libc++: A Standard Library for C++0x

2010 LLVM Developers' Meeting

Why?

Why?

- Another C++ standard library?

Why?

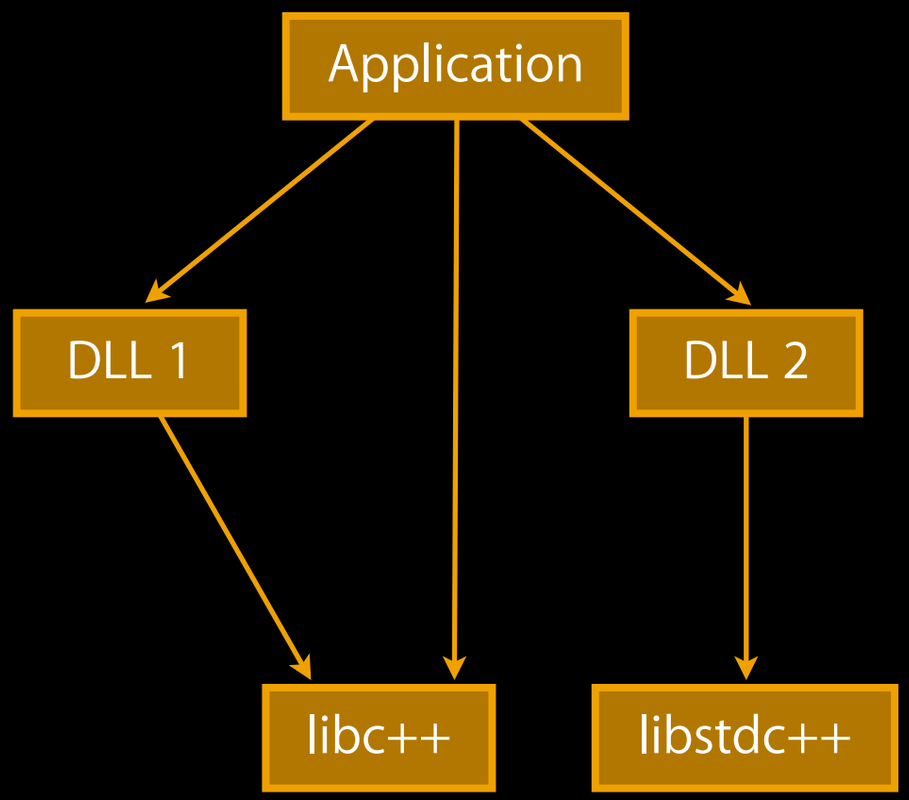
- Another C++ standard library?
- The C++0x spec introduces several fundamentally new ideas at the language level.
  - Move semantics
  - Perfect forwarding
  - Variadic templates

Why?

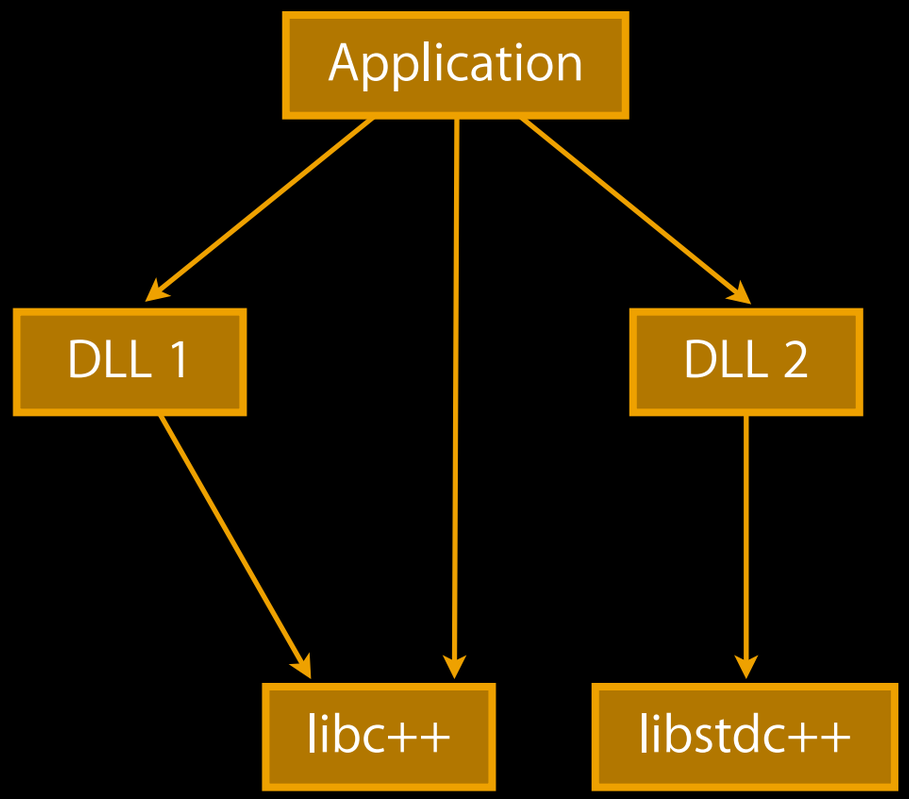
- libc++ is designed from the ground up to take advantage of these new language features.
  - This is not a C++0X implementation layered on top of a C++03 implementation.
  - It has been a C++0X implementation from the beginning.
  - This has driven several low-level design decisions.

