BoostCon 2010

C++0x: the Dawn of a new Standard

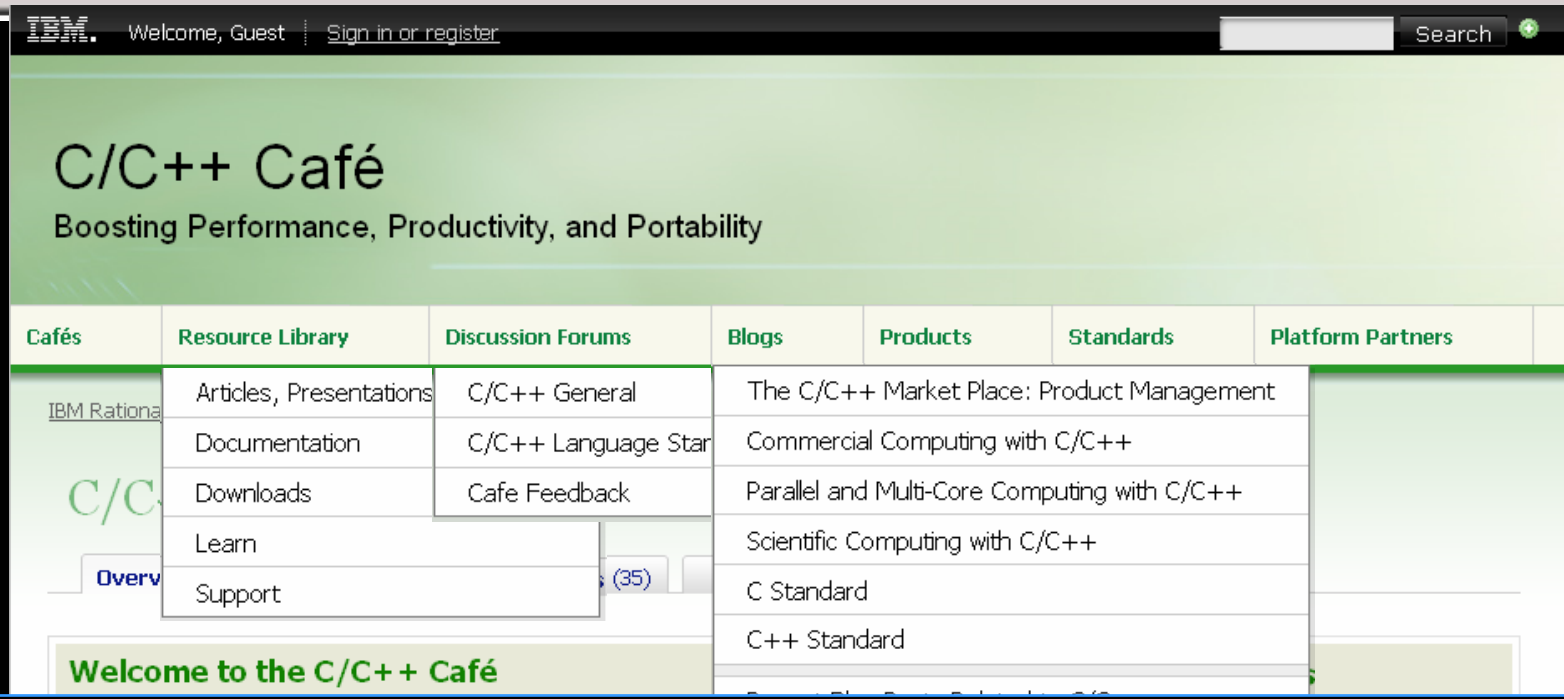
Michael Wong
michaelw@ca.ibm.com
IBM Toronto Lab
Canadian C++ Standard Committee

IBM Rational Disclaimer

- **The information contained in these materials is provided for informational purposes only, and is provided AS IS without warranty of any kind, express or implied. IBM shall not be responsible for any damages arising out of the use of, or otherwise related to, these materials. Nothing contained in these materials is intended to, nor shall have the effect of, creating any warranties or representations from IBM or its suppliers or licensors, or altering the terms and conditions of the applicable license agreement governing the use of IBM software. References in these materials to IBM products, programs, or services do not imply that they will be available in all countries in which IBM operates. Product release dates and/or capabilities referenced in these materials may change at any time at IBM's sole discretion based on market opportunities or other factors, and are not intended to be a commitment to future product or feature availability in any way. IBM, the IBM logo, Rational, the Rational logo, Telelogic, the Telelogic logo, and other IBM products and services are trademarks of the International Business Machines Corporation, in the United States, other countries or both. Other company, product, or service names may be trademarks or service marks of others.**

The IBM Rational C/C++ Café

ibm.com/rational/cafe/community/ccpp



Agenda

- **C++0x, goals, examples**
- **C++ Standard timeline, state, documents, features**
- **Compiler status**
- **Features and summary**
- **Q/A**

Agenda

- **C++0x, goals, examples**
- C++ Standard timeline, state, documents, features
- Compiler status
- Features and summary
- Q/A

C++0x goals

- Overall goals
  - Make C++ a better language
    - for systems programming
    - for library building
  - Make C++ easier to teach and learn
    - generalization
    - better libraries
- Massive pressure for
  - More language features
  - Stability / compatibility
    - Incl. C compatibility
- Insufficient pressure for
  - More standard libraries
    - The committee doesn't have the resources required for massive library development

