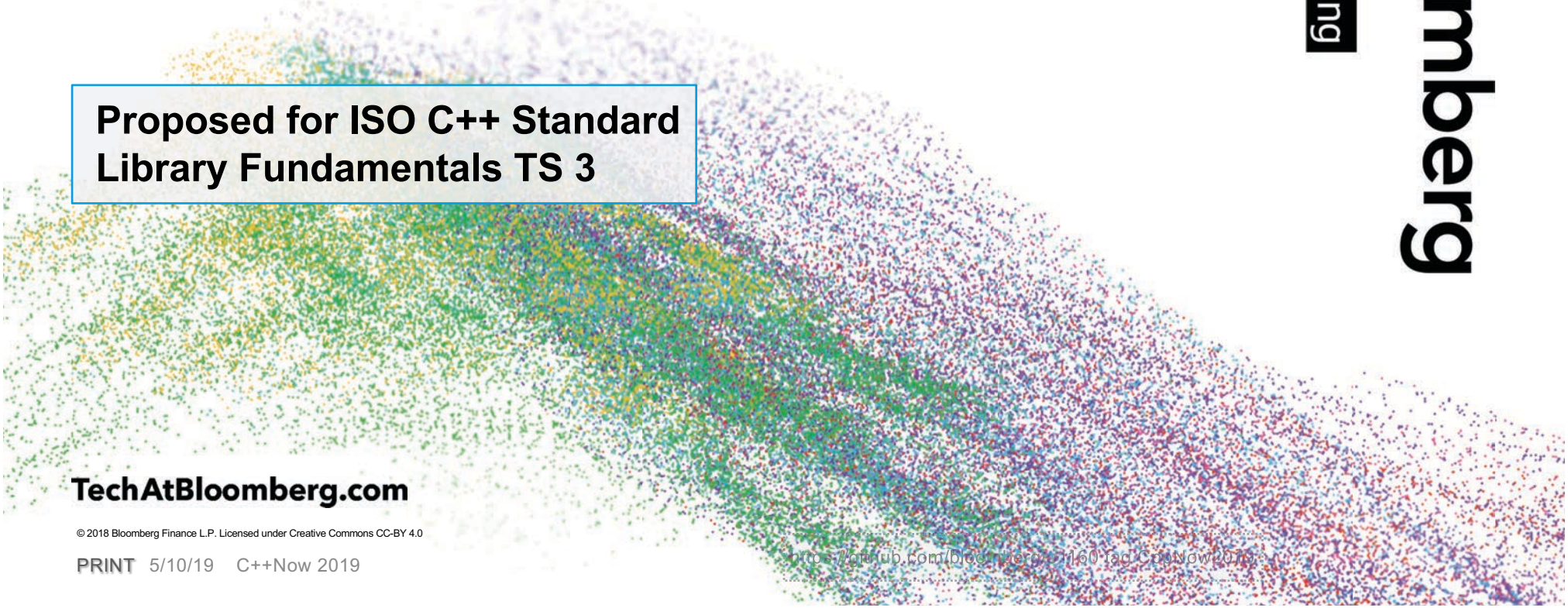
`test_resource`
the `pmr` detective

Bloomberg

Engineering

Proposed for ISO C++ Standard Library Fundamentals TS 3



TechAtBloomberg.com

PRINT 5/10/19 C++Now 2019

https://github.com/bloomberg/p2160-tad-CppNow2019

About me

- Attila (Farkas) Fehér
  [Attila (Wolf) White] – in English
- Hungarian
- Ő, him, them
- Bloomberg LP
- @wwolf @cppguru @attila_f_feher
- https://www.facebook.com/cppguru
- https://www.linkedin.com/in/cppguru/
- *Was* Attila aka WW on USENET
- C++11 alignment features

Pixabay License

Petra Blahoutová

TechAtBloomberg.com

https://github.com/bloomberg/p1160 tag CppNow2019

Bloomberg Engineering

Meta-talk about the talk

- First time speaking at a conference …
- … please be gentle and forgive the stumbles
- Presenting someone else's design that has 20 years of history
- I will be kind of telling a story (with code examples and conclusions)
- Questions that need answering for **understanding**: *ask any time*
- Questions going into tangents, comments, discussion: *ask at the end*

https://github.com/bloomberg/p1160 tag CppNow2019

Topics

I. Introduction

II. The Purpose of Test Resource

III. Polymorphic Memory Resource Recap

IV. Examples of Testing and Debugging

V. Applicability of Test Resource

VI. Enhancement Ideas

VII. A Look at The Code (https://github.com/bloomberg/p1160)

Topics

I. Introduction

II. The Purpose of Test Resource

III. Polymorphic Memory Resource Recap

IV. Examples of Testing and Debugging

V. Applicability of Test Resource

VI. Enhancement Ideas

VII. A Look at The Code (https://github.com/bloomberg/p1160)

Introduction

- I am about to present a polymorphic memory resource that plugs into the existing **`pmr`** framework and provides verification and telemetry for allocation related behavior
- As is with all of **`pmr`**, it works with **`std::allocator<>`** aware code
- We (Bloomberg and I) also have an ISO proposal for inclusion into **Library Fundamentals TS 3**, and written the code to accompany it
- The code is freely available (as in beer & speech), under the Apache 2.0 License, and you can use it **today**
- This talk is not trying to sell **`pmr`**. It aims to introduce **`test_resource`**.

https://github.com/bloomberg/p1160 tag CppNow2019

Introduction

- I am about to present
- This talk is

https://github.com/bloomberg/p1160
tag CppNow2019

https://github.com/bloomberg/p1160 tag CppNow2019

Topics

I. Introduction

II. The Purpose of Test Resource

III. Polymorphic Memory Resource Recap

IV. Examples of Testing and Debugging

V. Applicability of Test Resource

VI. Enhancement Ideas

VII. A Look at The Code (https://github.com/bloomberg/p1160)

What can `test_resource` verify?

- Do we leak?
- Do we use released memory?
- Do we write more bytes than we allocated? (overrun)
- Maybe write before the bytes we allocated? (underrun)
- Deallocate with different values? (ptr, size, alignment)
- Do we cleanly survive a memory allocation failure?

https://github.com/bloomberg/p1160 tag CppNow2019

What is `test_resource` designed for?

- Verifies and instruments memory allocation related behavior
- Made especially for **unit testing**, so it makes tests:
  - repeatable
  - automatable (no need for human interaction)
  - non-intrusive (don't even need the sources tested)
  - targeted (on individual objects)
- It is **not a test framework**, it works with any framework

https://github.com/bloomberg/p1160 tag CppNow2019

What can `test_resource` instrument?

- How many blocks/bytes *are* allocated *right now*?
- How many blocks/bytes *were* allocated in *total*?
- The *maximum* blocks/bytes *ever* allocated?
- *Last* allocation/deallocation values (ptr/size/align)?
- *Monitoring* statistic since a point in time/code
  - verify there were no new allocations
  - *or* verify that all allocation were also released etc.

https://github.com/bloomberg/p1160 tag CppNow2019

What can `test_resource` instrument?

- How many blocks/bytes *are* allocate
- How many blocks/bytes *were* alloca
- The *maximum* blocks/bytes *ever* all
- *Last* allocation/deallocation values (
- *Monitoring* statistic since a point in t
  - verify there were no new allocations
  - *or* verify that all allocation were also released

```
blocks_in_use
bytes_in_use

total_blocks
total_bytes

max_blocks
max_bytes

last_allocated_address
last_allocated_num_bytes
last_allocated_alignment

last_deallocated_address
last_deallocated_num_bytes
last_deallocated_alignment
```


https://github.com/bloomberg/p1160 tag CppNow2019

Topics

I. Introduction

II. The Purpose of Test Resource

III. Polymorphic Memory Resource Recap

IV. Examples of Testing and Debugging

V. Applicability of Test Resource

VI. Enhancement Ideas

VII. A Look at The Code (https://github.com/bloomberg/p1160)

The pmr::memory_resource

```
class memory_resource {
 public:
    virtual ~memory_resource() noexcept;
```

A (pure) virtual base class.
It puts the p into pmr.

https://github.com/bloomberg/p1160 tag CppNow2019

The pmr::memory_resource

```
class memory_resource {
 public:
    virtual ~memory_resource() noexcept;

    void *allocate(const size_t bytes, const size_t align =
                                                    alignof(max_align_t));

    void deallocate(void *const ptr, const size_t bytes, const size_t align =
                                                    alignof(max_align_t));

    bool is_equal(const memory_resource& _That) const noexcept;
```

