C++ and Beyond

Meyers Sutter Alexandrescu

December 13-16, 2010

Snoqualmie
Washington
USA

artima

Image: NASA

C++ and Beyond 2010 *Sample*

Thank you for downloading this sample from the official presentation materials for the *C++ and Beyond 2010* event, presented by Scott Meyers, Herb Sutter, and Andrei Alexandrescu. If you'd like to purchase the complete copy of these materials, please visit:

http://www.artima.com/shop/cpp_and_beyond_2010

Artima Press is an imprint of Artima, Inc.
P.O. Box 305, Walnut Creek, California 94597

All information and materials in this document are provided "as is" and without warranty of any kind.

C++ and Beyond

Snoqualmie, Washington, USA
October 24-27, 2010

Schedule

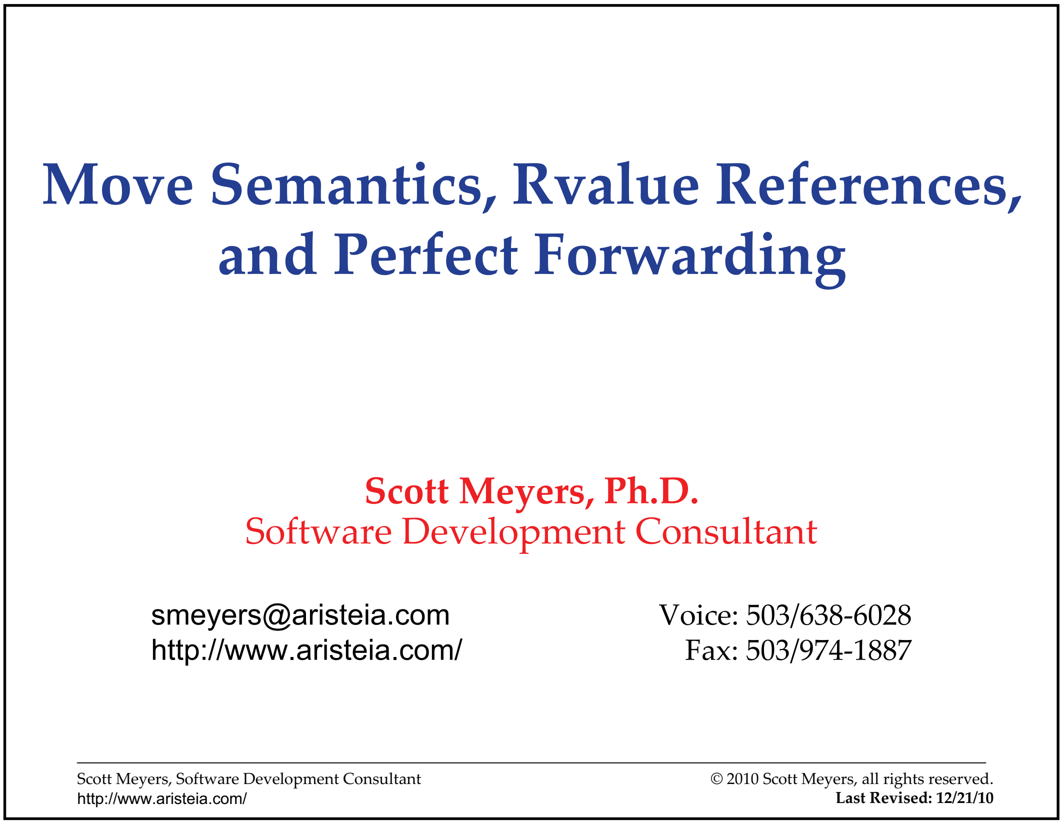
	Sunday	Monday	Tuesday	Wednesday
8:00 - 9:00		**Group Breakfast** [Attic]	**Group Breakfast** [Attic]	**Group Breakfast** [Attic]
9:00 - 10:30		**Welcome** [Andrei] **Move Semantics, Rvalue References, and Perfect Forwarding, Part 1** [Scott]	**A Fresh Look at Containers and Iterators** [Andrei]	**Elements of Design, Part 1** [Herb]
		Break	Break	Break
10:45 - 12:00		**Move Semantics, Rvalue References, and Perfect Forwarding, Part 2** [Scott]	**CAS-Based Concurrency** [Andrei]	**Elements of Design, Part 2** [Herb]
12:00 - 2:30		**Group Lunch and Mid-Day Activity**	**Group Lunch and Mid-Day Activity**	**Group Lunch and Mid-Day Activity**
2:30 - 3:45		**Lambdas, Lambdas, Everywhere** [Herb]	**CPU Caches and Why You Care** [Scott]	**Super Size Me: Lessons Learned Working at a Web Company** [Andrei]
		Break	Break	Break
4:00 - 5:00		**Ask us anything...in advance** [Panel]	**Points and Counterpoints** [Panel]	**Ask us anything...live!** [Panel]
5:00 - 7:30		Free Time (No Official C&B-Related Activities)		
7:30 - 9:30	**Reception** [Ballroom]	**Informal Discussions** [Falls Terrace & Ballroom]	**Informal Discussions** [Falls Terrace & Ballroom]	

