The Birth of Study Group 14

Towards Improving C++ for Games & Low Latency

Speakers

Nicolas Guillemot
(Intel)

@nlguillemot

Michael Wong
(OpenMP/IBM)

http://wongmichael.com

Sean Middleditch
(Wargaming.net)

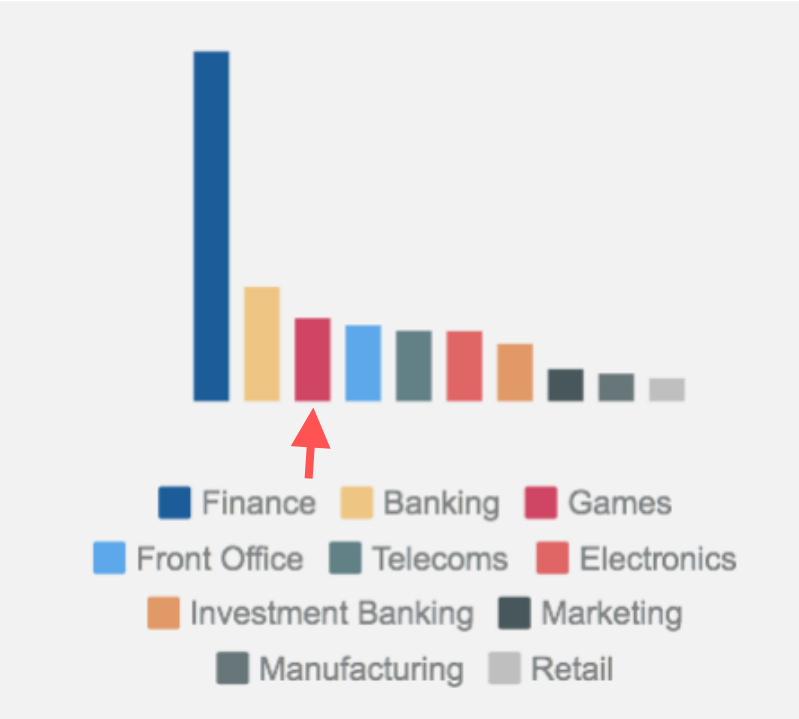
@stmiddleditch
http://seanmiddleditch.com

Overview

1. About SG14
2. Control & Reliability
3. Metrics & Performance
4. Fun & Productivity
5. Current Efforts
6. The Future


WORLD OF WARSHIPS
WARGAMING.NET

Among the top users of C++!



http://blog.jetbrains.com/clion/2015/07/infographics-cpp-facts-before-clion/

About SG14

1. About SG14
2. Control & Reliability
3. Metrics & Performance
4. Fun & Productivity
5. Current Efforts
6. The Future

The Breaking Wave: N4456

CppCon 2014

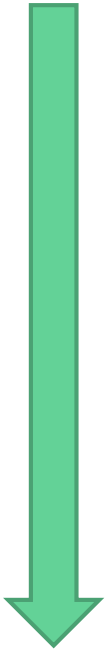
C++ committee panel leads to impromptu game developer meeting.

Google Group created.

Discussions have outstanding industry participation.

N4456 authored and published!

N4456	Towards improved support for games, graphics, real-time, low latency, embedded systems



Formation of SG14

N4456 presented at Spring 2015 Standards Committee Meeting in Lenexa.

Very well received!

