C++17 & C++22: the future of C++11

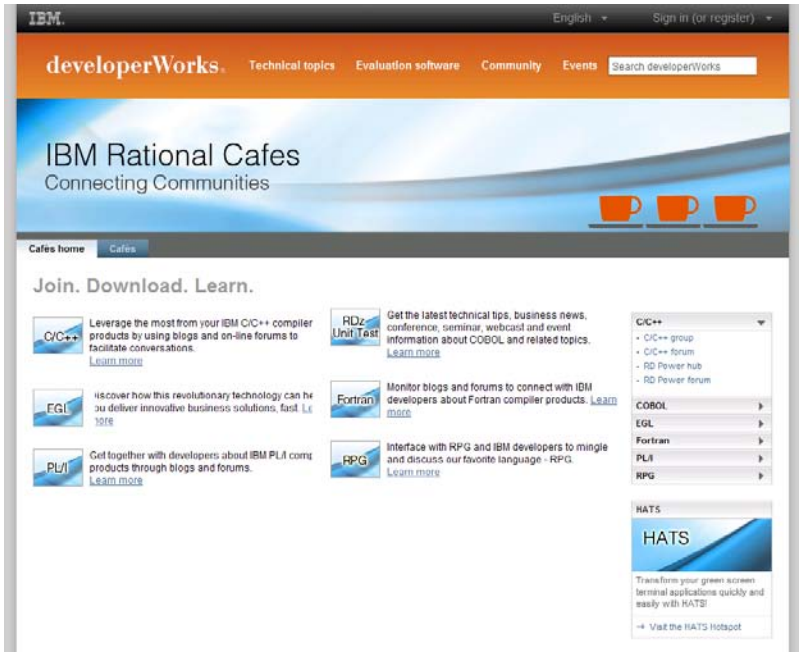
Michael Wong
michaelw@ca.ibm.com
IBM, Canada C++ Standard
Chair of WG21 SG5 Transactional Memory
OpenMP CEO
IBM xlC compiler Senior Technical Lead

IBM Rational Disclaimer

IBM Rational Cafes – Connecting Communities

- Accelerate your enterprise modernization efforts by becoming a member of the Cafe communities
- Ask questions, get free distance learning, browse the resources, attend user group webcasts, read the blogs, download trials, and share with others
- Cafes have forums, blogs, wikis, and more

- **Languages covered:**
  C/C++, COBOL, Fortran, EGL, PL/I, and RPG
- **Products covered:**
  - COBOL for AIX®
  - Enterprise COBOL for z/OS®
  - Enterprise PL/I for z/OS
  - Host Access Transformation Services
  - PL/I for AIX
  - Rational® Business Developer
  - Rational Developer for Power Systems Software™
  - Rational Developer for i for SOA Construction
  - Rational Developer for System z
  - Rational Developer for System z Unit Test
  - Rational Team Concert™
  - XL C for AIX
  - XL C/C++ for AIX/Linux ®
  - XL Fortran for AIX/Linux
  - XL C/C++ for z/VM
  - z/OS XL C/C++



Become a member and join the conversation!
ibm.com/rational/café

Continue the conversation

facebook.com/IBMcompilers

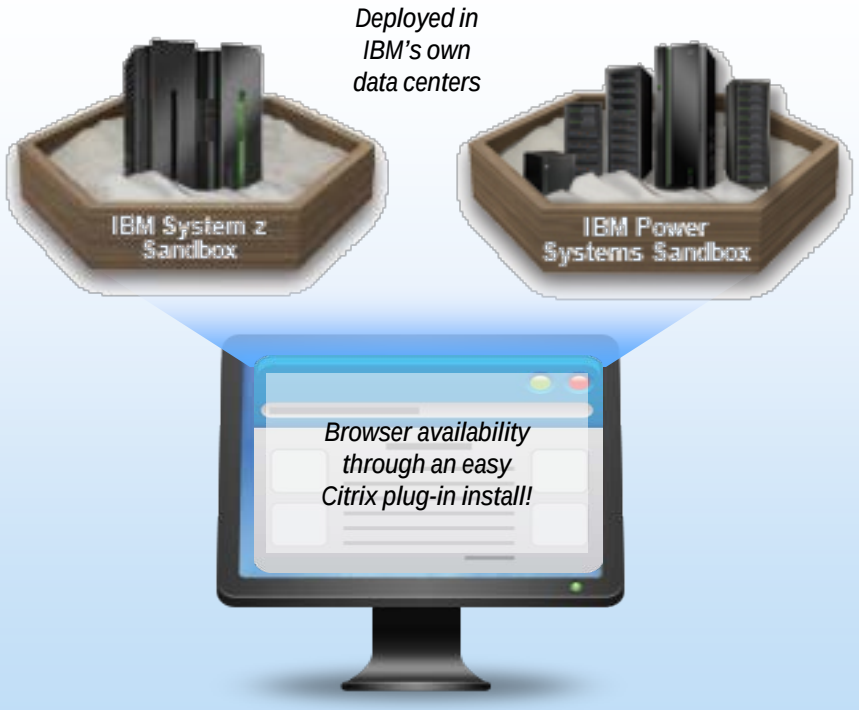
@IBM_Compilers

xl_compiler@ca.ibm.com

IBM Enterprise Modernization Sandbox

Realize the value of your investments in assets, skills and infrastructure within minutes

- Learn how to revitalize applications, empower people, unify teams and exploit infrastructure based on your knowledge and experiences
- See firsthand the tangible value Rational tools can bring to your business
- Get a fast start with scripted scenarios and best practice education materials at no cost available 24x7
- Access a low risk way to try several new offerings and integrated solutions without disturbing your existing environment

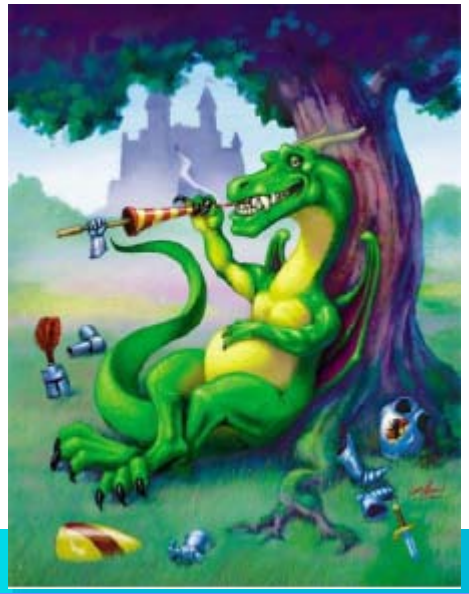


Try it firsthand within minutes today!

ibm.com/playinthesandbox

Agenda

- The future of C++ Standard
- Bonus 1: Update from Feb C++ Std meeting
- Bonus 2: current C++11 compiler status
- Fragen?



The future after ratification

- Adopt a bus/train delivery schedule
  - 2017, 2022
- The most successful test-run was a TR which was pretested by Boost
  - E.g. Boost.functional
  - Propose we have library and language TR
- New standard every 3-5 years, but not 11
- Invite /review proposals for 1.5 years (until 2014?)
  - 3 meetings: Hawaii, Portland, Italy?
- Refine and integrate accepted proposals for 1.5 years
  - 3 meetings
- C++1x?

What is missing?

- We do not have a system
  - We (mostly) have an unordered sea of non-interoperable libraries
    - The standard library was supposed to help
    - Boost was supposed to help

What do you want in the future?

- C++1x?
  - stabilize for a few years until compilers catchup
  - Start immediately on new work?
- Concepts?
- A full conservative Garbage Collector?
- Standardize dynamic libraries
- Improve compile time through Module?
- Advanced Concurrency Abstractions?
  - Cilk++, Continuation (then, await), TBB/PPL, TM
- Other failed features from C++0x
  - Contract programming
  - Scoped preprocessor directives

What new feature for C++1x?