Overall Design

libstdc++ (gcc) Interoperability

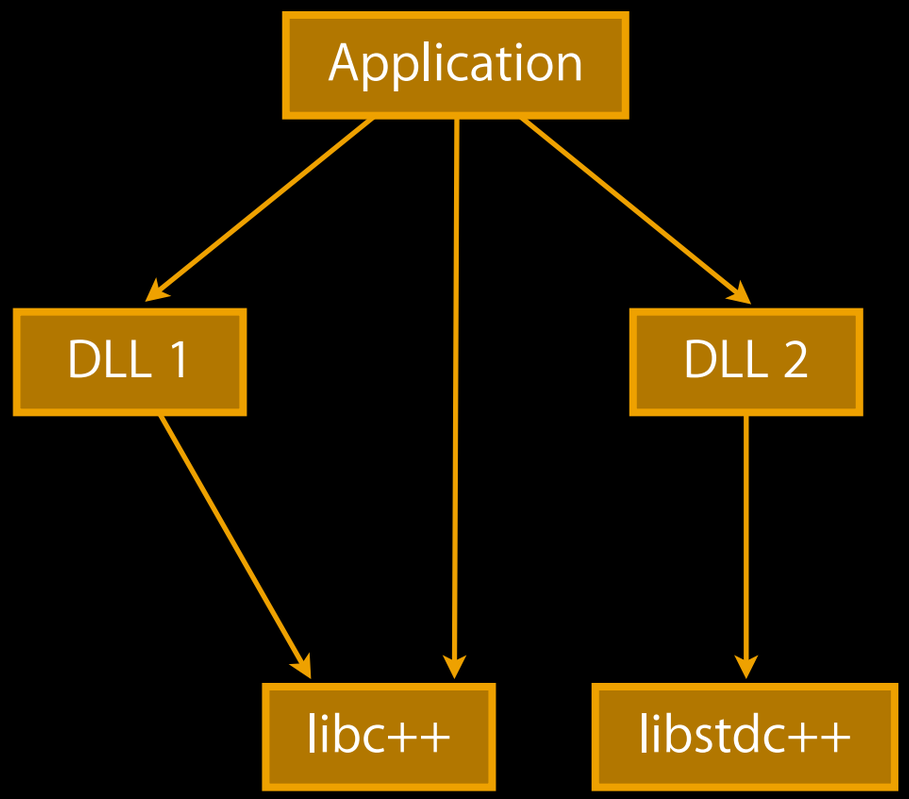


libstdc++ (gcc) Interoperability



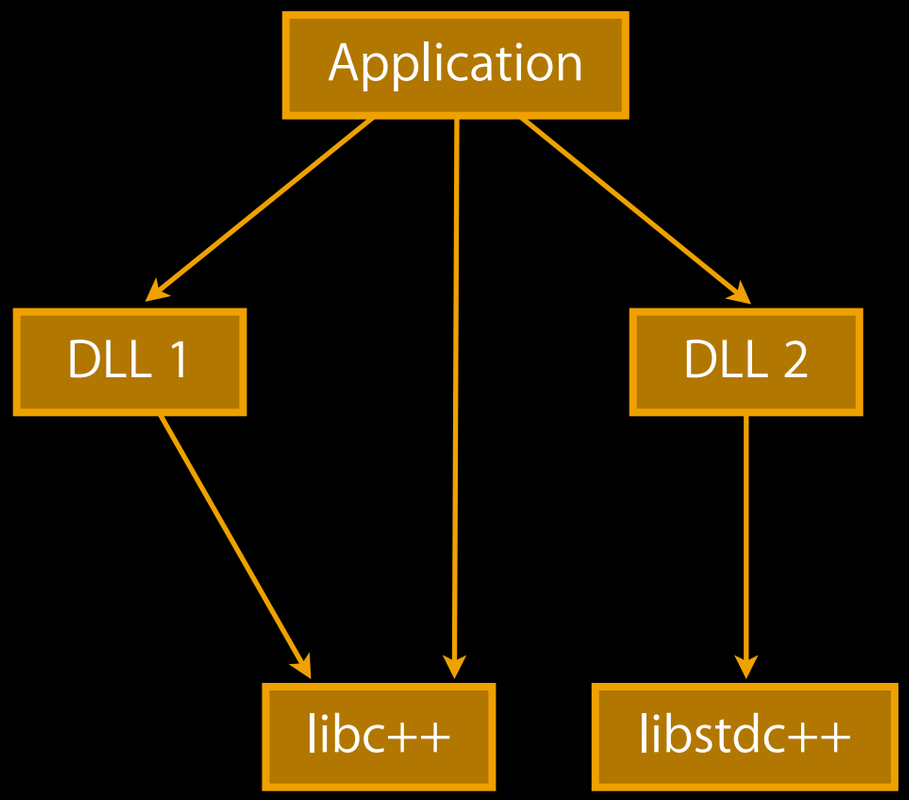
- We expect that both libc++ and libstdc++ will end up in the same application.

libstdc++ (gcc) Interoperability



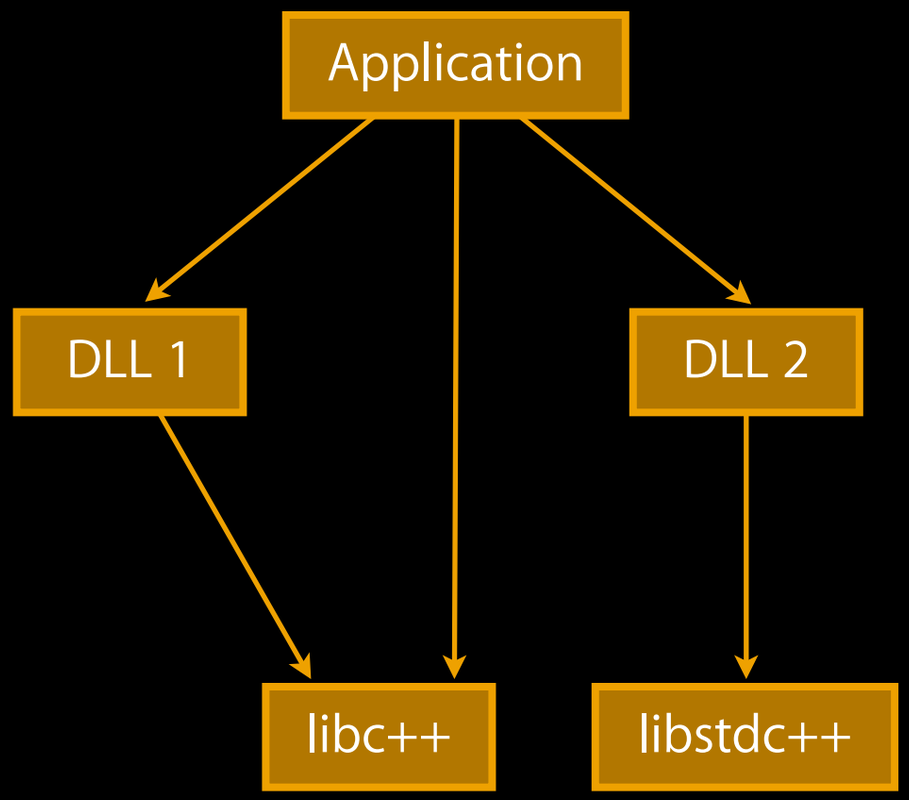
- We expect that both libc++ and libstdc++ will end up in the same application.
- libc++ is versioned using inline namespaces so that ABI-incompatible objects can not accidentally be mistaken for one another.

libstdc++ (gcc) Interoperability



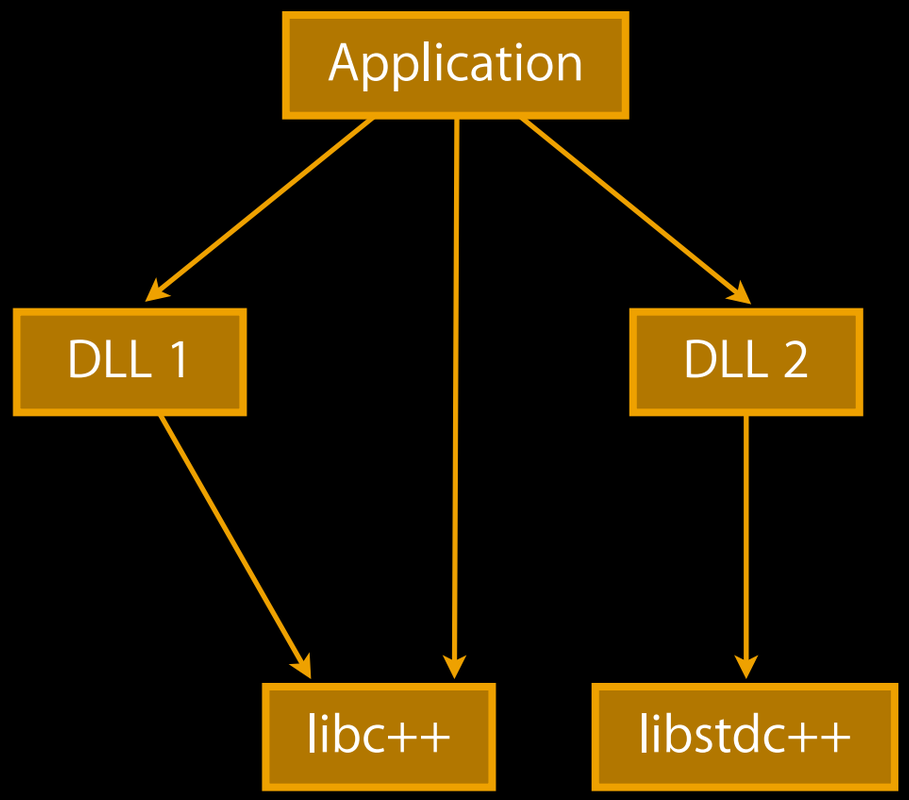
- We expect that both libc++ and libstdc++ will end up in the same application.
- libc++ is versioned using inline namespaces so that ABI-incompatible objects can not accidentally be mistaken for one another.
- Some low-level facilities are not versioned, and are ABI-compatible with libstdc++.

libstdc++ (gcc) Interoperability



- We expect that both libc++ and libstdc++ will end up in the same application.
- libc++ is versioned using inline namespaces so that ABI-incompatible objects can not accidentally be mistaken for one another.
- Some low-level facilities are not versioned, and are ABI-compatible with libstdc++.
  - Operator new/delete

libstdc++ (gcc) Interoperability



- We expect that both libc++ and libstdc++ will end up in the same application.
- libc++ is versioned using inline namespaces so that ABI-incompatible objects can not accidentally be mistaken for one another.
- Some low-level facilities are not versioned, and are ABI-compatible with libstdc++.
  - Operator new/delete
  - Exceptions

libc++ <headers>

libc++ <headers>

- Every public header begins with a synopsis of that header in comments (a quick reference).

libc++ <headers>

- Every public header begins with a synopsis of that header in comments (a quick reference).
- Every header that is not a public header has a name beginning with "__" (e.g. <__hash_table>).

libc++ <headers>

- Every public header begins with a synopsis of that header in comments (a quick reference).
- Every header that is not a public header has a name beginning with "__" (e.g. <__hash_table>).
- The number of private headers is kept to a minimum to enable faster compile times.
  - A private header is only introduced when needed to break cyclic dependencies, or to factor out code needed in two places.

libc++ <headers>

- Every public header begins with a synopsis of that header in comments (a quick reference).
- Every header that is not a public header has a name beginning with "__" (e.g. <__hash_table>).
- The number of private headers is kept to a minimum to enable faster compile times.
  - A private header is only introduced when needed to break cyclic dependencies, or to factor out code needed in two places.
  - Headers are not used to "moduralize" code.
    - All of the regular expression library is in <regex>.
    - All of the random number library is in <random>.
    - etc.