C++ and Beyond

Snoqualmie, Washington, USA
December 13-16, 2010

Schedule

	Monday	Tuesday	Wednesday	Thursday
8:00 - 8:45		**Group Breakfast** [Attic]	**Group Breakfast** [Attic]	**Group Breakfast** [Attic]
8:45 - 9:00		**Welcome**	**Announcements**	**Announcements**
9:00 - 10:30		**Move Semantics, Rvalue References, and Perfect Forwarding, Part 1** [Scott]	**Elements of Design, Part 1** [Herb]	**Scalable Use of the STL** [Andrei]
		Break	Break	Break
10:45 - 12:00		**Move Semantics, Rvalue References, and Perfect Forwarding, Part 2** [Scott]	**CAS-Based Concurrency** [Andrei]	**Elements of Design, Part 2** [Herb]
12:00 - 2:30		**Group Lunch** **and** **Mid-Day Activity**	**Group Lunch** **and** **Mid-Day Activity**	**Group Lunch** **and** **Mid-Day Activity**
2:30 - 4:00		**Lambdas, Lambdas, Everywhere** [Herb]	**CPU Caches and Why You Care** [Scott]	**Super Size Me: Lessons Learned Working at a Web Company** [Andrei]
		Break	Break	Break
4:15 - 5:00		**Informal C++0x Feature Overview** [Scott, Herb, Andrei]	**Q&A** [Andrei, Scott, Herb]	**Q&A** [Herb, Andrei, Scott]
5:00 - 7:30		Free Time (No Official C&B-Related Activities)		
7:30 - 9:30	**Reception** [Falls Terrace]	**Informal Discussions** [Falls Terrace]	**Informal Discussions** [Falls Terrace]	



Move Semantics, Rvalue References, and Perfect Forwarding

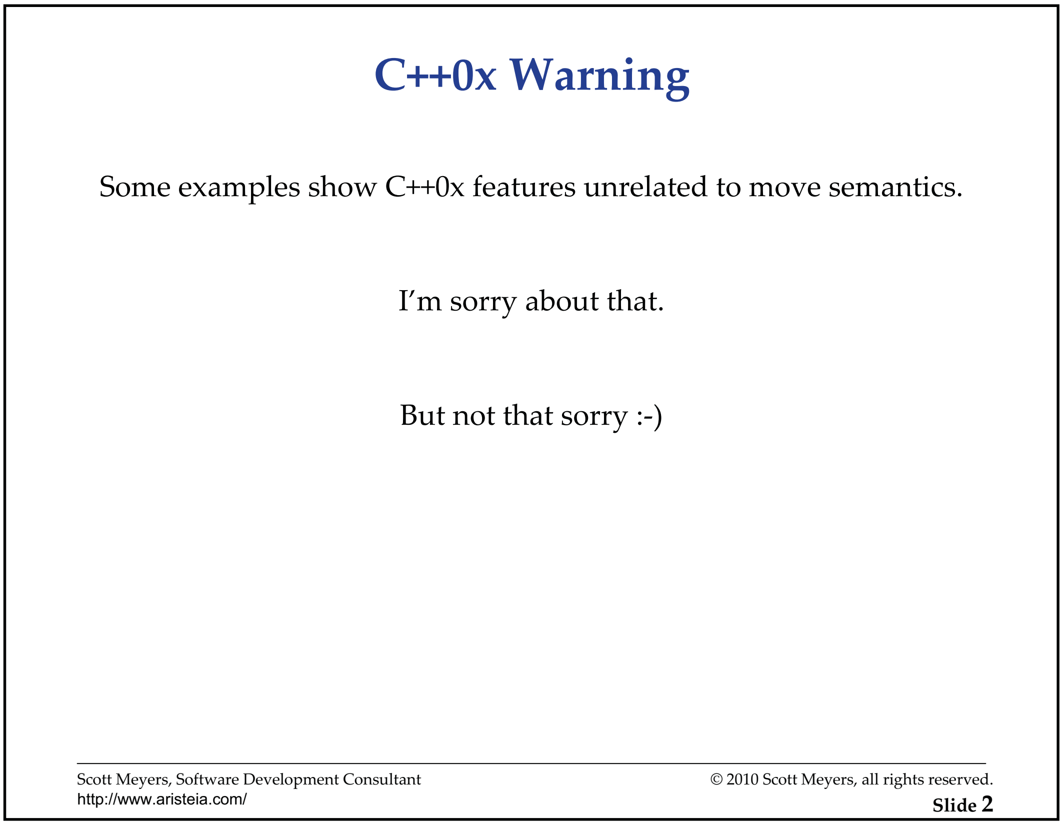
Scott Meyers, Ph.D.
Software Development Consultant