- “People” don’t clearly distinguish between
  - Library components
  - Language features
    - E.g., dynamic linking and loading support
  - Tools
    - E.g., leak detector
  - Programming environment
    - E.g., debugging support
- For many, those distinctions are artificial
  - Solutions tend to cut across the distinctions
    - E.g., is “lambda” a library or a language feature?
- But great libraries may require tool and/or language support

Potential C++1x proposals

- Networking
- Threading
- Unicode
  - “much more than C++0x offers”
- Date and time
- File system
  - “more protocols than boost”
- Safe casts

GUI

- GUI (gtkmm?)
- 2D rendering (Cairo?) Bezier curves
- 3D rendering (OpenGL?)
- 2D layout engine
- Bridge to other systems/libraries
- Computational geometry
- Rational numbers, real, fixed-point math

Business

- XML (parsing, generating, validating, transforming)
- Web programming (HTTP, HTTPS, email)
- Web services (SOAP, WSDL, UDDI)
- Bridge to other systems/libraries
- Pluginf ramework
- GUI
- Distribution (communications, serialization, resource discovery)
- Generic database connectivity and transactions
- Cryptography
- Authentication
- Generic scripting language interface (call, load and execute)
- Audio/video streams
- VB-like string manipulation

Concurrency/distribution

- Concurrency “beyond locks and semaphores”
  - lock-free, wait-free containers and algorithms
- Multi-core/throughput
  - TBB, ArBB, C++AMP, PPL
- Generic database connectivity and transactions
  - Transactional memory, hierarchical locking
- Web programming
  - HTTP, HTTPS, email
- Distribution
  - communications, serialization, resource discovery
- Web services
  - SOAP, WSDL, UDDI
- Cryptography
- Authentication
- Shared memory
- Library for querying machine about hardware and OS resources
- Bridge to other systems/libraries

Math/Science

- Matrix library
- Bigint, rational numbers, real, fixed-point math
- Numerical methods
- Computational geometry
- A Mathlab library
- Physical units
- Better formatting (“type-safe printf”)
- Bridge to other systems/libraries
- Math special functions (as in TR1)

Other

- Generic database connectivity and transactions
- VB-like string manipulation
- Command-line parser
- Neural networks
- Library for querying machine about hardware and OS resources
- Plug-in framework
- C++ parser and transformer
- Memory mapped files, random-access

Meta Evolution

- Features into 4 bins

1. Cleanup/small features for C++17
2. Stretch features for C++17
3. Features for C++22
4. Not considered at this time

- When a new feature is proposed, we should ask ourselves the following questions:
  - What do we get from this proposal?
    - Does it enable doing something that could not (reasonably) be done before?
    - Does it complete another feature?
    - Does it make life easier for beginners or teachers?
    - Does it help avoid bugs?
    - Guideline: A proposal should be rejected unless it provides clear benefits.
  - What is the cost of the proposal?
    - How much committee time will it take to refine and finish the wording?
    - What is the ripple affect on the rest of the standard?
    - How hard is it to implement?
    - Guideline: cheap proposals might go into C++17, more costly proposals might go into our C++17 stretch goals or into C++22.
    - Guideline: A very costly proposal should provide huge benefits, or else we should reject it.
  - Would the proposed feature be included with or obsoleted by some larger feature (e.g., concepts or modules)?
    - Would accepting a specific small feature now interfere with the specification of the larger feature later?
    - Would waiting for the larger feature (e.g. until C++22) causes us to give up something that we really want in C++17?
    - Can the proposed feature be structured so that it will be harmonious with the larger feature in the future?
    - How certain are we that the larger feature will happen?
    - Does the proposed feature fit with a principled solution to the larger problem?
  - Etc.
    - Does it fit with the rest of the language?
    - Does it allow a programmer to write more type-safe code
    - Does it interract well with exceptions and resource management.

Core Feature category

- metaprogramming (small audience, working facility)
  - metacode (better metaprogramming), N1471
  - AST operator (access to expression tree)
  - compile-time if
- Templates
  - member templates of local classes/local templates (not commonly useful, just makes some things more regular)
  - cv templates (template over const, volatile status)
  - parameter packs not in last position
  - default arguments for partial specializations
  - function partial specialization
- parameter pack interfaces (typelists)
  - (want to pass packs around as objects,
  - first-class parameter packs for multiple return values, ...)
- literal types
  - generalizing constexpr function bodies
  - literal types as template value arguments (type-rich prgmg)
- Reflection
  - compile-time (remove boilerplate)
  - run-time, rich pointers (possibly implementable with compile-time)
- automatic code
  - implicit comparison operators
- large-scale abstraction
  - modules (because of template pollution of headers)
  - #nomacro
  - namespace regions, remove namespaces
  - #once
  - dynamic libraries
- memory management
  - garbage collection
  - VLA/dynarray (performance, looks bad if C has it, experience in practice)
- proxies (wanted for a long time)
  - overloaded operator
  - operator auto (allow replacement of template argument deduction for a type)
- network
  - async I/O - web
  - XML
  - email/sms
  - compression
- other
  - range
  - database access
  - Boost.Optional, Variant, Any
- interface specification
  - concepts (improve experience of end-users) (maybe not for this standard)
    - concept-based overloading
    - concept-based optimization
  - preconditions/postconditions (make concepts cleaner)
  - contracts
- control generalization
  - multimethods (hard to do with libraries only)B 2
  - pattern matching (advanced type switch) (simplifies lots of code, does not need linker support)
  - complex control flow (asynchronous)B
    - more general continuationsB
    - resumable functions
- call generalization
  - f(x) finds x.f() (simplifies generic programming)
  - multiple return values (library solutions, pattern matching will help more)
  - named parameters (library solutions but not good)
  - designators (C99) (few classes with that many constructor parameters)
  - argument deduction for constructors, N2332
- call optimization?
  - pure function qualifier (compilers can infer it, but not always)
  - restrict
- Concurrency
  - data/task parallelism (modern hardware requires them)
  - transactional memory
  - heterogenous execution (modern hardware)
  - heterogenous memory (modern hardware, C has it as TR)
- lambda, et. al.
  - polymorphic lambda (easy to get types wrong, redundant)
  - deduce function return type (easy to get types wrong, redundant)
  - move captures in lambdas
- Lexical
  - digit separators
- syntactic/semantic
  - exponentiation operator
  - swap operator
- Misc
  - inner classes with captures
  - parameterized constant expressions
  - ADL not finding unconstrained templates
  - scoped operator new
  - allow comparison to empty initializer list
  - remove restrictions on converting pointers-to-members (CH 1 on FCD)
  - local variables in member initializers of local classes
- numeric
  - big int
  - fixed-point (N3352)
  - rational
  - fixed-slash
  - constructive reals
  - linear algebra

WG21 Study Groups

- SG1: Concurrency
- SG2: Modules
- SG3: Filesystems
- SG4: Networking
- SG5: Transactional Memory

C++ Advanced Parallel Summit May 7-9

- Asynchronous Operations
  - then, await
- Executors
  - Work Executors (N3378) (Jeffrey Yasskin et al)
- Transactional Memory
- Task-Level Parallelism
  - Cilk (Robert Geva)
  - TBB and PPL (Artur Laksberg?)
  - TLS interactions (Pablo Halpern)
- Vector Parallelism
  - Based on Intel
- Completing the synchronization library
  - Latches and Barriers (Alasdair Mackintosh)
  - N3353-5 progress (Lawrence Crowl)
- Voluntary cancellation

Transactional Memory

- Addressing concerns expressed at Kona meeting. Locks are impractical for generic programming, because ordering of locks is generally not visible. Transactional memory solves the problem. It also helps for fine-grained locking on irregular data structures.
- Discussion on composability. Relaxed transaction can expose partial state. Discussion on interaction with existing locks.
- Helps with read-mostly structures.
- Presentation of student-based study of error rates with transactions versus mutexes and condition variables. Error rates fall from 50% to 10%.
- Presentation of longer-term study of graduate students on an indexing application. Overall 14% improvement in overall programming effort with transactional memory. Good performance using transactional memory.
- Is transactional memory fast enough? Many different software transactional memory systems with different performance characteristics, so probably one fits your needs. Recent work on adaptive algorithms. Can change system without changing client code. Unknown about ABI changes.
- Performance study of a multiplayer game with >100k concurrent players. Requirement is atomicity and consistency for all actions.
- Much discussion on appropriateness to standards, industry adoption, compatibility with existing code, meaning of a Technical Specification, possibility of a Study Group, etc.
- No objection to creating a Study Group for transactional memory. We will create one. Chair to be selected.