C++0x: areas of language change

- Machine model and concurrency
  - Memory model
  - Threads library, thread pools, futures
  - Atomic API
  - Thread-local storage
- Support for generic programming
  - concepts
  - **auto**, **decltype**, template aliases, Rvalue references, …
  - initialization
- Etc.
  - improved **enum**s
  - **long long**, C99 character types, etc.
  - …
- Modules and dynamically linked libraries
  - Postponed for a TR

Removed June 2009

Is this legal C++03 syntax?

```
template<class T> using Vec = vector<T,My_alloc<T>>;
Vec<double> v = { 2.3, 1.2, 6.7, 4.5 };
sort(v);
for(auto p = v.begin(); p!=v.end(); ++p)
        cout << *p << endl;
```

Hello Concurrent World

```
#include <iostream>
#include <thread> //#1
void hello() //#2
{
    std::cout<<"Hello Concurrent World"<<std::endl;
}
int main()
{
    std::thread t(hello); //#3
    t.join(); //#4
}
```

Is this valid C++ today? Are these equivalent?

```
int x = 0;

atomic<int> y = 0;

Thread 1:
  x = 17;
  y.store(1,
  memory_order_release);
  // or:       y.store(1);


Thread 2:
  while
  (y.load(memory_order_acquir
  e) != 1)
  // or:       while
  (y.load() != 1)

  assert(x == 17);
```

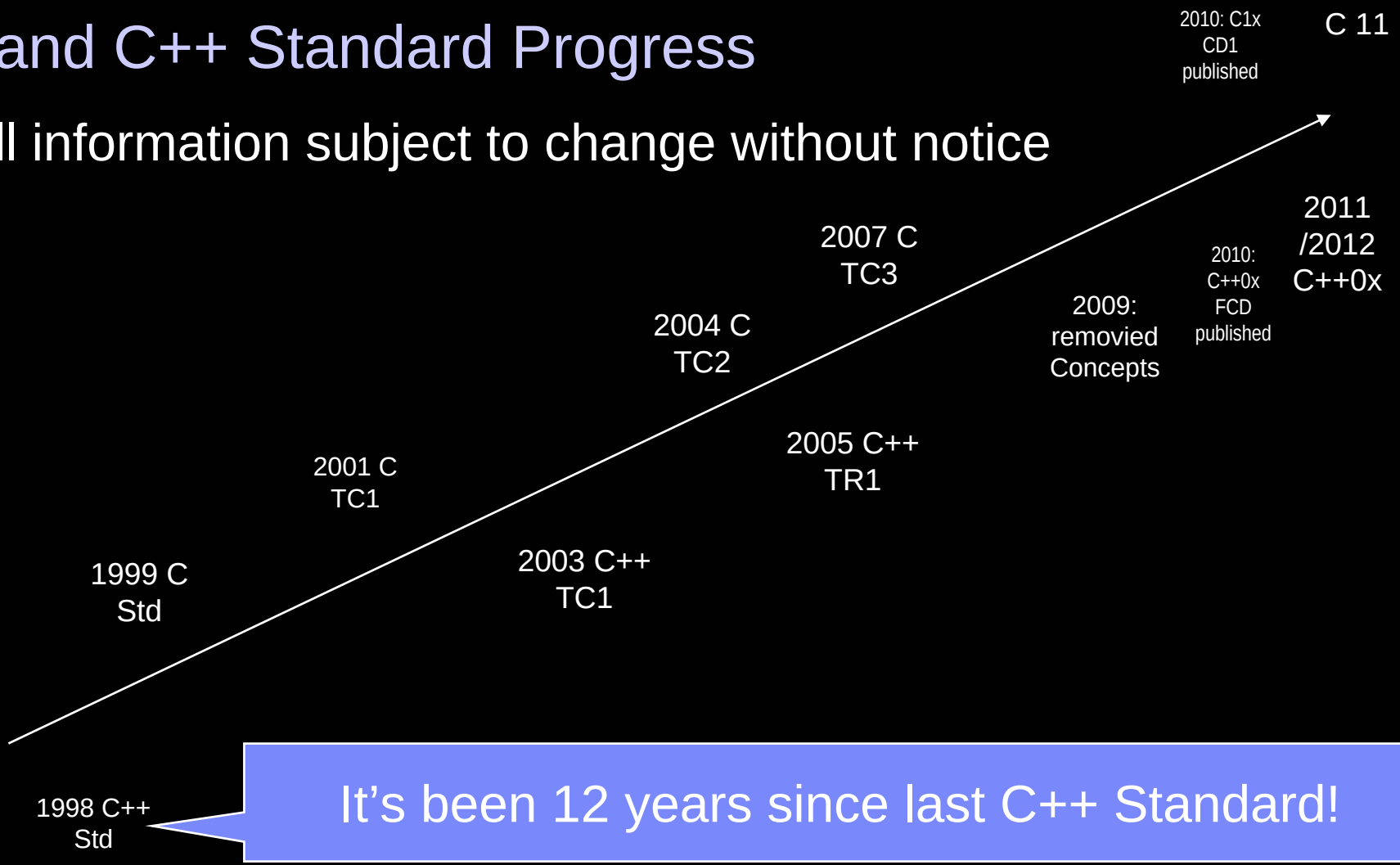
```
int x = 0;

atomic<int> y = 0;

Thread 1:
  x = 17;
  y = 1;

Thread 2:
  while (y != 1)
        continue;
  assert(x == 17);
```