The Build System

The Build System

- The build system is purposefully simplistic.
  - Downright primitive!

The Build System

- The build system is purposefully simplistic.
  - Downright primitive!
  - You don’t need to build llvm or clang to build libc++.

The Build System

- The build system is purposefully simplistic.
  - Downright primitive!
  - You don’t need to build llvm or clang to build libc++.
  - No configure.
  - Configuration is accomplished via the header <__config>.

The Build System

- The build system is purposefully simplistic.
  - Downright primitive!
  - You don’t need to build llvm or clang to build libc++.
  - No configure.
  - Configuration is accomplished via the header <__config>.
- Consequently the entire library builds in **45–60 seconds.**

The Test System

The Test System

- Every single sub-section in the C++0X draft standard is represented by a test subdirectory.

The Test System

- Every single sub-section in the C++0X draft standard is represented by a test subdirectory.
- The draft standard section hierarchy is mirrored in the libc++ test directory.

The Test System

- Every single sub-section in the C++0X draft standard is represented by a test subdirectory.
- The draft standard section hierarchy is mirrored in the libc++ test directory.
- If a test subdirectory has no tests in it, the libc++ test harness counts that part of the library as unimplemented.

The Test System

- Every single sub-section in the C++0X draft standard is represented by a test subdirectory.
- The draft standard section hierarchy is mirrored in the libc++ test directory.
- If a test subdirectory has no tests in it, the libc++ test harness counts that part of the library as unimplemented.
- The test harness is purposefully simplistic…
  - $ cd test; testit

The Test System

- Every single sub-section in the C++0X draft standard is represented by a test subdirectory.
- The draft standard section hierarchy is mirrored in the libc++ test directory.
- If a test subdirectory has no tests in it, the libc++ test harness counts that part of the library as unimplemented.
- The test harness is purposefully simplistic…
  - $ cd test; testit
- One can cd into any subdirectory and run the tests just for that section and its subdirectories.
  - $ cd <where ever>; testit

A Few libc++ Examples of Excellence…

string

string

- Not reference counted:
  - No atomic increment/decrement

string

- Not reference counted:
  - No atomic increment/decrement
- Fast default constructor:
  - The default constructed string can hold up to 22 chars before needing to allocate memory (on 64-bit platforms)

string

- Not reference counted:
  - No atomic increment/decrement
- Fast default constructor:
  - The default constructed string can hold up to 22 chars before needing to allocate memory (on 64-bit platforms)

```
movq  $0, (%rdi)
movq  $0, 8(%rdi)
movq  $0, 16(%rdi)
```

string

- Not reference counted:
  - No atomic increment/decrement
- Fast default constructor:
  - The default constructed string can hold up to 22 chars before needing to allocate memory (on 64-bit platforms)
- Fast move constructor:
  - Copy 3 words, zero 3 words.
  - No branching, no allocation.

```
movq  $0, (%rdi)
movq  $0, 8(%rdi)
movq  $0, 16(%rdi)
```

string

- Not reference counted:
  - No atomic increment/decrement
- Fast default constructor:
  - The default constructed string can hold up to 22 chars before needing to allocate memory (on 64-bit platforms)
- Fast move constructor:
  - Copy 3 words, zero 3 words.
  - No branching, no allocation.
- This design considered the importance of move semantics from the beginning: minimum `sizeof` leads to faster moving.

```
movq  $0, (%rdi)
movq  $0, 8(%rdi)
movq  $0, 16(%rdi)
```

Allocator Aware Containers

Allocator Aware Containers

- All dynamically sized containers have an O(1), non-throwing default constructor—extremely fast.
  - Embedded sentinel nodes for all node-based containers that require a sentinel node.

Allocator Aware Containers

- All dynamically sized containers have an O(1), non-throwing default constructor—extremely fast.
  - Embedded sentinel nodes for all node-based containers that require a sentinel node.
- All containers meet all of the latest allocator requirements which fully support stateful allocators.
  - Space for stateless allocators optimized away.

Allocator Aware Containers

- All dynamically sized containers have an O(1), non-throwing default constructor—extremely fast.
  - Embedded sentinel nodes for all node-based containers that require a sentinel node.
- All containers meet all of the latest allocator requirements which fully support stateful allocators.
  - Space for stateless allocators optimized away.
- Associative containers optimize away the space for stateless comparators (and stateless hash functions for the unordered containers).

libc++ vs g++-4.2 libstdc++

(64 bit platform)

libc++ vs g++-4.2 libstdc++

(64 bit platform)

deque<int>	libc++	libstdc++
sizeof	48 bytes	80 bytes
Default ctor allocation	0 bytes	576 bytes

libc++ vs g++-4.2 libstdc++

(64 bit platform)

map<int, int>	libc++	libstdc++
sizeof	24 bytes	48 bytes
Default ctor allocation	0 bytes	0 bytes

libc++ vs g++-4.2 libstdc++

(64 bit platform)

unordered_map<int, int>	libc++	libstdc++
sizeof	40 bytes	48 bytes
Default ctor allocation	0 bytes	96 bytes

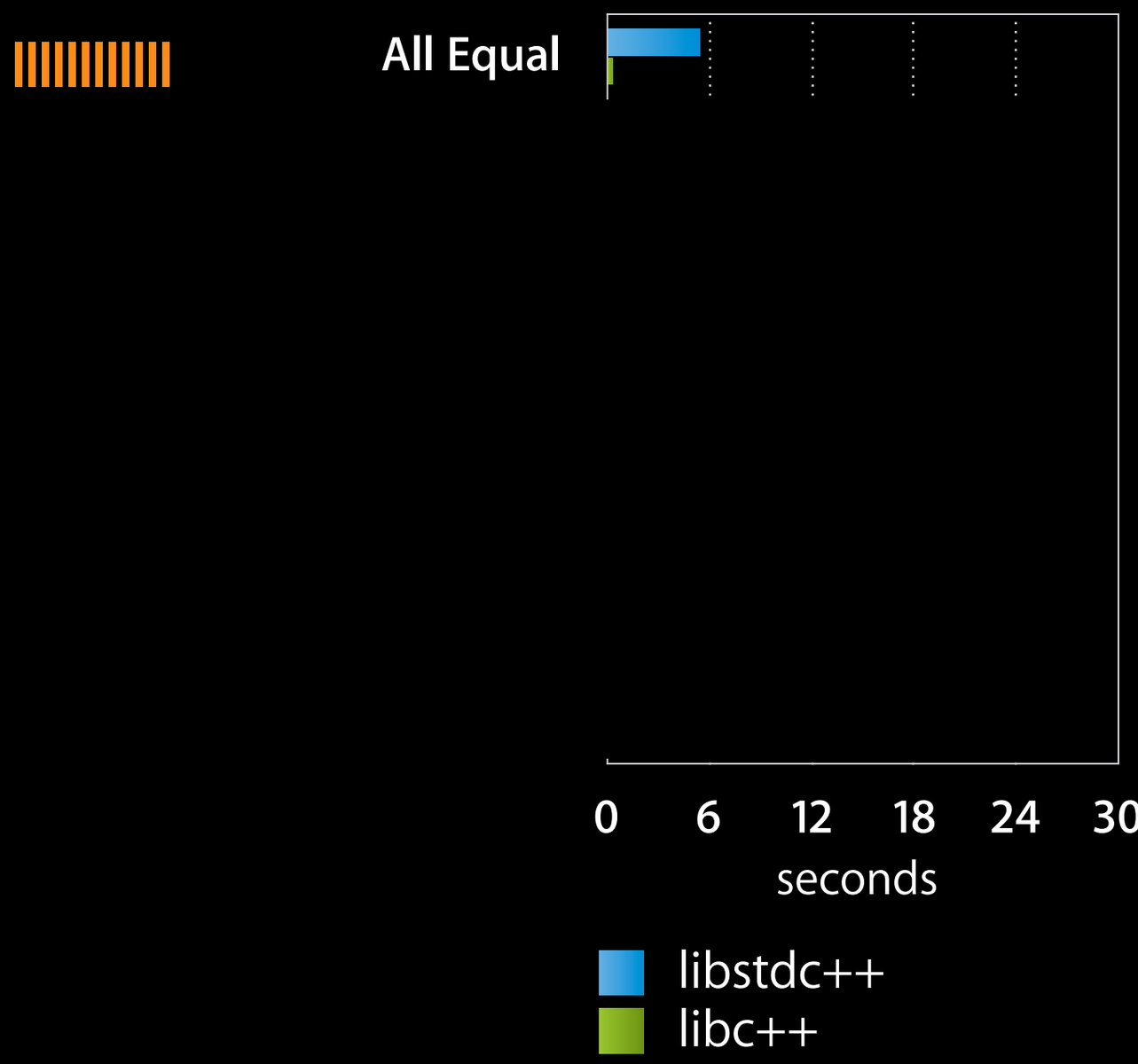
sort

sort

- Fast
  - Never copies, only swaps or moves.