Look ma', no virtuals!

https://github.com/bloomberg/p1160 tag CppNow2019

The pmr::memory_resource

```
class memory_resource {
 public:
    virtual ~memory_resource() noexcept;
    void *allocate(const
                                                    nof(max_align_t));
    void deallocate(voi
                                                    st size_t align =
                                                    nof(max_align_t));
    bool is_equal(const memory_resource& _That) const noexcept;
```

Tells whether `_That` and `this` can deallocate each other's allocations

https://github.com/bloomberg/p1160 tag CppNow2019

The pmr::memory_resource

```
class memory_resource {
 public:
    virtual ~memory_resource() noexcept;

    void *allocate(const size_t bytes, const size_t align =
                                                    alignof(max_align_t));

    void deallocate(void *const ptr, const size_t bytes, const size_t align =
                                                    alignof(max_align_t));

    bool
  private:
    virtual void *do_allocate(size_t bytes, size_t align) = 0;
    virtual void do_deallocate(void * ptr, size_t bytes, size_t align) = 0;
    virtual bool do_is_equal(const memory_resource& that) const noexcept = 0;
};
```

They are smart, they make it go. (Well, do).

https://github.com/bloomberg/p1160 tag CppNow2019

The pmr::memory_resource

```
class memory_resource {
 public:
    virtual ~memory_resource() noexcept;

    void *allocate(const size_t bytes, const size_t align =
                                                  alignof(max_align_t));

    void deallocate(void *const ptr, const s

    bool is_equal(const memory_resource& _That) const noexcept;

  private:
    virtual void *do_allocate(size_t bytes, size_t align) = 0;
    virtual void do_deallocate(void * ptr, size_t bytes, size_t align) = 0;
    virtual bool do_is_equal(const memory_resource& that) const noexcept = 0;
};
```

Users care about these.

Implementers of memory resources care about these.

https://github.com/bloomberg/p1160 tag CppNow2019

The traditional containers and allocators

```
// in the <deque> standard header

template<class T,
         class Allocator = allocator<T>>
  class deque;
```


https://github.com/bloomberg/p1160 tag CppNow2019

The pmr::polymorphic_allocator

```
template <class Target>
struct polymorphic_allocator {
    using value_type = Target;
    polymorphic_allocator() noexcept;
    polymorphic_allocator(memory_resource *mem_res) noexcept;
  ~~~
    Target *allocate(size_t count);
    void deallocate(Target *ptr, size_t count) noexcept;
    memory_resource *resource() const noexcept;
  private:
    memory_resource *__resource = ::std::pmr::get_default_resource();
};
```

A glorious global that (almost) every type uses if no specific memory resource is supplied during construction.

https://github.com/bloomberg/p1160 tag CppNow2019

The pmr::polymorphic_allocator – P0339R5

```
template <class Target>
struct polymorphic_allocator {
    using value_type = Target;
    polymorphic_allocator() noexcept;
    polymorphic_allocator(memory_resource *mem_

  ~~~

    Target *allocate(size_t count);
    void deallocate(Target *ptr, size_t c
    memory_resource *resource() const n
  private:
    memory_resource *__resource = ::std::pmr::g
};                                          urce();
```

P0339R5 introduces **polymorphic_allocator<>** that has **std::byte** as **Target**,

It also adds additional member functions and function templates to avoid having to use **rebind** .

You are going to see these used in the code examples later.

Approved for C++20

Pablo Halpern
Dietmar Kühl

https://github.com/bloomberg/p1160 tag CppNow2019

The pmr containers

```
// in the <deque> standard header

template<class T,
         class Allocator = allocator<T>>
  class deque;

namespace pmr {
  template <class T>
  using deque = deque<T, pmr::polymorphic_allocator<T>>;
}
```


https://github.com/bloomberg/p1160 tag CppNow2019

Topics

I. Introduction

II. The Purpose of Test Resource

III. Polymorphic Memory Resource Recap

IV. Examples of Testing and Debugging

V. Applicability of Test Resource

VI. Enhancement Ideas

VII. A Look at The Code (https://github.com/bloomberg/p1160)

https://github.com/bloomberg/p1160 tag CppNow2019

The parts in the `test_resource` "system"

1. The test memory resource class type
2. The test resource monitor class type
3. The default resource guard class type
4. The test resource exception class type
5. An algorithm to test out of memory behavior

https://github.com/bloomberg/p1160 tag CppNow2019

The parts in the `test_resource` "system"

1. **The test memory resource class type**
2. The test resource monitor class type
3. The default resource guard class type
4. The test resource exception class type
5. An algorithm to test out of memory behavior

https://github.com/bloomberg/p1160 tag CppNow2019

Example: A broken string class (no destructor defined)

This `polymorphic_allocator<>` is from P0339R5

```
class pstring {
  public:
    using allocator_type = std::pmr::polymorphic_allocator<>;

    pstring(const char *cstr, allocator_type allocator = {});
~~~
  private:
    allocator_type  m_allocator_;
    size_t          m_length_;
    char           *m_buffer_;
};
```

No destructor is defined!

https://github.com/bloomberg/p1160 tag CppNow2019

Example: Constructor implementation (broken string class)

```
pstring::pstring(const char *cstr, allocator_type allocator)
: m_allocator_(allocator)
, m_length_(std::strlen(cstr))
, m_buffer_(
    static_cast<char *>(allocator.allocate_bytes(m_length_, 1)))
{
    std::strcpy(m_buffer_, cstr);
}
```

The `allocate_bytes` function is from P0339R5

Just copied N+1 bytes into an N bytes buffer. That's usually bad, mkay?

Note that we allocate too small space. No space for the closing NUL character.

https://github.com/bloomberg/p1160 tag CppNow2019

Example: Leaking destructor (test code)

```
{
    std::pmr::test_resource tr{ "stringtr", verbose };
    tr.set_no_abort(true);

    pstring astr{ "baroof", &tr };

    ASSERT_EQ(astr.str(), "baroof");
}   astr.~pstring();      // leaks
    tr.~test_resource(); // reports
```

The **test_resource** detect leaks at the time of its destruction.

https://github.com/bloomberg/p1160 tag CppNow2019

Example: Leaking destructor (non-verbose output)

```
MEMORY_LEAK from stringtr:
  Number of blocks in use = 1
    Number of bytes in use = 6
```

Number of *blocks* means number of *allocations*

https://github.com/bloomberg/p1160 tag CppNow2019

Example: Leaking destructor (test code verbose output)

```
test_resource stringtr [0]: Allocated 6 bytes (aligned 1) at 00543F48.
==========================================================
                TEST RESOURCE stringtr STATE
----------------------------------------------------------
       Category        Blocks  Bytes
       --------        ------  -----
         IN USE          1       6
            MAX          1       6
          TOTAL          1       6
     MISMATCHES          0
  BOUNDS ERRORS          0
  PARAM. ERRORS          0
----------------------------------------------------------
Indices of Outstanding Memory Allocations:
0
```

```
MEMORY_LEAK from stringtr:
  Number of blocks in use = 1
   Number of bytes in use = 6
```


https://github.com/bloomberg/p1160 tag CppNow2019

Example: Adding destructor (fixing the leak)

```
pstring::~pstring()
{
    m_allocator_.deallocate_bytes(m_buffer_, m_length_, 1);
}
```

```
void deallocate_bytes(void *ptr, size_t bytes, size_t alignment=alignof(max_align_t));
```

The `deallocate_bytes` function is from **P0339R5**

https://github.com/bloomberg/p1160 tag CppNow2019

Example: Buffer overrun (test code)

```
{
    std::pmr::test_resource tr{ "stringtr", verbose };
    tr.set_no_abort(true);

    pstring astr{ "baroof", &tr };

    ASSERT_EQ(astr.str(), "baroof");
}--> astr.~pstring(); // calls deallocate now
            └ ◦ ◦ ◦ ─►tr.do_deallocate(…); // detects & reports
```


https://github.com/bloomberg/p1160 tag CppNow2019

Buffer overrun report – triggered by the deallocation

```
*** Memory corrupted at 1 bytes after 6 byte segment at 00332CC8. ***

Header:
00332CA0:       ef be ad de   06 00 00 00   01 00 00 00   cd cd cd cd
00332CB0:       00 00 00 00   00 00 00 00   48 b6 32 00   d8 fc 23 00
00332CC0:       b1 b1 b1 b1   b1 b1 b1 b1

User segment:
00332CC8:       62 61 72 6f   6f 66
                 b  a  r  o    o  f