Agenda

- C++0x, goals, examples
- **C++ Standard timeline, state, documents, features**
- Compiler status
- Features and summary
- Q/A

C and C++ Standard Progress

All information subject to change without notice



C++0x

- **Codename for the planned new standard for the C++ programming language**
  - Will replace existing ISO/IEC 14882 standard published in 1998 (C++98) and updated in 2003 (C++03)
  - Many new features to core language
  - Many library features: most of C++ Technical Report 1 (TR1)
  - Was aiming for ratification 2009, now is looking at 2010/2011
  - X=9,A,B,C,D,E,F?

Major stages of C++0x remaining

Major Stages

SC22 Reg. Ballot (complete)	Ideally all major features present. Usually few comments.
SC22 CD Ballot (optional, 3 months)	Nearly all major features in near-final form. After ballot, need to allow time for disposition of comments and completion of all major features.
SC22 FCD Ballot * (required, 4 months)	All major features in essentially final form. After ballot, need to allow time for disposition of comments.
JTC1 FDIS Ballot (required, 2 months + publication time)	Final text. **Note:** This is an up-down ballot, and no comments are allowed. The only way for a NB to express displeasure is to vote No on the whole standard.

* JTC1 is planning to replace the FCD stage with the ISO DIS stage. This would extend the ballot period to 5 months, but the change is not expected to happen in time to affect us.

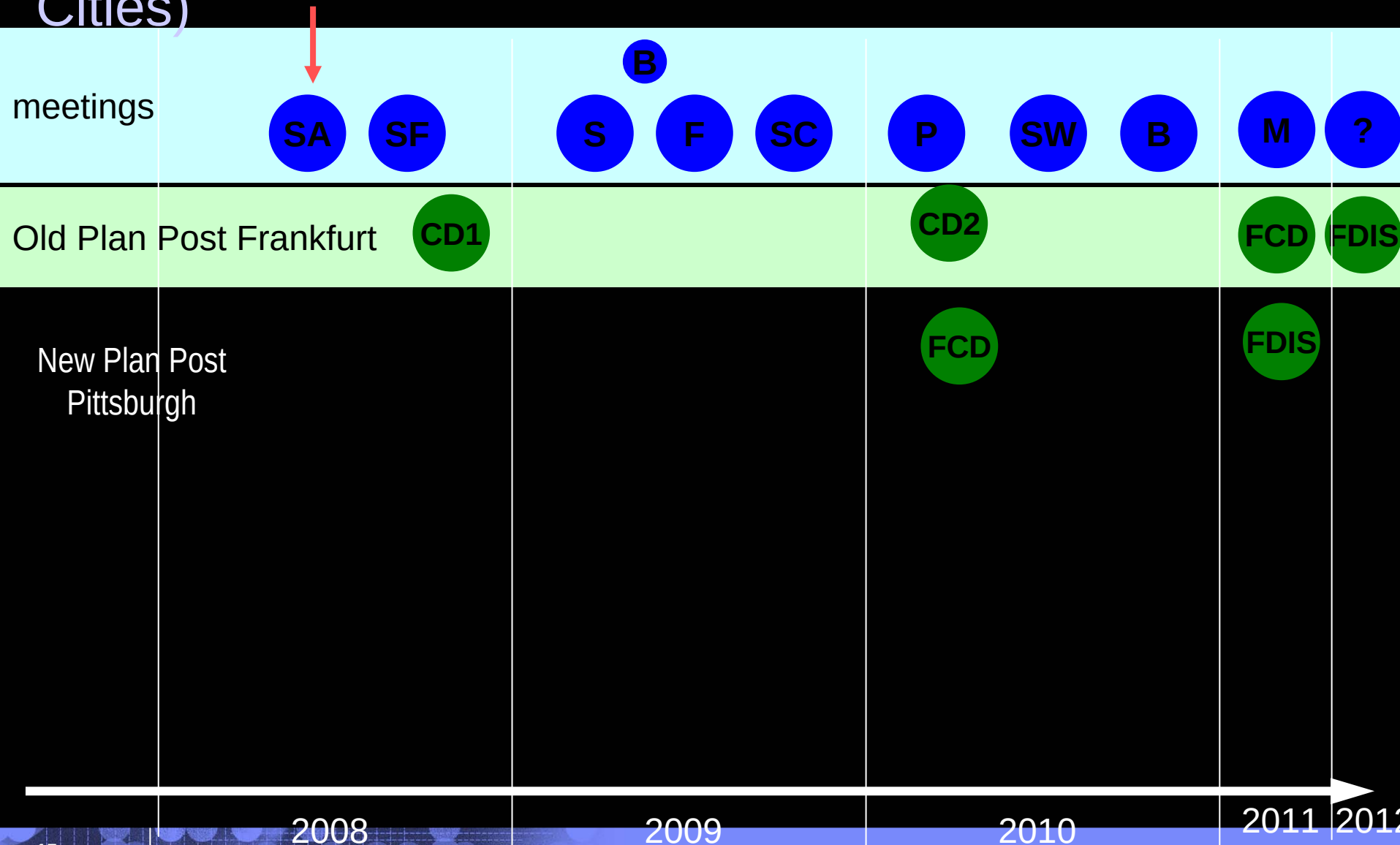
DONE in 9/2007

DONE in 9/2008

Delayed 3/2011? NO!