smeyers@aristeia.com
http://www.aristeia.com/

Voice: 503/638-6028
Fax: 503/974-1887

Scott Meyers, Software Development Consultant
http://www.aristeia.com/

Last Revised: 12/21/10

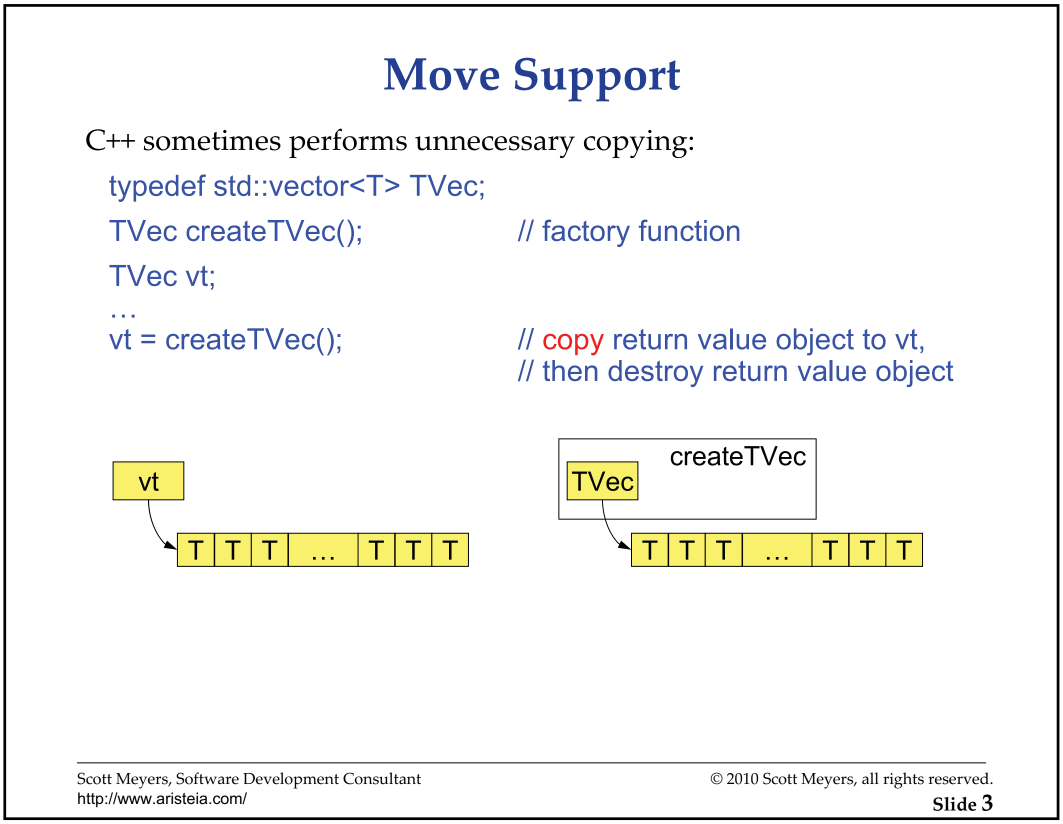


C++0x Warning

Some examples show C++0x features unrelated to move semantics.

I'm sorry about that.

But not that sorry :-)

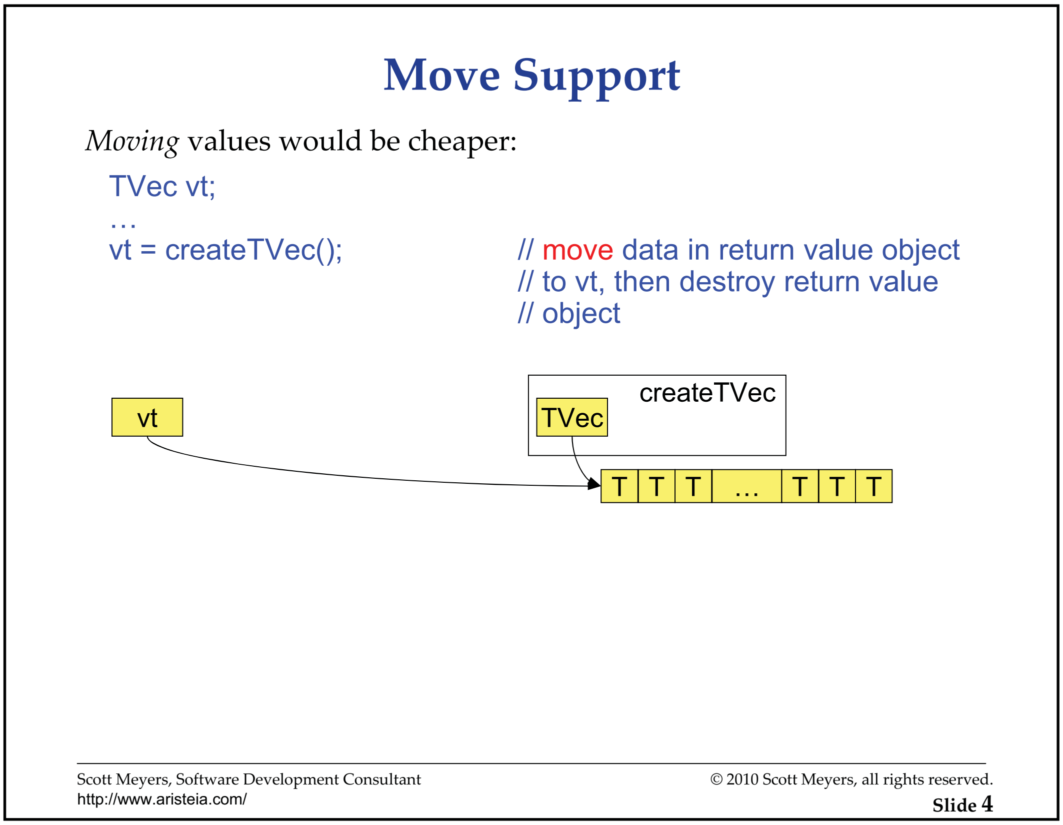


Move Support

C++ sometimes performs unnecessary copying:

```
typedef std::vector<T> TVec;
TVec createTVec();              // factory function
TVec vt;
…
vt = createTVec();              // copy return value object to vt,
                                // then destroy return value object
```