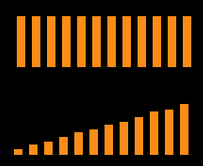
sort

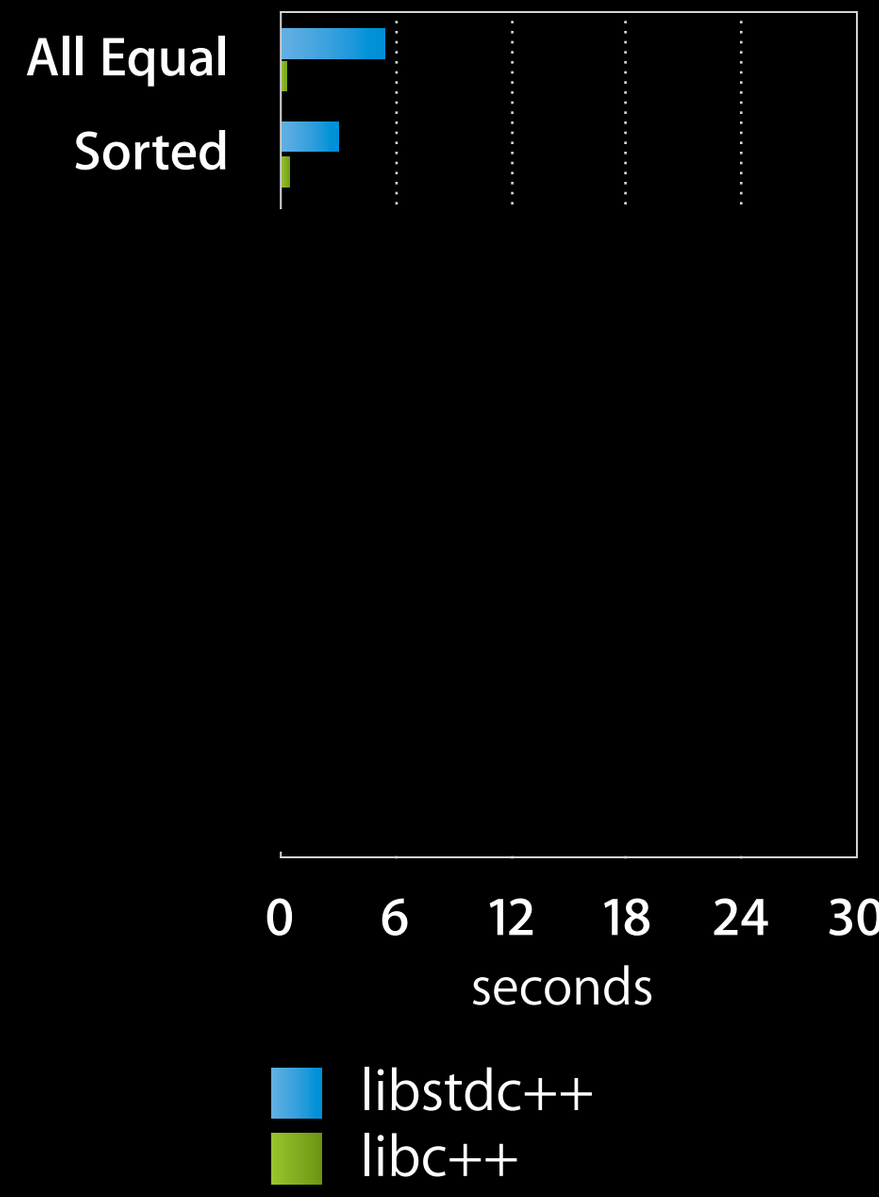
- Fast
  - Never copies, only swaps or moves.
  - Automatically recognizes and adapts to patterns.



sort

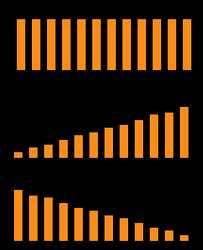
- Fast
  - Never copies, only swaps or moves.
  - Automatically recognizes and adapts to patterns.

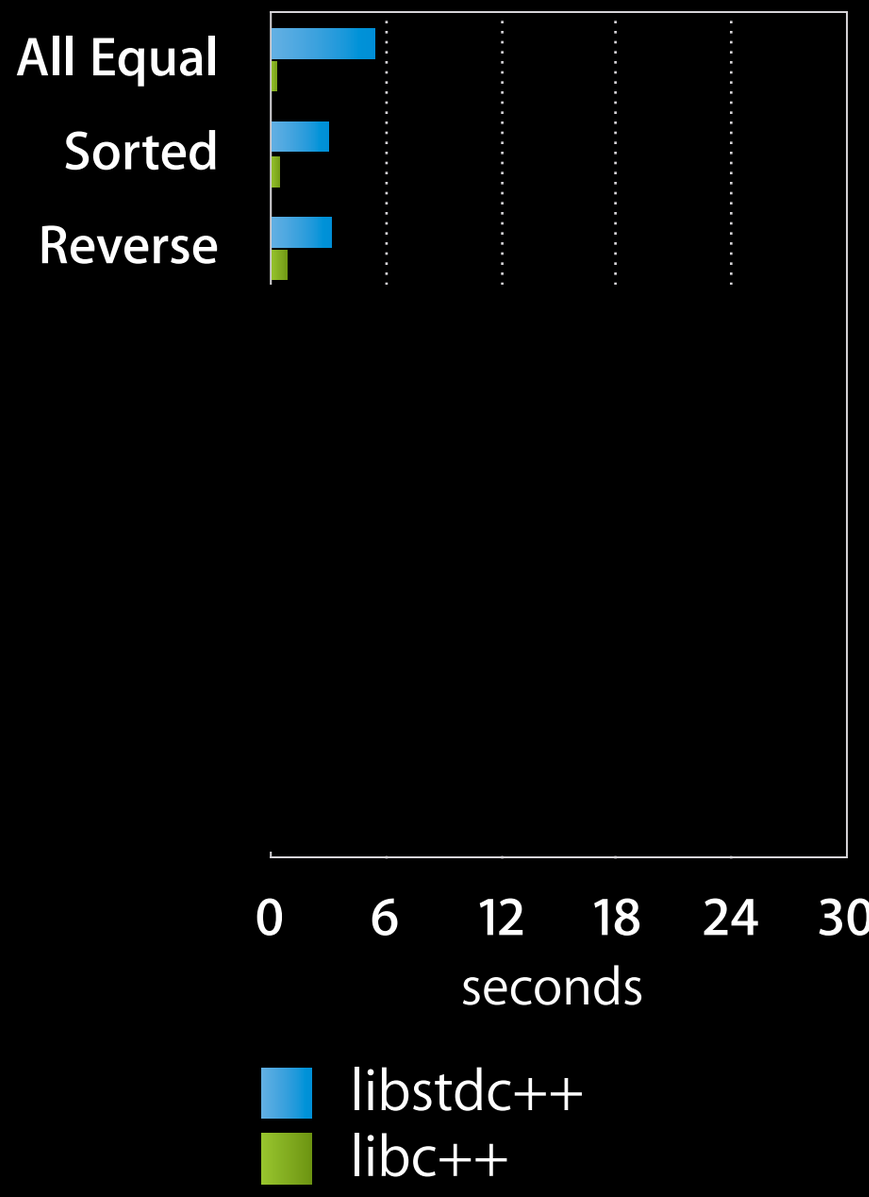




sort

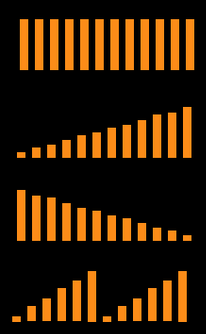
- Fast
  - Never copies, only swaps or moves.
  - Automatically recognizes and adapts to patterns.

