Mitigating Tony Hoare's CAD$2.16 billion mistake

C++ Ottawa

27 March 2025

Marc Pawlowsky

2009

Null References: The Billion Dollar Mistake - Tony Hoare

Tony Hoare introduced Null references in ALGOL W back in 1965 "simply because it was so easy to implement"

https://www.infoq.com/presentations/Null-References-The-Billion-Dollar-Mistake-Tony-Hoare/

What is int* p?

- Pointer to a piece of data
- Pointer to a collection of data, *(p+4)
- No data is available,
  int *p = 0; p == nullptr;
- Pointer to a resource that must be released
- Pointer to a resource that must not be released

```
void
do_not_delete(int const * const p)
{
  delete p;
}
```

https://godbolt.org/z/zMd6YsqMr

Today’s goals

- Reduce the chance of following a null-pointer
- Reduce your cognitive load
  - Make the implicit explicit

Backlog

- Pointer to a piece of data
- Pointer to a collection of data, *(p+4)
- 0 as nullptr
- No data is available
- Pointer to a resource that must be released
- Pointer to a resource that must **not** be released

Backlog

- Pointer to a piece of data
- Pointer to a collection of data, *(p+4)
- 0 as nullptr
- No data is available
- Pointer to a resource that must be released
- Pointer to a resource that must **not** be released
- **What is the binary representation of 0**

Inspiration

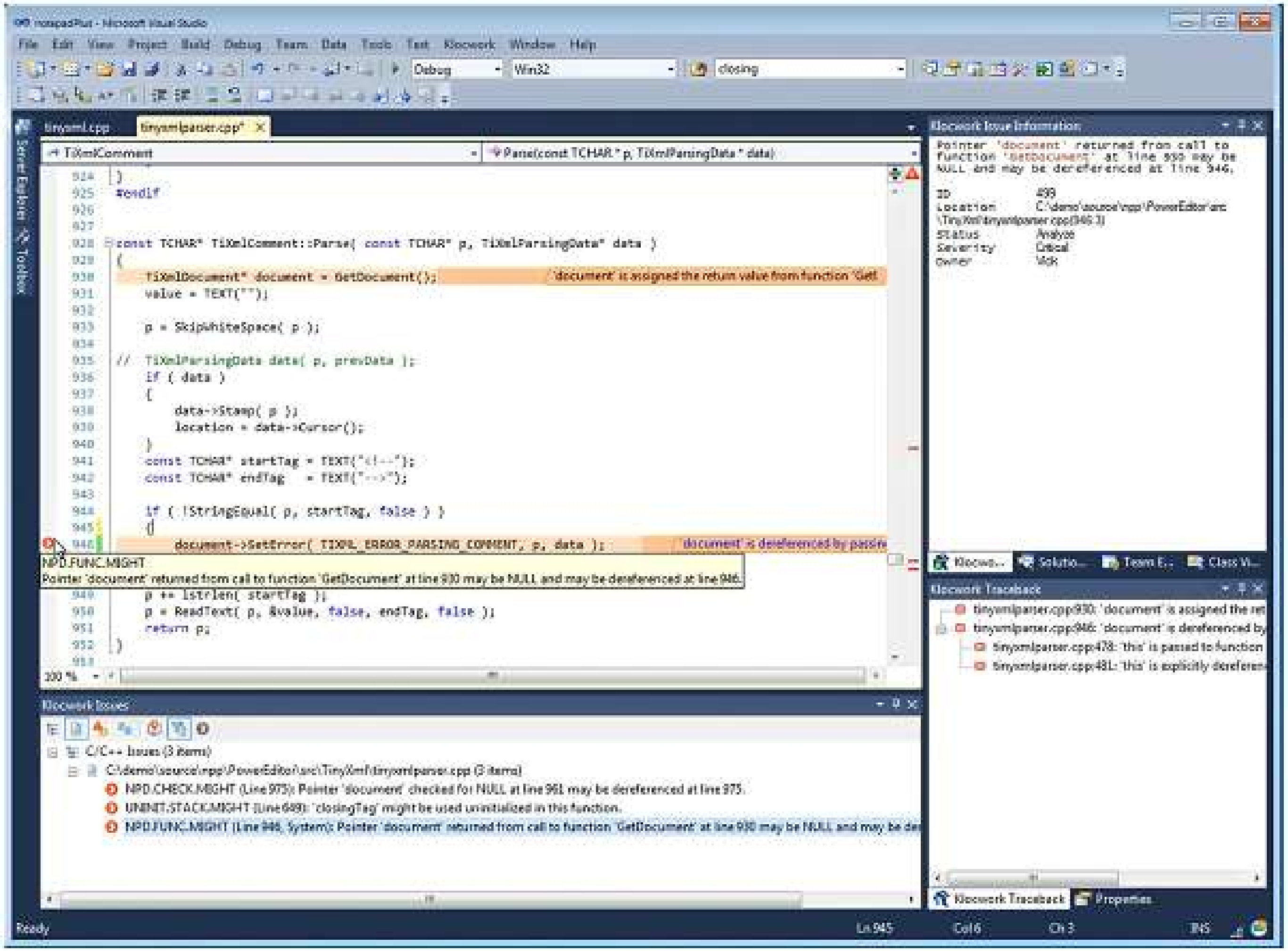
- `Java Programmers FAQ  post by Marc Pawlowsky (~1996)`
  `  Does Java have the equivalent of "const" arguments in C and C++?`
- `Correct by Construction: APIs That Are Easy to Use and Hard to Misuse :`
  ` C++ On Sea,Matt Godbolt, 2020`
- gsl::not_null, and gsl::owner
  Guidelines support library

