v.end());
```

Conclusion

- Contribution
  - The Algorithms Library
  - Runtime Concepts on Container elements
- Future Work
  - Template instantiation at runtime
  - Integration with Open-Multimethods

Thank you!

T. Becker.

Type erasure in C++: The glue between object oriented and generic programming.
In *MPOOL Workshop*, July 2007.

L. Bourdev and J. Järvi.

Efficient run-time dispatching in generic programming with minimal code bloat.
In *Workshop of Library-Centric Software Design at OOPSLA'06, Portland Oregon*, Oct. 2006.

J. C. Dehnert and A. A. Stepanov.

Fundamentals of Generic Programming.
In *International Seminar on Generic Programming*, pages 1–11, London, UK, 2000. Springer-Verlag.

D. Gregor, J. Järvi, J. Siek, B. Stroustrup, G. Dos Reis, and A. Lumsdaine.

Concepts: linguistic support for generic programming in C++.
In *OOPSLA '06*, pages 291–310, New York, USA, 2006. ACM Press.

J. Järvi, M. A. Marcus, and J. N. Smith.

Library composition and adaptation using C++ concepts.
In *GPCE '07*, pages 73–82, New York, NY, USA, 2007. ACM Press.

M. Marcus, J. Järvi, and S. Parent.

Runtime polymorphic generic programming—mixing objects and concepts in ConceptC++.
In *MPOOL Workshop*, July 2007.

S. Parent.

Beyond objects: Understanding the software we write.
Presentation at C++ connections, November 2005.

P. Pirkelbauer, Y. Solodkyy, and B. Stroustrup.

Open multi-methods for C++.
In *GPCE '07*, pages 123–134, New York, NY, USA, 2007. ACM Press.

J. Smith.

Draft proposal for adding multimethods to C++.
Technical Report N1463, JTC1/SC22/WG21 C++ Standards Committee, 2003.