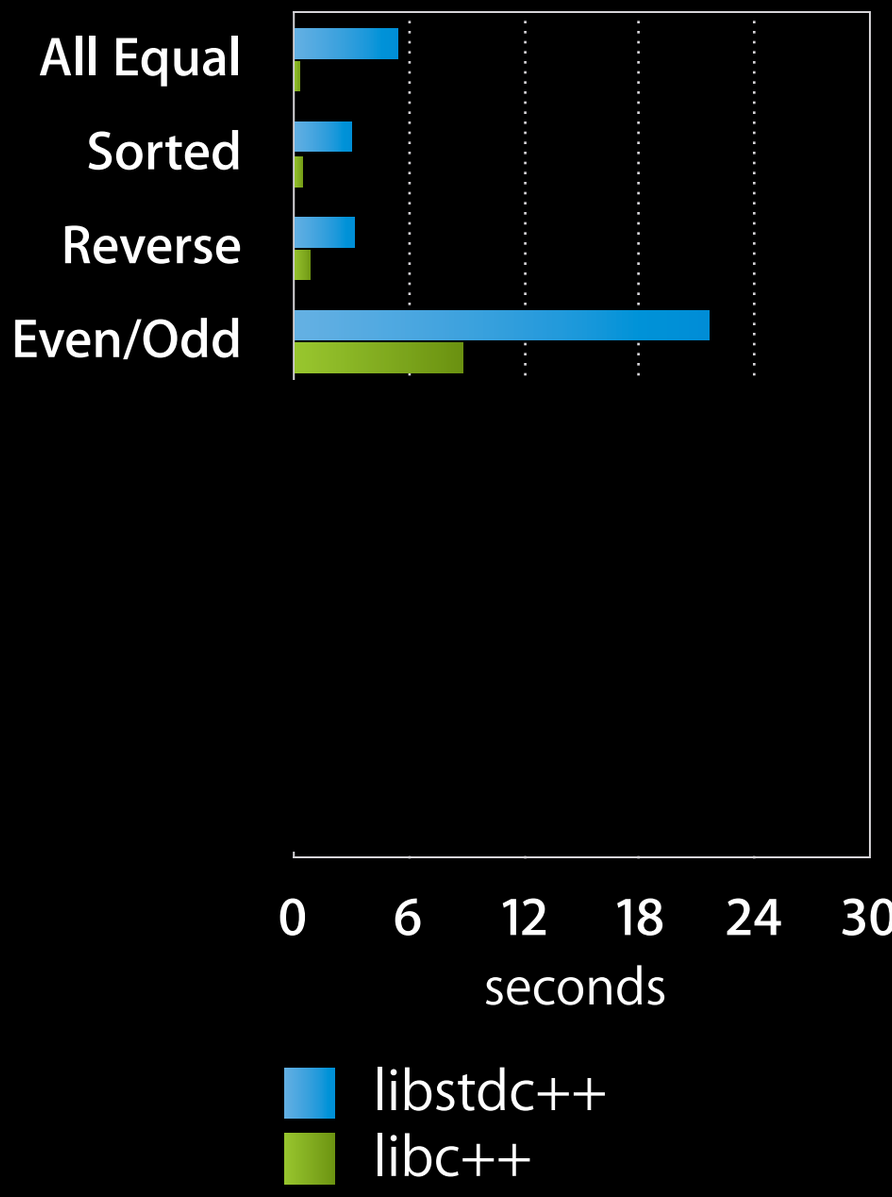




sort

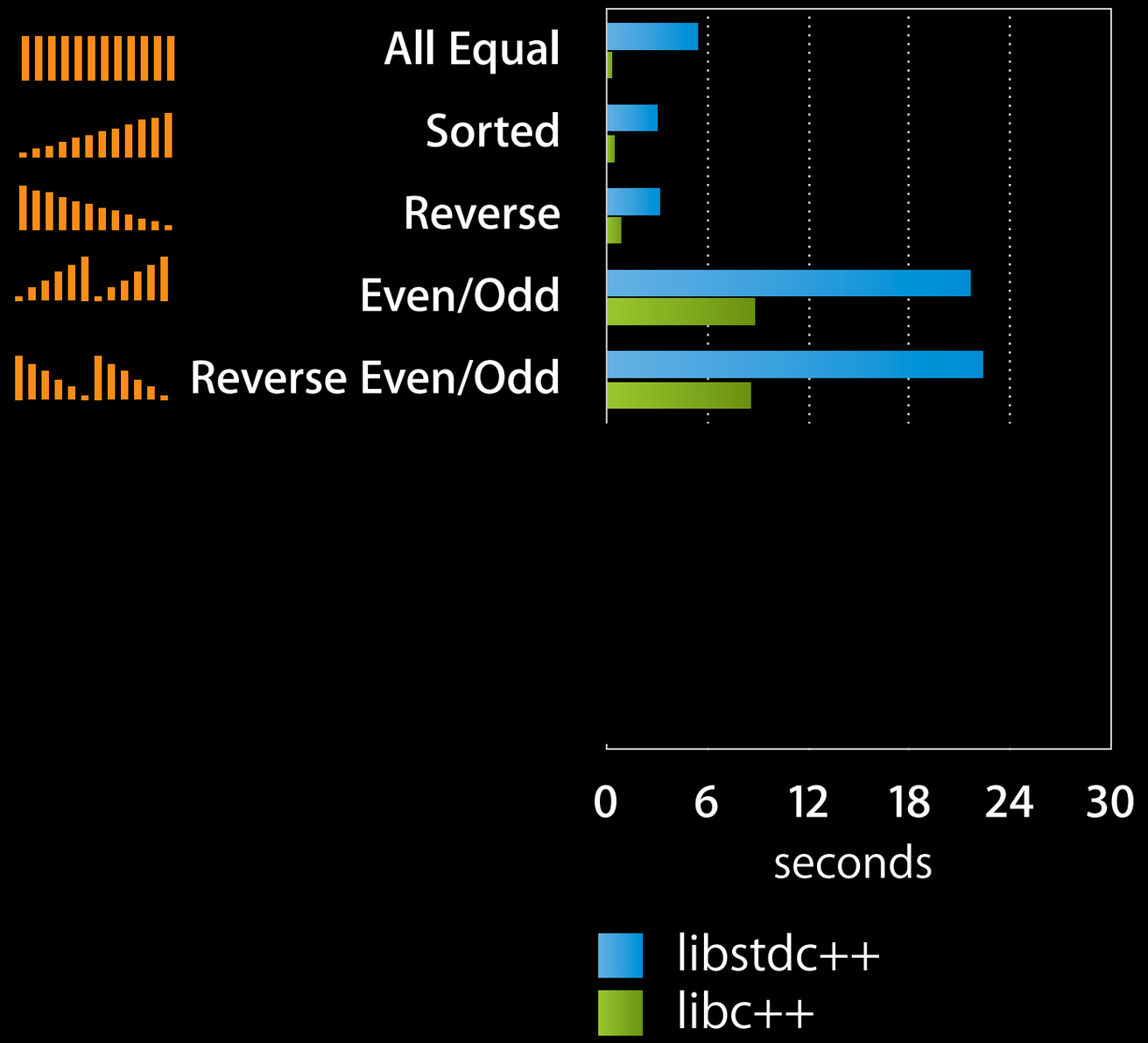
- Fast
  - Never copies, only swaps or moves.
  - Automatically recognizes and adapts to patterns.