Technique

- Step by step refactoring of legacy code to catch the possible misuse pointers
  - Real world:
    - C++98 using Boost
    - New code in C++17
- Introduce a compiler failure in an API and fixit.

Tools?

- **GSL : Guidelines Support Library**
- clang-tidy
- static analyzers



```
924  }
925  #endif
926
927
928  const TCHAR* TiXmlComment::Parse( const TCHAR* p, TiXmlParsingData* data )
929  {
930      TiXmlDocument* document = GetDocument();          'document' is assigned the return value from function 'Get
931      value = TEXT("");
932
933      p = SkipWhiteSpace( p );
934
935  //  TiXmlParsingData data( p, prevData );
936      if ( data )
937      {
938          data->Stamp( p );
939          location = data->Cursor();
940      }
941      const TCHAR* startTag = TEXT("<!--");
942      const TCHAR* endTag   = TEXT("-->");
943
944      if ( !StringEqual( p, startTag, false ) )
945      {
946          document->SetError( TIXML_ERROR_PARSING_COMMENT, p, data );     'document' is dereferenced by passin
949      p += lstrlen( startTag );
950      p = ReadText( p, &value, false, endTag, false );
951      return p;
952  }
953
```

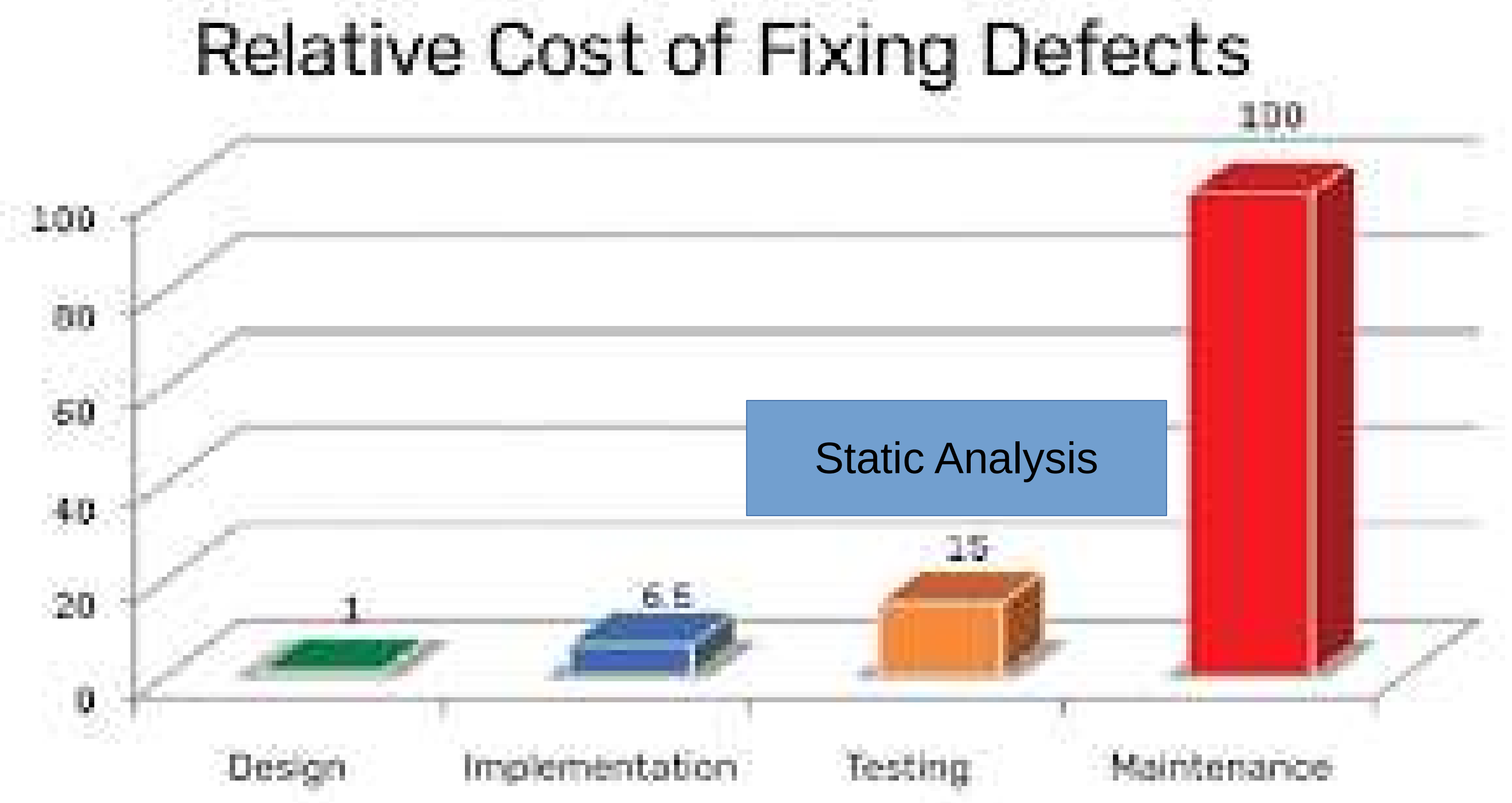
NPD.FUNC.MIGHT
Pointer 'document' returned from call to function 'GetDocument' at line 930 may be NULL and may be dereferenced at line 946.

```
Pointer 'document' returned from call to
function 'GetDocument' at line 930 may be
NULL and may be dereferenced at line 946.

ID          499
Location    C:\demo\source\npp\PowerEditor\src
\TinyXml\tinyxmlparser.cpp(946:3)
Status      Analyze
Severity    Critical
Owner       Vick
```

- tinyxmlparser.cpp:930: 'document' is assigned the ret
- tinyxmlparser.cpp:946: 'document' is dereferenced by
  - tinyxmlparser.cpp:478: 'this' is passed to function
  - tinyxmlparser.cpp:481: 'this' is explicitly dereferen