Pad area after user segment:
00332CCE:       00 b1 b1 b1   b1 b1 b1 b1
```

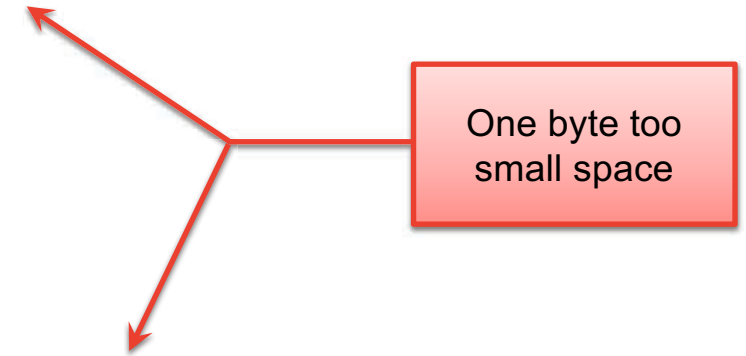
This should also be **b1**, but it is overwritten by the closing NUL character

https://github.com/bloomberg/p1160 tag CppNow2019

Example: A broken string class (*before* fixing the overrun)

```
pstring::pstring(const char *cstr, allocator_type allocator)
: m_allocator_(allocator)
, m_length_(std::strlen(cstr))
, m_buffer_(m_allocator_.allocate_bytes(m_length_, 1))
{
    std::strcpy(m_buffer_, cstr);
}

pstring::~pstring()
{
    m_allocator_.deallocate_bytes(m_buffer_, m_length_, 1);
}
```



https://github.com/bloomberg/p1160 tag CppNow2019

Example: A broken string class (buffer overrun *fixed*)

```
pstring::pstring(const char *cstr, allocator_type allocator)
: m_allocator_(allocator)
, m_length_(std::strlen(cstr))
, m_buffer_(m_allocator_.allocate_object<char>(m_length_ + 1))
{
    std::strcpy(m_buffer_, cstr);
}

pstring::~pstring()
{
    m_allocator_.deallocate_object(m_buffer_, m_length_ + 1);
}
```

The `allocate_object` and `deallocate_object` functions are from **P0339R5**

https://github.com/bloomberg/p1160 tag CppNow2019

Example: Undeclared copy constructor (test code)

```
{
    std::pmr::test_resource tr{ "stringtr", verbose };
    tr.set_no_abort(true);
    pstring astr{ "baroof", &tr };
    pstring astr_copied{ astr };
    ASSERT_EQ(astr_copied.str(), "baroof");
}--> astr_copied.~pstring(); // deletes
    astr.~pstring(); // detects & reports via do_deallocate
```


https://github.com/bloomberg/p1160 tag CppNow2019

Interlude -- About Crashing

- While driver by faulty code, the **`test_resource`** may cause a crash
- But that happens with faulty code ***only***
- When the tested code isn't broken, there will be no crash
- Bad code may start crashing when we use the **`test_resource`**
- We either crash or detect the memory handling bug (best effort)
- In either case, we make more bugs **visible**

https://github.com/bloomberg/p1160 tag CppNow2019

Example: Bad copy constructor (non-verbose output)

```
*** Invalid magic number 0xdead0022 at address 00815858. ***

Header:
00815830:       22 00 ad de   07 00 00 00   01 0          00 00 00
00815840:       00 00 00 00   00 00 00 00   b0 d          f8 3b 00
00815850:       b1 b1 b1 b1   b1 b1 b1 b1

User segment:
00815858:       a5 a5 a5 a5   a5 a5 a5
```

That was `0xDEADBEEF` before something overwrote it. UB is at play!

https://github.com/bloomberg/p1160 tag CppNow2019

Example: Bad copy constructor (test verbose output)

```
test_resource stringtr [0]: Allocated 7 bytes (aligned 1) at 00815858.
test_resource stringtr [0]: Deallocated 7 bytes (aligned 1) at 00815858.
*** Invalid magic number 0xdead0022 at address 00815858. ***
Header:
00815830:       22 00 ad de   07 00 00 00   01 00 00 00   00 00 00 00
00815840:       00 00 00 00   00 00 00 00   b0 d8 80 00   f8 f8 3b 00
00815850:       b1 b1 b1 b1   b1 b1 b1 b1
User segment:
00815858:       a5 a5 a5 a5   a5 a5 a5
```


https://github.com/bloomberg/p1160 tag CppNow2019

Example: Bad copy constructor (test verbose output)

```
test_resource stringtr [0]: Allocated 7 bytes (aligned 1) at 00815858.
test_resource stringtr [0]: Deallocated 7 bytes (aligned 1) at 00815858.
*** Invalid magic number 0xdead0022 at address 00815858. ***
Header:
00815830:       22 00 ad de   07 00 00 00   01 00 00 00   00 00 00 00
00815840:       00 00 00 00   00 00 00 00   b0 d8 80 00   f8 f8 3b 00
```

```
================================================
          TEST RESOURCE stringtr STATE
------------------------------------------------
       Category        Blocks  Bytes
       --------        ------  -----
         IN USE        0       0
            MAX        1       7
          TOTAL        1       7
```


https://github.com/bloomberg/p1160 tag CppNow2019

Example: Adding copy-constructor (fix the double delete) 1

```
class pstring {
 public:
   using allocator_type = std::pmr::polymorphic_allocator<>;
   pstring(const char *cstr, allocator_type allocator = {});
   pstring(const pstring& other, allocator_type allocator = {});
~~~
  private:
    allocator_type  m_allocator_;
    size_t          m_length_;
    char           *m_buffer_;
};
```


https://github.com/bloomberg/p1160 tag CppNow2019

Example: Adding copy-constructor (fix the double delete) 2

```
inline
pstring::pstring(const pstring& other, allocator_type allocator)
: m_allocator_(allocator)
, m_length_(other.m_length_)
, m_buffer_(m_allocator_.allocate_object<char>(m_length_ + 1))
{
    std::strcpy(m_buffer_, other.m_buffer_);
}
```


https://github.com/bloomberg/p1160 tag CppNow2019

`test_resource` constructors, virtuals, parameters

`test_resource([name][, verbose][, upstream])` All parameters are optional (currently) Default upstream is *not* `default_resource`!	`set_verbose(bool) / bool is_verbose()` Prints on every allocation/deallocation Print summary on destruction
`~test_resource()` Leak detection happens here Summary report in verbose mode	`set_quiet(bool) / bool is_quiet()` Do not print when errors are detected
	`set_no_abort() / bool is_no_abort()` Do not abort the executable on errors
`void *do_allocate(bytes, alignment)` Allocates memory, metadata, control block Fills in all the metadata, fills in padding	`string_view name()` For custom printing
`void do_deallocate(ptr, bytes, alignment)` Most of the magic happens here Metadata checked & matched, padding checked	`memory_resource *upstream_resource()` Usually a resource using `malloc/free`
	`print()` Print statistics summary

TechAtBloomberg.com

https://github.com/bloomberg/p1160 tag CppNow2019

`test_resource` status and detection

`long long mismatches()` *Counter*
Different allocator or bad magic number
	`long long allocations()`
	`long long deallocations()`
	Counts attempts, also failed ones

`long long bounds_errors()` *Counter*
Buffer underrun/overrun
	`void *last_allocated_address()`
	`size_t last_allocated_num_bytes()`
	`size_t last_allocated_alignment()`

`long long bad_deallocate_params()` *Counter* **Size or alignment is wrong**	**Changed on successul allocations only**

`long long status()` **The destructor uses this**
Returns zero if no errors and no leaks
	`void *last_deallocated_address()`
	`size_t last_deallocated_num_bytes()`
	`size_t last_deallocated_alignment()`

`bool has_allocations()` **Used to to detect leaks by `status()`**	**Changed on successful deallocations only**

https://github.com/bloomberg/p1160 tag CppNow2019

The parts in the `test_resource` "system"