vt
T T T … T T T
createTVec
TVec
T T T … T T T

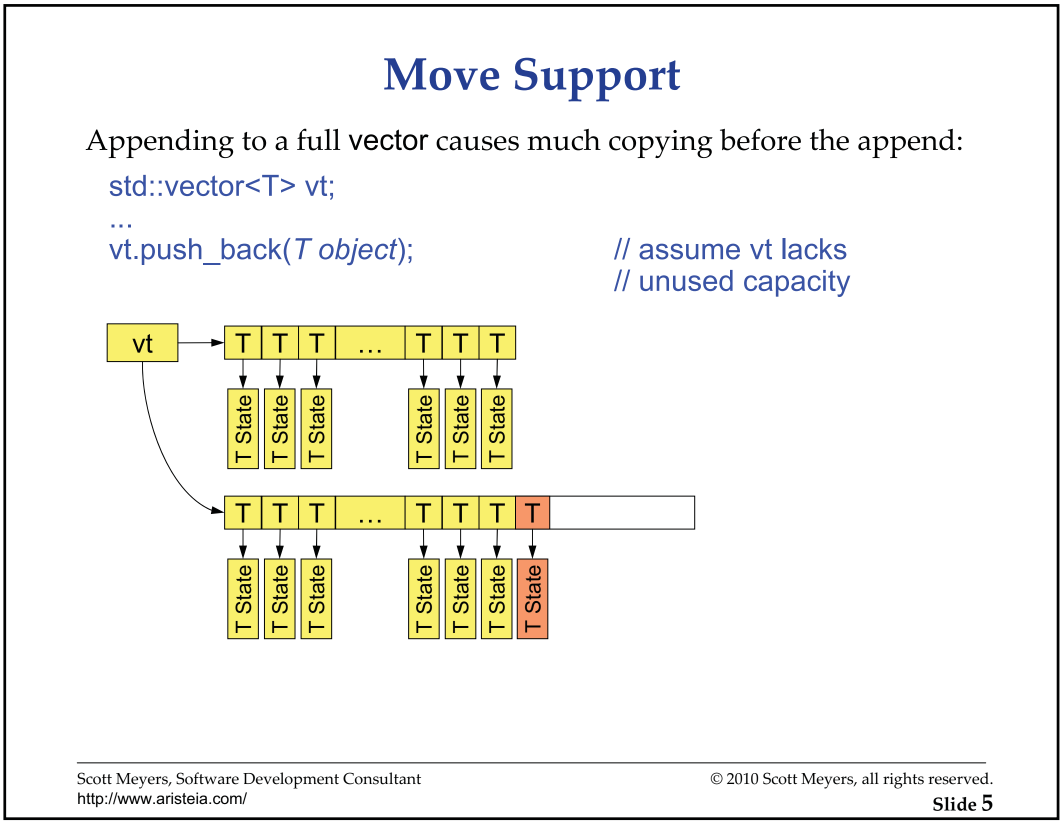


Move Support

Moving values would be cheaper:

```
TVec vt;
…
vt = createTVec();              // move data in return value object
                                // to vt, then destroy return value
                                // object
```