SG14
Game Dev &
Low Latency

Formation of Study Group 14:
Game Dev & Low Latency

Chair: Michael Wong (IBM)

cppcon
the c++ conference

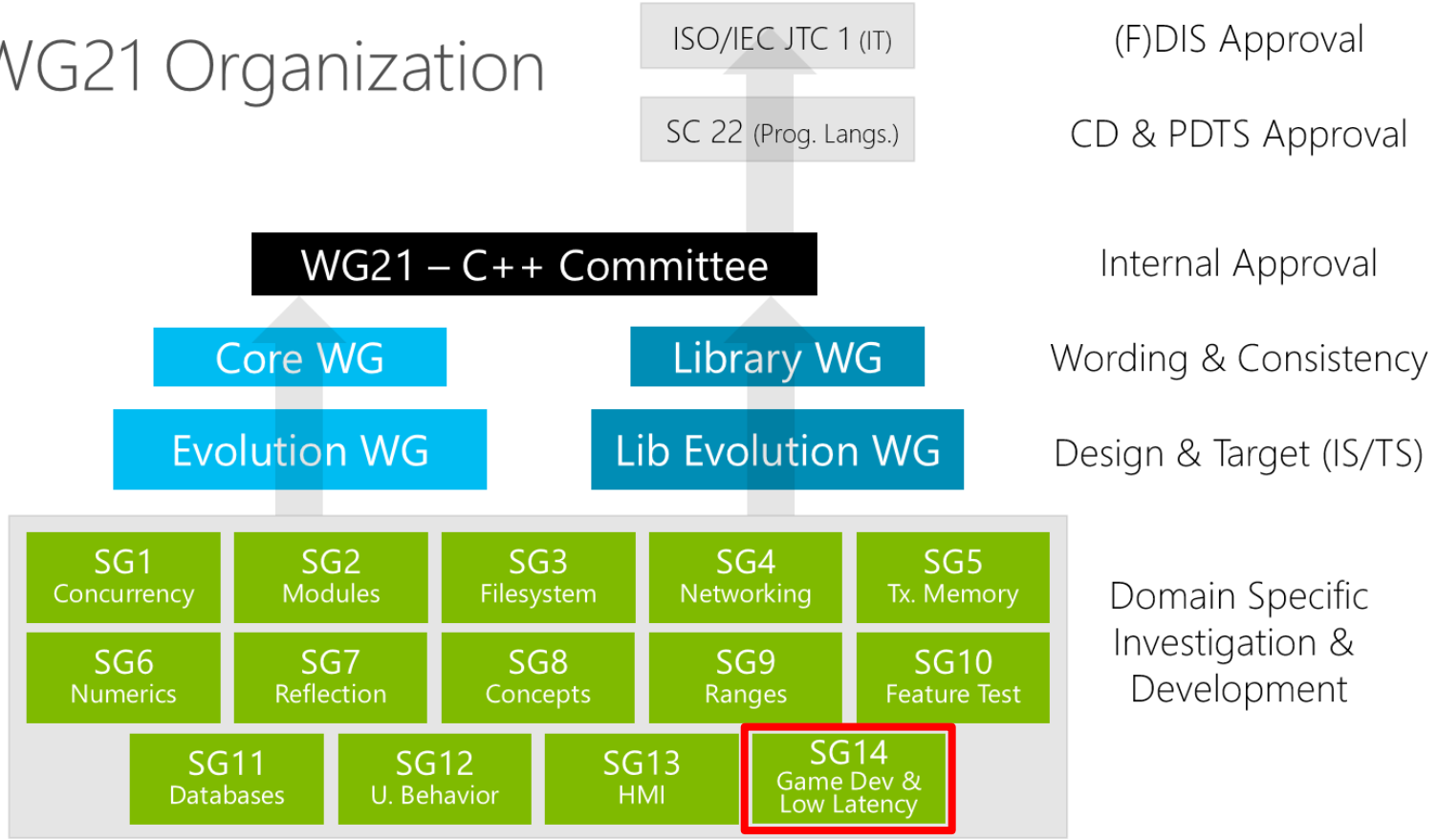
GDC

SONY

Two SG14 meetings planned:

- CppCon 2015 (this Wednesday)
- GDC 2016, hosted by SONY

WG21 Organization



(F)DIS Approval

CD & PDTS Approval

Internal Approval

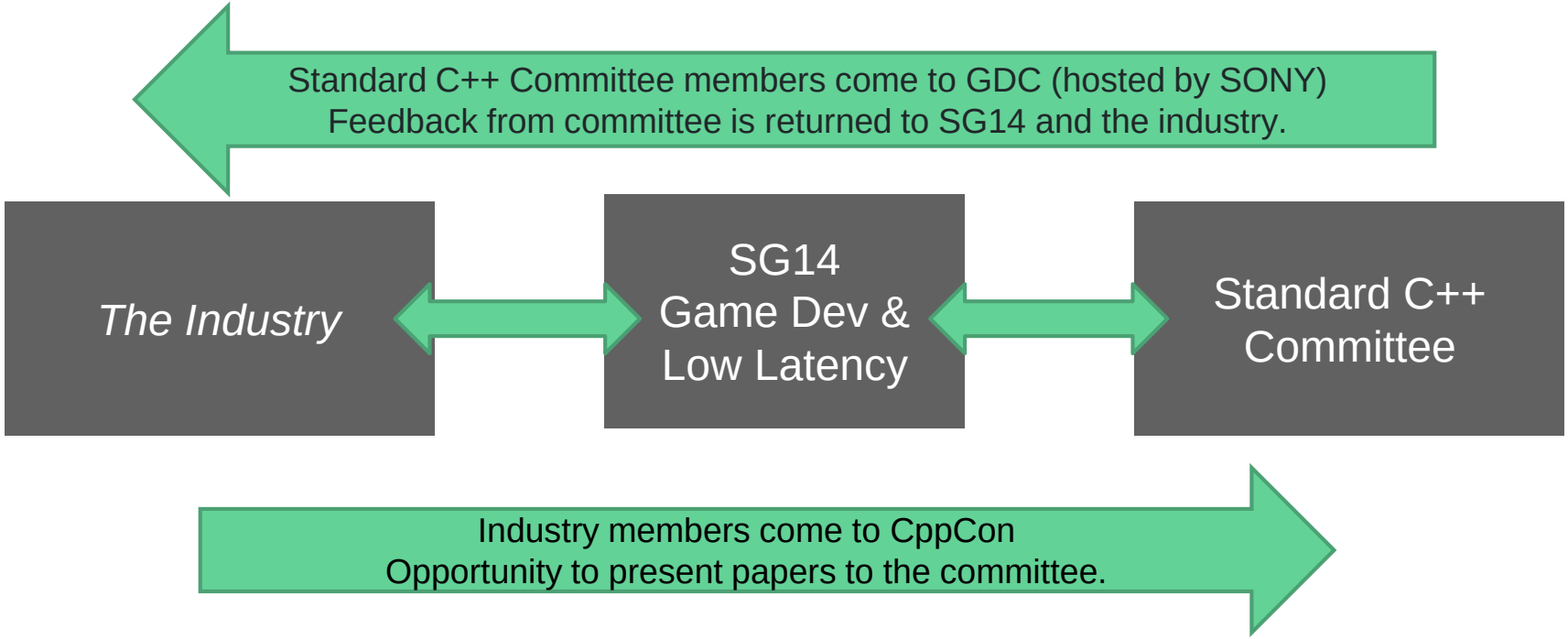
Wording & Consistency

Design & Target (IS/TS)

