C++ Standard Library ‘Little Things’

Billy O’Neal, Visual C++ Libraries

bion@Microsoft.com

@MalwareMinigun

microsoft / STL

Unwatch 84 | Unstar 1,651 | Fork 87

Code | Issues 78 | Pull requests 2 | Actions | Projects 0 | Wiki | Security | Insights | Settings

MSVC's implementation of the C++ Standard Library.

Edit

Manage topics

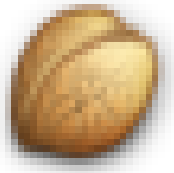
18 commits | 1 branch | 0 releases | 3 contributors | View license

Branch: master | New pull request | Create new file | Upload files | Find File | Clone or download

BillyONeal and CaseyCarter Create a CODEOWNERS (#102) ...

Latest commit e94b4e7 22 minutes ago

docs	Create a CODEOWNERS (#102)	22 minutes ago
stl	Reflect directory structure of includes in ${PROJECT_BINARY_DIR}/out/...	1 hour ago
.clang-format	Initial commit.	13 days ago
.gitattributes	Tell Github's linguist tool that our files are C++, including extensi...	12 days ago
.gitignore	Apply useful CMake changes from the subninja attempt that work on the...	12 days ago
CMakeLists.txt	Remove need for user to specify triplet, and enable ARM and ARM64 bui...	11 days ago
CMakeSettings.json	Remove need for user to specify triplet, and enable ARM and ARM64 bui...	11 days ago



This talk in a nutshell

Go to the compiler feature support page on cppreference:

https://en.cppreference.com/w/cpp/compiler_support

and look at *all* the things voted in, not just headline ones about which people usually write talks.

Walnut.png from: https://en.wikipedia.org/wiki/Template:Nutshell

In this talk

- Feature test macros (C++20)
- starts_with + ends_with (C++20)
- std::clamp (C++17)
- std::exchange (C++14)
- Variadic lock_guard (C++17)
- Parallel Algorithms (C++17)
- Associative contains (C++20)
- Splicing Maps and Sets (C++17)
- Unordered transparency (C++20)

Feature Test Macros (C++20)

- Test if the compiler or standard library implements a given feature for the current build mode
- Core features: http://eel.is/c++draft/cpp.predefined
- Library features: http://eel.is/c++draft/support.limits.general
- Examples of proper use from http://eel.is/c++draft/cpp.cond

Feature Test Macros (C++20)

15 [*Example:* This demonstrates a way to include a library `optional` facility only if it is available:

```
#if __has_include(<optional>)
#  include <optional>
#  if __cpp_lib_optional >= 201603
#    define have_optional 1
#  endif
#elif __has_include(<experimental/optional>)
#  include <experimental/optional>
#  if __cpp_lib_experimental_optional >= 201411
#    define have_optional 1
#    define experimental_optional 1
#  endif
#endif
#ifndef have_optional
#  define have_optional 0
#endif
```

— *end example*]

starts_with and ends_with (C++20)

- Exactly what it says:

```
using namespace std::string_view_literals;

constexpr auto s = "hello world"sv;
static_assert(s.starts_with("hello"sv));
static_assert(!s.ends_with("hello"sv));
static_assert(!s.starts_with(" world"sv));
static_assert(s.ends_with(" world"sv));
```

std::clamp (C++17)

```
#include <algorithm>

constexpr int a = std::clamp(1234, 10, 20);
constexpr int b = std::clamp(11, 10, 20);
constexpr int c = std::clamp(0, 10, 20);
static_assert(a == 20);
static_assert(b == 11);
static_assert(c == 10);
```

std::exchange (C++14)

- Equivalent to moving out a T, and move assigning over it

```
template <class T, class Other = T>
T exchange(T& val, Other&& new_val) {
    T old_val = static_cast<T&&>(val);
    val       = static_cast<Other&&>(new_val);
    return old_val;
}
```

std::exchange (C++14)

```
template <class T, class Other = T>
T exchange(T& val, Other&& new_val) {
    T old_val = static_cast<T&&>(val);
    val       = static_cast<Other&&>(new_val);
    return old_val;
}
```

== std::move

== std::forward

== std::move!

std::exchange (C++14)

```
std::string Detach() {
    std::string old = std::move(member);
    member = {};
    return old;
}
```

```
std::string Detach() {
    return std::exchange(member, {});
}
```

Variadic lock_guard: scoped_lock (C++17)

- Takes multiple locks using a deadlock avoidance algorithm
- Oblivious to mutex type, scheduler, etc.

