

C++ Club Meeting Notes

Gleb Dolgich

2018-05-03

- ▶ Website

- ▶ YouTube channel

GCC 8.1 released

► Announcement

John Bandela:

This is a truly exciting time to be a C++ programmer. We are getting a new standard every 3 years with 3 popular compilers that are all committed to implementing the standard as fast as possible. This is truly a welcome change from C++98 <...> Kudos to the GCC team for this release.

Webkit is on C++17

- ▶ Announcement

Microsoft GSL 1.0.0

- ▶ Release
- ▶ Reddit
 - ▶ GSL = Gnu Scientific Library
 - ▶ Beware

Span-lite

A single-file header-only version of a C++20-like span for C++98, C++11 and later

- ▶ [GitHub](#) (MIT)
- ▶ [Reddit](#)

Trailing Return Types, East Const, and Code Style Consistency

- ▶ Post by Arne Mertz

PGI Community Edition Version 18.4

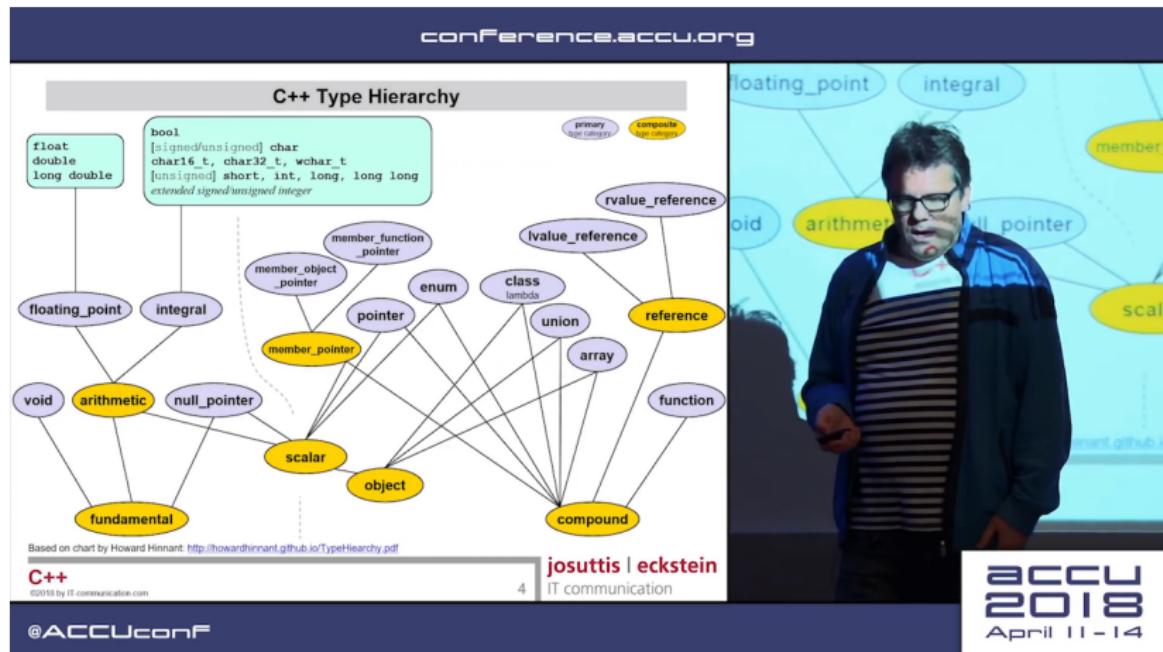
- ▶ [Website](#)
- ▶ [OpenACC YouTube channel](#)
- ▶ New in this version:
 - ▶ Support for Intel Skylake, IBM POWER9, and AMD Zen
 - ▶ AVX-512 code generation on compatible Intel processors
 - ▶ Full OpenACC 2.6 directives-based parallel programming on both NVIDIA® Tesla® GPUs and multicore CPUs
 - ▶ OpenMP 4.5 for x86-64 and OpenPOWER multicore CPUs
 - ▶ Integrated NVIDIA CUDA® 9.1 toolkit and libraries for Tesla GPUs, including V100 with NVIDIA Volta
 - ▶ Partial C++17 support and GCC 7.2 interoperability
 - ▶ New PGI fastmath intrinsics library, including AVX-512 support

Rea

Rea is a lightweight library of data structures implemented in C++11, designed for constant time insertion, erasure, lookup, and fastest possible iteration. Great for using in games or any other software which needs to manage thousands upon thousands of objects.