3/2010

Post-Pittsburgh schedule (Letters are Meeting Cities)



C++ Standard timelines

Past Meetings	**Target Progress**
June 8-14, 2008 Sophia Antipolis	Complete features All papers voted into draft
Sept 14-20, 2008 San Francisco	Complete features Ship CD 1 5 months (3 ballots + 1 buffer)
Mar 1-8, 2009 Summit	Resolve comments
July 12-18, 2009 Frankfurt	Resolve comments Removed Concepts
Oct 18-24, 2009 **Santa Cruz**	**Resolve comments**

Future Meetings	Target Progress
Mar 8-13, 2010 Pittsburg	Resolve Comments Ship FCD 5 months (4 ballots + 1 buffer)
Aug 2-7, 2010 Rapperswil, Switzerland	Resolve Comments
Nov 8-13, 2010 Batavia, IL	Resolve comments
Mar 21-26, 2011 Madrid	Resolve Comments Ship FDIS? >= 6 months (2 ballots +>=4 publications)

What are the STD documents and their status?

- **Library TR1:** Draft Technical Report
- **C++0x:** Final Committee Draft, has 13/14 TR1 libraries
- **Special Math Library:** Final Committee Draft
- **Decimal Floating Point TR:** Proposed Draft Technical Report
- **POSIX C++:** working draft, target 2011
- **C++ ABI:** working draft, ongoing discussion on mangling, and common-vendor interoperability

What's in?

- **Memory Model and Concurrency [N2138]**
- **Concurrent Libraries [N2094]**
- **Initialization [N2116]**
- **Rvalue references [N2118]**
- **Other primary features**
  - Constant expressions, automatic types
- **Expanded Library from most of TR1**
- **140 features, 600 bug fixes to the standard**
- **What's out?**
  - Concepts [N2081]
  - Garbage Collection (Replaced by smaller proposal)

Feature and defect count

- **Language**
  - 70 features
  - 300 defects (in the C++ Standard)
- **Runtime**
  - 70 features
  - 230 defects (in the C++ Standard)
- **Too many features to be done in one release**
  - stage across many compiler releases over several years
  - not all Standard defects translate into compiler issues

C++ CD1 National Body Comment status

	Unresolved	Accepted	Modified	Rejected	Total
CWG	93	32	8	43	176
LWG	128	36	5	35	204
Ed	28	148	14	21	211
Total	249	216	27	99	591

Agenda

- C++0x, goals, examples
- C++ Standard timeline, state, documents, features
- **Compiler status**
- Features and summary
- Q/A

Overall C++0x Delivery Strategy

- **Phase in features over many compiler releases, and several year**
- **Ratification is still 1-2 yrs out, but we need to start NOW to finish in reasonable time!**
- **Feature selection criteria**
  - Features that are furthest along in standardization, or the simplest that won't likely change before ratification
  - Features that are base dependencies of more complex features
  - Features that are requested by customers, or provide an immediate benefit, such as performance, or compiler time, or porting
  - Features required by the C++ 0x Standard Library
  - Features that are already exposed by other compilers, and may show up in customer or Boost code
  - Bjarne Stroustrup (father of C++) has given his thought on implementation order
    - http://www.research.att.com/~bs/C++0xFAQ.html#order

IBM XL C++ V11.1 status (May, 2010)

- Released in XL C/C++ for AIX/Linux V10.1 in mid 2008
  - -qlanglvl=extended0x option (umbrella option for all future 0x features)
  - long long,
  - sync C99 preprocessor (Empty macro arguments, Variadic macros, Trailing comma in enum definition, Concatenation of mixed-width string literals)
- in zOS XL C/C++ V1R10 (include all of above)
  - Extern template
  - Extended friend
  - -qwarn0x
- **In** C/C++ for AIX for V11.1, in 2Q 2010 (include all of above)
  - Variadic template
  - Auto
  - Decltype
  - Namespace association
  - Delegating constructor
  - Static assert

All information subject to change without notice

Other Compilers

- **C++0x support publicly available in 100509**
  - Micsrosoft Visual C++ 2010, Apr 12, 2010
  - GNU 4.5, Apr 14, 2010
  - IBM xlC++ 11.1, Apr 23, 2010
  - Intel 11.1 (EDG), June 23, 2009
  - HP aC++ A.06.22 (EDG), Dec, 2008
  - Comeau 4.3.10.1 (EDG), Oct 6, 2008
  - Borland/CodeGear C++ Builder 2009 Compiler 6.10, 2H 2008
- **No C++0x features available publicly as of 100509 on their latest compiler, but we do know from their blogs about their future plans**
  - Sun Studio 12