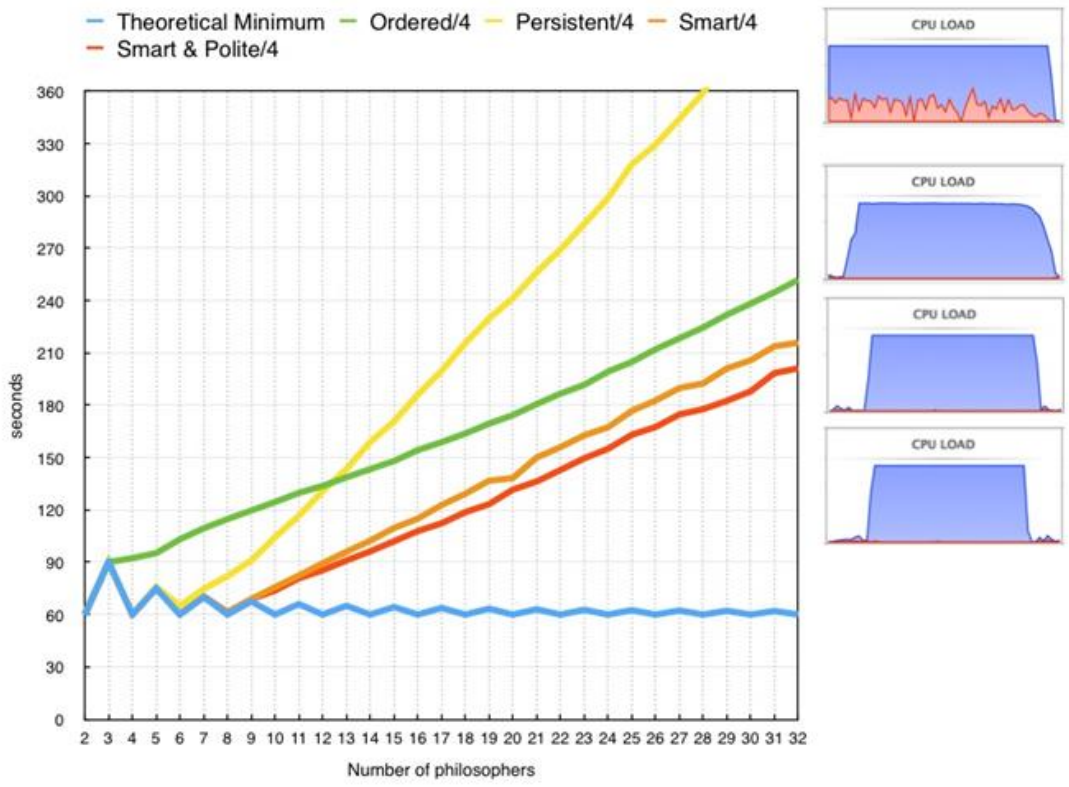
```
template<class... MutexTypes>
class scoped_lock;
```

scoped_lock (C++17)

```
struct DataValue {
    mutable std::shared_mutex m;
    int theValue;
};
void modify_both(DataValue& a, DataValue& b) {
    std::scoped_lock lock(a.m, b.m); // takes both
        // mutexes without deadlock
    std::swap(a.theValue, b.theValue);
    // both mutexes are unlocked upon exiting the scope
}
```

scoped_lock (C++17)

- More efficient than defining a lock ordering, see http://howardhinnant.github.io/dining_philosophers.html#Explanation



Parallel Algorithms (C++17)

std::sort(a, b, pred) => std::sort(std::execution::par, a, b, pred)

See blog post for more nitty gritty:

https://devblogs.microsoft.com/cppblog/using-c17-parallel-algorithms-for-better-performance/

See my talk last year for how this works inside MSVC:

https://youtu.be/nOpwhTbulmk

Parallel Algorithms (C++17)

```
state.PauseTiming();
vector<T> data(r0);
fill_with_random(data);
state.ResumeTiming();
sort(execution::par,
    data.begin(), data.end(), less<>{});
```

Parallel Algorithms (C++17)

x64 Native Tools Command Prompt for VS 2019 Int Preview

```
C:\Dev\vclib-benchmarks\build.x64.debug>.\parallel_sort.exe
06/13/19 19:23:33
Running .\parallel_sort.exe
Run on (64 X 3000 MHz CPU s)
CPU Caches:
  L1 Data 32K (x32)
  L1 Instruction 65K (x32)
  L2 Unified 524K (x32)
  L3 Unified 8388K (x8)
***WARNING*** Library was built as DEBUG. Timings may be affected.
-----------------------------------------------------------------------------------------------
Benchmark                                                    Time             CPU   Iterations
-----------------------------------------------------------------------------------------------
serial_sort<unsigned int>/256/real_time                    81454 ns         75006 ns         8541
serial_sort<unsigned int>/512/real_time                   182987 ns        178405 ns         3766
serial_sort<unsigned int>/1024/real_time                  404678 ns        390852 ns         1719
serial_sort<unsigned int>/2048/real_time                  891912 ns        921474 ns          780
serial_sort<unsigned int>/4096/real_time                 1958092 ns       1838235 ns          357
serial_sort<unsigned int>/32768/real_time               19727433 ns      20833333 ns           36
serial_sort<unsigned int>/262144/real_time             188767225 ns     195312500 ns            4
serial_sort<unsigned int>/1000000/real_time            808893000 ns     812500000 ns            1
parallel_sort<unsigned int>/256/real_time                 153574 ns        104655 ns         4479
parallel_sort<unsigned int>/512/real_time                 246906 ns        141502 ns         2773
parallel_sort<unsigned int>/1024/real_time                397775 ns        201006 ns         1753
parallel_sort<unsigned int>/2048/real_time                [illegible] ns   281172 ns         1[illegible]2
parallel_sort<unsigned int>/4096/real_time               1022056 ns       542[illegible] ns         6[illegible]
parallel_sort<unsigned int>/32768/real_time              5174165 ns       84[illegible]75 ns         1[illegible]
parallel_sort<unsigned int>/262144/real_time             3919158 ns      3536[illegible]84 ns         [illegible]9
parallel_sort<unsigned int>/1000000/real_time          125173900 ns     125000000 ns            4
ppl_sort<unsigned int>/256/real_time                       82301 ns         85491 ns         7859
ppl_sort<unsigned int>/512/real_time                      184560 ns        177182 ns         3792
ppl_sort<unsigned int>/1024/real_time                     411068 ns        421307 ns         1706
ppl_sort<unsigned int>/2048/real_time                     904653 ns        844038 ns          759
ppl_sort<unsigned int>/4096/real_time                     397374 ns        3404[illegible]30 ns          179
ppl_sort<unsigned int>/32768/real_time                   13135525 ns       85937[illegible]00 ns            4
ppl_sort<unsigned int>/262144/real_time                  31210510 ns       533[illegible]250[illegible]00 ns            1
ppl_sort<unsigned int>/1000000/real_time                 33218[illegible]364[illegible]0 ns       578[illegible]25[illegible]000 ns            1
serial_sort<double>/256/real_time                          [illegible]267 ns         87[illegible]66 ns         8[illegible]25
serial_sort<double>/512/real_time                          [illegible]784 ns        153[illegible]46 ns         3[illegible]
serial_sort<double>/1024/real_time                        512835 ns        468750 ns         1000
serial_sort<double>/2048/real_time                        954666 ns        935759 ns          718
serial_sort<double>/4096/real_time                       2107433 ns       2187500 ns          350
serial_sort<double>/32768/real_time                     23807687 ns      23437500 ns           30
serial_sort<double>/262144/real_time                   196067825 ns     191406250 ns            4
serial_sort<double>/1000000/real_time                  850158800 ns     843750000 ns            1
parallel_sort<double>/256/real_time                       179387 ns        137567 ns         3521
parallel_sort<double>/512/real_time                       271619 ns        159156 ns         2258
parallel_sort<double>/1024/real_time                      456326 ns        272874 ns         1317
parallel_sort<double>/2048/real_time                      722018 ns        969608 ns          983
parallel_sort<double>/4096/real_time                     1280739 ns        666182 ns          516
parallel_sort<double>/32768/real_time                    5293592 ns       4770992 ns          131
parallel_sort<double>/262144/real_time                  38504216 ns      39473684 ns           19
parallel_sort<double>/1000000/real_time                149724850 ns     148437500 ns            4
ppl_sort<double>/256/real_time                             85683 ns         80980 ns         7718
ppl_sort<double>/512/real_time                            187213 ns        215081 ns         3705
ppl_sort<double>/1024/real_time                           415844 ns        384991 ns         1664
ppl_sort<double>/2048/real_time                           916288 ns        927606 ns          758
ppl_sort<double>/4096/real_time                          4002487 ns       3161127 ns          173
ppl_sort<double>/32768/real_time                       169725450 ns      93750000 ns            4
ppl_sort<double>/262144/real_time                     3297779700 ns    2343750000 ns            1
ppl_sort<double>/1000000/real_time                   33700343400 ns    6125000000 ns            1

C:\Dev\vclib-benchmarks\build.x64.debug>
```