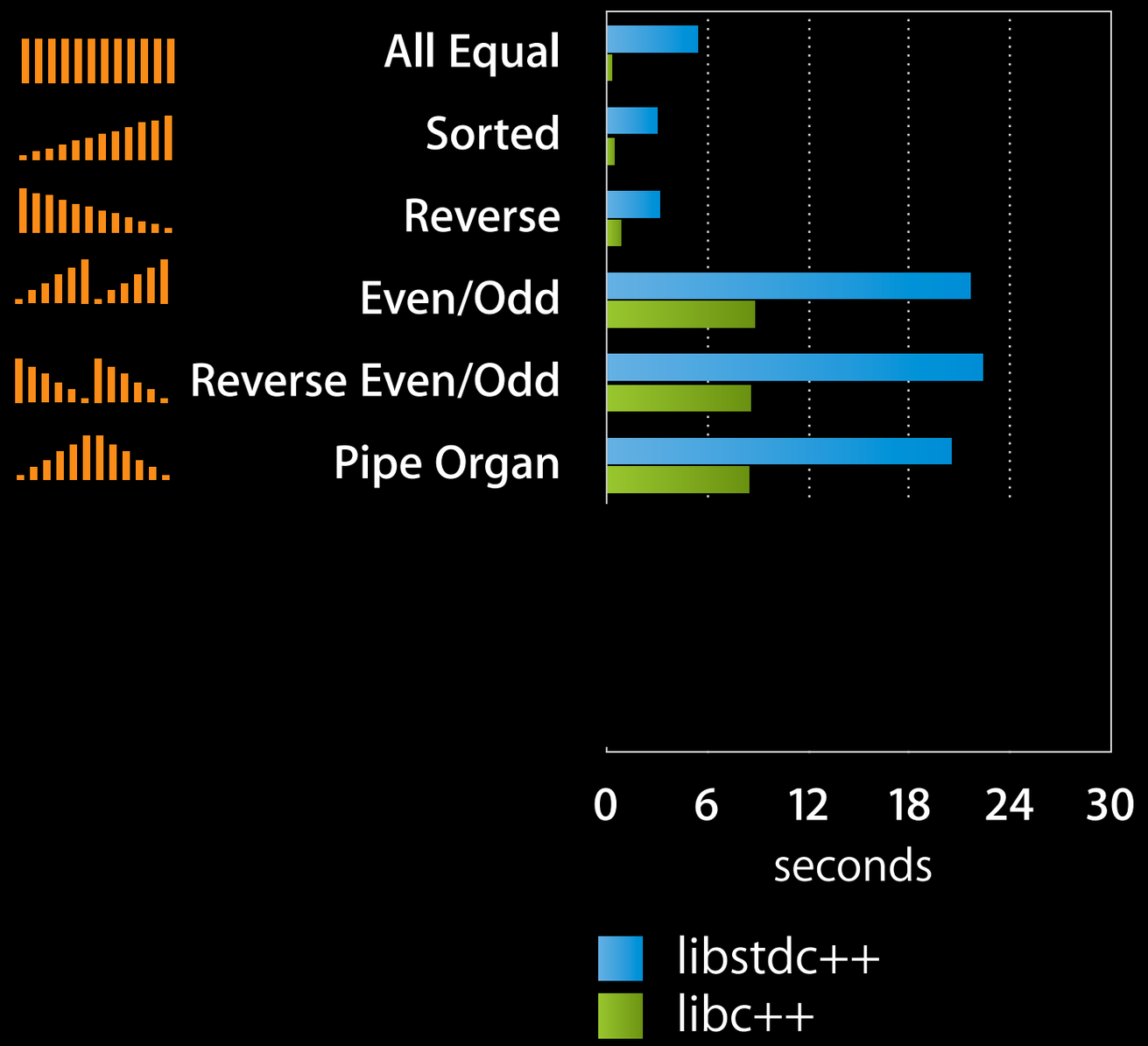
sort

- Fast
  - Never copies, only swaps or moves.
  - Automatically recognizes and adapts to patterns.



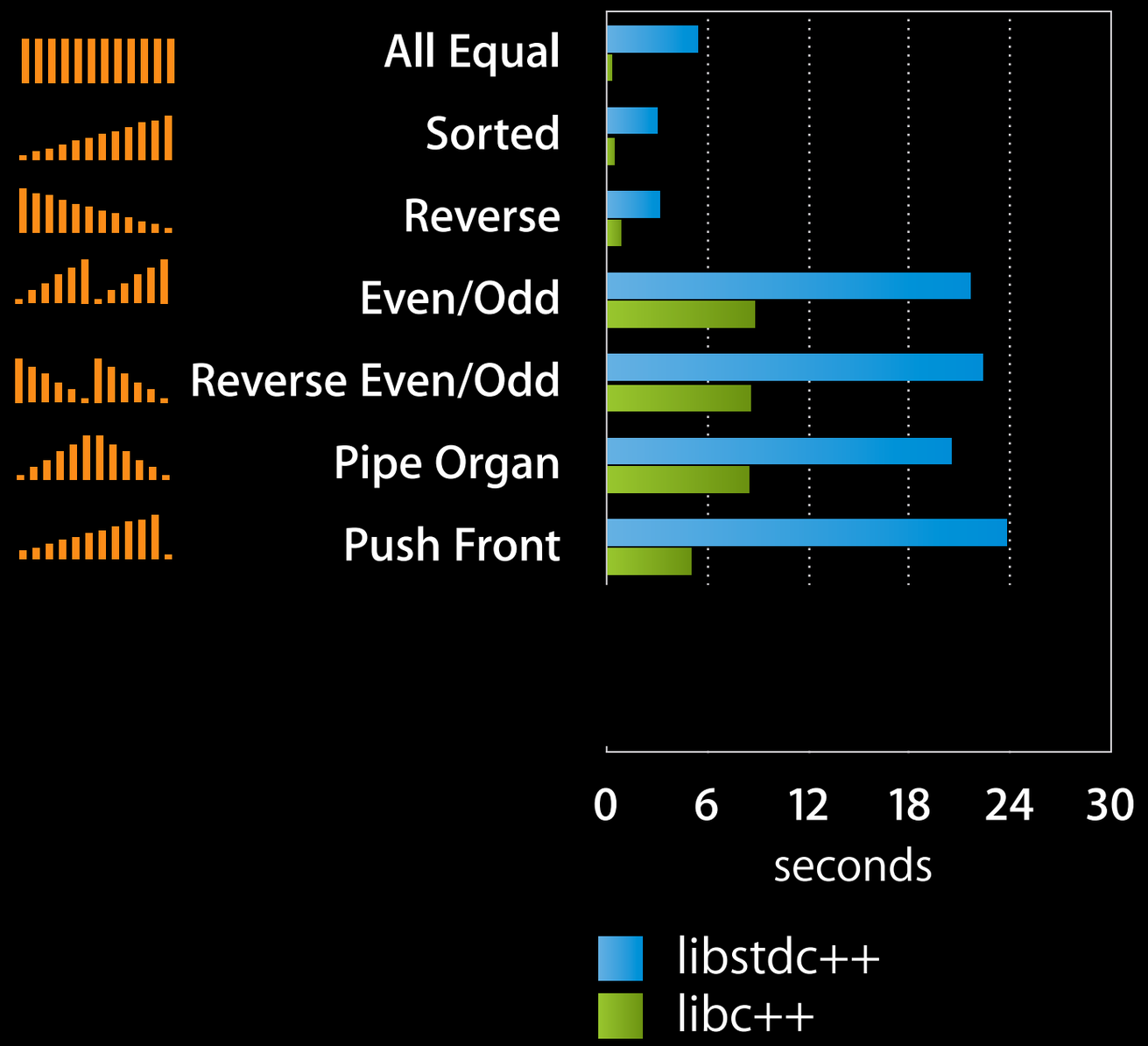
sort

- Fast
  - Never copies, only swaps or moves.
  - Automatically recognizes and adapts to patterns.