1. The test memory resource class type
2. **The test resource monitor class type**
3. **The default resource guard class type**
4. The test resource exception class type
5. An algorithm to test out of memory behavior

https://github.com/bloomberg/p1160 tag CppNow2019

Allocation monitoring

- **Sometimes we want to verify that our design does what we think it does**
  - Like **`push_back(longString);`** on a full **`vector<string>`** allocates 2 blocks
  - Pushing back again (with free capacity) allocates 1 block only, for the new string
- **Monitoring does not have to be exact, we can ask the monitor:**
  - Are there more (less) blocks allocated now?
  - Has the maximum allocations increased?
  - Were there any blocks allocated (and perhaps released as well)?
  - Of course can also ask for exact deltas (like 4 more blocks are allocated now)
- **We may want to monitor the default resource as well**

https://github.com/bloomberg/p1160 tag CppNow2019

Example: Broken string (no move constructor defined)

```
std::pmr::test_resource          tr { "object" };
std::pmr::test_resource_monitor trm{ &tr };
{
   pstring astring{ "barfool", &tr };
   ASSERT   (trm.is_total_up());
   ASSERT_EQ(trm.delta_blocks_in_use(), 1);
   trm.reset();
   pstring bstring{ std::move(astring) };
}
ASSERT(trm.is_total_same());
```

We want this to fail, as we have not made a **move** constructor, so the **copy** constructor is called.

https://github.com/bloomberg/p1160 tag CppNow2019

Example: Broken string (no move constructor defined)

```
std::pmr::test_resource          tr { "object" };
std::pmr::test_resource_monitor trm{ &tr };
{
    pstring astring{ "ba
    ASSERT    (trm.is_to
    ASSERT_EQ(trm.delta
    trm.reset();
    pstring bstring{ std::move(astring) };
}
ASSERT(trm.is_total_same());
```

But it will **not** fail, because we did not pass &tr to the copy constructor, so it uses the default resource.

We want this to fail, as we have not made a **move** constructor, so the **copy** constructor is called.

https://github.com/bloomberg/p1160 tag CppNow2019

Example: Monitoring the default_resource

```
std::pmr::test_resource           tr { "object" };
std::pmr::test_resource_monitor   trm{ &tr };
std::pmr::test_resource           dr { "default" };
std::pmr::test_resource_monitor   drm{ &dr };
std::pmr::default_resource_guard  drg{ &dr };
```

1. In its constructor saves the default resource, then installs the new one.
2. The destructor restores the saved old resource as the default resource.

https://github.com/bloomberg/p1160 tag CppNow2019

Example: Monitoring the default_resource

```
std::pmr::test_resource           tr { "object" };
std::pmr::test_resource_monitor   trm{ &tr };
std::pmr::test_resource           dr { "default" };
std::pmr::test_resource_monitor   drm{ &dr };
std::pmr::default_resource_guard  drg{ &dr };

{
  /* The moving test code remains the same */
}

ASSERT(trm.is_total_same()); // Th
ASSERT(drm.is_total_same());
```

This new assertion is going to fail, as the **copy** constructor is called, and it uses the default resource.

https://github.com/bloomberg/p1160 tag CppNow2019

`test_resource` & `monitor` statistics, telemetry

`std::pmr::test_resource`
`long long blocks_in_use() / bytes_in_use()`
Current memory use in this resource
`long long total_blocks() / total_bytes()`
Running sum during resource lifetime
`long long max_blocks() / max_bytes()`
Maximum number of active allocations/bytes

`std::pmr::test_resource_monitor`	
`bool is_in_use_up() / *_same() / *_down()` `long long delta_allocated_blocks()`	
The number of allocated blocks	
`bool is_max_up() / is_max_same()` `long long delta_max_blocks()`	
Did the maximum change, and how much?	
`bool is_total_up() / is_total_same()` `long long delta_total_blocks()`	It is a sum
Did we allocate any new memory?	

Yes, there are no bytes-statistics monitored. We (Bloomberg) never needed to monitor them for changes, only needed their current value, at the very end of a test.

https://github.com/bloomberg/p1160 tag CppNow2019

TechAtBloomberg.com

The parts in the `test_resource` "system"

1. The test memory resource class type
2. The test resource monitor class type
3. The default resource guard class type
4. **The test resource exception class type**
5. **An algorithm to test out of memory behavior**

https://github.com/bloomberg/p1160 tag CppNow2019

The exception test loop introduction (code example follows)

- The `test_resource` has a configurable allocation limit number
- When the limited number of allocations is reached it throws a `test_resource_exception` that inherits from `std::bad_alloc`
- The test loop repeatedly calls a supplied lambda with increasing limit (0, then 1, 2, …) until no `test_resource_exception` is thrown
- Therefore the algorithm throws an exception from all allocation points
- So we may verify the basic exception safety guarantee through all code paths in face of allocation failures

https://github.com/bloomberg/p1160 tag CppNow2019

Example: Failed allocation testing code

```
std::pmr::test_resource tpmr{ "tester" };
std::pmr::string lstr{ "A very long string that hopefully allocates memory" };

std::pmr::exception_test_loop(&tpmr,
                              [lstr](std::pmr::memory_resource& tpmr) {
    std::pmr::deque<std::pmr::string> deq{ &tpmr };
    deq.push_back(lstr);
    deq.push_back(lstr);

    ASSERT_EQ(deq.size(), 2);
    });
}
```


https://github.com/bloomberg/p1160 tag CppNow2019

Example: Introducing The allocations

Disclaimer

This slide will show how the standard library I used has implemented **`std::pmr::deque<>`** at a certain date. Not all standard library implementations behave the same way and they also change.

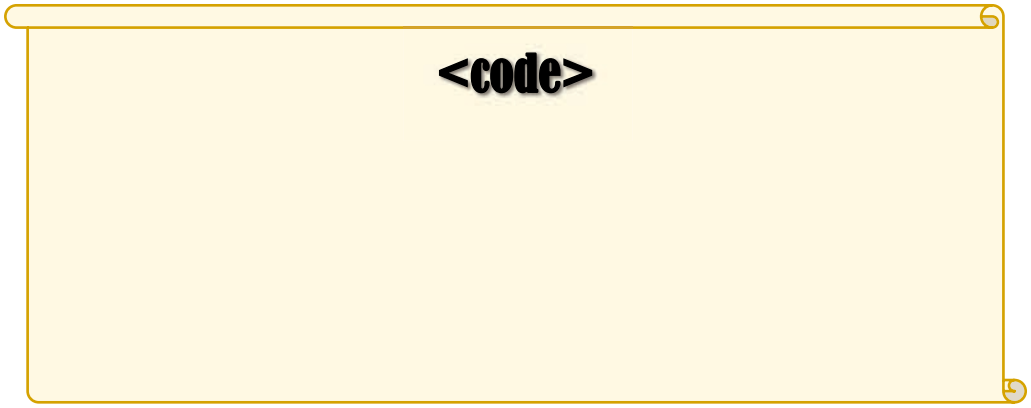
https://github.com/bloomberg/p1160 tag CppNow2019