x64 Native Tools Command Prompt for VS 2019 Int Preview

```
C:\Dev\vclib-benchmarks\build.x64.release>.\parallel_sort.exe
06/13/19 19:17:39
Running .\parallel_sort.exe
Run on (64 X 3000 MHz CPU s)
CPU Caches:
  L1 Data 32K (x32)
  L1 Instruction 65K (x32)
  L2 Unified 524K (x32)
  L3 Unified 8388K (x8)
-----------------------------------------------------------------------------------------------
Benchmark                                                    Time             CPU   Iterations
-----------------------------------------------------------------------------------------------
serial_sort<unsigned int>/256/real_time                     6507 ns          5919 ns        108238
serial_sort<unsigned int>/512/real_time                    14750 ns         14426 ns         47656
serial_sort<unsigned int>/1024/real_time                   32990 ns         30082 ns         21296
serial_sort<unsigned int>/2048/real_time                   73314 ns         78403 ns          9566
serial_sort<unsigned int>/4096/real_time                  161539 ns        168822 ns          4350
serial_sort<unsigned int>/32768/real_time                1648073 ns       1768868 ns           424
serial_sort<unsigned int>/262144/real_time              16091407 ns      16335227 ns            44
serial_sort<unsigned int>/1000000/real_time             68745680 ns      70312500 ns            10
parallel_sort<unsigned int>/256/real_time                  25388 ns         22928 ns         27941
parallel_sort<unsigned int>/512/real_time                  32662 ns         31465 ns         21353
parallel_sort<unsigned int>/1024/real_time                 42612 ns         33232 ns         15986
parallel_sort<unsigned int>/2048/real_time                 70736 ns         59854 ns          9920
parallel_sort<unsigned int>/4096/real_time                102167 ns         90227 ns          6927
parallel_sort<unsigned int>/32768/real_time               607018 ns        449070 ns          1183
parallel_sort<unsigned int>/262144/real_time             3443112 ns       3001847 ns           203
parallel_sort<unsigned int>/1000000/real_time           12243628 ns      15625000 ns            57
ppl_sort<unsigned int>/256/real_time                        6550 ns          6594 ns        104268
ppl_sort<unsigned int>/512/real_time                       14779 ns         14814 ns         47465
ppl_sort<unsigned int>/1024/real_time                      33122 ns         32367 ns         21241
ppl_sort<unsigned int>/2048/real_time                      73492 ns         67137 ns          9542
ppl_sort<unsigned int>/4096/real_time                      118[illegible]34 ns         95[illegible]81 ns          5916
ppl_sort<unsigned int>/32768/real_time                     524[illegible]65 ns        346[illegible]23 ns          1264
ppl_sort<unsigned int>/262144/real_time                   2944[illegible]61 ns       2[illegible]00[illegible]0 ns           238
ppl_sort<unsigned int>/1000000/real_time                  3097[illegible]86 ns       7[illegible]12[illegible]00 ns            56
serial_sort<double>/256/real_time                          8363 ns          8[illegible]15 ns         83309
serial_sort<double>/512/real_time                          18[illegible]47 ns         20[illegible] ns         37131
serial_sort<double>/1024/real_time                         42240 ns         37831 ns         16521
serial_sort<double>/2048/real_time                         93805 ns         87664 ns          7486
serial_sort<double>/4096/real_time                        205923 ns        220523 ns          3401
serial_sort<double>/32768/real_time                      2089583 ns       1871257 ns           334
serial_sort<double>/262144/real_time                    20333171 ns      19761029 ns            34
serial_sort<double>/1000000/real_time                   86469263 ns      87890625 ns             8
parallel_sort<double>/256/real_time                        25158 ns         20546 ns         28138
parallel_sort<double>/512/real_time                        33339 ns         21979 ns         21327
parallel_sort<double>/1024/real_time                       46488 ns         32910 ns         15193
parallel_sort<double>/2048/real_time                       74957 ns         77597 ns          9464
parallel_sort<double>/4096/real_time                      115878 ns         87062 ns          6102
parallel_sort<double>/32768/real_time                     578243 ns        450474 ns          1214
parallel_sort<double>/262144/real_time                   4722628 ns       4823826 ns           149
parallel_sort<double>/1000000/real_time                 16313214 ns      15252976 ns            42
ppl_sort<double>/256/real_time                              8358 ns          8818 ns         69107
ppl_sort<double>/512/real_time                             18751 ns         21338 ns         37346
ppl_sort<double>/1024/real_time                            41789 ns         39261 ns         16715
ppl_sort<double>/2048/real_time                            92540 ns        100687 ns          7604
ppl_sort<double>/4096/real_time                           123278 ns        107969 ns          5644
ppl_sort<double>/32768/real_time                          541416 ns        440831 ns          1276
ppl_sort<double>/262144/real_time                        4727719 ns       3715035 ns           143
ppl_sort<double>/1000000/real_time                      17488350 ns      12500000 ns            40

C:\Dev\vclib-benchmarks\build.x64.release>
```