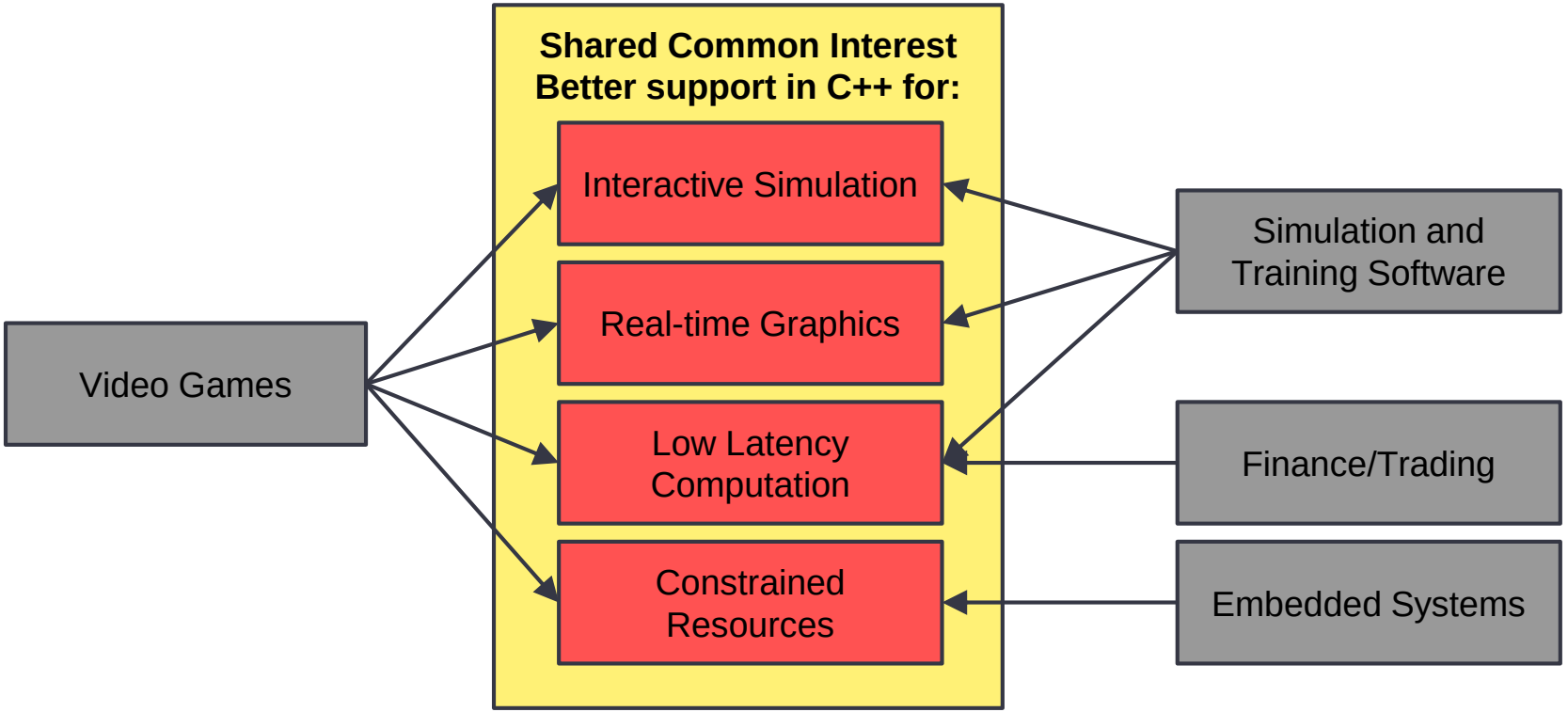
Domain Specific Investigation & Development

https://isocpp.org/std/the-committee

Improving Communication/Feedback/review cycle



Audience of SG14 Goals and Scopes: Not just games!



Where We Are

- Google Groups
  - https://groups.google.com/a/isocpp.org/forum/?fromgroups#!forum/sg14
- GitHub
  - https://github.com/WG21-SG14/SG14
  - Created by Guy Davidson

Control & Reliability

1. About SG14
2. Control & Reliability
3. Metrics & Performance
4. Fun & Productivity
5. Current Efforts
6. The Future

Memory Usage

- Fixed memory budgets
  - 100’s of MB to a couple GB
- Shared CPU/GPU memory
  - 100’s of MB in texture data, animations, framebuffers…
- No swap space and no temporary disk scratch space
- Upgrading hardware isn’t an option
  - Users may not even have the option (eg: game consoles)