All information based on publicly available data

Updated page of C++0x support

- **http://wiki.apache.org/stdcxx/C%2B%2B0xCompilerSupport**
  - Maintained by Martin Sebor

GNU

- **http://gcc.gnu.org/projects/cxx0x.html**
- **4.3/4.4/4.5 support:**
  - http://gcc.gnu.org/gcc-4.3/cxx0x_status.html
  - http://gcc.gnu.org/gcc-4.4/cxx0x_status.html
  - http://gcc.gnu.org/gcc-4.5/cxx0x_status.html
- **-std=c++0x or -std=gnu++0x**
- **GNU will write their own C++0x library, libstdC++, as they have always done:**
  - http://gcc.gnu.org/onlinedocs/libstdc++/manual/status.html#id476343
  - Boost 1.42.0
- **Additional Branch**
  - Concepts
  - Lambda
  - Delegating constructors
  - Raw strings

All information based on publicly available data

GNU 4.3/4.4/4.5 (100509)

GNU 4.3	GNU 4.4	GNU 4.5
Rvalue Reference	Extending variadic template template parameters	Initializer lists
Variadic Template	Initializer lists	Lambdas
Static Assert	Auto, multideclarator auto, removing auto as storage-class specifier, new function declarator syntax	Explicit conversion
Decltype	Propagating exceptions	Raw string literals
Right Angle Bracket	Strongly-typed enums	UCN Literals
C99 Preprocessor	New character types	Extending sizeof
Extern Templates	Unicode string literals	Local and unamed types as template arguments
__func__	Standard Layout types	
Long long	Default and deleted functions	
	Inline namespaces	

Intel and likely HP/Comeau (use EDG frontend)

- **Intel C++ 11.1 has**
  - -qstd=c++0x (Linux/Mac OS X), /Qstd:c++0x (Windows)
  - Static_assert
  - Auto type specifier/decltype operator
  - Extern template
  - Long long
  - Extended friend
  - C99 Preprocessor
  - Use of ">>" to close two template argument lists
  - compliant __func__
  - lambda expressions (pre-0x)
- **Intel C++ Standard Library is based on Microsoft on Windows (uses Dinkwumare) and GNU on Linux (uses GNU's libstdC++), Boost 1.39**
- **HP aC++ V6 has been quiet about their C++ support, but will likely peggy-back on EDG as they move to new versions, uses STLport 5.1.7 as C++ Library, libstd runtime library matches Rogue Wave Version 1.2.1. , libstd_v2 runtime library matches Rogue Wave Version 2.02.01. Boost 1.38**
- **Comeau is also very active in delivering C++0x as soon as EDG delivers it to them, runs on multiple platforms, uses their own libcomo 36 based on an old SGI C++ Std Library**

All information based on publicly available data

MS VS C++ 2010

- **http://blogs.msdn.com/vcblog/archive/2010/04/06/c-0x-core-language-features-in-vc10-the-table.aspx**
  - Lambdas
  - Auto
  - Static_assert
  - Rvalue references
  - decltype
  - Nullptr
  - Extern termplates
  - Right angle brackets
  - Local and unamed types as template arguments
  - Long long
  - Exception_ptr

All information based on publicly available data

Sun Studio (Version 13 and higher?)

- Steve Clamage's post (080516):
  - http://forums.sun.com/thread.jspa?threadID=5296590
  - "Right now, we are working on providing binary compatibility with g++ as an option in the next compiler release. "
  - "We won't release an official (stable, fully-supported) product with C++0X features until the standard is final. Until then, any feature could change in unpredictable ways. "
  - "Beginning some time next year, we expect to have Express releases with some C++0X features. Express releases are our way of providing compilers with experimental features that might not be stable yet. It gives our customers a chance to try them out and provide feedback before they become part of a stable release. "
- No known plans on C++0x Library based on Steve Clamage's post (070917):
  - http://forums.sun.com/thread.jspa?threadID=5165721
  - Ships with libCstd, an ancient version of Rogue Wave C++ library from 1999 for binary compatibility
  - Ships with STLport 4.5.3 for enhanced performance
  - Boost 1.34.1
  - Can work with open source Apache C++ Standard Library derived from Rogue Wave 4.1.2
  - "A new C++ standard is in progress, planned for completion in 2009. We will release a new compiler, C++ 6.0, conforming to the new standard, including a fully-conforming standard library as the default. The new library will be shipped as part of Solaris.
    We also plan to maintain compatibility with C++ 5.x and libCstd as an option. Details are still in the planning stage. "

All information based on publicly available data

Borland/CodeGear C++Builder Compiler 6.10 2009

- http://www.codegear.com/article/38534/images/38534/CBuilder2009Datasheet.pdf
- Rvalue references
- decltype
- Variadic templates (in testing)
- Scoped enumerations
- static_assert
- explicit conversion operators
- Attributes [[final]] and [[noreturn]]
- alignof
- Type traits
- Unicode character types and literals
- long long
- variadic macros
- Dinkumware C++Std Library
- Boost 1.35

All information based on publicly available data

What is C++0x?