- ▶ [GitHub](#)
- ▶ [Reddit](#)

C++ Templates Revised - Nicolai Josuttis (ACCU 2018)



C++ Templates Revised - Nicolai Josuttis (ACCU 2018) (cont.)

conference.accu.org

C++14: Return-Type auto

```
template <typename T>
typename C::iterator foo (const T& c) {
    ...
    return c.begin();
}
```

C++11

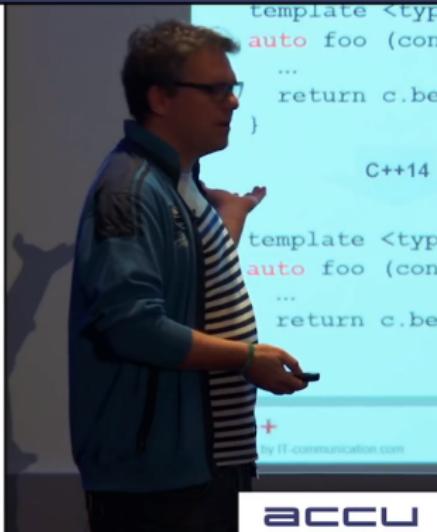
```
template <typename T>
auto foo (const T& c) -> decltype(c.begin()) {
    ...
    return c.begin();
}
```

C++14

```
template <typename T>
auto foo (const T& c) {
    ...
    return c.begin();
}
```

josuttis | eckstein

7 IT communication



The slide shows a presentation by Nicolai Josuttis at ACCU 2018. The title is "C++ Templates Revised - Nicolai Josuttis (ACCU 2018) (cont.)". The main content is a comparison of C++14's `auto` return type feature with its predecessors. It includes three code snippets: C++11's use of `decltype`, C++14's direct `auto` return type, and the original C++14 proposal. The slide is from conference.accu.org and includes the copyright notice "©2018 by IT-communication.com". The bottom right corner features the ACCU 2018 logo and date.

C++ Templates Revised - Nicolai Josuttis (ACCU 2018) (cont.)

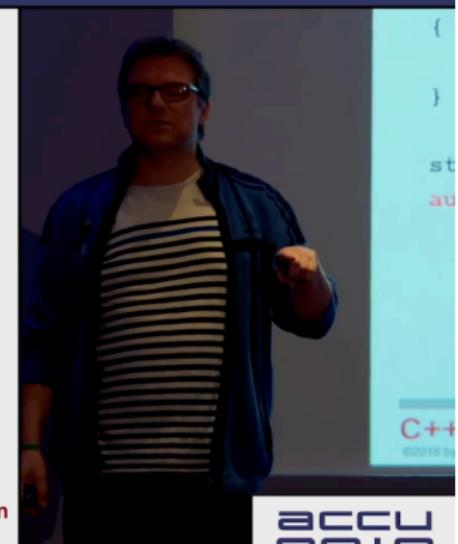
conference.accu.org

C++17: string_view Considered Harmful

```
std::string operator+ (std::string_view sv1, std::string_view sv2)
{
    return std::string(sv1) + std::string(sv2);
}

template<typename T>
T concat (const T& x, const T& y)
{
    return x + y;
}

std::string_view hi = "hi";
auto xy = concat(hi, hi); // OOPS: xy is std::string_view
```



C++
©2018 by IT communication.com

josuttis | eckstein
10 IT communication

@ACCUconf

ACCU
2018
April 11 - 14

C++ Templates Revised - Nicolai Josuttis (ACCU 2018) (cont.)

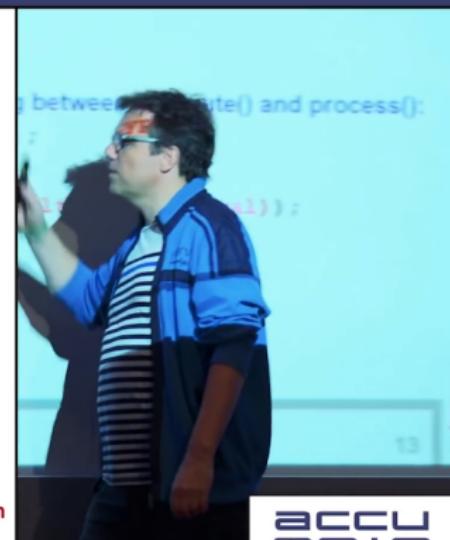
conference.accu.org

Perfect Forwarding of Local Objects

```
// pass return value of compute() to process():
process(compute(t));
```



```
// same, but doing something between compute() and process():
auto&& val = compute(t);
...
process(std::forward<decltype(val)>(val));
```