Asynchronous Operations

- Discussion of attributes of proposal. Much of the discussion was around the right strategy for integrating something along these lines into the standard. Is asynchrony better done with changes to I/O operations?
- Futures may have limits to performance for large-scale parallelism. See Space-efficient scheduling of multithreaded computations.
- Straw polls to 'ask author to do more work'.
- Asynchronous library '.then' extension to futures.
  - 22 SF, 9 WF, 1 N, 0 WA, 0 SA
- Asynchronous language 'await' extension to futures.
  - 12 SF, 11 WF, 8 N, 0 WA, 0 SA

The Serial Equivalence of Cilk Plus

- Three keywords: cilk_spawn, cilk_sync, cilk_for. The serial elision of a Cilk program is well defined. A Cilk program without a determinacy race behaves the same as its serial elision when running on any number of threads.
- FIX: Needs definition of a determinacy race. Is it a data race?
- For most library solutions, and equivalent property is not well defined.
- Example with race. Correct by using a Cilk-provided abstraction, e.g. cilk::reducer_list_append.
- Language support for serial equivalence.
- No language construct that breaks equivalence
- Parent stealing
- Arguments evaluated by parent
- Implicit sync to structured fork-join parallelism
- After Cilk became part of Intel, the implementation changed so that the ABI is the same between spawnable and non-spawnable functions.

Serial Equivalence's Impact on Space Bounds

- Example of quicksort. Regular approaches require more space and more threads. Do depth-first serial execution and steal breadth first.
- Discussion of equivalence to async/future.
- Discussion of overhead of Cilk primitives on algorithms designed to be parallel.
- Discussion on modifying or extending the standard library with Cilk.
- Performance difference in that paper is unknown, but if inherent likely due to serial equivalence property.
- Much discussion on performance tradeoffs, particularly with respect to additional guarantees.
- Okay to call locks as long as they commute within the domain.
- Discussion on alternate implementations and the burden of implementation.
- Two basic tasks of compiler: injecting code around constructs and optimizing reducer based on source.
- There is a macro/library equivalent of Cilk that is useful for evaluating and understanding Cilk. But it is not the library you are looking for.

PPL

- Heavily uses lambda to represent work passed to PPL. Has task_group, structured_task_group, parallel_invoke, etc. Can '.run' a lambda in a group. Can '.wait' for all runs to finish in a group. Can '.run_and_wait' a lambda in a group.
- Parallel_for partitioning:
  - static: number of chunks == number of cores
  - simple: user-specified chunk size
  - auto: range stealing

Concurrent objects

- Current standard library data structures are no concurrency safe.
  - concurrent_queue, concurrent_priority_queue
    - push, pop
  - concurrent_unordered_set, concurrent_unordered_multiset, concurrent_unordered_map, concurrent_unordered_multimap
    - insert, find, iterate
  - concurrent_vector (elements are not in contiguous memory)
    - push_back, operator[], grow_by, grow_to_at_least
- extern concurrent_vector v; copy(u.begin(), u.end(), v.grow_by(u.size()));
- combinable objects provide reduction.
- Open Issues:
  - value-oriented containers versus reference-oriented containers?
  - separate "parallel" and "serial" views?
  - selecting concurrent requirements more precisely?
- Much discussion on the specificity of data structures to patterns.

Problems of Thread Local Storage

- Three use cases for thread-local storage.
  - session-specific information
    - one thread per session, store info in TLS
  - dynamic cache
    - cache previously computed values in TLS to avoid synchronization
  - task-specific variables
    - function receives one of its arguments or result through a global variable
- Fourth use case suggested: merge multiple single-threaded processes into a single multi-threaded process.
- What should thread_local specify?
  - There are at least three candidates.
  - Should be associated with std::thread.
- Conclusions
  - No single concept sufficies.
  - We will need a task formalisim and possibly a worker formalism
  - All three candidates should be supported.
- Much discussion on "what is a thread", "what is a thread-local variable", "what about POD thread-local variables", etc.

Async and Thread-Local Storage

- Benchmarking fib(30) example. The std::async with default launch policy is useful for parallel decomposition. The std::async with launch::async policy is useful for "get this work of my GUI thread".
- Microsoft implemented async(launch::async) with thread pools. Reinitializes thread-locals. Does not handle destruction.
- Extensive discussion of semantics and implementation of thread-local variables. Did I mention extensive?
- Straw Polls:
  - Shall we reconsider non-POD thread-local variables?
    - SF 5, WF 5, N 5, WA 7, SA 3

Vector Parallelism

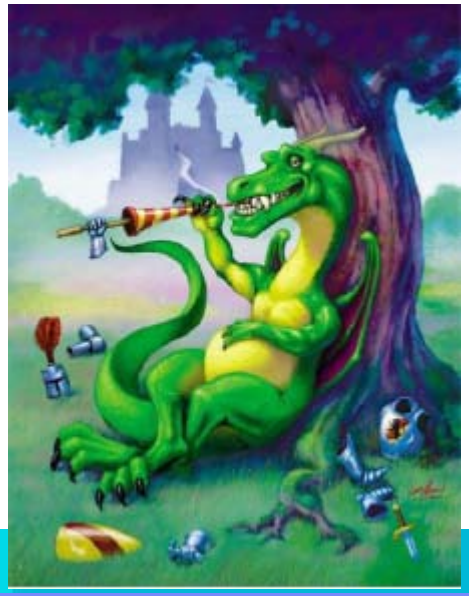
- Core counts on servers will continue to grow. Core counts on (mobile) clients are not likely to get much higher. Vectors are getting wider.
- Ability to program tasks versus vectors explicitly. Ability to express intent for parallel execution and let compiler map to hardware resources.
- Several currently available technologies for vectorization.
- Discussion of appropriate syntax and semantics for vectorization.
- Need language support for efficient code.
- Data parallelism is a programming pattern; vectorization is an implementation.

Other discussions

- C++ AMP
- C++ Latches and Barriers
- Counters
- Concurrent Queues
- Stream mutexes
- Cancellation
- Blocking in future destructors

Agenda

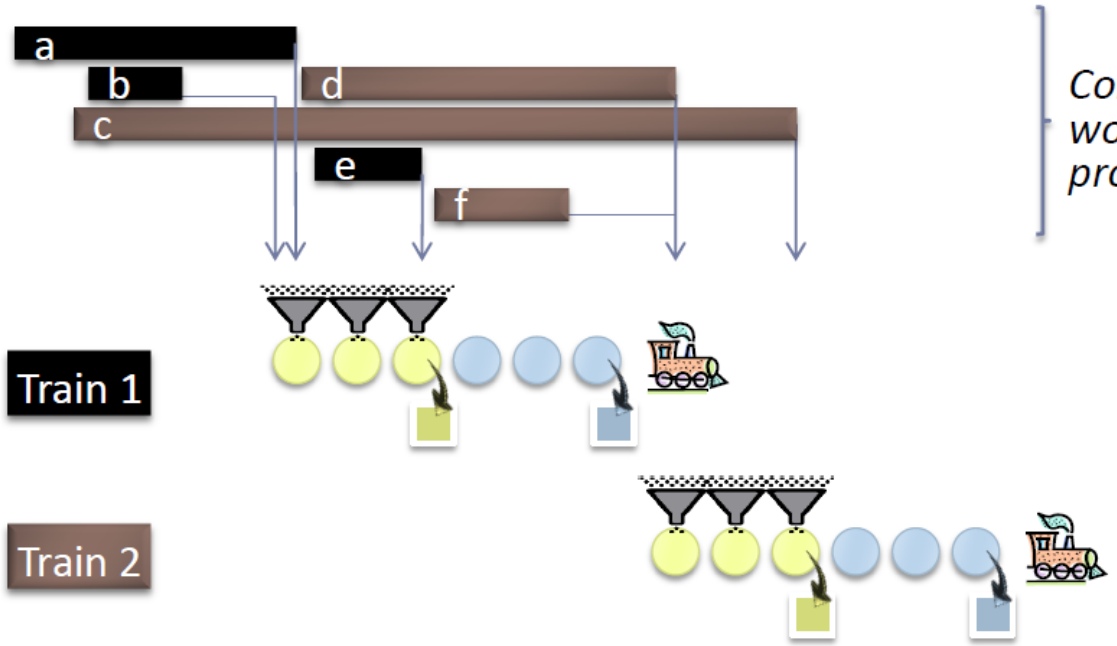
- The future of C++ Standard
- Bonus 1: Update from Feb C++ Std meeting
- Bonus 2: current C++11 compiler status
- Fragen?