Example: Introducing The allocations

https://github.com/bloomberg/p1160 tag CppNow2019

Example: Introducing The allocations

deque



https://github.com/bloomberg/p1160 tag CppNow2019

Example: Introducing The allocations

deque

m
a
p

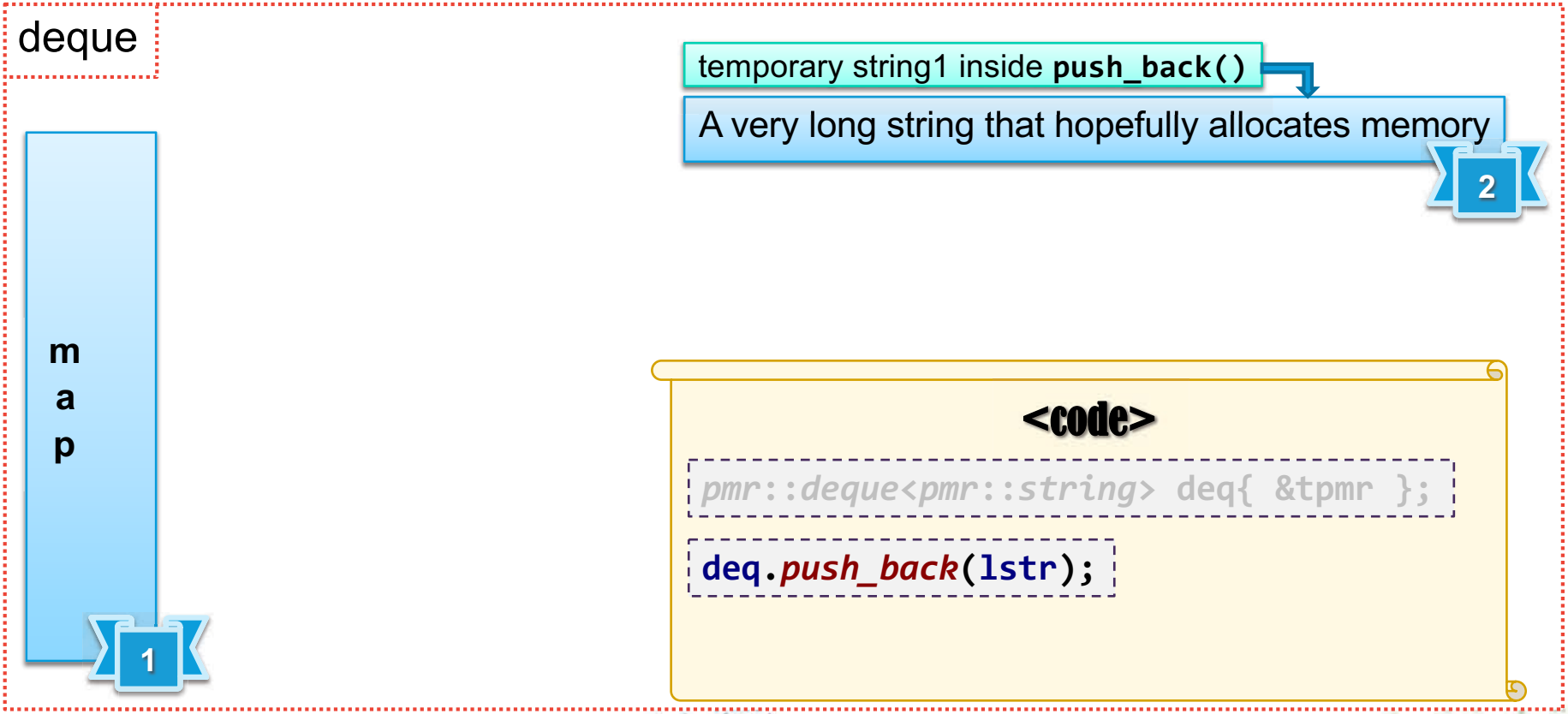
1

<code>

```
pmr::deque<pmr::string> deq{ &tpmr };
```

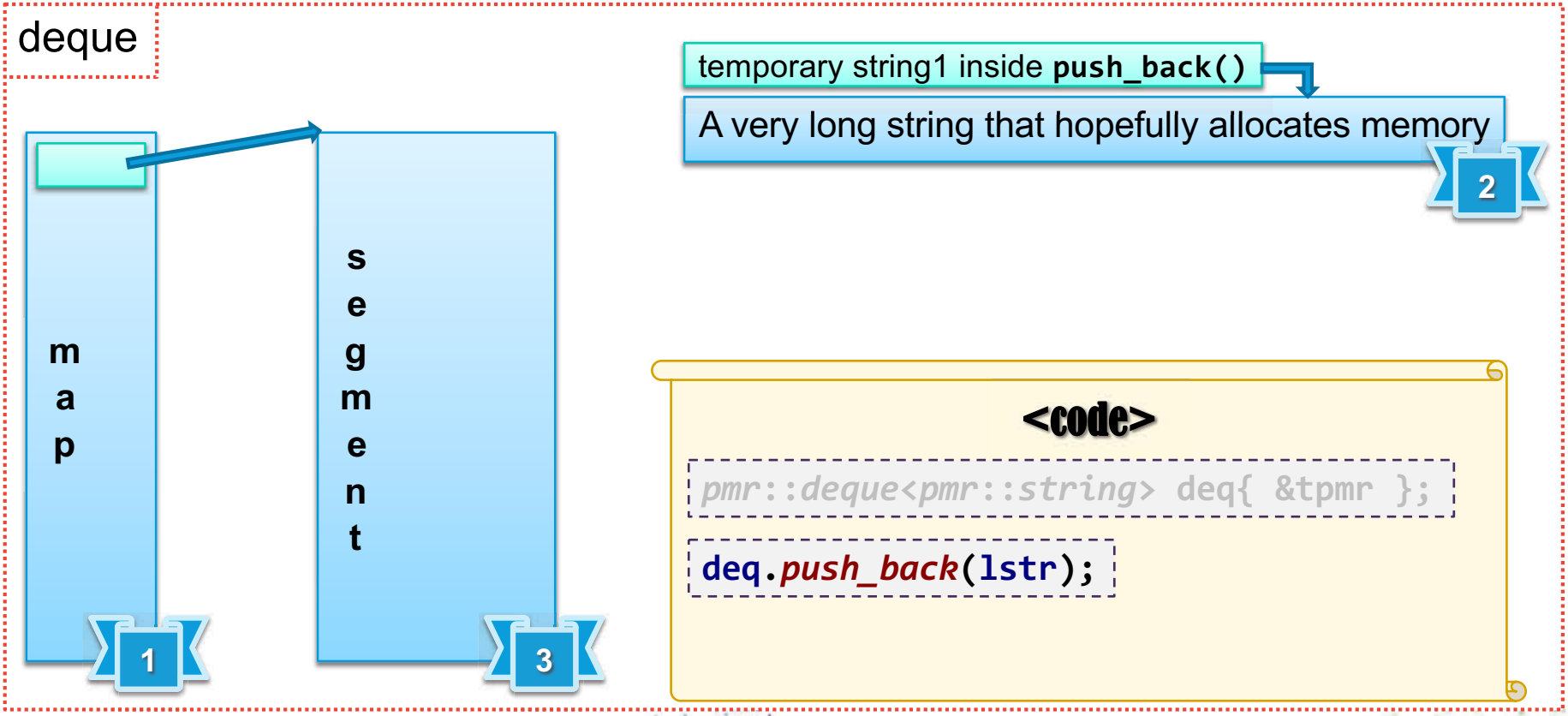

https://github.com/bloomberg/p1160 tag CppNow2019

Example: Introducing The allocations



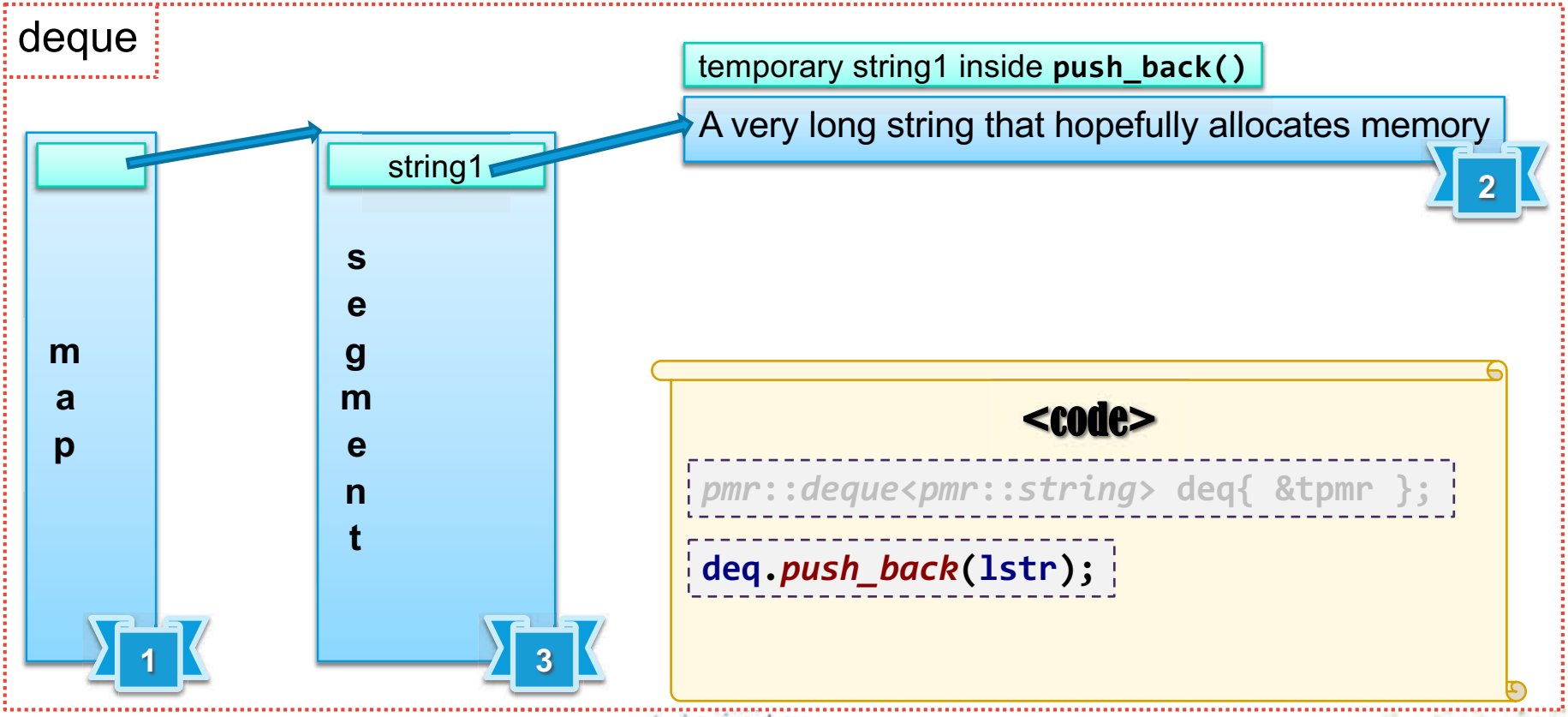
https://github.com/bloomberg/p1160 tag CppNow2019