Computation Time

- Cost of debug iterators in vendor libraries
  - Many game engines replace even std::vector
  - Each implementation has a different magic incantation to turn off unwanted “features”
- `dynamic_cast` versus home-grown reflection systems
- Not all O(log N) are made the same
  - `boost::flat_map` vs the node-based `std::map`

Inconsistent Allocation Patterns

- Container implementation differences
  - Does an empty container allocate?
  - Vector growth rate and initial capacity?
  - Small string and small object optimizations?
- What size functor will require `std::function` to allocate?
- Behavior is unpredictable when porting between platforms and C++ implementations

Implementation Details

- `std::async` uses a thread pool?
- Standard library features often re-implemented
  - EASTL (Electronic Arts' implementation)
  - STLport (based on SGI's implementation)
  - `folly::FBVector` (Facebook's custom `std::vector`)
  - llvm/ADT (LLVM's custom containers)

Traditionally Costly Features

- RTTI
  - Excessive data generated by eg. dynamic_cast
- Virtual functions
  - Less important these days, but still worth noting
- Poor inlining
  - C++ abstractions not always as free as we are taught to believe
- Exceptions
  - Restrict some optimizations for unwinding

Exceptions & RTTI

- Games often use `-fno-exceptions` and `-fno-rtti`
  - Some important platforms don't support exceptions reliably or at all
- Behavior of `try`/`throw`/`dynamic_cast` not defined when disabled
  - Their being disabled is not even acknowledged by the standard, despite common practice
  - Usually results in a compile error making many libraries unusable without modification
- Not just a games thing or a niche concern
  - http://llvm.org/docs/CodingStandards.html#do-not-use-rtti-or-exceptions
  - https://google-styleguide.googlecode.com/svn/trunk/cppguide.html#Exceptions

Metrics & Performance

1. About SG14
2. Control & Reliability
3. Metrics & Performance
4. Fun & Productivity
5. Current Efforts
6. The Future

Memory budgets

- Content creator and production focus
  - Artists, designers, distribution/publishing/QA
  - Let them answer questions of memory budget on their own (programmers' time is expensive and precious)
- Capture memory stats in the middle of a 3-hour test session without expensive or slow instrumentation
- Need finely-grained accounting and budgeting

Example: Custom Allocators

```
// pseudo-code for allocator declaration similar to many engines
DEFINE_ALLOCATOR(MeshData, "Graphics/Meshes");
DEFINE_ALLOCATOR(Enemies, "GameObjects/Characters/NPCs/Enemies");

// class-based allocators
class VertexMesh {
        USE_ALLOCATOR(MeshData);
};

// runtime allocator assignment, tied
RegisterGameObjectAllocator("objects/enemies/*.json", Enemies());
GameObject* guard = LoadGameObject("objects/enemies/evil_guard.json");
```

Allocation Interfaces

- Standard allocator usage is rare in games
  - Interface is non-ideal
  - Built-in accounting support for distinct memory regions
- Custom allocators with an innate knowledge of alignment
  - Global `new` and `delete` on many platforms not aligned for SIMD
- Simpler interface for custom allocators, of which we have many
  - C++11 was a big improvement on this item, at least, though not perfect
  - Rebinding for node-based allocators is crazy
  - Allocator has no reason to know what it's allocating
    - Even if the allocator has strict size or alignment limits

Performance

- Some hardware has ~~terrible~~ no branch prediction
- Cache locality increasingly critical
- Small inefficiencies permittable in desktop software unacceptable for us
- Performance matters even when debugging
- Memory usage and performance are tightly coupled
- Need algorithms and data structure designed for real hardware
    - Pure math is great and never changes, but hardware certainly does

Some missing algorithms

- Radix sort
  - Integer keys are king
  - Very efficient CPU comparison
  - Trumps `std::sort`
- Spatial and geometric algorithms
- Imprecise but faster alternatives for math algorithms