Herb Sutter's presentation of plans for the future

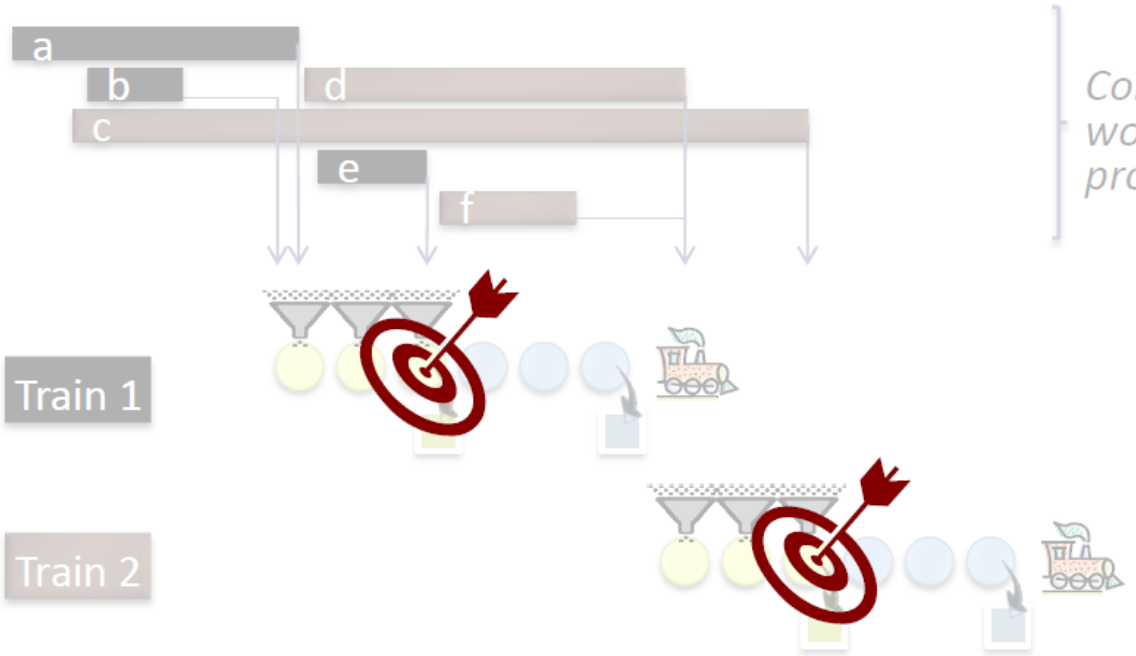
Predictability

- Please let's ship **lots** of good stuff but in **smaller and more predictable** chunks.



Predictability

- Please let's ship **lots** of good stuff but in **smaller and more predictable** chunks.



"C++11 Feels Like a New Language"

- Great!
- But significant change to coding **style/idioms/guidance**.
  - That's why it feels new. Style/idioms/guidance define a language.
  - Features that significantly change style/idioms/guidance include:

Core Language	
auto	nullptr
range-for	lambdas
move semantics, rvalue references	uniform initialization, initializer lists
noexcept	*constexpr*

Library
smart pointers
begin(x) / end(x)

We're All Learning C++11

- Includes everyone: WG21 members, authors, developers.
- Example: Move semantics.
  - 2002-09-10: N1377, Initial move proposal. (Hinnant, Dimov, Abrahams)
  - **2011-11-09: "We are still in an infancy stage with regard to having a good working knowledge of how to use rvalue refs. I would very much like us to strongly consider field experience** in nailing down corner cases like this." (Hinnant)
- Discovering missing fingers and toes (e.g., cbegin(x)/cend(x)).
- Discovering guidance for even "easy to use" features (e.g., {} init).
- Handling our issues-list bug tail and Defect Reports.

Libraries, Compilers, and Books: Oh My?

- Using C++11: Implementations.
  - C++11 library: Several available now or soon (2012).
  - C++11 language: First reasonably fully conforming compiler in 2013?
- Learning C++11: Books and other materials.
  - Scott Meyers' "Overview of the New C++ (C++11)" – *bit.ly/meyers11*
  - ETAs for the major books…

C++ Primer (Moo)	Aug 2012	Reference
The C++ Programming Language (Stroustrup)	Late 2012	Reference
Programming: Principles & Practice Using C++ (Str.)	Late 2013?	Introductory
Effective C++ (Meyers)	2013-14?	Style guidance
C++ Coding Standards (Sutter, Alexandrescu)	2015?	Established guidance

Suggestion: Preserve Style/Idiom/Guidance

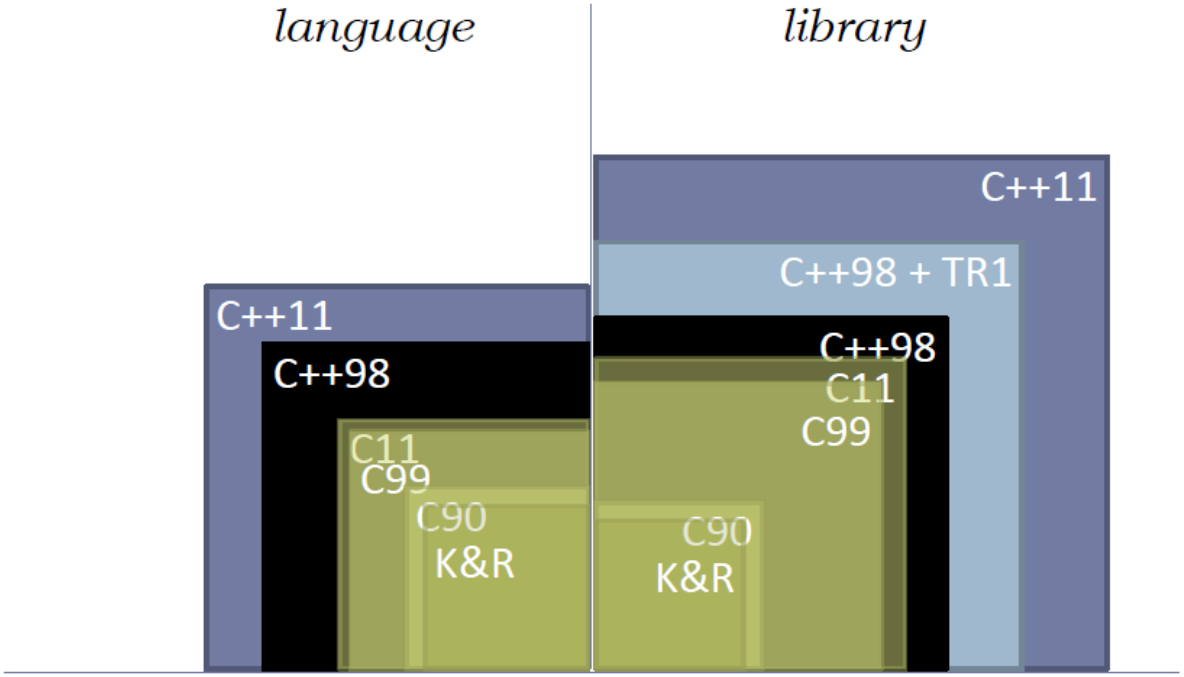
- Can still remove reasons to resort to preprocessor macros (they're outside the language anyway).
  - Example: **"#once"** to replace #include guard macros.
- Can still remove reasons to resort to template metaprogramming (arguably outside language, inaccessible to most developers anyway).
  - Example: **"static if"** to replace specialization/termination/SFINAE.
- Can still make usability improvements to existing features (e.g., simpler spelling or wider use, not new style/idiom/guidance).
  - Polymorphic lambdas.