Example: Introducing The allocations



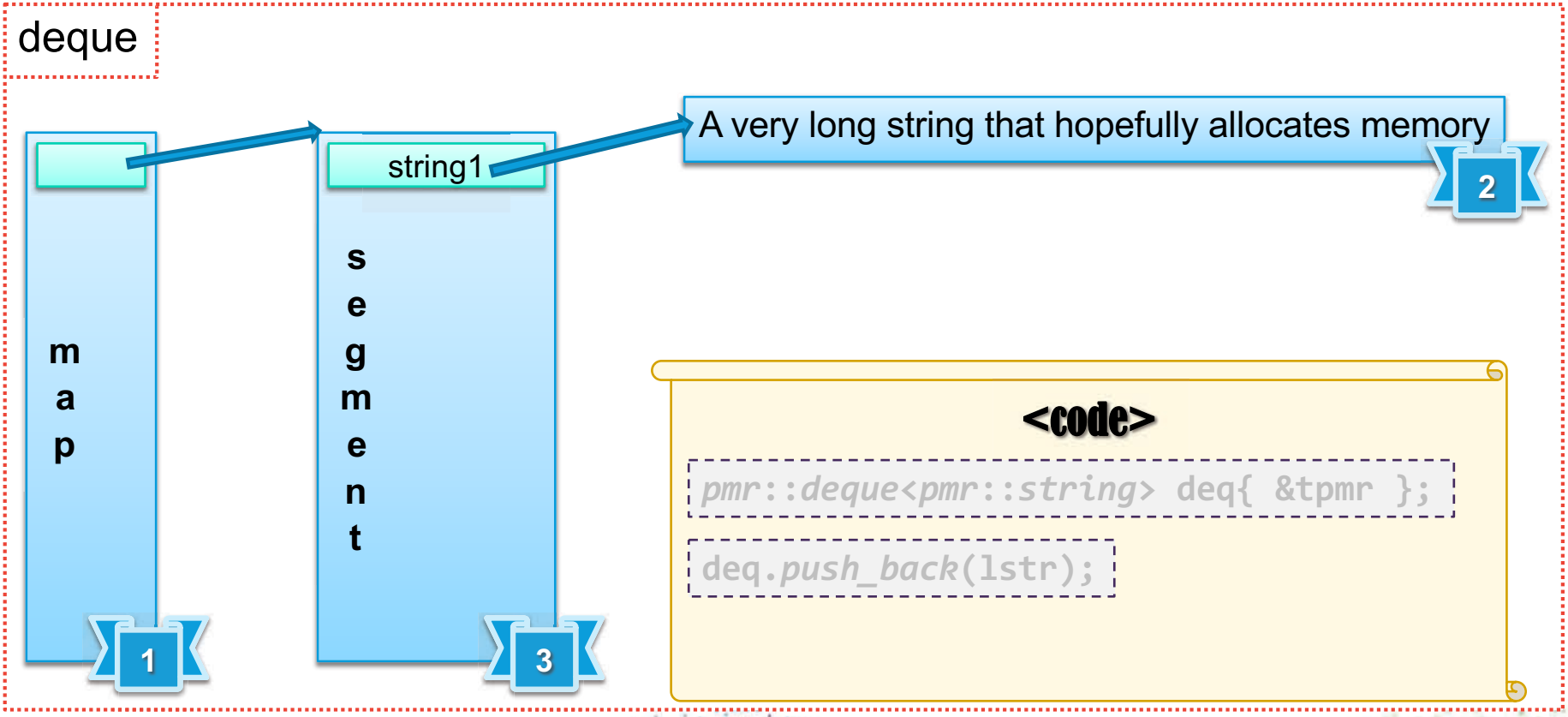
https://github.com/bloomberg/p1160 tag CppNow2019

Example: Introducing The allocations



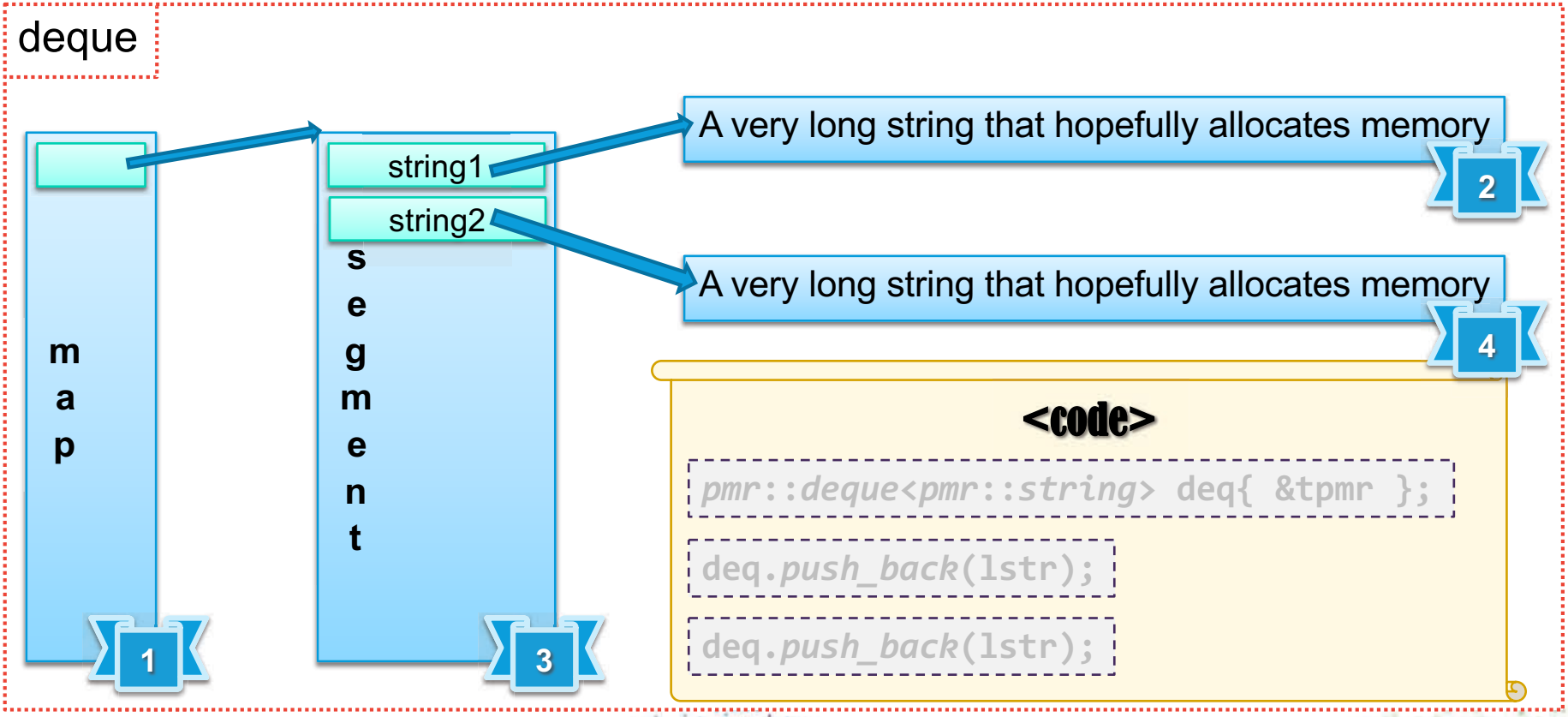
https://github.com/bloomberg/p1160 tag CppNow2019