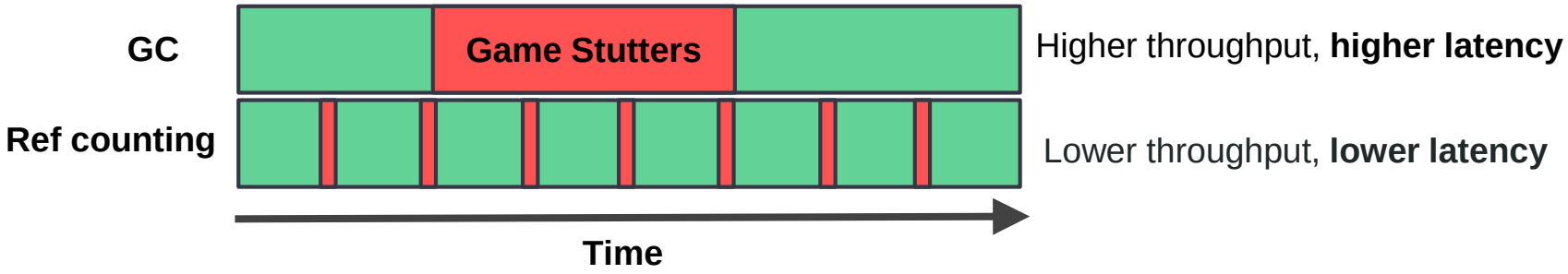
Some missing containers

- Intrusive linked list container
  - Fewer allocations and static initialization
  - No “self iterators”
- Cache-friendly hash table
- Contiguous containers
- Stack containers

Bounded worst case time

- Worst case time vs average case time
  - In general, steady 30fps > jittery 60fps
  - Especially important for VR (jitter = nausea)

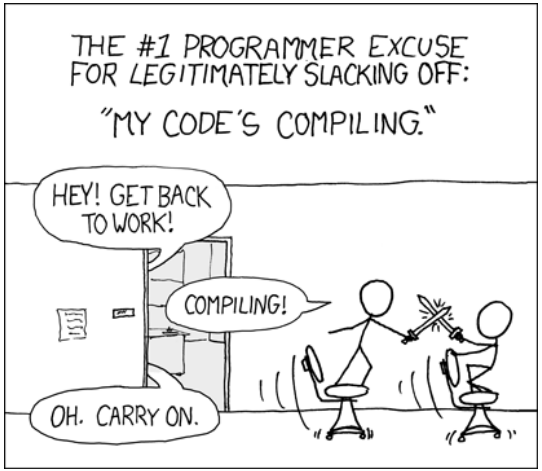
- Note: garbage collection trade-off



Fun & Productivity

Long Compilation Times

- Template/include bloat
  - `std::unique_ptr`/`std::array` vs C pointer/array
  - `<memory>` over 2 KLOC in VC14 (+ dependencies)
- “C with classes”-style code compiles *much* faster
- File I/O, complex grammar, template instantiation, optimizations



https://xkcd.com/303/

Long Loading Times

- Long level loading impacts productivity
  - Loading takes much longer in debug modes
- Want to iterate without loading screens in between
  - C++ “Edit and Continue” feature
    - Integrated in IDE or game engine
  - Live-reloading scripting languages (Lua, Scheme, Python)
  - Live-reloading shading languages (HLSL, GLSL)