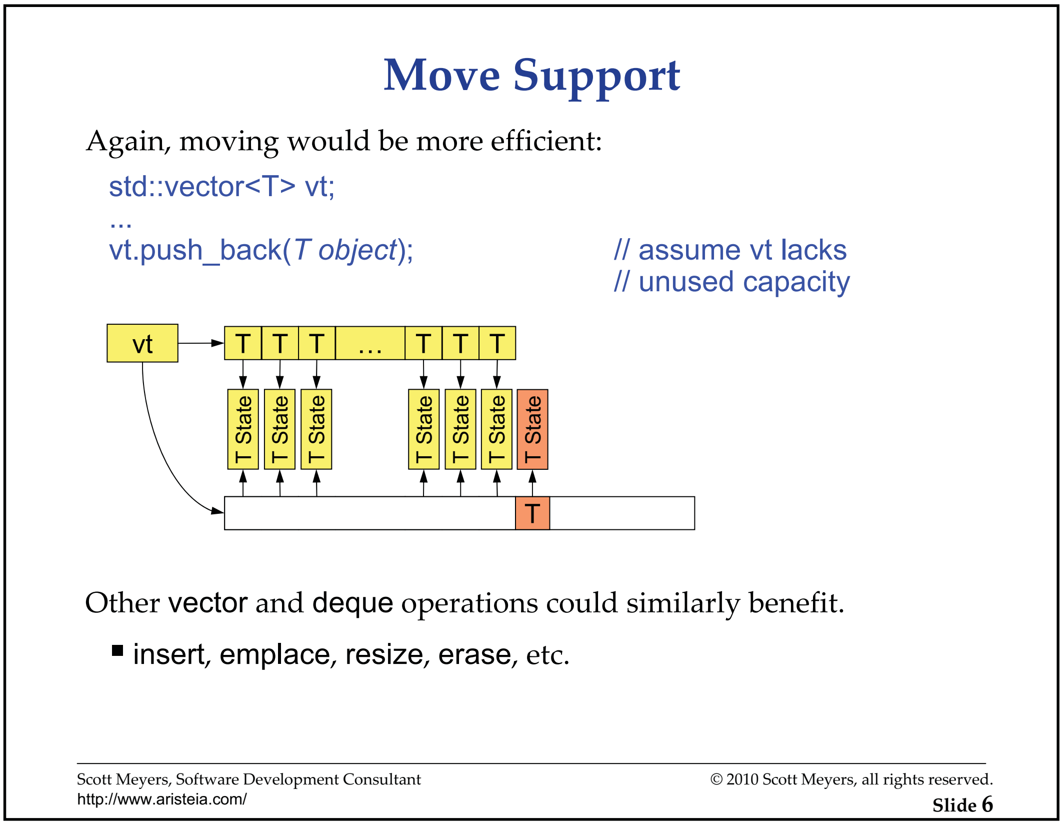
vt
createTVec
TVec
T T T … T T T



Move Support

Appending to a full vector causes much copying before the append:

```
std::vector<T> vt;
...
vt.push_back(T object);                 // assume vt lacks
                                        // unused capacity
```



Move Support

Again, moving would be more efficient:

```
std::vector<T> vt;
...
vt.push_back(T object);                 // assume vt lacks
                                        // unused capacity
```

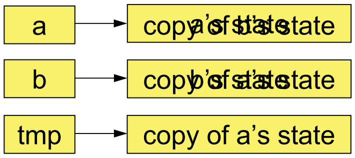
Other vector and deque operations could similarly benefit.

- insert, emplace, resize, erase, etc.

Move Support

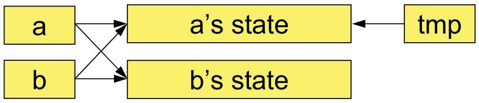
Still another example:

```
template<typename T>          // straightforward std::swap impl.
void swap(T& a, T& b)
{
  T tmp(a);                   // copy a to tmp (⇒ 2 copies of a)
  a = b;                      // copy b to a (⇒ 2 copies of b)
  b = tmp;                    // copy tmp to b (⇒ 2 copies of tmp)
}                             // destroy tmp
```