C++
©2018 by IT communication

13 josuttis | eckstein
IT communication

@ACCUconf

ACCU
2018
April 11 - 14

C++ Templates Revised - Nicolai Josuttis (ACCU 2018) (cont.)

conference.accu.org

Perfect Returning

```
template<typename Callable, typename... Args>
auto call(Callable op, Args&&... args)
{
    return op(std::forward<Args>(args)...);
}
```

auto

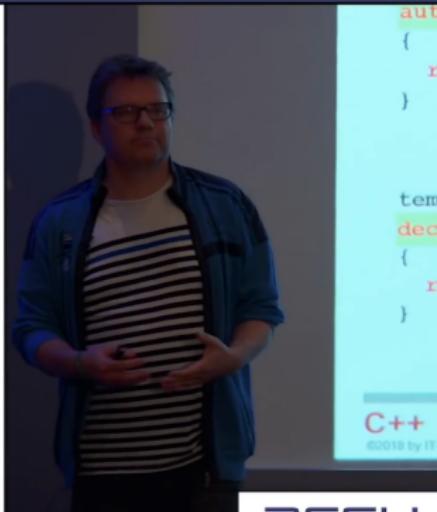
- can't return a reference
- because it decays (which usually is good)

```
template<typename Callable, typename... Args>
auto&& call(Callable op, Args&&... args)
{
    return op(std::forward<Args>(args)...);
}
```

**Fatal runtime error
(compilers may warn)**

```
template<typename Callable, typename... Args>
decltype(auto) call(Callable op, Args&&... args) // since C++14
{
    return op(std::forward<Args>(args)...);
}
```

returns perfectly



C++
©2018 by IT-communication.com

14 josuttis | eckstein
IT communication

@ACCUconf

ACCU
2018
April 11 - 14

C++ Templates Revised - Nicolai Josuttis (ACCU 2018) (cont.)

conference.accu.org

C++14: decltype(auto) and parentheses in return statements

```
decltype(auto) foo1()
{
    int i;
    return i;      // returns int (type of i)
}

decltype(auto) foo2()
{
    int i;
    return i + i; // returns int (type of i+i)
}

decltype(auto) foo3()
{
    int i;
    return (i);    // returns int& (i is lvalue)
}

std::is_same<decltype(foo1()), int>::value    // true
std::is_same<decltype(foo2()), int>::value    // true
std::is_same<decltype(foo3()), int&gt;::value // true!
```



Remember: decltype(auto)
uses the rules of decltype:
- type of entity e

- value category of expression e:
 - for prvalues: type
 - for lvalues: type&
 - for xvalues: type&&

Don't put parentheses around
return statements
(when returning named objects
and using decltype(auto))

C++

©2018 by IT communication.com

josuttis | eckstein

15

IT communication

@ACCUconf



ACCU
2018
April 11 - 14

C++ Templates Revised - Nicolai Josuttis (ACCU 2018) (cont.)

conference.accu.org

C++17: Perfect Returning of Local Objects

```
#include <utility>      // for std::invoke()
#include <functional>    // for std::forward()
#include <type_traits>   // for std::is_same<> and std::invoke_result<>

template<typename Callable, typename... Args>
decltype(auto) call(Callable&& op, Args&&... args)
{
    if constexpr(std::is_same_v<std::invoke_result_t<Callable, Args...>, void>)
        // return type is void:
        std::invoke(std::forward<Callable>(op),
                   std::forward<Args>(args)...);
    ...
    return;
}
else {
    // return type is not void:
    decltype(auto) ret{std::invoke(std::forward<Callable>(op),
                                  std::forward<Args>(args)...)};
    ...
    return ret;
}
```

invalid for void (because it is an incomplete type)



C++
©2018 by IT communication

16 josuttis | eckstein
IT communication

@ACCUconf

ACCU
2018
April 11 - 14

C++ Templates Revised - Nicolai Josuttis (ACCU 2018) (cont.)