Example output, see parallel_sort.cpp on GitHub

Parallel Algorithms (C++17)

Debug, 32 cores

element count	256	512	1024	2048	4096	262144	1000000
serial, unsigned int (µs)	81	183	405	892	1958	188767	808893
parallel, unsigned int (µs)	154	247	398	645	1022	35919	125174
Relative Time	1.90	1.35	0.98	0.72	0.52	0.19	0.15
Win, Times	0.53	0.74	1.02	1.38	1.92	5.26	6.46
ppl, unsigned int (µs)	82	184	411	905	3874	3121075	33218766
Relative Time	1.01	1.01	1.01	1.01	1.98	16.53	41.07
Win, Times	0.99	0.99	0.99	0.99	0.51	0.06	0.02

Parallel Algorithms (C++17)

Release, 32 cores

element count	256	512	1024	2048	4096	262144	1000000
serial, unsigned int (µs)	7	15	33	73	161	16091	68746
parallel, unsigned int (µs)	25	33	46	71	102	3443	12244
Relative Time	3.57	2.20	1.39	0.97	0.63	0.21	0.18
Win, Times	0.28	0.45	0.72	1.03	1.58	4.67	5.61
ppl, unsigned int (µs)	7	15	33	73	119	2945	13093
	1.00	1.00	1.00	1.00	0.74	0.18	0.19
	1.00	1.00	1.00	1.00	1.35	5.46	5.25

Parallel Algorithms (C++17)

Release (4 Core Laptop)

element count	256	512	1024	2048	4096	262144	1000000
serial, unsigned int (µs)	8	17	39	87	190	19405	80760
parallel, unsigned int (µs)	14	23	36	55	92	4798	19591
Relative Time	1.75	1.35	0.92	0.63	0.48	0.25	0.24
Win, Times	0.57	0.74	1.08	1.58	2.07	4.04	4.12
ppl, unsigned int (µs)	8	17	43	85	116	4814	18836
	1.00	1.00	1.10	0.98	0.61	0.25	0.23
	1.00	1.00	0.91	1.02	1.64	4.03	4.29

Contains (C++20)

- New member of maps and sets that does just what it says – bool to ask if an element is in the container
- Alternative to find() != end() or count() != 0

Contains (C++20)

```
printf("find != end 24: %s\n",
    s.find(24) != s.end() ? "true" : "false");
printf("find != end 67: %s\n",
    s.find(67) != s.end() ? "true" : "false");

printf("count 24: %s\n", s.count(24) != 0 ? "true" : "false");
printf("count 67: %s\n", s.count(67) != 0 ? "true" : "false");

printf("contains 24: %s\n", s.contains(24) ? "true" : "false");
printf("contains 67: %s\n", s.contains(67) ? "true" : "false");
```

Splicing Maps and Sets (C++17)

- Extract elements from maps or sets, and mutate them or reinsert them later
- Combine maps and sets into each other with the merge member
- No allocations

Splicing Maps and Sets (C++17)