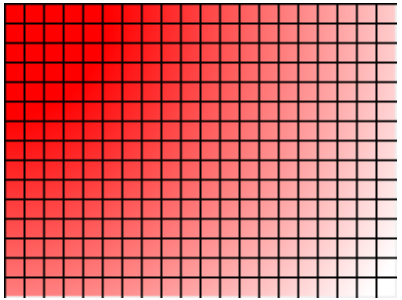
Current Efforts

Fixed-point Numbers

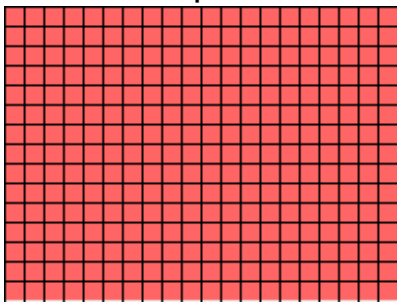
- Effort led by Lawrence Crowl and John McFarlane
  - Overlap with SG6 “Numerics”
  - https://github.com/johnmcfarlane/fixed_point (McFarlane)
  - N3352 “C++ Binary Fixed-Point Arithmetic“ (Crowl)
- Example uses:
  - Platforms slow at floating point (eg: no FPU present)
  - Uniform precision (as opposed to float’s varying precision)
- Proposed:
  - std::fixed_point<Repr,Exponent>
  - std::make_fixed<IntegerBits, FractionBits>
- Papers to be discussed at Kona

Precision of screen coordinates
(not actual precision, just example visualization)

floating point

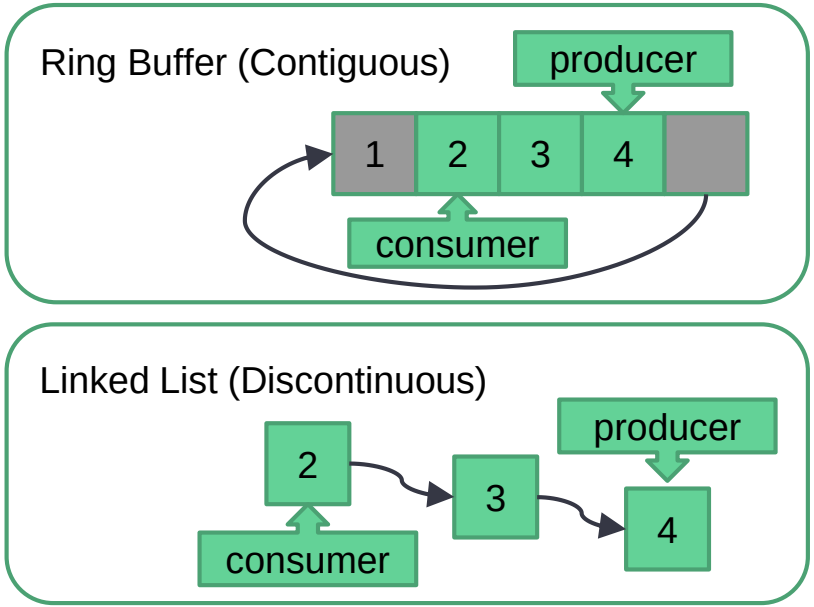


fixed point



Ring Buffer

- Effort led by Guy Davidson
  - https://github.com/WG21-SG14/SG14/blob/master/Docs/Proposals/RingProposal.pdf
- Contiguous FIFO buffer
- Examples uses:
  - Feeding audio samples to a DAC
  - Queuing up network packets to be sent
- Proposed:
  - `std::static_ring<T,N>`
  - `std::dynamic_ring<T,std::vector<T>>`
- Approved for Kona

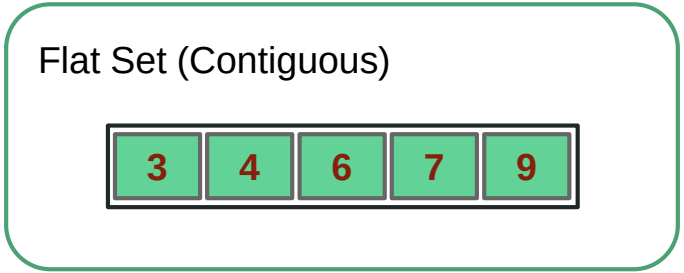


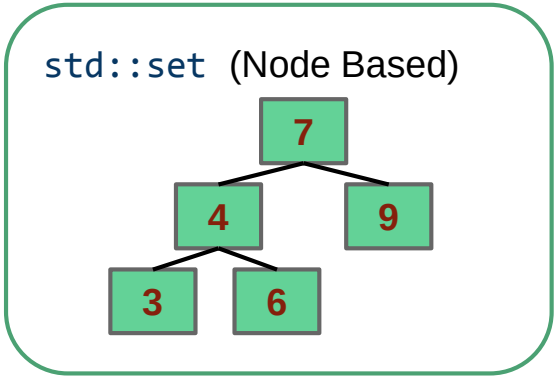
Extended Memory Operations and Algorithms