conference.accu.org

C++11: Variadic Templates

```
template<typename T>
void print (T arg)
{
    std::cout << arg << ' ';
}

template<typename T, typename... Types>
void print (T arg1, Types... args)
{
    print(arg1);
    print(args...);
}
```

Formally OK
since C++17

```
std::string str = "world";
print("hi", 7.5, str); // hi 7.5 world
=> print("hi");
print(7.5, str);

=> print("hi");
print(7.5)
print(str);
```

Code effectively compiled:

```
std::string str = "world";
std::cout << "hi" << ' ';
std::cout << 7.5 << ' ';
std::cout << str << ' ';
```

C++

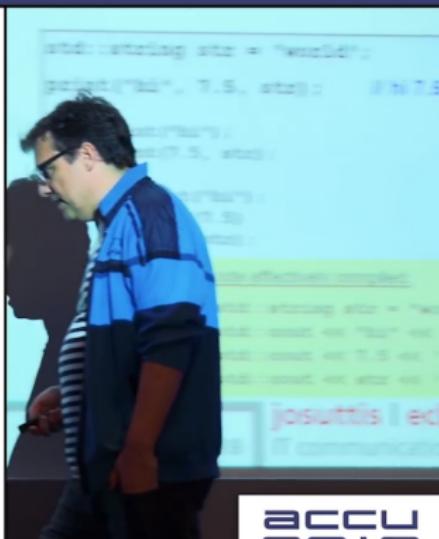
©2018 by IT-communication.com

josuttis | eckstein

19

IT communication

@ACCUconf



ACCU
2018
April 11 - 14

C++ Templates Revised - Nicolai Josuttis (ACCU 2018) (cont.)

conference.accu.org

Strings as Template Arguments

- Since C++17, non-type template parameters can be instantiated with strings with no linkage

```
template <const char* str>
class Message
{
    ...
};

extern const char hello[] = "Hello World!";           // external linkage
const char hello11[] = "Hello World!";                // internal linkage

void foo()
{
    Message<hello> msg;                            // OK, all versions
    Message<hello11> msg11;                          // OK since C++11

    static const char hello17[] = "Hello World!";     // no linkage
    Message<hello17> msg17;                          // OK since C++17
}
```

see N4268

The slide content includes a code snippet demonstrating the use of string literals as non-type template parameters. A callout bubble points to the string literal "Hello World!" in the first template instantiation with the text "see N4268". The right side of the slide shows a photograph of Nicolai Josuttis, the speaker, pointing towards a projection screen. The projection screen displays the ACCU logo and the text "C++ ©2018 by IT-communication.com". The bottom of the slide features the C++ logo, the author's name "josuttis | eckstein", the page number "28", and the copyright notice "©2018 by IT-communication.com". The footer also includes the ACCU 2018 logo and the date "April 11 - 14".

C++ Templates Revised - Nicolai Josuttis (ACCU 2018) (cont.)

conference.accu.org

C++17: auto and Strings for Non-Type Template Parameters

```
template <auto Msg>
class Message {
public:
    void print() {
        std::cout << Msg << '\n';
    }
};

Message<'x'> m1;
m1.print();           // outputs: x

Message<42> m2;
m2.print();           // outputs: 42

Message<7.5> m3;    // Error: template parameter type still cannot be double
static const char s[] = "hello";
Message<s> m4;       // OK since C++17
m4.print();           // outputs: hello
```

C++

©2018 by IT communication

29

josuttis | eckstein
IT communication

@ACCUconf



ACCU
2018
April 11 - 14

C++ Templates Revised - Nicolai Josuttis (ACCU 2018) (cont.)

conference.accu.org

C++17: Using Fold Expressions with auto Template Parameters

```
template<auto sep = ' ',  
        typename T1, typename... Types>  
void print (const T1& arg1, const Types&... args)  
{  
    std::cout << arg1;  
    (... , [](const auto& arg) {  
        std::cout << sep << arg;  
    })(args);  
}
```

```
std::string str = "world";  
print("hi", 7.5, str);           // hi 7.5 world  
  
print<'->("hi", 7.5, str);   // hi-7.5-world  
  
static const char sep[] = ", ";  
print<sep>("hi", 7.5, str);   // hi, 7.5, world
```

C++ Templates Revised - Nicolai Josuttis (ACCU 2018) (cont.)