- C/C++ Issues (3 items)
  - C:\demo\source\npp\PowerEditor\src\TinyXml\tinyxmlparser.cpp (3 items)
    - NPD.CHECK.MIGHT (Line 975): Pointer 'document' checked for NULL at line 961 may be dereferenced at line 975.
    - UNINIT.STACK.MIGHT (Line 649): 'closingTag' might be used uninitialized in this function.
    - NPD.FUNC.MIGHT (Line 946, System): Pointer 'document' returned from call to function 'GetDocument' at line 930 may be NULL and may be de


Relative Cost of Fixing Defects
100
80
60
40
20
0
1
6.5
15
100
Design
Implementation
Testing
Maintenance
Static Analysis

mp

- **namespace mp = marcpawl::pointers**
- **https://github.com/marcpawl/nullptr/tree/20250327**

Wrapping gotchas

- operator bool()

Take a look at:

2020 LLVM Developers’ Meeting

Using Clang-tidy for Customized Checkers and Large Scale Source Tree Refactoring - Vince Bridgers

- How to write your own rules
- Fix one clang-tidy rule at a time across a whole project

Step 1: nullptr

- Replace 0 with nullptr
  - **clang-tidy - modernize-use-nullptr**

```
int* p = 0;
=====>
int* p = nullptr;
```

Step 1: nullptr

- Replace implicit use in conditional
  - **clang-tidy - readability-implicit-bool-conversion**

```
int* p;
If (p) {}
=====>
if (p != nullptr) {}
```

Backlog

- Pointer to a piece of data
- Pointer to a collection of data, *(p+4)
- ~~0 as nullpt~~r
- No data is available
- Pointer to a resource that must be released
- Pointer to a resource that must **not** be released
- What is the binary representation of 0

Step 2 : Describe data availability

Data may not available

Comments as a type

```
//  a flag for saying go fetch the data
void foo(int* ptr) {
  if (nullptr == ptr) {
   ptr = new int(4);
  }
}

//  a flag for no-op
void fee(int* ptr) {
  if (nullptr == ptr) {
     return;
  }
}
```

Data must be available

Comment as type

```
// @param p may not be null
int square(int* p) {
return (*p) * (*p);
}
```

Data must be available

Soft failure

```
// @param p must not be null, ...
// @return -1 if p is nullptr, else ...
int square(int* p) {
  if (nullptr != p) {
     // Fix for BUG12345
     return -1; // in-band encoding
  }
  return (*p) * (*p);
}
```

Data must be available

Hard failure

```
// @param p must not be null
int square(int* p) {
  assert(nullptr != p);
  return (*p) * (*p);
}
```

Backlog

- ~~Pointer to a piece of data~~
- **Runtime type description of data must be available**
- Pointer to a collection of data, *(p+4)
- ~~No data is available~~
- Pointer to a resource that must be released
- Pointer to a resource that must **not** be released
- What is the binary representation of 0

Step 3: Use a type

- Have the compiler tell you when data must be available with run time checks.

Step 3.1 : gsl::not_null

- Use gsl::not_null to indicate data MUST be available

  void hum(gsl::not_null<int*> p)

  - Opinionated: prevents pointer arithmetic
- **clang-tidy - cppcoreguidelines-owning-memory**

https://github.com/microsoft/GSL/blob/3.0.1/include/gsl/pointers#L69

Step 3.2 : gsl::strict_not_null

- Use gsl::strict_not_null
- Removes explicit constructor and automatic conversion.

  ```
  void hum(gsl::strict_not_null<int*> p)
  ```

Cost gsl::not_null

- No extra space
- void foo(int *p) {   auto np = gsl::strict_not_null<int*>(p);}
  we have a conditional, equivalent to the assert.
- void bar(gsl::strict_not_null<int*> q);
  void foo(gsl::strict_not_null<int*> p) {   bar(p);}
  No conditional, no need for asserts.
- Compiler optimizes out extra check.