- Simplifying simple tasks
  - **Deducing types, ranged for loops,**
- Initialization
  - **Uniform { }, no accidental narrowing**
- Support for low-level programming
  - **Standard layout types, unions, generalized constant expr**
- Tools for writing classes
  - **Init list constructor, inheriting constructor, move, user-defined literals**
- Concurrency
  - **Memory model, threads, locks, atomics, mutex, future, shared_future, atomic_future, promise, async()**
- Standard library components
  - **Containers, regular expression, random numbers, time, resource mgmt, utility, metaprogramming, Garbage collection ABI**

Sum of all things C++0x FCD

- • __cplusplus
- • alignments
- • attributes
- • atomic operations
- • auto (type deduction from initializer)
- • C99 features
- • enum class (scoped and strongly typed **enum**s)
- • copying and rethrowing exceptions
- • constant expressions (generalized and guaranteed; **constexpr**)
- • decltype
- • default template parameters for function
- • defaulted and deleted functions (control of defaults)
- • delegating constructors
- • Dynamic Initialization and Destruction with Concurrency
- • explicit conversion operators
- • extended **friend** syntax
- • extended integer types
- • extern templates
- • for statement ; see range **for** statement
- • generalized SFINAE rules
- • Uniform initialization syntax and semantics
- • unions (generalized)
- • user-defined literals
- • variadic templates
- • in-class member initializers
- • inherited constructors
- • initializer lists (uniform and general initialization)
- • lambdas
- • local classes as template arguments
- • long long integers (at least 64 bits)
- • memory model
- • move semantics; see rvalue references
- • Namespace Associations (Strong using)
- • Preventing narrowing
- • null pointer (**nullptr**)
- • PODs (generalized)
- • range for statement
- • raw string literals
- • right-angle brackets
- • rvalue references
- • static (compile-time) assertions (**static_assert**)
- • suffix return type syntax (extended function declaration syntax)
- • template alias
- • template typedef ; see template alias
- • thread-local storage (**thread_local**)
- • unicode characters

Performance Opportunities in future C++0x features

- **Improve Execution Time**
  - memory model, concurrency/atomics, rvalue references, pods, variadic template, Concepts, auto
- **Increase Compile Time**
  - Concepts, most template features (except variadic template)
- **Decrease Compile Time**
  - Variadic template
- **Improve usage/teachability**
  - Auto, initialization, decltype
- **Supports concurrency**
  - Atomics, fences, basic mutlithreading library, futures

FCD: Features for whom?

- **Library enhancements**
- **For Class writers**
  - Move, rvalue ref, deleted and default functions, delegating, inheriting
- **For Library writers**
  - Static assert, explicit conversion, variadic template, decltype
- **For you**
  - >>, auto, range-based for, nullptr, unicode, raw strings, uniform init, init lists, lambda, trailing return, template aliases, concurrency
- **For everyone else**
  - Class enum, unrestricted union, time library, local types as template args, C99 compat, scoped allocators, constexpr, user-defined literals, relaxed POD, extern template, sizeof on class data members, & and && member functions, in-class init of static data member, attributes

C++0x Library

- **Start with original C++98 library**
  - Improved performance with rvalue reference
  - Used variadic templates to improve compile time
  - Potential binary incompatibility with C++98 library strings
  - Reference counting not allowed
- **Added 13/14 TR1 libraries**
  - Reference wrapper, smart ptrs, return type determination, enhanced member pointer adapter, enhanced binder, generalized functors, type traits, random numbers, tuples, fixed size array, hash tables, regular expressions, C99 cmpat
- **Added threading, unique_ptr,forward_list, many new algorithms**