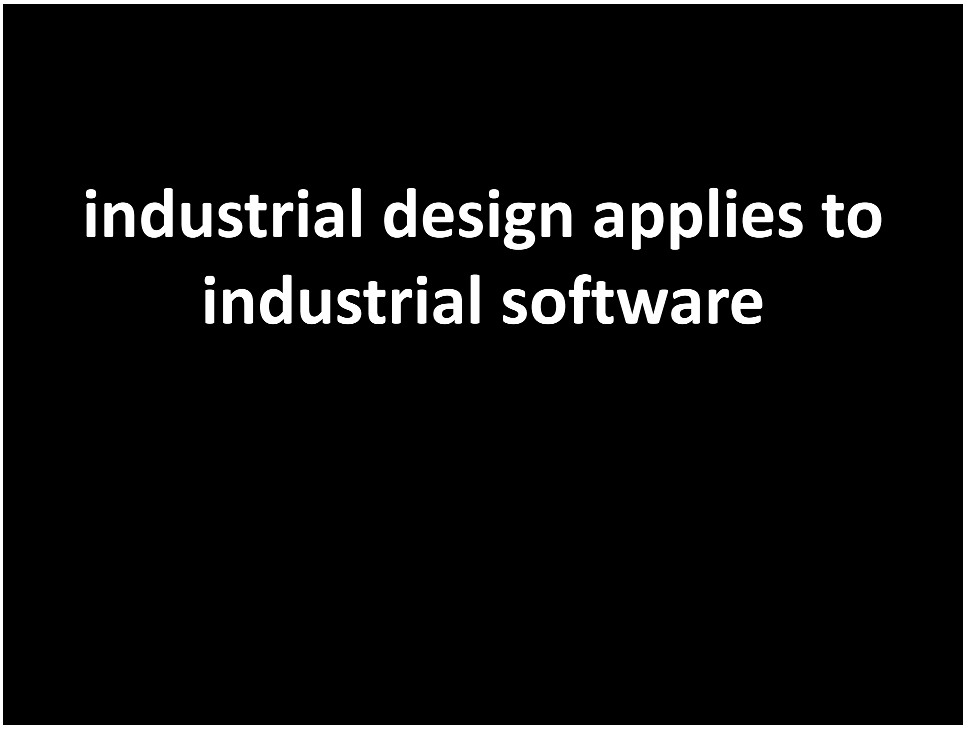


Move Support

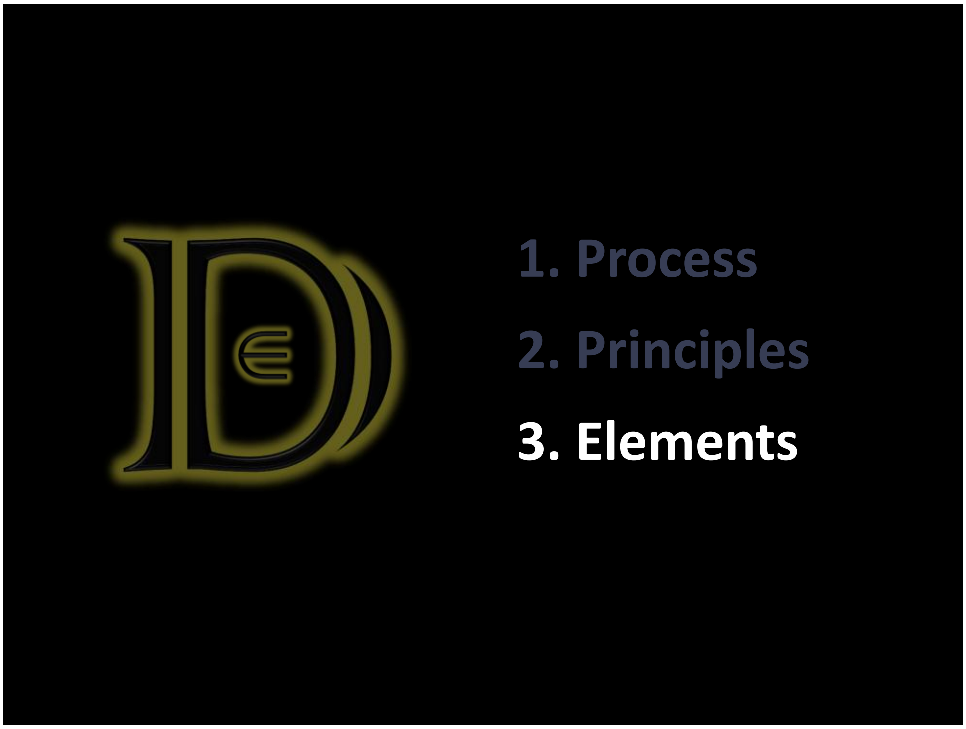
```
template<typename T>          // straightforward std::swap impl.
void swap(T& a, T& b)
{
  T tmp(std::move(a));        // move a's data to tmp
  a = std::move(b);           // move b's data to a
  b = std::move(tmp);         // move tmp's data to b
}                             // destroy (eviscerated) tmp
```