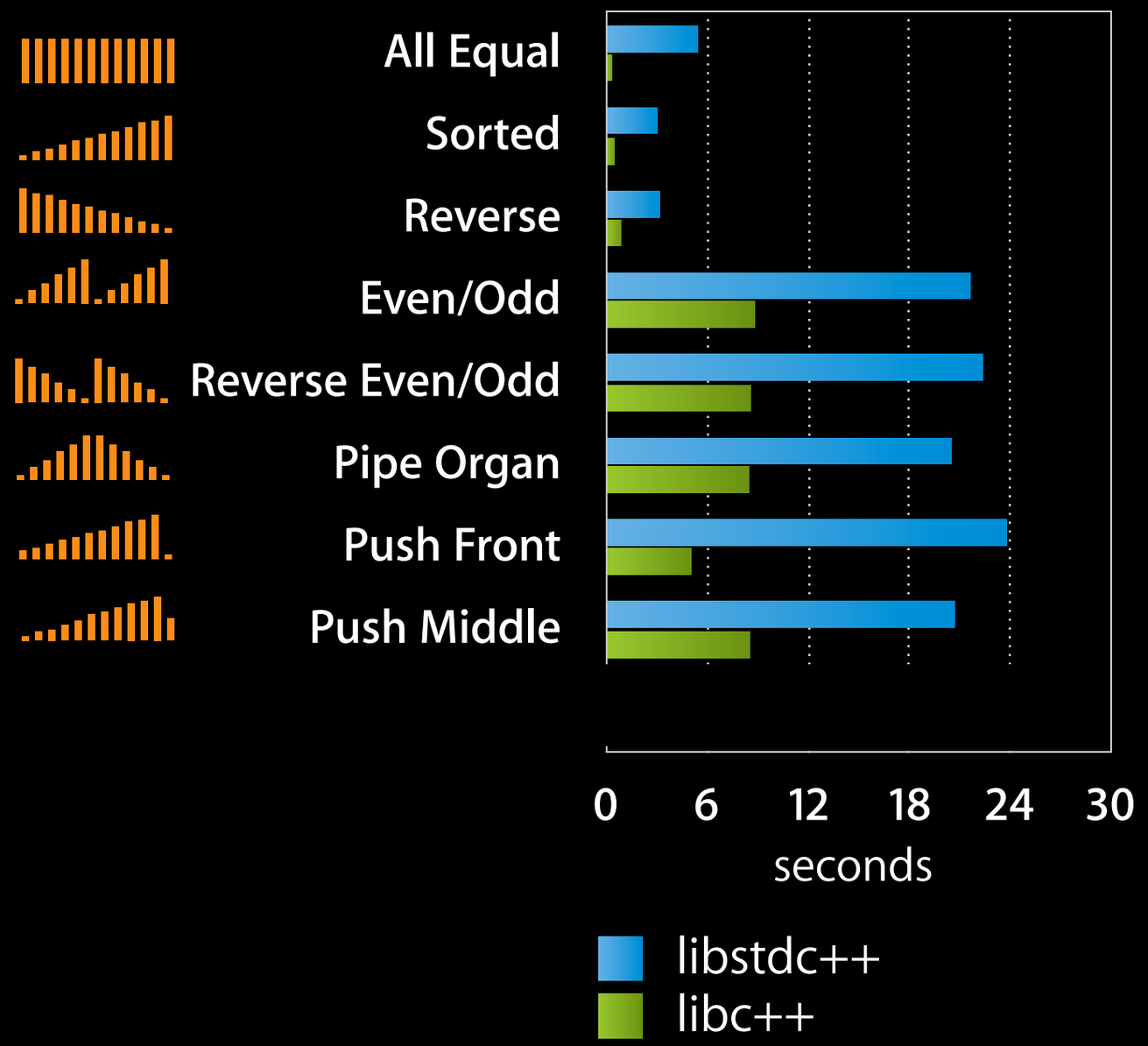
sort

- Fast
  - Never copies, only swaps or moves.
  - Automatically recognizes and adapts to patterns.



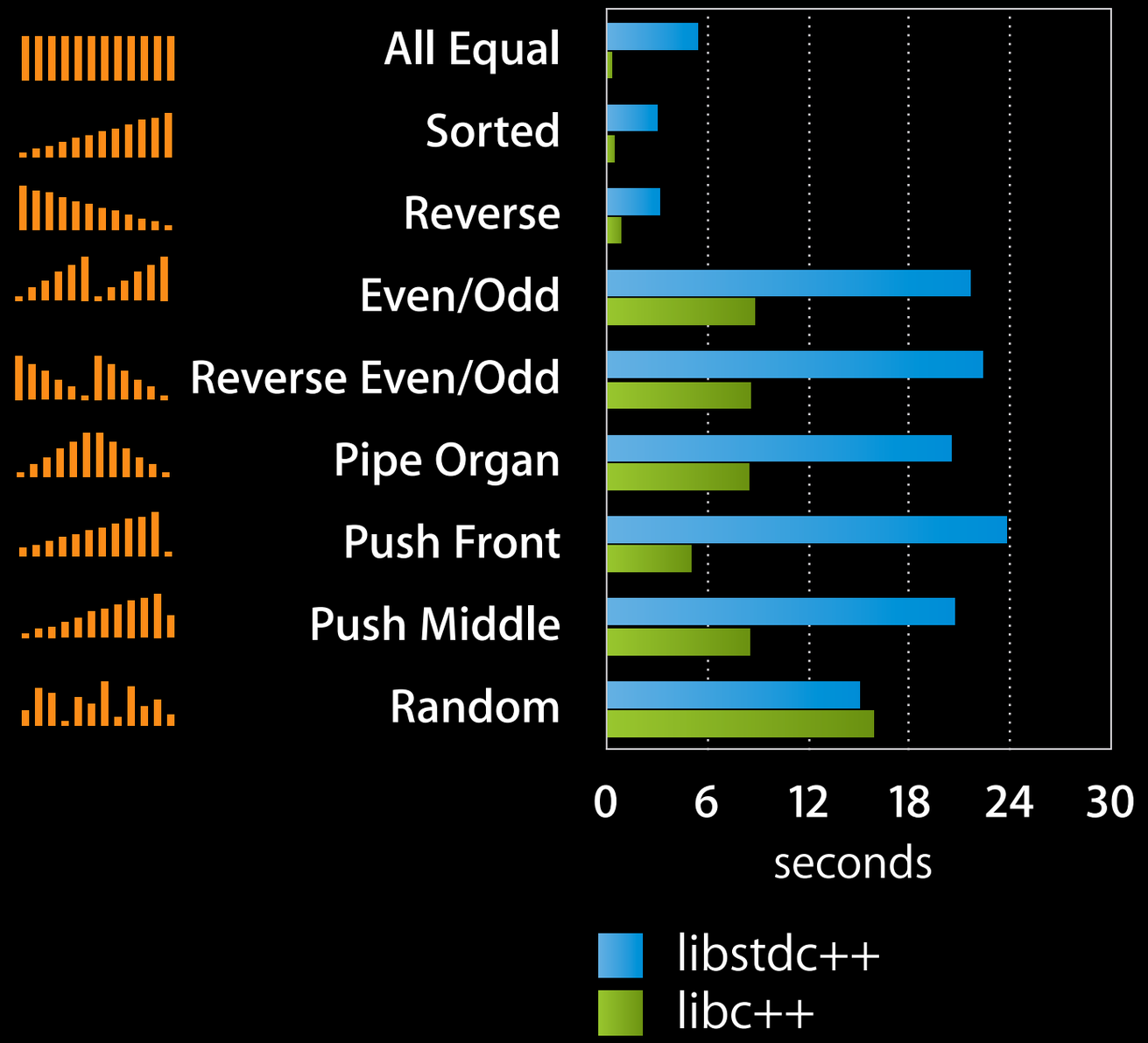
sort

- Fast
  - Never copies, only swaps or moves.
  - Automatically recognizes and adapts to patterns.



sort

- Fast
  - Never copies, only swaps or moves.
  - Automatically recognizes and adapts to patterns.
  - Pattern recognition isn't free, but the cost is quite reasonable.



New Facilities in C++0x…

Smart Pointers

Smart Pointers

- shared_ptr
  - Bullet-proof reference counting.

Smart Pointers

- shared_ptr
  - Bullet-proof reference counting.
  - Weak references.

Smart Pointers

- shared_ptr
  - Bullet-proof reference counting.
  - Weak references.
  - Can be as efficient as intrusive reference counting.

Smart Pointers

- shared_ptr
  - Bullet-proof reference counting.
  - Weak references.
  - Can be as efficient as intrusive reference counting.
- unique_ptr
  - Unique ownership (like auto_ptr).

Smart Pointers

- shared_ptr
  - Bullet-proof reference counting.
  - Weak references.
  - Can be as efficient as intrusive reference counting.
- unique_ptr
  - Unique ownership (like auto_ptr).
  - Zero overhead.

Smart Pointers

- shared_ptr
  - Bullet-proof reference counting.
  - Weak references.
  - Can be as efficient as intrusive reference counting.
- unique_ptr
  - Unique ownership (like auto_ptr).
  - Zero overhead.
  - Can safely be put into containers and used with algorithms.