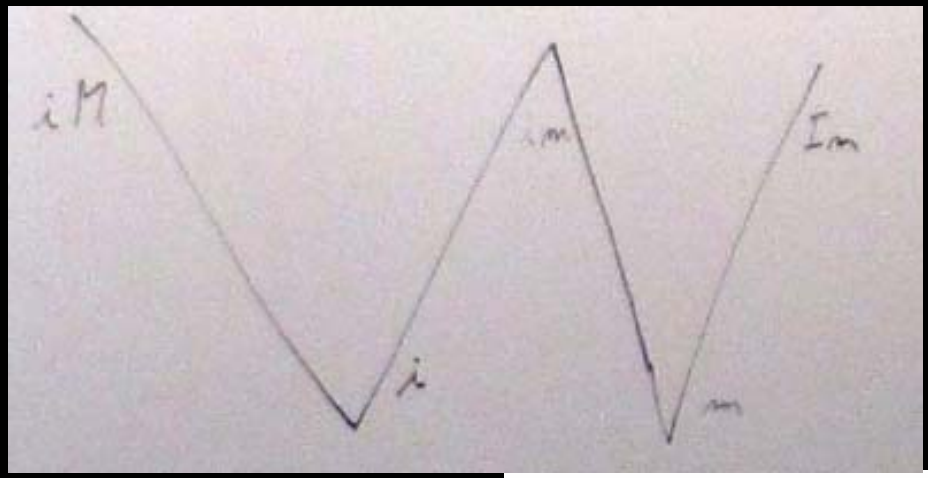
Removed or Deprecated features

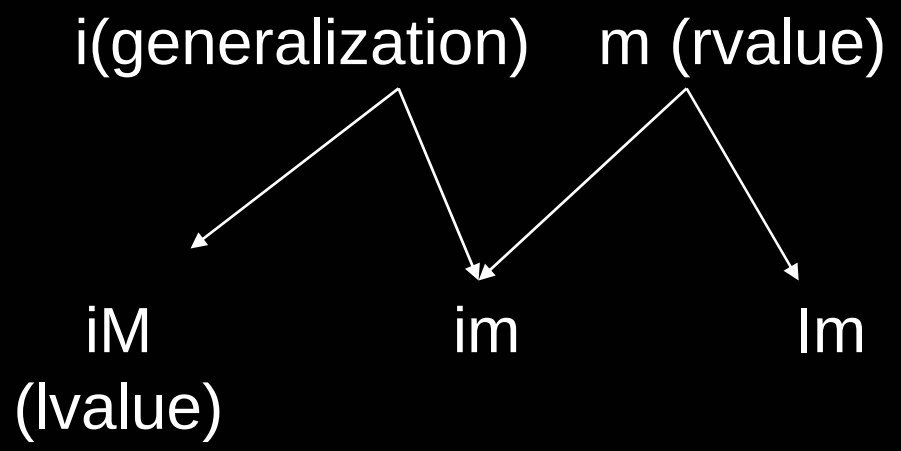
- **Auto as a storage class**
- **Export semantics**
- **Register semantics**
- **Exception specification**
- **Auto_ptr**
- **Bind1st/bind2nd**

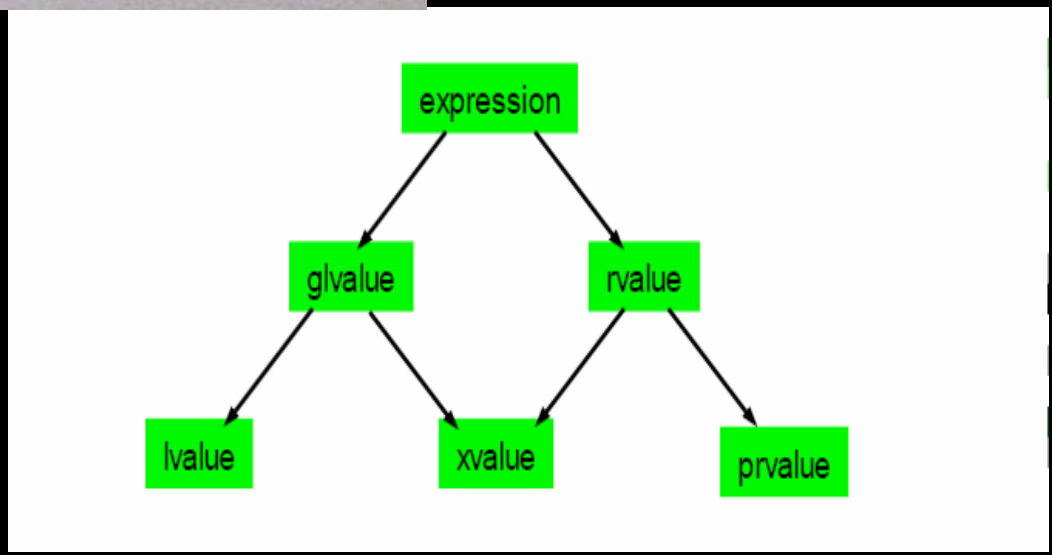
Updates from Pittsburgh 03/2010

- Value categories (was "Funny lvalues")
- **We spent a lot of time talking about the "Funny Lvalues" paper, which reworked the specification of how objects created by rvalue reference operations work.**
- **Some fix needs to be made, because the current specification doesn't deal properly with the dynamic type and the identity aspects of rvalue reference objects. It also suggests there are rvalue functions, which are not defined.**
- **The initial proposal was naive about the impact of a change of terminology on the library section and on teachability/textbooks. In particular, lots of uses of "rvalue" might have had to be changed. LWG was asked to investigate changes required.**
- **We rethought the terminology, and came up with an approach that:**
  - Solves the dynamic type, identity, and function problems.
  - Retains the current meanings of "lvalue" and "rvalue" for the library section and the outside world. No LWG changes needed.
  - Has the nice property that what an rvalue reference binds to is an rvalue, and what a lvalue reference (to nonconst) binds to is an lvalue.
- **All expressions are now divided into three "value categories":**
  - "lvalues" are the same as what's meant traditionally by lvalue.
  - "xvalues" are objects created by rvalue reference operations (sometimes previously called "rvalue reference objects"). The "x" can be for "eXpiring value" or a cross between lvalues and rvalues.
  - "prvalues" are the new name for traditional rvalues (i.e., excluding the xvalue cases). The "p" is for "pure rvalue".
- **There are two compound categories:**
  - "glvalues" are lvalues and xvalues. These are things that have dynamic type and object identity.
  - "rvalues" are xvalues and prvalues. These are (as now in the draft) things that are potentially movable.
- **The final paper is N3055.**