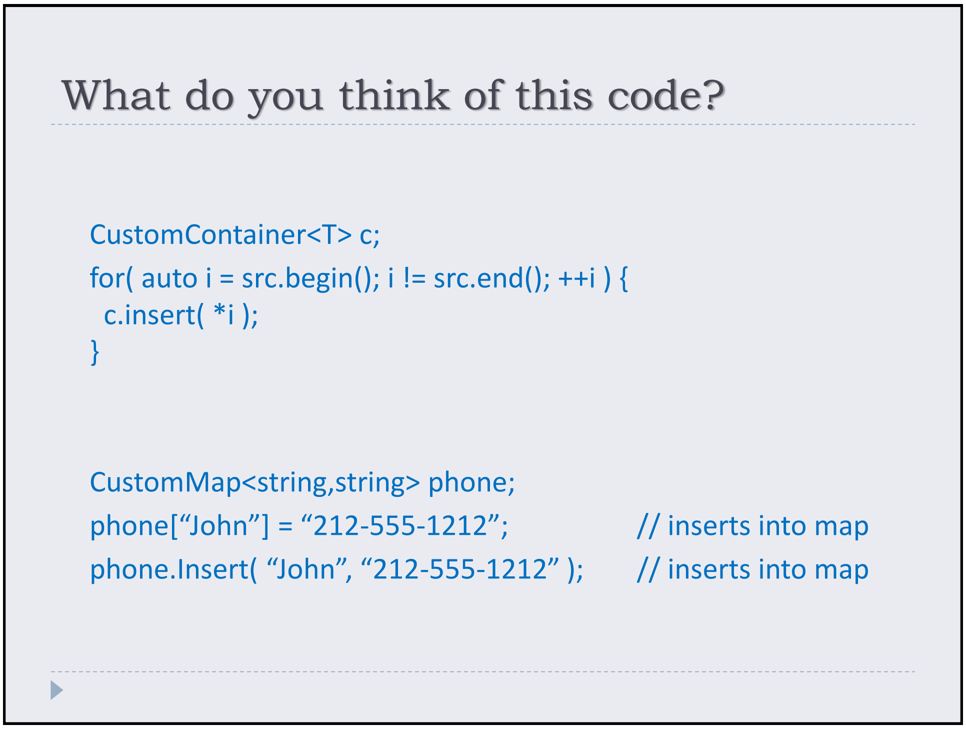
Herb Sutter



industrial design applies to industrial software



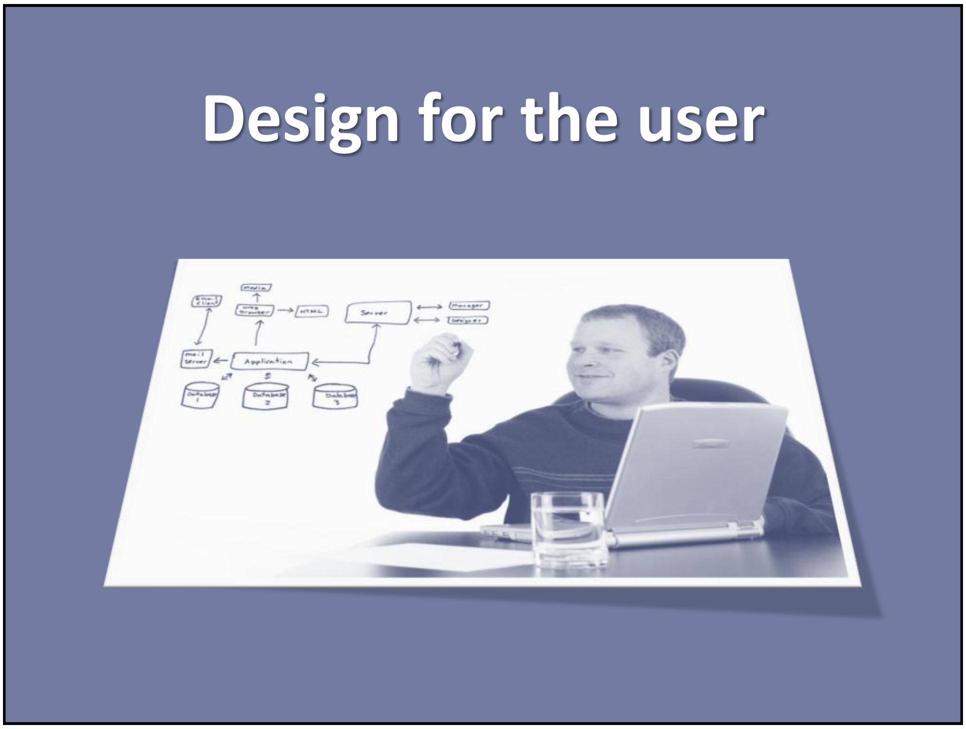
1. Process
2. Principles
3. Elements



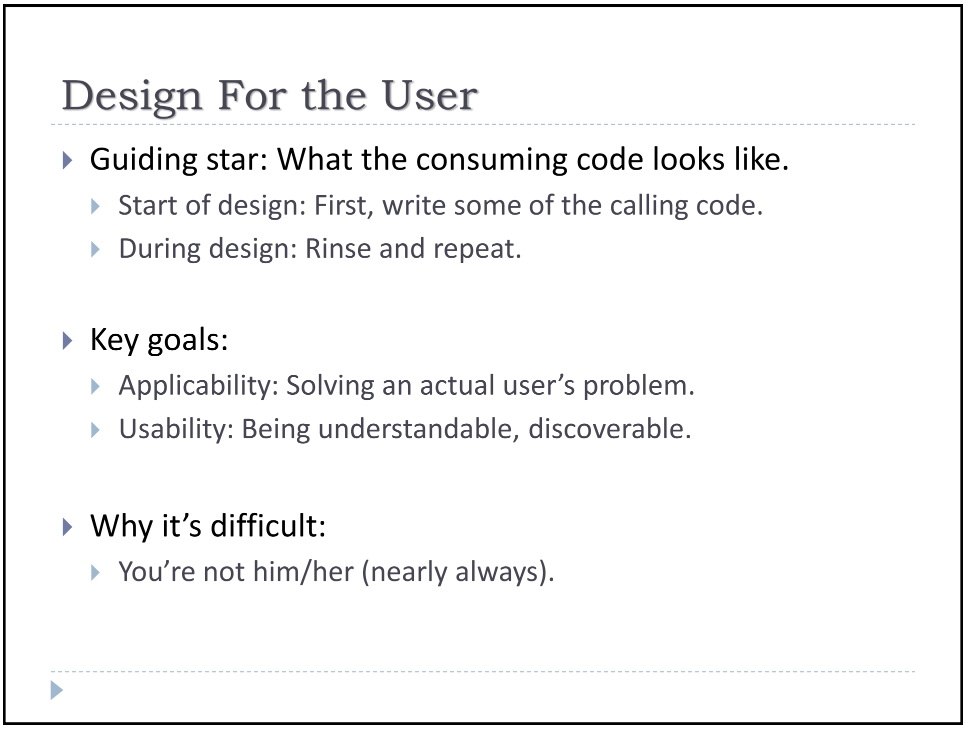
What do you think of this code?

```
CustomContainer<T> c;
for( auto i = src.begin(); i != src.end(); ++i ) {
  c.insert( *i );
}


CustomMap<string,string> phone;
phone[“John”] = “212-555-1212”;               // inserts into map
phone.Insert( “John”, “212-555-1212” );       // inserts into map
```