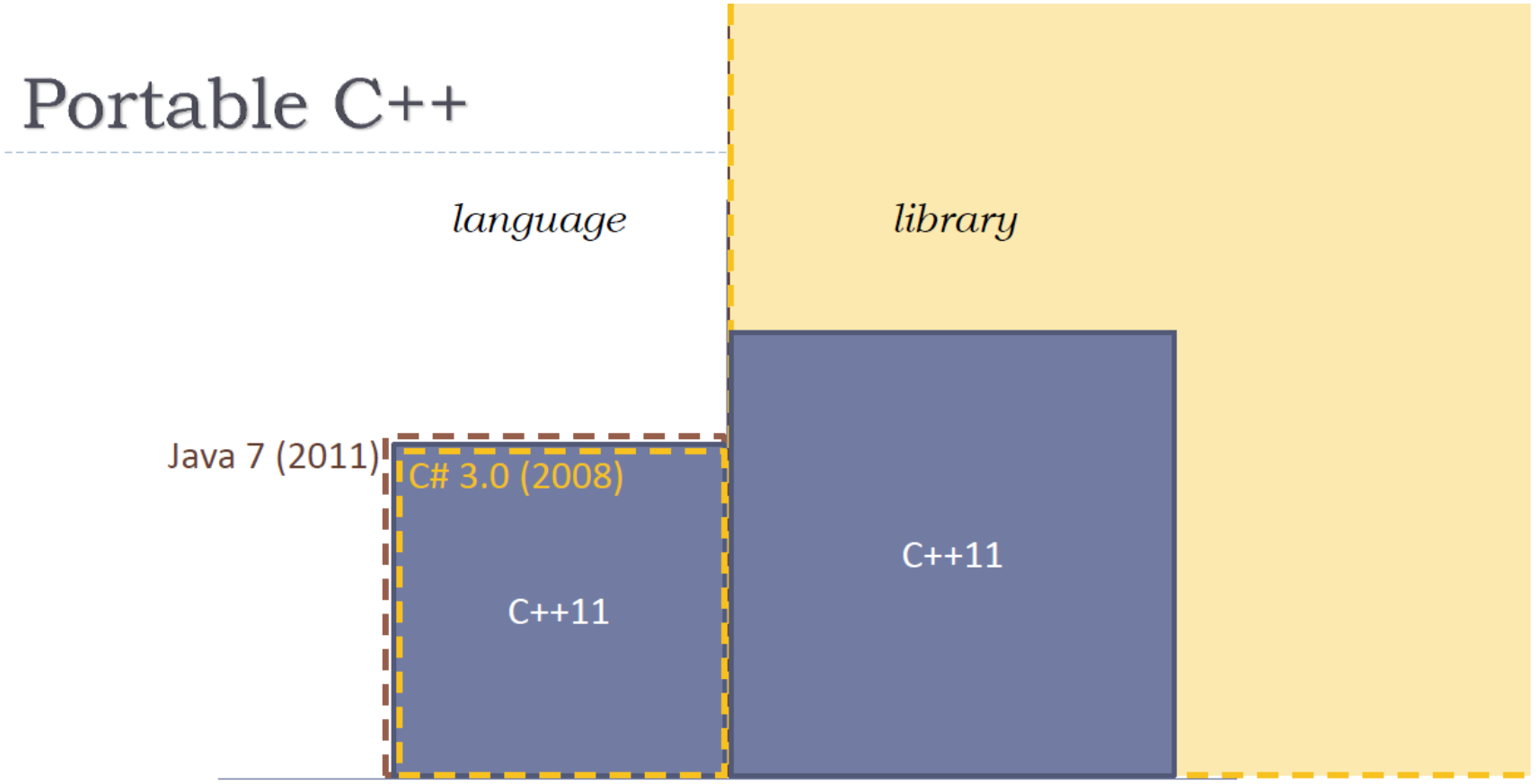
```
[]( some_typename& x ) { return x.f() + g(x); }
➔ []( auto& x ) { return x.f() + g(x); }
➔ []( x ) { x.f() + g(x) }
```

Q: What is Standard C++'s biggest weakness?