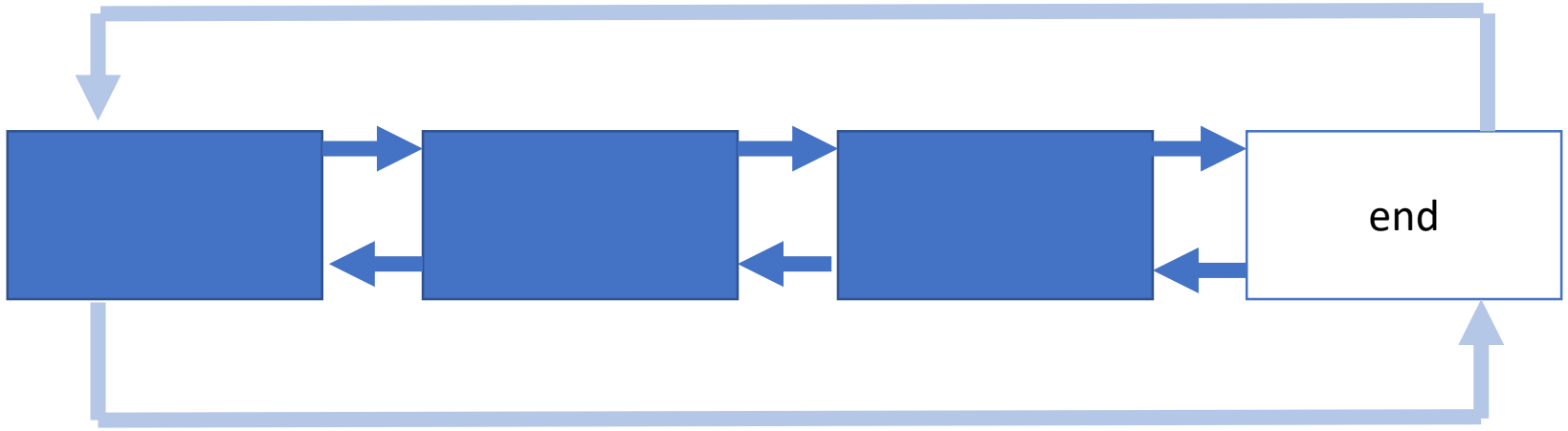
Node-based containers

- list (C++98/03)
- forward_list (C++11)
- (multi)set
- (multi)map
- unordered_(multi)set
- unordered_(multi)map

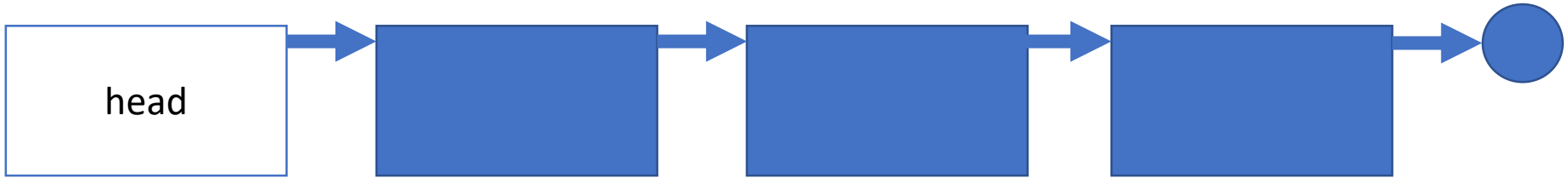
Not node-based

- array
- deque*
- vector
- vector<bool>
- string
- bitset
- valarray

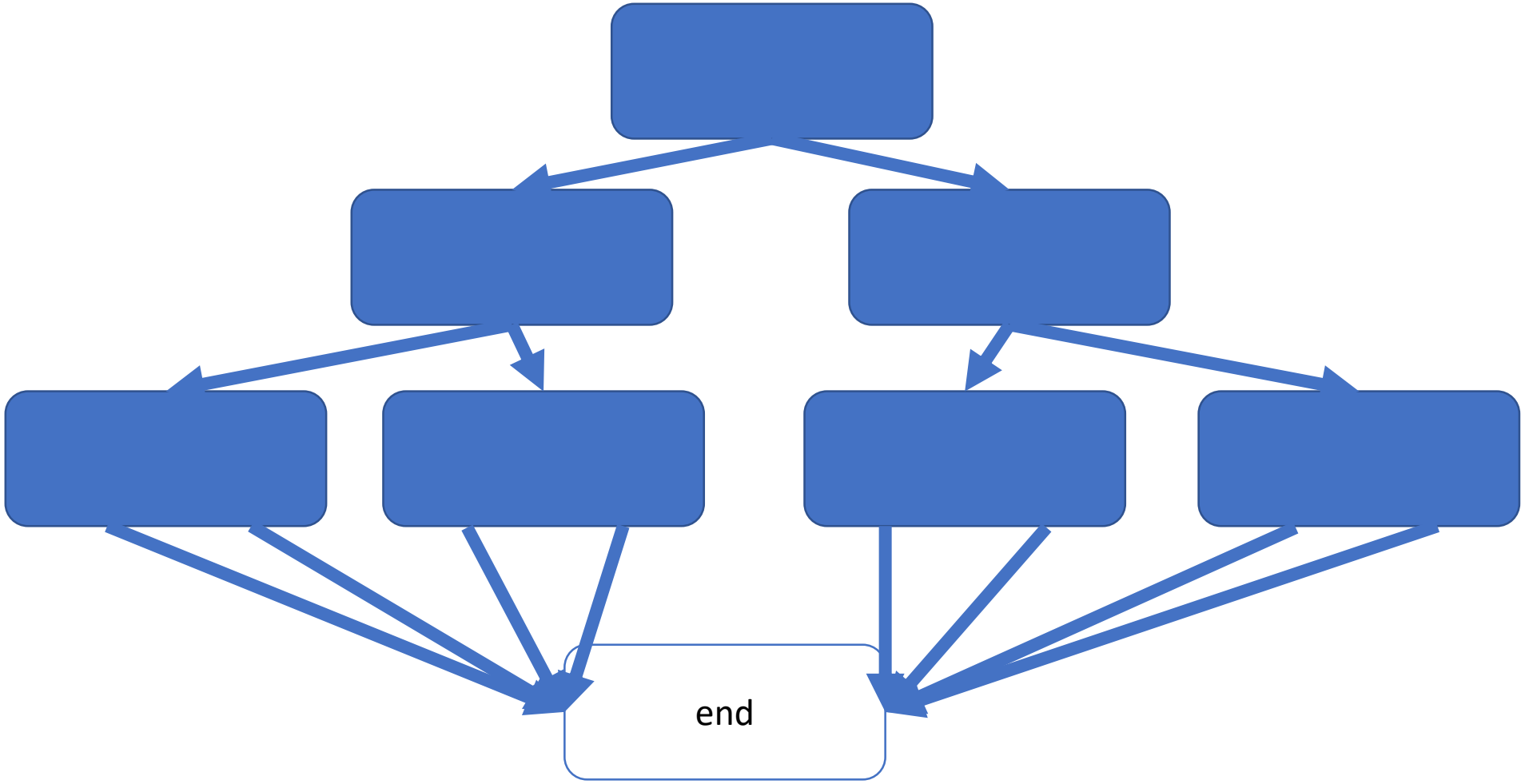
Splicing Maps and Sets (C++17)