conference.accu.org

C++17: auto and Strings for Non-Type Template Parameters

```
template <decltype(auto) Msg>
class Message {
public:
    void print() {
        std::cout << Msg << '\n';
    }
};

Message<42> m2;
m2.print();           // outputs: 42

extern int x;         // external linkage!
x = 42;
Message<(x)> m5;    // OK: Msg deduced as int&
m5.print();          // outputs: 42
x *= 10;
m5.print();          // outputs: 420
```

Remember: decltype(auto)
uses the rules of decltype:
- type of entity e
- value category of expression e:
- for prvalues: type
- for lvalues: type&
- for xvalues: type&&



C++ Templates Revised - Nicolai Josuttis (ACCU 2018) (cont.)

conference.accu.org

```
C++17: std::variant<> Visitors  
C++17 Features:  
• std::variant<>  
• Aggregates can have base classes  
• Deduction guides for class template arguments  
  
std::variant<int, std::string> var{42};  
...  
  
template<typename... Functors>  
struct Ovld : Functors... {  
    using Functors::operator()...; // derive op() from all functors  
};  
// deduce template arguments from initializers:  
template<typename... Functors> Ovld(Functors...) -> Ovld<Functors...>;  
  
  
std::visit(Ovld{[](const auto& val) { // call lambda for the appropriate type  
    std::cout << val << '\n';  
},  
    [](int i) {  
        std::cout << "int: " << i << '\n';  
},  
    ...  
},  
var);
```

C++

©2018 by IT communication.com

38

josuttis | eckstein
IT communication

@ACCUconf

....>;

type

stein

ACCU
2018
April 11 - 14

C++ Templates Revised - Nicolai Josuttis (ACCU 2018) (cont.)

conference.accu.org

Template Metaprogramming with C++98

```
template <int p, int d>           // p: prime, d: divisor
struct DoIsPrime {
    enum { value = (p%d != 0) && DoIsPrime<p,d-1>::value };
};

template <int p>                 // end recursion if divisor is 2
struct DoIsPrime<p,2> {
    enum { value = (p%2 != 0) };
};

template <int p>
struct IsPrime {
    enum { value = DoIsPrime<p,p/2>::value }; // start recursion with divisor from p/2 to 2
};

// handle special cases:
template <> struct IsPrime<0> { enum { value = false }; };
template <> struct IsPrime<1> { enum { value = false }; };
template <> struct IsPrime<2> { enum { value = true }; };
template <> struct IsPrime<3> { enum { value = true }; };

IsPrime<9>::value
⇒ DoIsPrime<9,4>::value
⇒ 9%4!=0 && DoIsPrime<9,3>::value
⇒ 9%4!=0 && 9%3!=0 && DoIsPrime<9,2>::value
⇒ 9%4!=0 && 9%3!=0 && 9%2!=0
```

- Compile-time "if"
- implemented via template function and its specialization



C++

©2018 by IT communication.com

josuttis | eckstein

40

IT communication

@ACCUconf

ACCU
2018
April 11 - 14

C++ Templates Revised - Nicolai Josuttis (ACCU 2018) (cont.)

conference.accu.org

Template Metaprogramming with static constexpr since C++11

```
template <int p, int d>           // p: prime, d: divisor
struct DoIsPrime {
    static constexpr bool value = (p+d != 0) && DoIsPrime<p,d-1>::value;
};

template <int p>                 // end recursion if divisor is 2
struct DoIsPrime<p,2> {
    static constexpr bool value = (p%2 != 0);
};

template <int p>
struct IsPrime {
    static constexpr bool value = DoIsPrime<p,p/2>::value; // start recursion with divisor from p/2 to 2
};

// handle special cases:
template <> struct IsPrime<0> { static constexpr bool value = false; };
template <> struct IsPrime<1> { static constexpr bool value = false; };
template <> struct IsPrime<2> { static constexpr bool value = true; };
template <> struct IsPrime<3> { static constexpr bool value = true; };

IsPrime<9>::value
⇒ DoIsPrime<9,4>::value
⇒ 9%4!=0 && DoIsPrime<9,3>::value
⇒ 9%4!=0 && 9%3!=0 && DoIsPrime<9,2>::value
⇒ 9%4!=0 && 9%3!=0 && 9%2!=0
```

Compile-time "if"
• implemented via
template function and
its specialization



C++

©2018 by IT communication.com

41

josuttis | eckstein

IT communication

@ACCUconf

ACCU
2018
April 11 - 14

C++ Templates Revised - Nicolai Josuttis (ACCU 2018) (cont.)