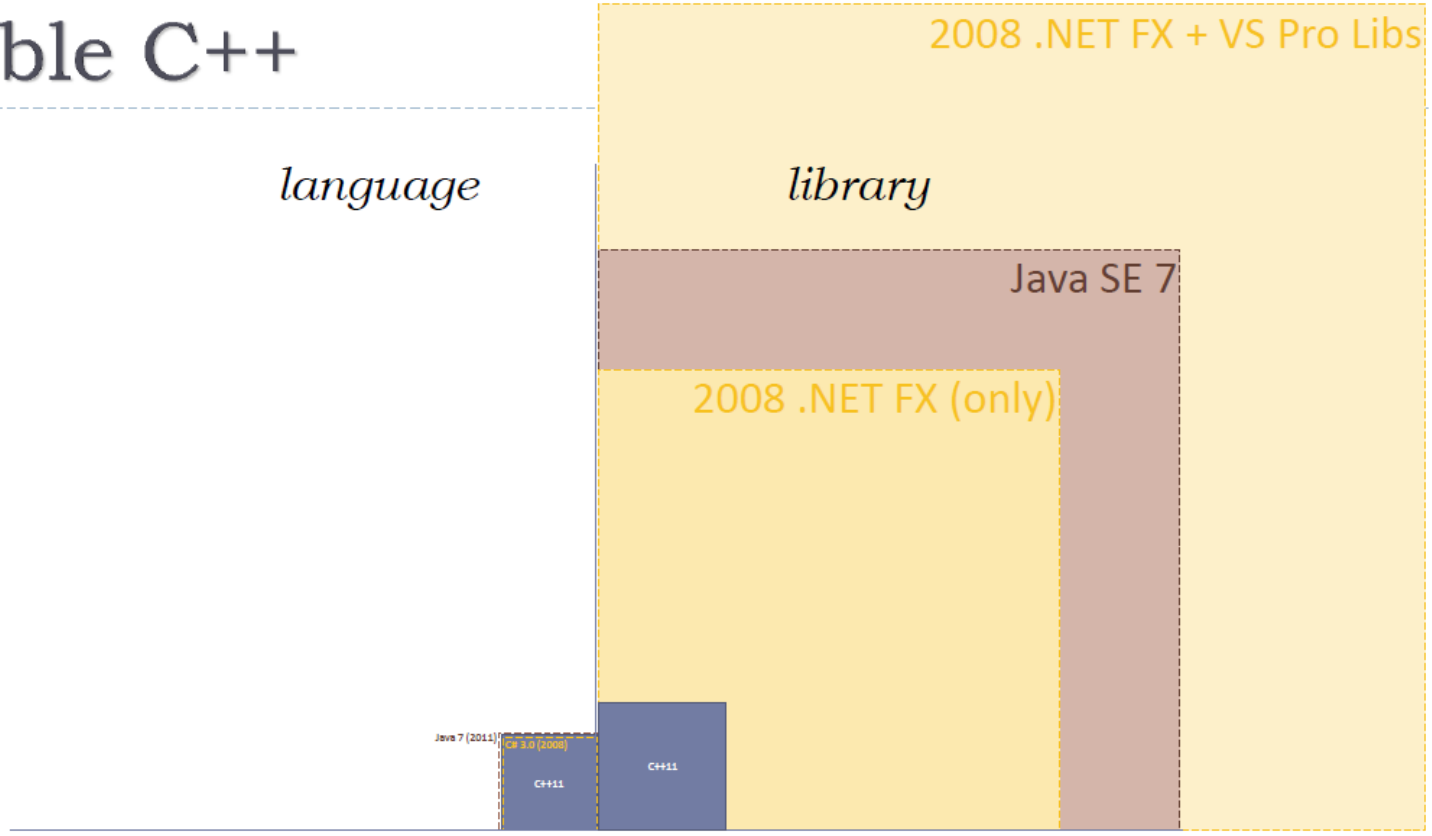
Portable C++

proxies for size comparisons:
spec #pages, book #pages




Portable C++
language
library
Java 7 (2011)
C# 3.0 (2008)
C++11
C++11

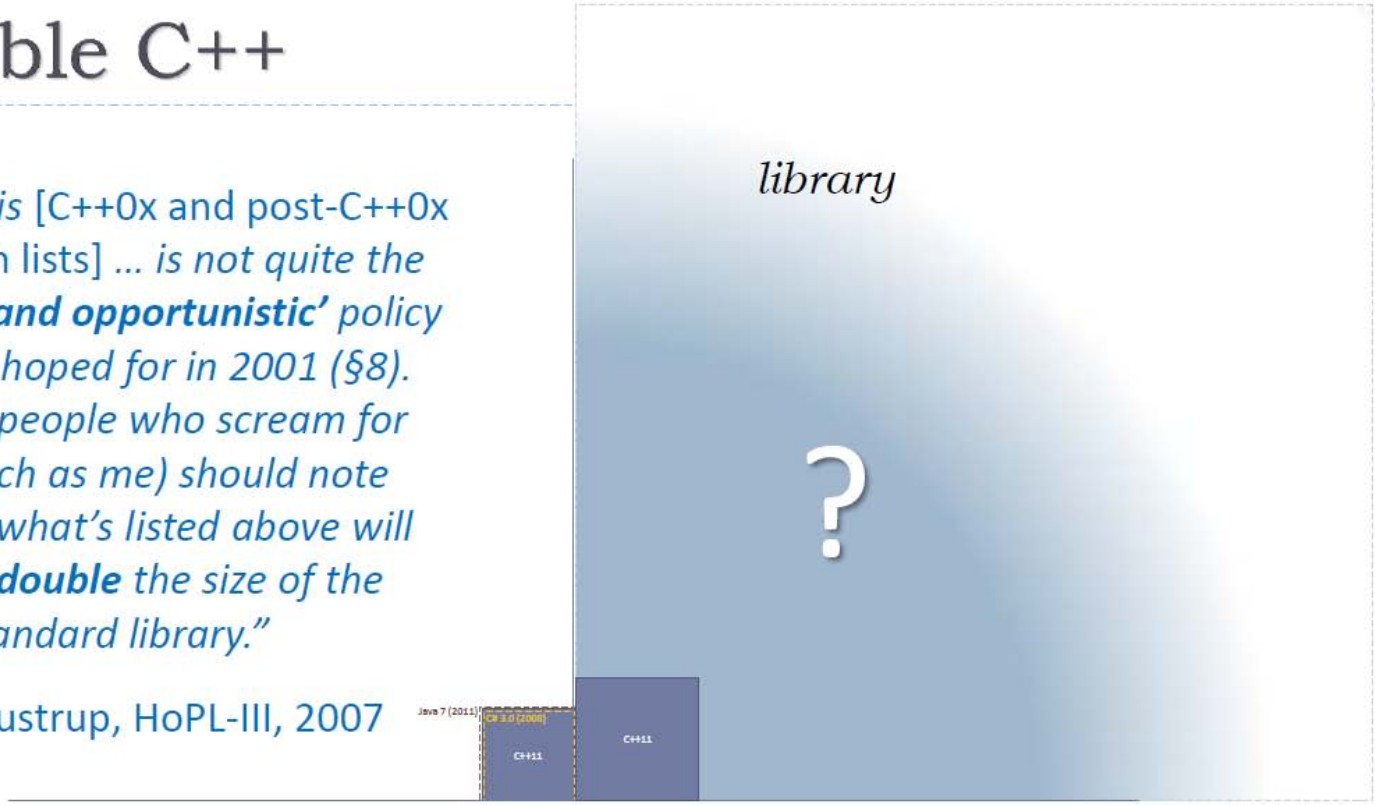
Portable C++


2008 .NET FX + VS Pro Libs
language
library
Java SE 7
2008 .NET FX (only)
Java 7 (2011)
C# 3.0 (2008)
C++11
C++11

Portable C++

"All in all, this [C++0x and post-C++0x library wish lists] *… is not quite the* ***'ambitious and opportunistic'*** *policy that I had hoped for in 2001 (§8). However, people who scream for more (such as me) should note that even what's listed above will* ***roughly double*** *the size of the standard library."*

– B. Stroustrup, HoPL-III, 2007



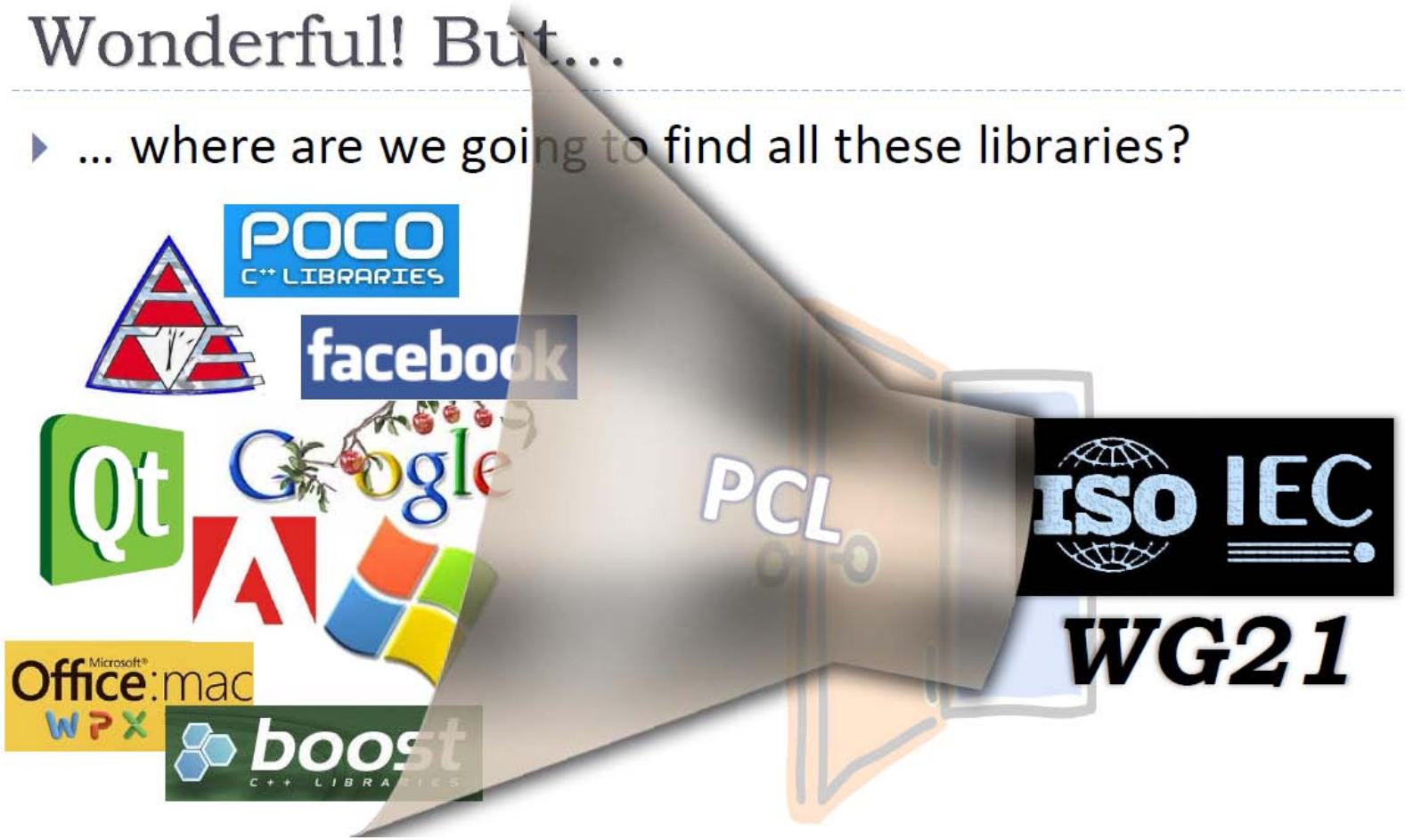
Portable C++ Library (PCL)

- Goals:
  - **Large** set of **useful** and **current** libraries.
  - Available on **all major platforms**.
  - **Shipped with** and **supported by** C++ implementations.
  - And **composable**, using consistent types and styles.

- Minimum: De facto availability as part of all major compiler products.
- Ideal: De jure inclusion in Standard C++.