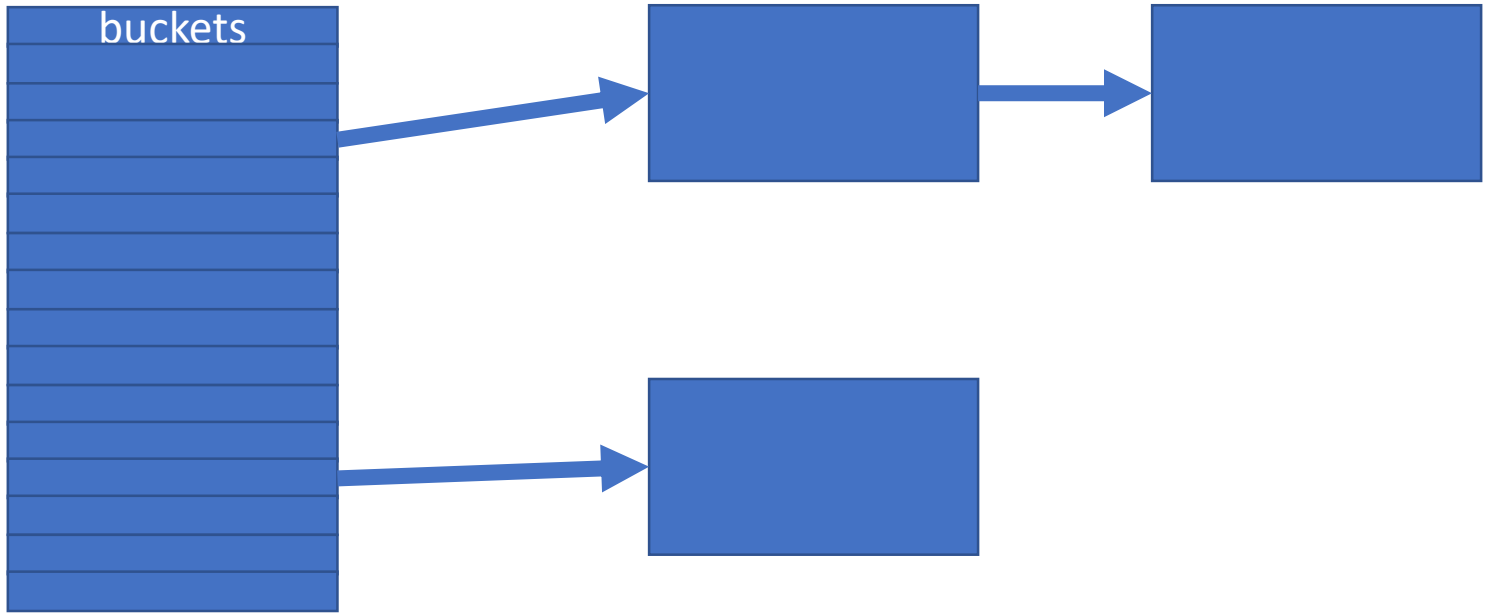
Splicing Maps and Sets (C++17)



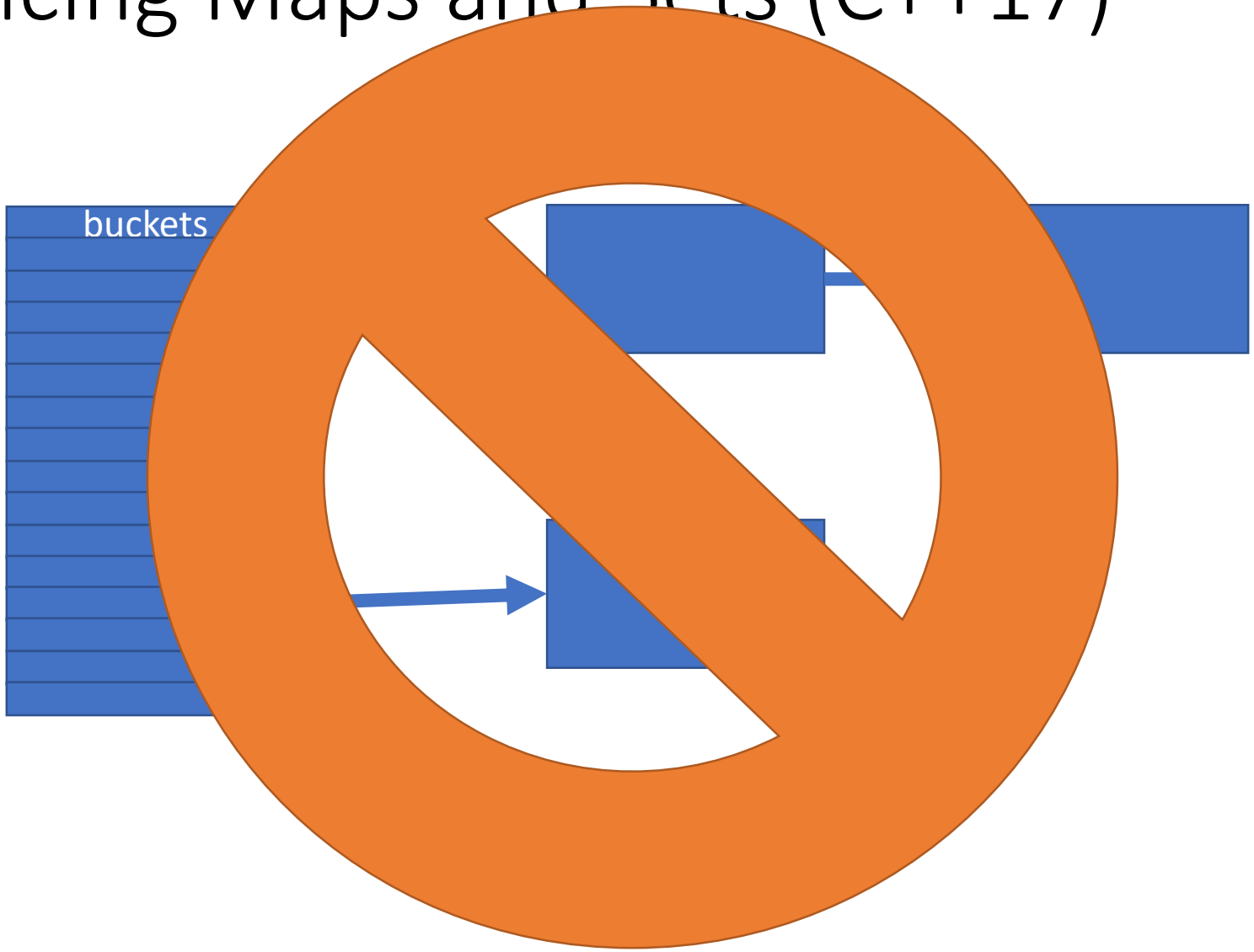
Splicing Maps and Sets (C++17)