- Effort led by Brent Friedman
- Better tools for implementing custom high-performance containers
- `std::raw_storage_iterator` improvements (move, emplace)
  - https://github.com/WG21-SG14/SG14/blob/master/Docs/Proposals/rawstorage.html
- `std::destroy, std::uninitialized_move, std::uninitialized_value_construct, std::uninitialized_default_construct`
  - https://github.com/WG21-SG14/SG14/blob/master/Docs/Proposals/uninitialized.html
- `std::unstable_remove`
  - https://github.com/WG21-SG14/SG14/blob/master/Docs/Proposals/unstable_remove.html
- Approved for Kona

“Flat” Associative Containers

- Effort led by Sean Middleditch
  - https://github.com/seanmiddleditch/CPlusPlus/blob/master/flat_containers.md
- Cache-friendly associative containers
  - Binary search in sorted contiguous memory
  - Similar to `std::lower_bound` but with associative container interface
- Approved for Kona





Thread-safe STL

- Early draft paper presented by Brett Searles
- Outlines a desire to investigate concurrency-related STL improvements
- Encouraged to further refine the paper for group discussion

Thread Stack Size

- Patrice Roy proposal
  - http://h-deb.clg.qc.ca/WG21/SG14/thread_ctor_stack_size.pdf
- Requirement in embedded and games to control stack size for new threads
- Stack sizes cannot be modified after a thread is created
  - `std::thread::native_handle()` doesn't help
- Presented case for making the request exact, not just suggested minimum
  - Embedded really needs to control memory budget
- Patrice may present in Kona himself

Exception Cost Analysis

- Patrice Roy presented preliminary and rough findings on the cost of exception handling on modern compilers
  - SG14 mailing list post
- Group discussed the ramifications of the problem and whether it was a real issue affecting the community
- Very initial brainstorming for possible ways of addressing the problem

Related Work

- Coroutines
  - N4499 by Gor Nishanov and Daveed Vandevoorde
- `noexcept` library additions
  - Use `std::error_code` for signaling errors
- Early SIMD in C++ investigation
  - No involvement with SG14 yet
  - There are existing SIMD papers suggesting eg. “Vec<T,N>” and “for simd (;;)”

The Future

1. About SG14
2. Control & Reliability
3. Metrics & Performance
4. Fun & Productivity
5. Current Efforts
6. The Future

Other Current Work

- These items to be discussed before position paper is written
- `plf::colony` and `plf::stack`
  - Matthew Bentley
- Intrusive Containers
  - Hal Finkel
- GPU/Accelerator support
  - Michael Wong, Nicolas Guillemot

Future Directions

- New containers and algorithms
  - New standard algorithms like radix sort
- Stricter requirements on existing containers and algorithms
  - Disallow allocations for default-empty construction of standard containers
  - Non-throwing moves for standard types
- Miscellaneous issues
  - Separate bloated headers like `<algorithm>`
  - Investigating the overhead and workarounds for exceptions and RTTI

What next?

- This is where you come in!
- Investigate pain points, study design space
  - https://groups.google.com/a/isocpp.org/forum/#!forum/sg14
  - SG14 meeting at GDC
- Write and present proposals
- Make awesome games using new features!

Thank you!

Questions? Comments?

Contact

Nicolas Guillemot

nlguillemot@gmail.com

@nlguillemot

Sean Middleditch

sean@seanmiddleditch.us

@stmiddleditch

Michael Wong

michaelw@ca.ibm.com