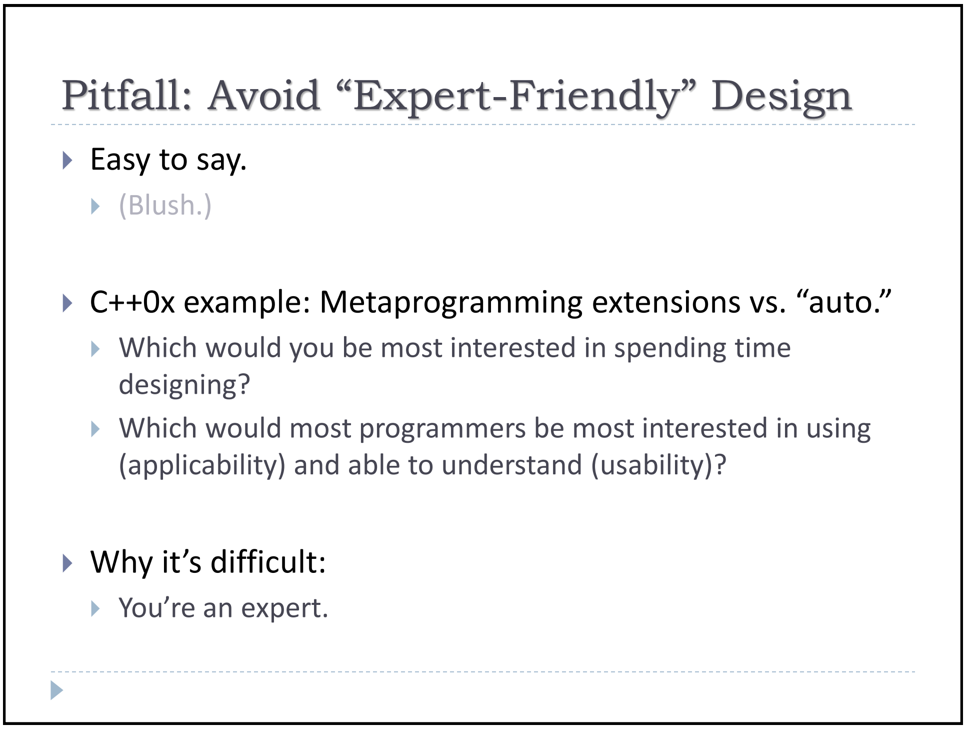


Design for the user



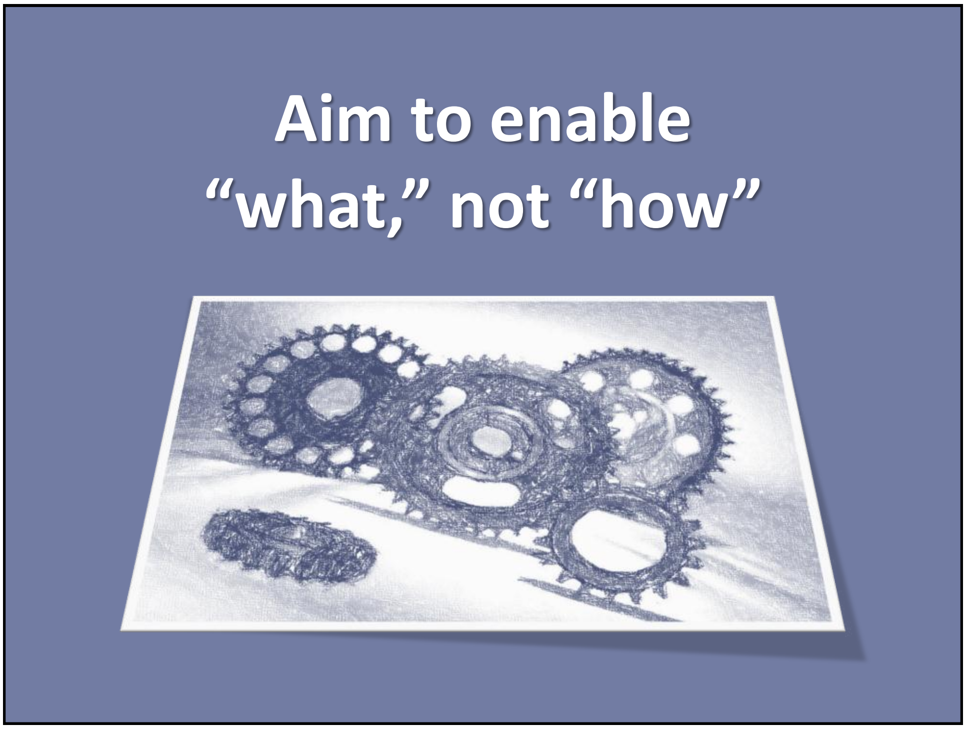
Design For the User

- Guiding star: What the consuming code looks like.
  - Start of design: First, write some of the calling code.
  - During design: Rinse and repeat.
- Key goals:
  - Applicability: Solving an actual user’s problem.
  - Usability: Being understandable, discoverable.
- Why it’s difficult:
  - You’re not him/her (nearly always).



Pitfall: Avoid “Expert-Friendly” Design

- Easy to say.
  - (Blush.)
- C++0x example: Metaprogramming extensions vs. “auto.”
  - Which would you be most interested in spending time designing?
  - Which would most programmers be most interested in using (applicability) and able to understand (usability)?
- Why it’s difficult:
  - You’re an expert.



Aim to enable “what,” not “how”

CAS-Based Concurrency

Prepared for C++ and Beyond 2010

Andrei Alexandrescu, PhD
`andrei@erdani.com`

This Talk

- Lock-free programming: Brief history and introduction
- CAS-based code
- A Singly-Linked Lock-Free List

Motto

“Multithreading is just one darn thing after, before, or simultaneously with another”.

Lock-free Programming: Brief History and Introduction

Defining Terms

- *Wait-free procedure:* completes in a bounded number of steps regardless of the relative speeds of other threads
- *Lock-free procedure:* at any time, at least one thread is guaranteed to make progress
  - Probabilistically, all threads will finish timely
- Mutex-based procedures
  - Not wait-free
  - Not lock-free

A Different Angle

- *Lock-based:* ask for synchronization device prior to operation
- Pessimistically assumes contention

- *Wait/Lock-free:* Perform operation, attempt to commit
- Optimistically assumes no contention
- “Better ask for forgiveness than permission”

Brief History

- Lock-based threading theory established in the 1960s
  - Still the dominant model today
- By 1972—efforts to avoid mutex-based pessimistic concurrency control
  - Atomic assignment
  - Use of atomic instructions: increment, test-and-set
- By 1990—search for universal atomic primitive that would enable all others
- 1991: “Wait-free synchronization” by Herlihy settles the matter

Impossibility/Universality

- Some primitives cannot synchronize any shared data structure for >2 threads
  - test-and-set
  - fetch-and-add
  - atomic queues!
- Some other primitives are enough to implement any shared data structure
  - e.g., CAS