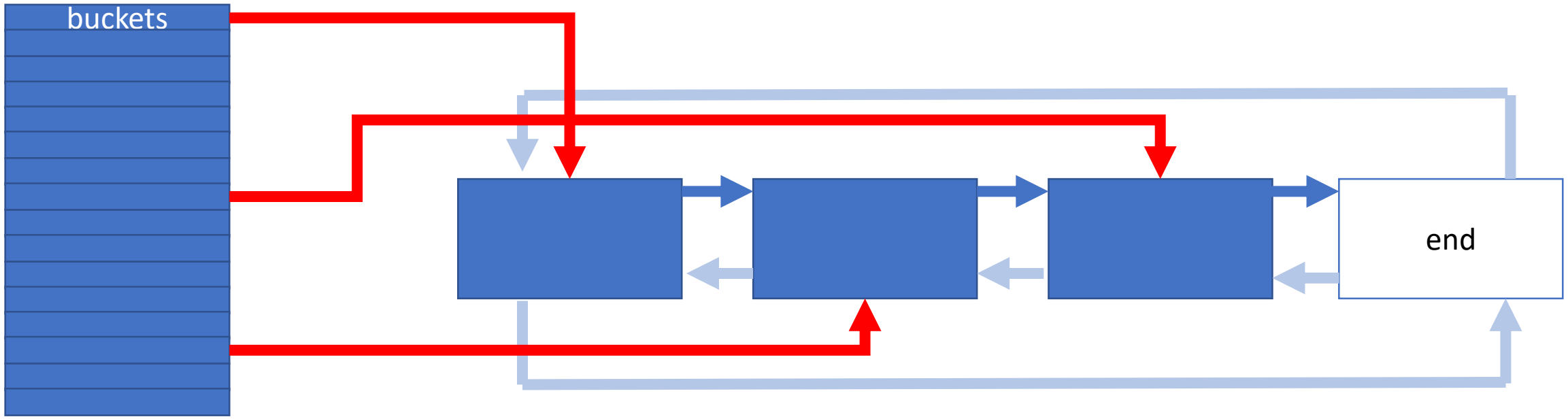
Splicing Maps and Sets (C++17)



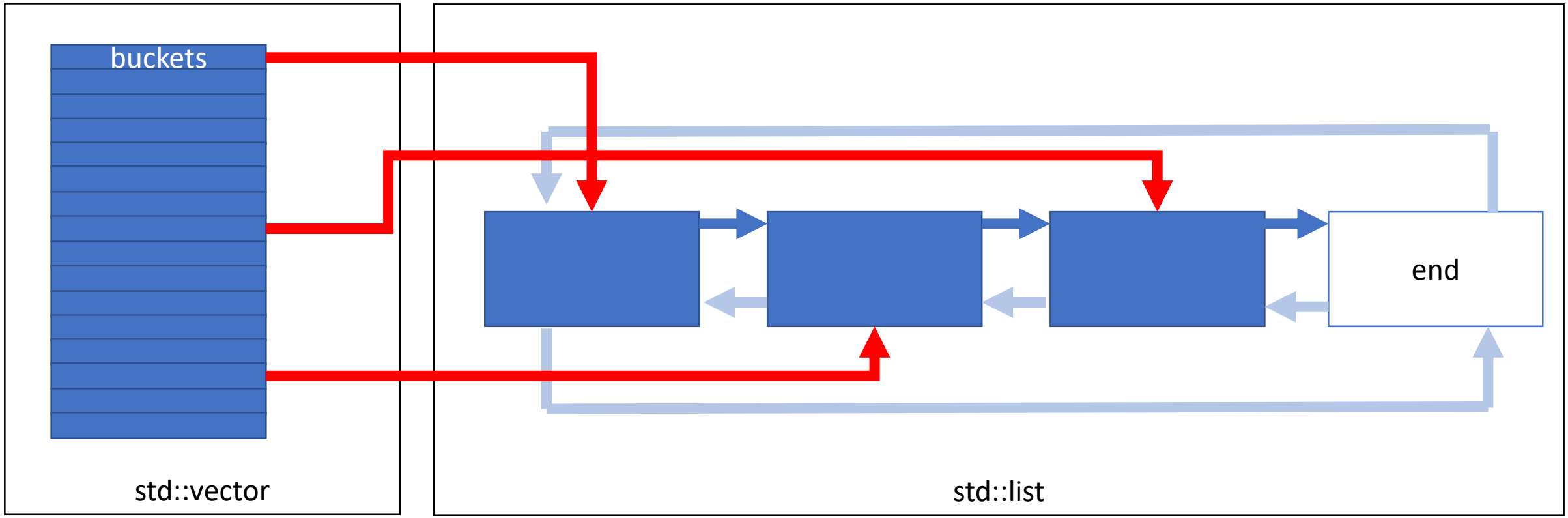
Splicing Maps and Sets (C++17)