Wonderful! But…

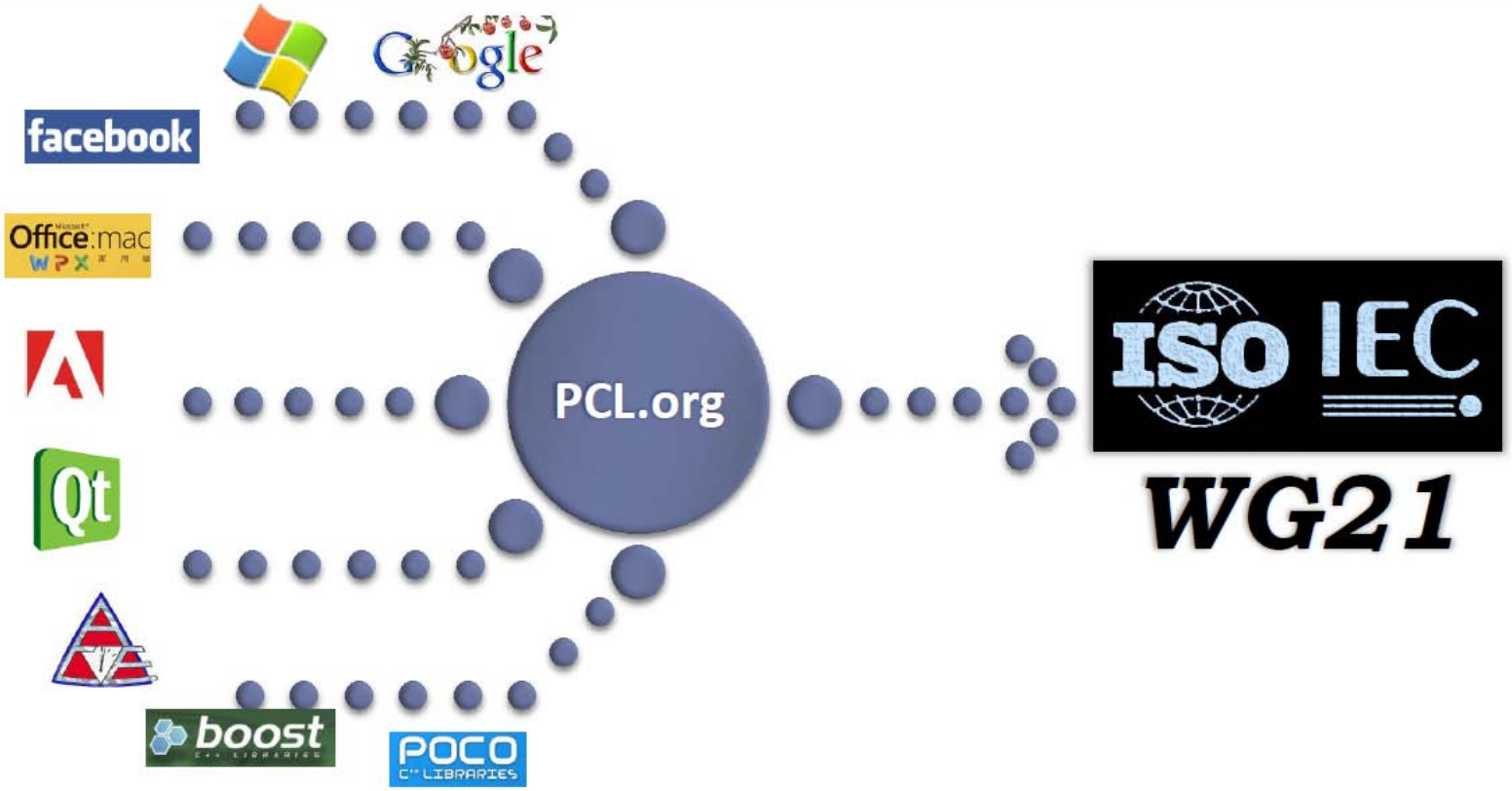
- … where are we going to find all these libraries?



PCL Scope (Suggestion)

- **Do** focus on pure libraries.
- **Do** focus on common modern tasks with state-of-the-art existing practice.
  - **Lower-level:** Message queue, ranges + range algorithms, parallel algorithms, thread-safe containers, continuations (future.then), async I/O, file system, networking/sockets, serialization.
  - **Higher-level:** REST web services, sensor fusion, HTTP, HTML, XML/XSLT, JSON, persistence, settings/preferences, compression, cryptography, audio/image/video, databases, SMS messaging.
- **Don't** target niche uses. (Example: Sci-eng linear algebra.)
- **Don't** become a platform = fat libs that duplicate native services.
- **Don't** attempt to define a "portable cross-platform" library that will be inferior to a native platform app. (Example: GUI WIMP widgets.)

"PCL.org" Open Interfaces

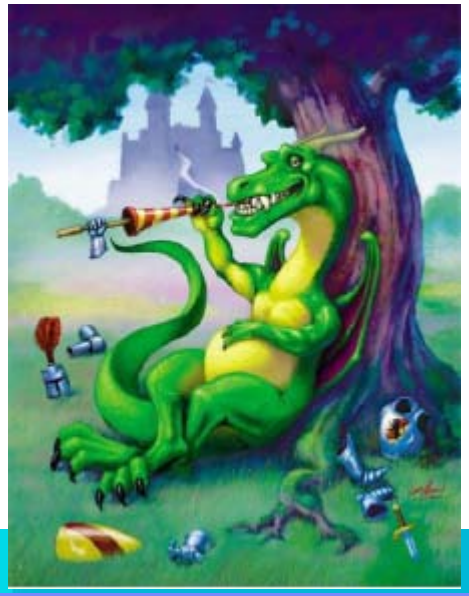


Some Key Deliverables

- **Library design guidelines,** esp. to ensure composability:
  - Write down what we "already know" are LWG design sensibilities. Includes: naming conventions; parameter orders; optional params vs. overloading; generic components vs. big"OO"; …
  - Add any additional guidelines. Pull selectively from existing sources: EC++, C++CS, .NET FX DG, etc.
- **Native FxCop tool** to enforce those guidelines on interfaces.
  - Benefit: Reason for us to specify checkable rules and compilable interfaces.
  - **Clang** seems well suited to get something useful quickly…
- Tracking mechanism: Status, assignments, etc.
- **Library submissions to WG21.**

Agenda

- The future of C++ Standard
- Bonus 1: Update from Feb C++ Std meeting
- Bonus 2: current C++11 compiler status
- Fragen?



Updated page of C++0x support

- Bjarne's C++0x FAQ:
  - http://www2.research.att.com/~bs/C++0xFAQ.html
- http://wiki.apache.org/stdcxx/C%2B%2B0xCompilerSupport
  - Maintained by Martin Sebor, me, and other compiler Tech leads from other company

XL Compiler status

- Our live C++0x status:
  - http://www-01.ibm.com/software/awdtools/xlcpp/aix/features/?S_CMP=rnav
    - https://www.ibm.com/developerworks/mydeveloperworks/blogs/5894415f-be62-4bc0-81c5-3956e82276f3/entry/xlc_compiler_s_c_11_support50?lang=en
- C++ published:
  - https://www.ibm.com/developerworks/mydeveloperworks/blogs/5894415f-be62-4bc0-81c5-3956e82276f3/entry/c_11_standard_now_available_for_purchase9?lang=en
- C++ ratified:
  - https://www.ibm.com/developerworks/mydeveloperworks/blogs/5894415f-be62-4bc0-81c5-3956e82276f3/entry/c_0x_c_11_standard_has_been_ratified2?lang=en

IBM XL c/C++ and z/OS c/C++ compiler status (May, 2012)

https://www.ibm.com/developerworks/mydeveloperworks/blogs/5894415f-be62-4bc0-81c5-3956e82276f3/entry/xlc_compiler_s_c_11_support50?lang=en

- **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)
  - Tested with Boost 1.34.1
- In **C/C++ for AIX/Linux for V11.1, in 2Q 2010 (include all of above)**
  - Variadic template
  - Auto
  - Decltype
  - Namespace association
  - Delegating constructor
  - Static assert
  - Extern template
  - extended friend
  - -qwarn0x
  - Tested with Boost 1.40
- In C/C++ for AIX/Linux for V12.1 in 2Q 2012 (include all of above)
  - Explicit conversion operator
  - Generalized const expr (phase1)
  - Rvalue reference
  - Right angle bracket
  - Scoped enum
  - Forward declaration of enum
  - Trailing return
  - C11 complex
  - C11 _Noreturn
  - C11 Anonymous structs
  - Tested with Boost 1.47
- **in zOS XL C/C++ V1R11**
  - Extern template
  - Extended friend
  - -qwarn0x
- V1R12 (include all of above)
  - Long long
  - Sync C99 preprocessor (Empty macro arguments, Variadic macros, Trailing comma in enum definition, Concatenation of mixed-width string literals)
  - Auto
  - Decltype
  - Variadic template
  - Namespace association
  - Delegating constructor
  - Static assert
- V1R13 (include all of above)
  - QOI message
  - Trailing Return