Example: Introducing The allocations



https://github.com/bloomberg/p1160 tag CppNow2019

Example: Introducing The allocations



https://github.com/bloomberg/p1160 tag CppNow2019

Example: Failed allocation testing

First iteration

```
*** exception: alloc limit = 0, last alloc size = 28, align = 4 ***
```


```
std::pmr::test_resource tpmr{ "tester" };
std::pmr::string lstr{ "A very long string t                allocates memory" };

std::pmr::exception_test_loop(&tpmr,
                              [lstr](std::pmr::memory_resource& tpmr) {
    std::pmr::deque<std::pmr::string> deq{ &tpmr };
    deq.push_back(lstr);
    deq.push_back(lstr);

    ASSERT_EQ(deq.size(), 2);
    });
}
```


https://github.com/bloomberg/p1160 tag CppNow2019

Example: Failed allocation testing Second iteration

```
Allocated 28 bytes (aligned 4) at 00641018.
```

1

```
std::pmr::test_resource tpmr{ "tester
std::pmr::string lstr{ "A very long st                at hopefully allocates memory" };

std::pmr::exception_test_loop(&tpmr,
                              [lstr](st  :pmr::memory_resource& tpmr) {
    std::pmr::deque<std::pmr::string> deq{ &tpmr };
    deq.push_back(lstr);
    deq.push
```

2

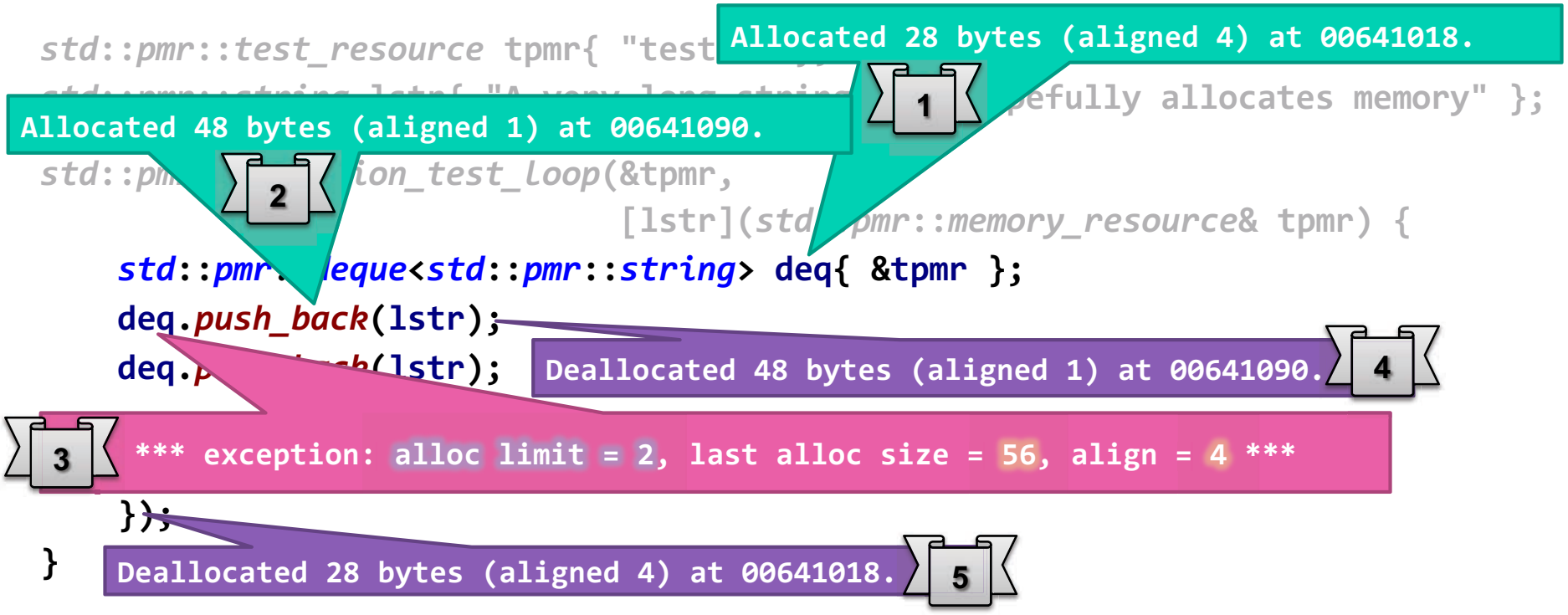
```
*** exception: alloc limit = 1, last alloc size = 48, align = 1 ***
    ASSERT_EQ(deq.size(), 2);
    });
}
```

```
Deallocated 28 bytes (aligned 4) at 00641018.
```

3

https://github.com/bloomberg/p1160 tag CppNow2019

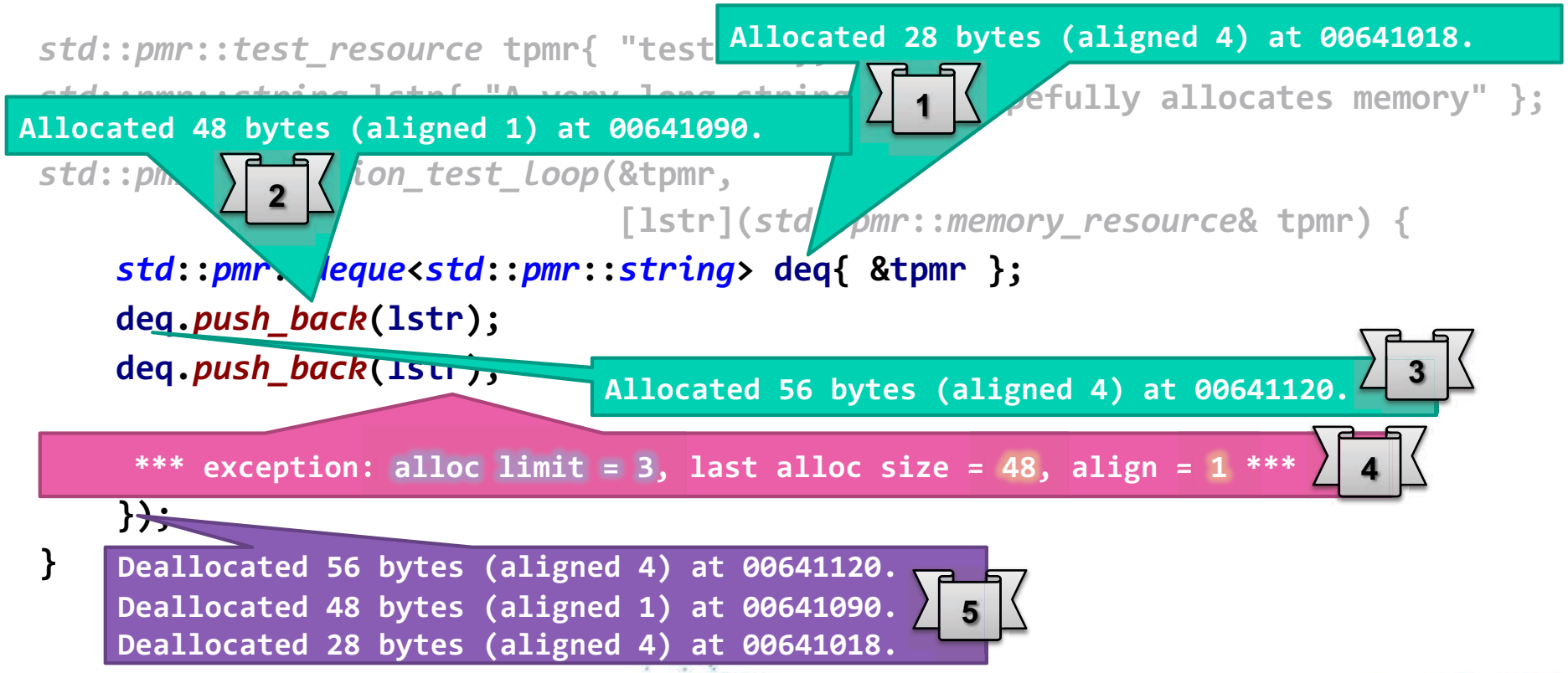
Example: Failed allocation testing — Third iteration



https://github.com/bloomberg/p1160 tag CppNow2019

Example: Failed allocation testing

Fourth iteration




```
std::pmr::test_resource tpmr{ "test                    pefully allocates memory" };
std::pm                  ion_test_loop(&tpmr,
                                       [lstr](std   pmr::memory_resource& tpmr) {
    std::pmr.  leque<std::pmr::string> deq{ &tpmr };
    deq.push_back(lstr);
    deq.push_back(lstr);
    });
}
```


https://github.com/bloomberg/p1160 tag CppNow2019

Example: Failed allocation testing

Fifth iteration

```
std::pmr::test_resource tpmr{ "test
                                                    pefully allocates memory" };
std::pm                    ion_test_loop(&tpmr,
                                         [lstr](std   pmr::memory_resource& tpmr) {
    std::pmr:  leque<std::pmr::string> deq{ &tpmr };
    deq.push_back(lstr);
    deq.push_back(lstr);

    ASSERT_EQ(deq.size(), 2);
    });
}
```