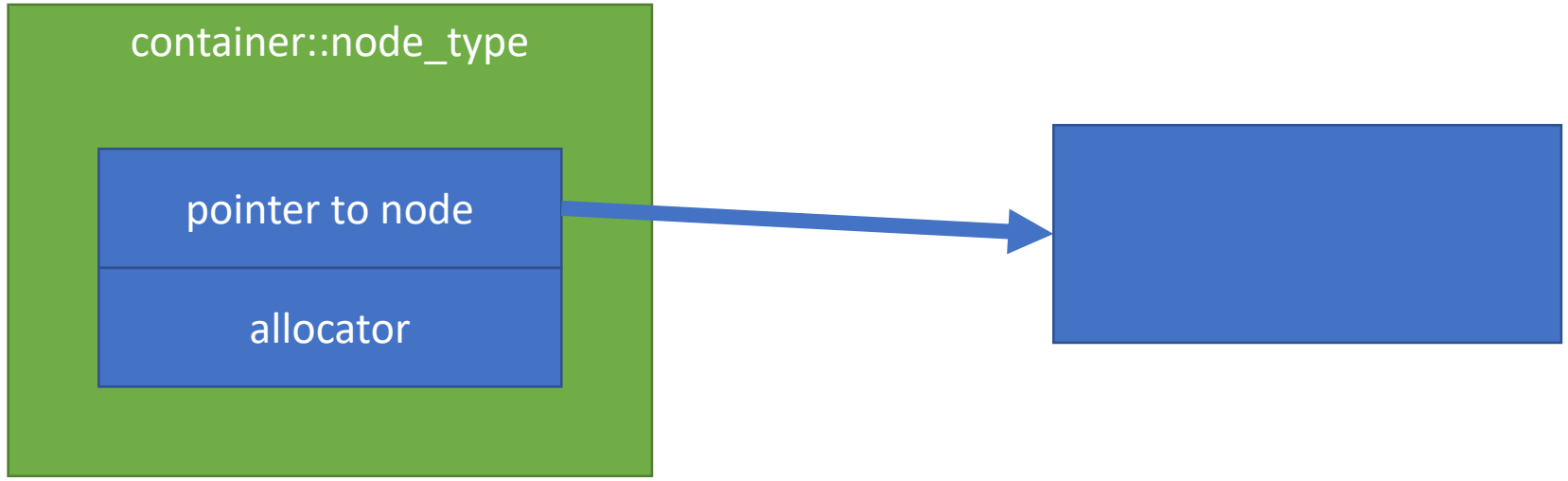
Splicing Maps and Sets (C++17)



Splicing Maps and Sets (C++17)



Splicing Maps and Sets (C++17)



Splicing Maps and Sets (C++17)

```
using UnwindMapData =
    UtcPool::map<T_ALLOC, EHSTATE, UWentry>;

for (auto &UWEntry : funcInfo.UnwindMap)
{
    // Change entry to be keyed off new state
    const_cast<T&>(UWEntry.first) = newState;
}
```

Splicing Maps and Sets (C++17)

```
using UnwindMapData =
    UtcPool::map<T_ALLOC, EHSTATE, UWentry>;

for (auto &UWEntry : funcInfo.UnwindMap)
{
    // Change entry to be keyed off new state
    auto movedEntry =
        funcInfo.UnwindMap.extract(UWEntry.first);
    movedEntry.key() = newState;
    funcInfo.UnwindMap.insert(move(movedEntry));
}
```

Splicing Maps and Sets (C++17)

```
std::set set1 = {1, 2, 4, 1000, 1234, -67, 1729};
std::set set2 = {-1, 2, 4, -1000, 1234, 7, -1729};
print_ints("set1", set1);
print_ints("set2", set2);
set1.merge(set2);
// set1 == {-1729, -1000, -67, -1, 1, 2, 4, 7, 1000,
//           1234, 1729}
// set2 == {2, 3, 1234}
print_ints("set1, after merge", set1);
print_ints("set2, after merge", set2);
```

Unordered Transparency (C++20)

```
void example(std::set<std::string>& s) {
    s.find("hello world"); // implicitly allocates memory :(
    s.find("hello world"sv); // doesn't compile X(
    s.find(std::string("hello world"sv)); // allocates memory :(
}

void example(std::set<std::string, std::less<>>& s) {
    s.find("hello world"); // doesn't allocate memory but strlen :/
    s.find("hello world"sv); // doesn't allocate memory :D
}
```

Unordered Transparency (C++20)

- Like C++14 transparent ordered associative containers extended to unordered containers

  Then: std::map<std::string, int, std::less<>>

  Now: std::unordered_map<std::string, int, ASpecialHash>

Unordered Transparency (C++20)

```
struct hasher {
// equal_to has is_transparent
using transparent_key_equal = std::equal_to<>;
    std::size_t operator()(std::string_view sv) {
        return std::hash<std::string_view>{}(sv);
    }
};

void example(std::unordered_set<std::string, hasher>& s) {
    s.find("hello world"); // doesn't allocate memory but strlen :/
    s.find("hello world"sv); // doesn't allocate memory :D
}
```

Questions?

Thanks all!

Talk materials at
https://github.com/BillyONeal/14_cpp_features_in_40_minutes