conference.accu.org

Compile-Time Programming with `constexpr` since C++11

```
constexpr bool doIsPrime (int p, int d)           // p: prime, d: divisor
{
    return d!=2 ? (p%d!=0) && doIsPrime(p,d-1)
                : (p%2!=0);                         // end recursion if divisor is 2
}

constexpr bool isPrime (int p)
{
    return p < 4 ? p>1                         // handle special cases
                  : doIsPrime(p,p/2); // start recursion with divisor from p/2 to 2
}

isPrime(9)          // false (at compile-time if needed)
isPrime(11)         // true (at compile-time if needed)

        isPrime(9)
        => doIsPrime(9,4)
        => 9%4!=0 && doIsPrime(9,3)
        => 9%4!=0 && 9%3!=0 && doIsPrime(9,2)
        => 9%4!=0 && 9%3!=0 && 9%2!=0
```

- Restrictions in C++11:
- only one statement
- which returns
- with literal types

// handle special cases
// start recursion with divisor from p/2 to 2

me if needed)
me if needed)

(9)

isPrime (9, 4)

!=0 && doIsPrime

!=0 && 9%3

!=0 && 9%

me (9, 2)

all ec

communicati



C++

©2018 by IT communication.com

josuttis | eckstein

42

IT communication

@ACCUconf

ACCU
2018
April 11 - 14

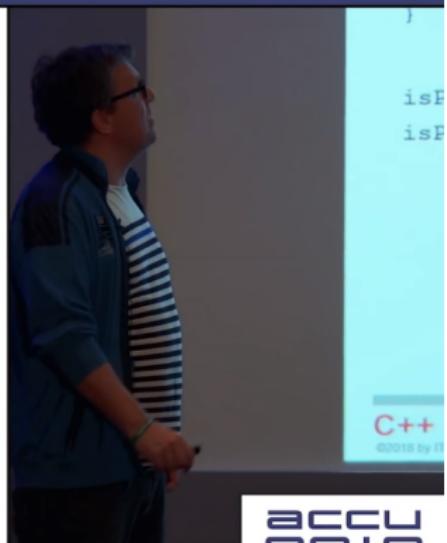
C++ Templates Revised - Nicolai Josuttis (ACCU 2018) (cont.)

conference.accu.org

Compile-Time Programming with `constexpr` since C++14

```
constexpr bool isPrime (unsigned int p)
{
    for (unsigned int d=2; d<=p/2; ++d) {
        if (p % d == 0) {
            return false; // found divisor without rest
        }
    }
    return p > 1; // is prime if no divisor found and > 1
}

isPrime(9)      // false (at compile-time if needed)
isPrime(11)      // true (at compile-time if needed)
```



Your handy cut-out-and-keep guide to std::forward and std::move

- ▶ Article

CRTP refresher by Jonathan Bocvara

- ▶ Part 1
- ▶ Part 2
- ▶ Part 3
- ▶ Part 4

Favourite C++ coding standards

- ▶ Reddit
- ▶ C++ Core Guidelines
- ▶ High Integrity C++ Coding Standard
- ▶ Geosoft C++ Programming Practice Guidelines
- ▶ Webkit Code Style Guidelines

Twitter

 Dan Luu
@danluu

↑ 1 Reply, 7 Quotes



A creator of the C++ STL says they might've used more cache-friendly data structures if HP Labs had the budget to buy HP PA-RISC machines?

(via [@daniel_bilar](#))

06/03/2018, 20:27 (6 days ago)
Twitter Web Client

Retweeted by [@AlisdairMered](#)
10/03/2018, 16:10

381 Likes	174 Retweets	Thread >
-----------	--------------	----------

Alex: STL is "only" two decades old, but yes, there have been important changes during that period that would lead to some different decisions. STL was actually designed on a Leading Edge PC with no cache and 640K memory. (Our group at HP Labs didn't have enough money in the budget for an HP PC. When HP CEO Lew Platt came to visit me, HP Labs' director rushed in beforehand to hide the Leading Edge PC.)

One of the biggest changes since then has been the growth of caches. Cache misses are very costly, so locality of reference is much more important now. Node-based data structures, which have low locality of reference, make much less sense. If I were designing STL today, I would have a different set of containers. For example, an in-memory B*-tree is a far better choice than a red-black tree for implementing an associative container.