All information subject to change without notice

GNU

- http://gcc.gnu.org/projects/cxx0x.html
- 4.3/4.4/4.5/4.6 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
  - http://gcc.gnu.org/gcc-4.6/cxx0x_status.html
  - http://gcc.gnu.org/gcc-4.7/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
  - Possibly the biggest holdback from their completion
- Usually supports latest Boost (Boost 1.49)
- Additional Branch
  - Concepts
  - Lambda
  - Delegating constructors
  - Raw strings

All information based on publicly available data

GNU 4.3/4.4/4.5/4.6/4.7 (120326)

- 4.3: Rvalue Reference, Variadic Template, Static Assert, Decltype, Right Angle Bracket, C99 Preprocessor, Extern Templates, __func__, Long long
- 4.4:Extending variadic template template parameters, Auto, multideclarator auto, removing auto as storage-class specifier, new function declarator syntax, Propagating exceptions, Strongly-typed enums, New character types, Unicode string literals, Standard Layout types, Default and deleted functions, Inline namespaces
- 4.5:Initializer lists, Lambdas, Explicit conversion, Raw string literals, UCN Literals, Extending sizeof, Local and unamed types as template arguments
- 4.6: null pointer, forward declaration of enums, constexpr, unrestricted unions, range-based for, noexcept, move special member functions,
- 4.7: non-static data member init, template aliases, delegating constructors, UDL, extended friend, explicit virtual overrides

Intel and likely HP/Comeau (use EDG frontend)

- Intel C++ 12.0 has
  - -qstd=c++0x (Linux/Mac OS X), /Qstd:c++0x (Windows)
    - rvalue references
    - Standard atomics
    - Support of C99 hexadecimal floating point constants when in —Windows C++ mode
    - Right angle brackets
    - Extended friend declarations
    - Mixed string literal concatenations
    - Support for long long
    - Variadic macros
    - Static assertions
    - Auto-typed variables
    - Extern templates
    - __func__ predefined identifier
    - Declared type of an expression (decltype)
    - Universal character name literals
    - Strongly-typed enums
    - Lambdas

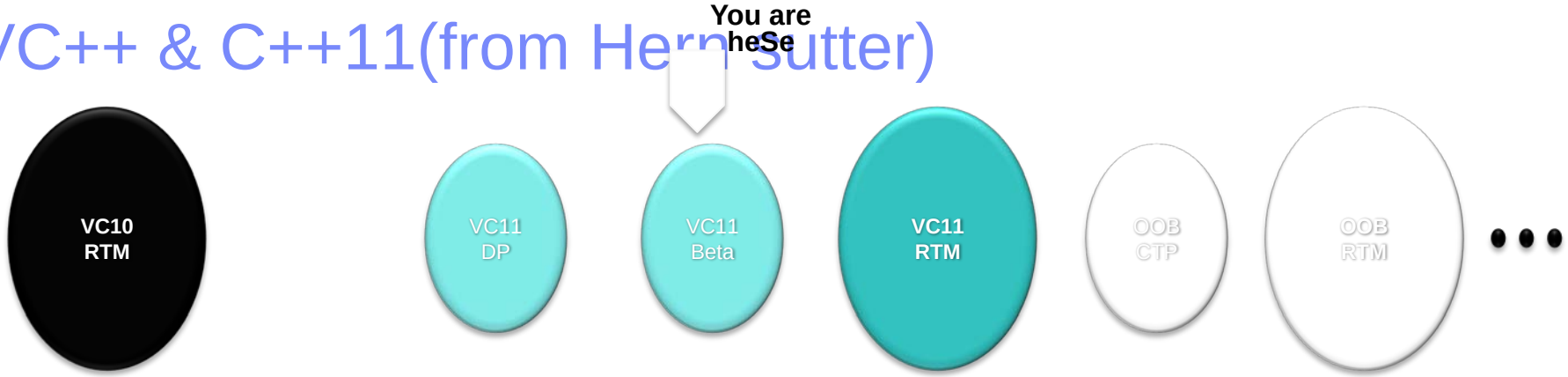
- Intel C++ Standard Library is based on Microsoft on Windows (uses Dinkwumare) and GNU on Linux (uses GNU's libstdC++), Boost 1.39
- HP aC++ V6 has been quiet about their C++ support, but will likely peggy-back on EDG as they move to new versions, uses STLport 5.1.7 as C++ Library, libstd runtime library 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 templates
  - Right angle brackets
  - Local and unamed types as template arguments
  - Long long
  - Exception_ptr
- Supports Boost 1.40
- Traditionally bought from Dinkumware C++ Library

VC++ & C++11(from Herb Sutter)



+ **lambdas, move/&&, auto, decltype**, auto fn decls, extern template, nullptr, …

+ complete C++11 stdlib: **thread/mutex, async, future**, … + *track changes (lambdas, &&, …)*

+ C++11 **range-for, final, override**

+ progressively roll out C++11 **initializer lists, template aliases, variadic templates, constexpr, noexcept, =default/delete, …**

Survey: **bit.ly/mscpp11**

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

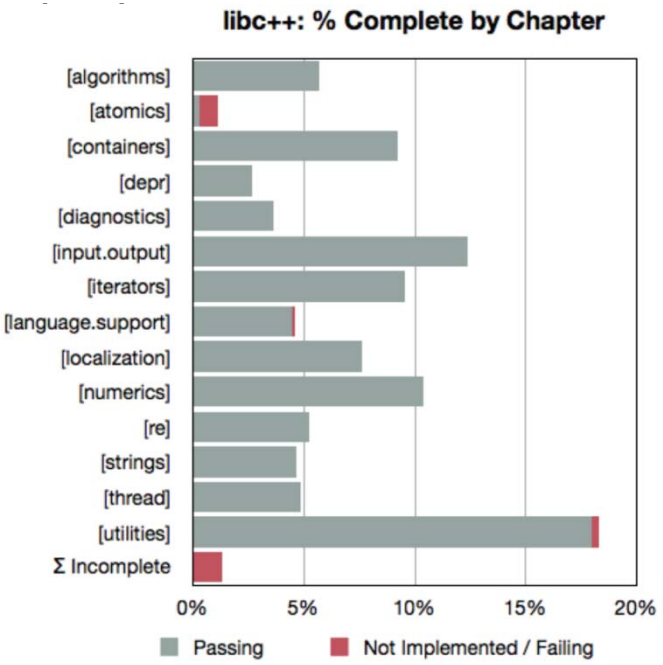
Borland/CodeGear C++Builder Compiler 6.10 2009

- 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

Clang/llvm

- Core language:http://clang.llvm.org/cxx_status.html
  - Very far from complete, can't compil
  - Variadic template, rvalue ref, extern long long
- Library:http://libcxx.llvm.org/index.l
- On Mac OS X/i386/x86_64
- Writes its own library libc++.a:
  - About 98% complete
  - Only missing atomics



Food for thought and Q/A

- This is the chance to get a copy before you have to pay for it:
  - C++ : http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2011/n3291.pdf
  - C++ (last free version): http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2011/n3242.pdf
  - C: http://www.open-std.org/jtc1/sc22/wg14/www/docs/n1570.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

- OpenMP CEO: http://openmp.org/wp/about-openmp/
  My Blogs: http://ibm.co/pCvPHR
  C++11 status:
  http://www.ibm.com/software/awdtools/xlcpp/aix/features/?S_CMP=rnav
  Boost test results
  http://www.ibm.com/support/docview.wss?rs=2239&context=SSJT9L&uid=swg27006911
  C/C++ Compilers Support/Feature Request Page
  http://www.ibm.com/software/awdtools/ccompilers/support/
  http://www.ibm.com/support/docview.wss?uid=swg27005811
  STM:
  https://sites.google.com/site/tmforcplusplus/

- Tell us how you use OpenMP:
- http://openmp.org/wp/whos-using-openmp/

Acknowledgement

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