Lvalues, rvalues, and things that goes bump in the night







New expression properties

	Glvalue	Rvalue	Lvalue	Xvalue	Prvalue
Has object	**Yes**	**Depends**	**Yes**	**Yes**	yes if class rvalues, maybe if non-class rvalues
Has identity	**Yes**	**Depends**	**Yes**	**Yes**	**No**
Polymorphic	**Yes**	**Depends**	**Yes**	**Yes**	**Yes**
Gratuitously copyable	**No**	**Depends**	**No**	**No**	**Yes**
Moveable	**Depends**	**Depends**	**No**	**Yes**	**depends**
Can be cv-qualified	**Yes**	**Depends**	**Yes**	**Yes**	yes if class rvalues, no if non-class rvalues
Independet properties	**Identity**	**Movable**	**Identity, not movable**	**Identity, movable**	**No identify, movable**

Trigraphs

- Coming into this meeting, the suggested approach to solving the problem of trigraphs in raw strings was the one given in N2990, i.e., making the trigraphs more token-like.
- We decided on a different approach, which requires that after forming a pp-token for a raw string any transformations previously done for trigraphs, line splices, and UCNs must be reversed.
- That's essentially "here magic happens" but implementers say they can do it. This proposal preserves C compatibility. It does nothing about the problem that users may be surprised if they inadvertently write trigraphs in normal strings, and it does not deprecate trigraphs. The final paper is N3077.
- Note that we changed the [...] in raw strings to (...) so that the delimiter is not one that can be written using a trigraph.

Exceptions

- **We discussed two things related to exception specifications:**
  - noexcept (N3050), and
  - deprecating exception specifications (N3051).
- **There was some spirited discussion of what happens when you violate a promise not to throw. In the end, we decided that in such a circumstance terminate() gets called, and it's unspecified whether any cleanup is done for local variables between the throw point and the point of the violated noexcept.**
- **It's believed that for efficient implementations of exception handling a noexcept will not add any overhead and will not restrict optimization opportunities.**
- **The old-style exception specifications, including the throw() form, will now be deprecated. The library specification needs to be updated to use noexcept instead, and that won't happen at this meeting.**

Example 1: doesn’t throw – no overhead

```
 int f (int a, int b) noexcept
{
   return a+b;
}  // doesn't throw - should have no overhead
```

Example 2: throws – should get compile error

```
void f () noexcept
{
   throw E();
}  // throws - should get compile time error
```

Example 3: might throw but rarely, trust the programmer

```
int* f() noexcept
{
    return new int (7);
}  // might throw - ignore possibility of a throw,
```

1. Give a compile error because f might throw
2. Detect a throw at runtime and terminate
3. Ignore the possibility of a throw and compile

Example 4: might throw, but ignore

```
int* f(int i) noexcept
{
    string s = "   ";
    if (i) return new int[i];
    throw E();
}  // might throw
void g (int i)
{
    string ss = "   ";
    int* p = f(i);
}  // calls f which might throw
```

```
void h (int i)
{
    try {
            g(i);
    } catch (E e) {
            gs (i);  // alternative ____
    }
}
```

Example 5:

```
void f() noexcept;
void f()
{
   throw E();
}
```

```
void f(int i) noexcept
{
   if (i<1 || i<j) return;
   while (i>0) {
            if (--j<0) throw E();
            --i;
   }
   if (i<j) throw E();
}
```

List of Standard features and papers (100509)

- **C++0x FCD:**
  - http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2010/n3092.pdf
- **c++0x (CD1):**
  - http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2009/n3000.pdf
- **Summary of Core language and Library State:**
  - http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2009/n2869.html
  - http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2009/n2870.html
- **Summary of C++0x CD1**
  - http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2009/n2871.html
- **Summary of C++ TR1**
  - http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2007/n2364.html
- **TR1(DTR):**
  - http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2005/n1836.pdf
- **Decimaal TR(PDTR):**
  - http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2008/n2732.pdf
- **Math(FCD):**
  - http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2008/n2717.pdf

Food for thought and Q/A

- **This is the chance to make comments on the C++0x FCD through us or the National Body rep:**
  - http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2010/n3092.pdf
- **Participate and feedback to Compiler**
  - What features/libraries interest you or your customers?
  - What problem/annoyance you would like the Std to resolve?
  - Is Special Math important to you?
  - Do you expect 0x features to be used quickly by your customers?
- **Talk to me at my blog:**
  - http://www.ibm.com/software/rational/cafe/blogs/cpp-standard

My blogs and email address

- michaelw@ca.ibm.com
- Rational C/C++ cafe: http://www.ibm.com/software/rational/cafe/community/ccpp
- My Blogs:
- Parallel & Multi-Core Computing http://www.ibm.com/software/rational/cafe/blogs/ccpp-parallel-multicore
- C++ Language & Standard http://www.ibm.com/software/rational/cafe/blogs/cpp-standard
- Commercial Computing http://www.ibm.com/software/rational/cafe/blogs/ccpp-commercial
- Boost test results http://www.ibm.com/support/docview.wss?rs=2239&context=SSJT9L&uid=swg27006911
- C/C++ Compilers Support Page http://www.ibm.com/software/awdtools/ccompilers/support/
- C/C++ Feature Request Interface http://www.ibm.com/support/docview.wss?uid=swg27005811
- XL Fortran Compiler Support Page http://www.ibm.com/software/awdtools/fortran/xlfortran/support/
- XL Fortran Feature Request Interface http://www.ibm.com/support/docview.wss?uid=swg27005812

Acknowledgement

- **Some slides are borrowed from committee presentations by various committee members, their proposals, and private communication**