Smart Pointers

- shared_ptr
  - Bullet-proof reference counting.
  - Weak references.
  - Can be as efficient as intrusive reference counting.
- unique_ptr
  - Unique ownership (like auto_ptr).
  - Zero overhead.
  - Can safely be put into containers and used with algorithms.
  - Custom deallocation support.

Smart Pointers

- shared_ptr
  - Bullet-proof reference counting.
  - Weak references.
  - Can be as efficient as intrusive reference counting.
- unique_ptr
  - Unique ownership (like auto_ptr).
  - Zero overhead.
  - Can safely be put into containers and used with algorithms.
  - Custom deallocation support.
  - Array support.

<chrono>

<chrono>

Type-safe time arithmetic facilities

<chrono>

Type-safe time arithmetic facilities

- Separate types for time durations and points in time.

<chrono>

Type-safe time arithmetic facilities

- Separate types for time durations and points in time.
- Full suite of common units: hours, minutes…nanoseconds.

<chrono>

Type-safe time arithmetic facilities

- Separate types for time durations and points in time.
- Full suite of common units: hours, minutes…nanoseconds.
- Add and subtract durations and time points with natural syntax.

```
system_clock::time_point t0 = system_clock::now();
...
auto t1 = system_clock::now();
nanoseconds ns = t1 - t0;
```

Much easier and safer than working with timeval, timespec, or C's new xtime.

Multithread Support

Multithread Support

- Thread class
  - Can launch any functor with arbitrary number of arguments.

Multithread Support

- Thread class
  - Can launch any functor with arbitrary number of arguments.
- mutexes

Multithread Support

- Thread class
  - Can launch any functor with arbitrary number of arguments.
- mutexes
- Condition variables

Multithread Support

- Thread class
  - Can launch any functor with arbitrary number of arguments.
- mutexes
- Condition variables
- Deadlock-free multi-lock algorithms

Multithread Support

- Thread class
  - Can launch any functor with arbitrary number of arguments.
- mutexes
- Condition variables
- Deadlock-free multi-lock algorithms
- Futures

Multithread Support

- Thread class
  - Can launch any functor with arbitrary number of arguments.
- mutexes
- Condition variables
- Deadlock-free multi-lock algorithms
- Futures
  - Ability to get a return value from a thread.
- Everything works with the new <chrono> facility for timed locking, timed sleeping, etc.

<random>

<random>

- If it has anything to do with random number generation, it is here:

<random>

- If it has anything to do with random number generation, it is here:
  - Six different random number generator generators, each templated on “tweaking” parameters.

<random>

- If it has anything to do with random number generation, it is here:
  - Six different random number generator generators, each templated on “tweaking” parameters.
  - Nine concrete random number generators.
    - e.g. mt19937 and knuth_b

<random>

- If it has anything to do with random number generation, it is here:
  - Six different random number generator generators, each templated on “tweaking” parameters.
  - Nine concrete random number generators.
    - e.g. mt19937 and knuth_b
  - Twenty (yes 20!) random number distributions:
    - uniform_real_distribution
    - bernoulli_distribution
    - gamma_distribution
    - fisher_f_distribution
    - etc.

<regex>

<regex>

- If it has anything to do with regular expressions, it is here:

<regex>

- If it has anything to do with regular expressions, it is here:
  - Both Perl and POSIX style search engines.
    - Multiple flavors of both.

<regex>

- If it has anything to do with regular expressions, it is here:
  - Both Perl and POSIX style search engines.
    - Multiple flavors of both.
  - Many options, including case (in)sensitivity.

<regex>

- If it has anything to do with regular expressions, it is here:
  - Both Perl and POSIX style search engines.
    - Multiple flavors of both.
  - Many options, including case (in)sensitivity.
  - Full locale support for case mapping, character equivalence, and collation.

<regex>

- If it has anything to do with regular expressions, it is here:
  - Both Perl and POSIX style search engines.
    - Multiple flavors of both.
  - Many options, including case (in)sensitivity.
  - Full locale support for case mapping, character equivalence, and collation.
  - Full results data structure including list of matched sub-expressions.

<regex>

- If it has anything to do with regular expressions, it is here:
  - Both Perl and POSIX style search engines.
    - Multiple flavors of both.
  - Many options, including case (in)sensitivity.
  - Full locale support for case mapping, character equivalence, and collation.
  - Full results data structure including list of matched sub-expressions.
  - Exact match, search and search/replace algorithms.

<regex>

- If it has anything to do with regular expressions, it is here:
  - Both Perl and POSIX style search engines.
    - Multiple flavors of both.
  - Many options, including case (in)sensitivity.
  - Full locale support for case mapping, character equivalence, and collation.
  - Full results data structure including list of matched sub-expressions.
  - Exact match, search and search/replace algorithms.
  - Full iterator support for iterating to the next match, and for tokenizing keywords/expressions out of a stream.

Miscellaneous

Miscellaneous

- <type_traits>
  - Compile-time testing and introspection of types.

Miscellaneous

- <type_traits>
  - Compile-time testing and introspection of types.
- <ratio>
  - Compile-time rational arithmetic.

Miscellaneous

- <type_traits>
  - Compile-time testing and introspection of types.
- <ratio>
  - Compile-time rational arithmetic.
- <tuple>
  - pair on steroids.
  - Implements empty member optimization as an extension.
  - Currently requires variadic templates and rvalue reference.

Wrap Up…

How to Use libc++ from clang

How to Use libc++ from clang

- $ clang++ -stdlib=libc++ test.cpp

How to Use libc++ from clang

- $ clang++ -stdlib=libc++ test.cpp
- Currently requires libc++abi on Snow Leopard.

How to Use libc++ from clang

- $ clang++ -stdlib=libc++ test.cpp
- Currently requires libc++abi on Snow Leopard.
  - Unsupported preview here:
    - http://home.roadrunner.com/~hinnant/libcppabi.zip

How to Use libc++ from clang

- $ clang++ -stdlib=libc++ test.cpp
- Currently requires libc++abi on Snow Leopard.
  - Unsupported preview here:
    - http://home.roadrunner.com/~hinnant/libcppabi.zip
  - Future: We would like to run on top of gcc's libsupc++ in addition to libc++abi.

libc++ Status

libc++ Status

- Every major feature of C++0x has been implemented except <atomic>. And that is in progress.

libc++ Status

- Every major feature of C++0x has been implemented except <atomic>. And that is in progress.
- Every feature is backed by unit tests.

libc++ Status

- Every major feature of C++0x has been implemented except <atomic>. And that is in progress.
- Every feature is backed by unit tests.
- Lacking the following features:
  - Make use of some language features such as constexpr and delegating constructors.

libc++ Status

- Every major feature of C++0x has been implemented except <atomic>. And that is in progress.
- Every feature is backed by unit tests.
- Lacking the following features:
  - Make use of some language features such as constexpr and delegating constructors.
  - Ability to run on top of libsupc++.

libc++ Status

- Every major feature of C++0x has been implemented except <atomic>. And that is in progress.
- Every feature is backed by unit tests.
- Lacking the following features:
  - Make use of some language features such as constexpr and delegating constructors.
  - Ability to run on top of libsupc++.
  - Debug mode

libc++ Status

- Every major feature of C++0x has been implemented except <atomic>. And that is in progress.
- Every feature is backed by unit tests.
- Lacking the following features:
  - Make use of some language features such as constexpr and delegating constructors.
  - Ability to run on top of libsupc++.
  - Debug mode
  - Performance tests

libc++ Status

- Every major feature of C++0x has been implemented except <atomic>. And that is in progress.
- Every feature is backed by unit tests.
- Lacking the following features:
  - Make use of some language features such as constexpr and delegating constructors.
  - Ability to run on top of libsupc++.
  - Debug mode
  - Performance tests
  - Porting to more platforms

Summary

Summary

- libc++ is a largely complete C++0x standard library with a very lenient open source license.

Summary

- libc++ is a largely complete C++0x standard library with a very lenient open source license.
- libc++ is high performance.

Summary

- libc++ is a largely complete C++0x standard library with a very lenient open source license.
- libc++ is high performance.
- libc++ is full featured.

Summary

- libc++ is a largely complete C++0x standard library with a very lenient open source license.
- libc++ is high performance.
- libc++ is full featured.
- libc++ is here today:
  - http://libcxx.llvm.org/