Allocated 28 bytes (aligned 4) at 00641018. 1

Allocated 48 bytes (aligned 1) at 00641090. 2

Allocated 56 bytes (aligned 4) at 00641120. 3

Allocated 48 bytes (aligned 1) at 00644030. 4

Deallocated 48 bytes (aligned 1) at 00644030.
Deallocated 56 bytes (aligned 4) at 00641120.
Deallocated 48 bytes (aligned 1) at 00641090.
Deallocated 28 bytes (aligned 4) at 00641018. 5

https://github.com/bloomberg/p1160 tag CppNow2019

Example: Failed allocation testing Leak detection

```
std::pmr::test_resource tpmr{ "tester" };
std::pmr::string lstr{ "A very long string that hopefully allocates memory" };

std::pmr::exception_test_loop(&tpmr,
                              [lstr](std::pmr::memory_resource& tpmr) {
    std::pmr::deque<std::pmr::string> deq{ &tpmr };
    deq.push_back(lstr);
    deq.push_back(lstr);

    ASSERT_EQ(deq.size(), 2);
    });
}
```

At the very end the test resource is destroyed and then we would detect if during the test loop we have failed to release memory.

https://github.com/bloomberg/p1160 tag CppNow2019

Topics

I. Introduction

II. The Purpose of Test Resource

III. Polymorphic Memory Resource Recap

IV. Examples of Testing and Debugging

V. Applicability of Test Resource

VI. Enhancement Ideas

VII. A Look at The Code (https://github.com/bloomberg/p1160)

https://github.com/bloomberg/p1160 tag CppNow2019

Why do we need a `test_resource`, there are tools! 1

- Tools need to be installed, compiler sanitizers need to be enabled (build-system change)
- Many tools change the code or emulate execution
- Different tools come free with different OS, portable tools usually need a license (logistics)
- Program or translation unit granularity
- Hard to automate the interpretation of the results

https://github.com/bloomberg/p1160 tag CppNow2019

Why do we need a `test_resource`, there are tools! 2

- Tools/sanitizers give no support for out-of-memory/exception safety guarantees testing
- Execution may get several orders of magnitude slower
- We are not testing the code that will run in production
- Tools usually report false positives in unrelated (startup) code
- We want everyone to run unit tests while coding, all the time
- The license, installation, training costs and time may be considerable to get a new person up and running

https://github.com/bloomberg/p1160 tag CppNow2019

Why do we need a `test_resource`, how is it different?

- About as fast as normal (**`pmr`**-using) code
- Very easy to use (in code)
- Test results are very easy to interpret
- Works with any unit test framework easily
- Requires no change to the tested code!
- I am not saying that you should switch all your allocating type to **`pmr`** right this moment. However if you do, unit testing dynamic memory allocations becomes a breeze.

https://github.com/bloomberg/p1160 tag CppNow2019

Topics

I. Introduction

II. The Purpose of Test Resource

III. Polymorphic Memory Resource Recap

IV. Examples of Testing and Debugging

V. Applicability of Test Resource

VI. Enhancement Ideas

VII. A Look at The Code (https://github.com/bloomberg/p1160)

https://github.com/bloomberg/p1160 tag CppNow2019

Applicability: When would I use `test_resource`?

- Making a standard library implementation
- Making STL compliant containers (rope, trie, etc.)
- Or the code tested supports allocation via a **`pmr::memory_resource`** or **`polymorphic_allocator`**

https://github.com/bloomberg/p1160 tag CppNow2019

Topics

I. Introduction

II. The Purpose of Test Resource

III. Polymorphic Memory Resource Recap

IV. Examples of Testing and Debugging

V. Applicability of Test Resource

VI. Enhancement Ideas

VII. A Look at The Code (https://github.com/bloomberg/p1160)

https://github.com/bloomberg/p1160 tag CppNow2019

Ideas that may be considered in the future

- Check integrity of leaked blocks as well
- Return a pointer with the weakest alignment that satisfies what was asked for
- Add callback-settings to natively integrate into test frameworks
- Strong exception safety test loop
- CRC on the metadata? (to detect if the head was overwritten)
- Configurable “deallocate later” # of blocks? (not needed for unit tests)
- Adding allocation stack trace addresses to the meta information? (could use lot of memory)

https://github.com/bloomberg/p1160 tag CppNow2019

Topics

I. Introduction

II. The Purpose of Test Resource

III. Polymorphic Memory Resource Recap

IV. Examples of Testing and Debugging

V. Applicability of Test Resource

VI. Enhancement Ideas

VII. A Look at The Code (https://github.com/bloomberg/p1160)

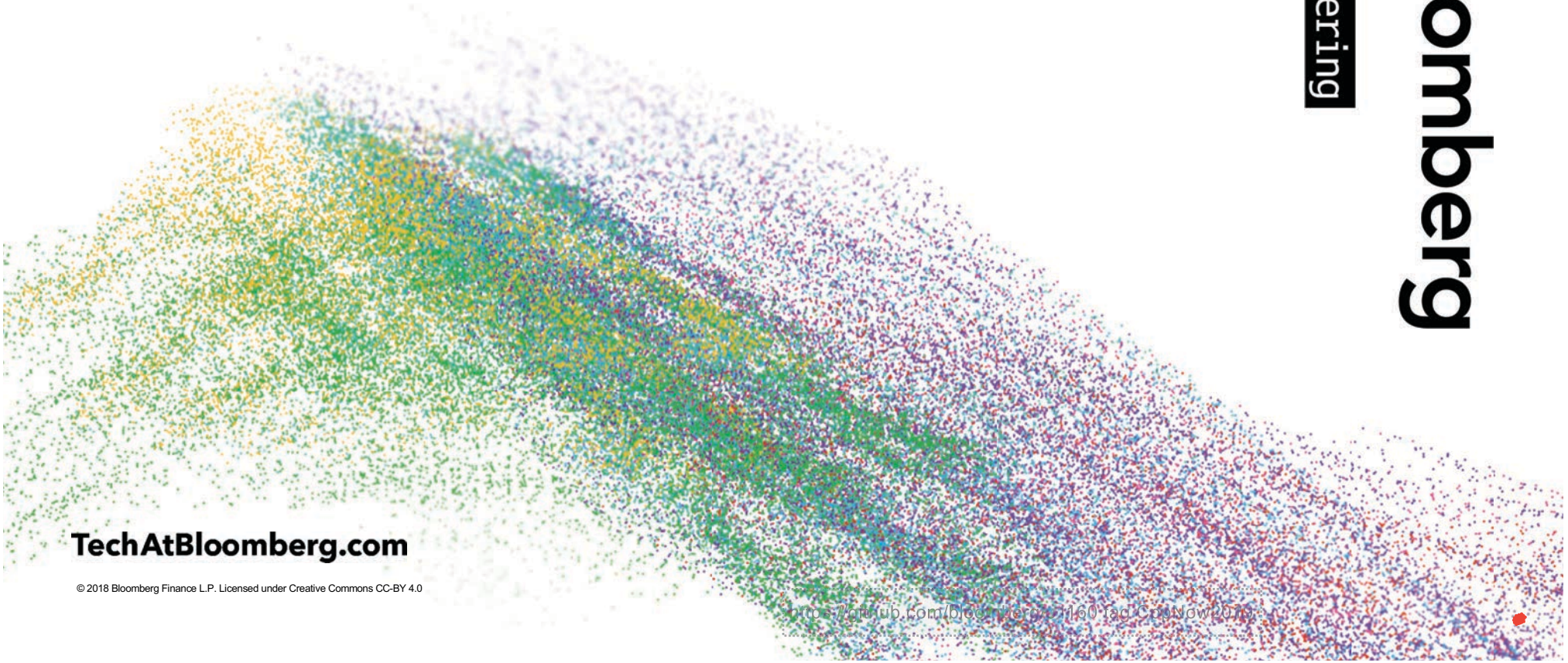
The tag **`CppNow2019`** anchors the state of the repository at the time of this presentation

https://github.com/bloomberg/p1160 tag CppNow2019

Questions?

Bloomberg

Engineering



TechAtBloomberg.com

https://github.com/bloomberg/160TechNow2018