Step 3.3 : mp::nullable

Type as a comment.

```
template<
  class T,
  std::enable_if_t<std::is_pointer<T>::value, bool> = true>
using nullable = T;

void foo(nullable<int*> ptr);
```

Step 3.4 Overload pointer functions

```
void foo(mp::maybe_null<int*> p)
{
  if (p == nullptr) {
    return;
  }
  // DO something big with p
}
```

```
void foo(gsl::not_null<int*> p)
{
  // DO something big with p
}

void foo(mp::maybe_null<int*> p)
{
  if (p == nullptr) {
    return;
  }
  foo(gsl::make_not_null(p));
}
```

https://godbolt.org/z/341vY991v

Where are we

- gsl::strict_not_null<int*>
- mp::nullable<int*>
  - it is intentional that null is a valid value
- int*
  - To be determined

Backlog

- ~~Compile time description of data must be available~~
- **Force nullable to be checked before use.**
- Pointer to a collection of data, *(p+4)
- Pointer to a resource that must be released
- Pointer to a resource that must **not** be released
- What is the binary representation of 0

Step 4 : std::span or gsl::span

- clang-tidy - cppcoreguidelines-pro-bounds-pointer-arithmetic

- Switch from legacy:

```
void print(Node const* nodes, size_t count)
```

To bounds check:

```
void print(std::span<Node const>& nodes)
```

https://godbolt.org/z/71Tn5s3KE

- std::span<T>::operator[](size_t)
  Bounds checking
  Coming in C++26

std::span cost

- Space is larger?
  - Pointer
  - Size
    - Maybe you only need a UINT8

Backlog

- Force nullable to be checked before use**.**
- ~~Pointer to a collection of data, *(p+4)~~
- Pointer to a resource that must be released
- Pointer to a resource that must **not** be released
- What is the binary representation of 0

Delete or not to delete?

```
struct Node {
  Node* next;
  Node(Node* next_):
    next(next_)
  {}
  ~Node() {
      delete next; // ?
  }
}
```

Step 5 : Comments as a type

```
struct Node {
  // next’s lifespan is distinct from this node.
  // should not be deleted.
  Node* next;
  Node(Node* next_):
    next(next_)
  {}
  ~Node() {
  }
}
```

Backlog

- Force nullable to be checked before use.
- ~~Pointer to a resource that must be released~~
- ~~Pointer to a resource that must **not** be released~~
- **Type for Pointer to a resource that must be released**
- **Type for Pointer to a resource that must not be released**
- What is the binary representation of 0

Step 6 : types for ownership

Types as comments

- gsl::owner

  ```
  template <class T, std::enable_if_t<std::is_pointer<T>::value,
  bool> = true>
  using owner = T;
  ```

  https://godbolt.org/z/ax45bfYxz
- gsl/pointers#L78

Types as comments

- mp::nonowner

  https://godbolt.org/z/raeTMcWvj

```
template <
  class T,
  std::enable_if_t<std::is_pointer<T>::value, bool> = true>
using nonowner = T;

void bar(nonowner<Node*> t) { ...}
```

Owner guidelines

- Do not use gsl::owner<T*> const&, use mp::nonowner<T*> const& instead.

  If you are not manipulating the pointer why are you identifying it as an owner.

Where are we

gsl::owner< gsl::strict_not_null<int*>>	gsl::owner< mp::nullable<int*>>	gsl::owner< std::span<int>>
mp::borrower< gsl::strict_not_null<int*>>	mp::nonowner< mp::nullable<int*>>	mp::nonowner< std::span<int>>
gsl::strict_not_null<int*>	mp::nullable<int*>	std::span<int>
gsl::owner<int*>	mp::nonowner<int*>	int*

Backlog

- Force nullable to be checked before use.
- ~~Type for Pointer to a resource that must be released~~
- ~~Type for Pointer to a resource that must not be released~~
- **Managed resources**
- What is the binary representation of 0

Step 7 : Managed resources

- std::unique_ptr
- std::shared_ptr
- std::weak_ptr

std::shared_ptr

- https://godbolt.org/z/z1hnxKzxE

std::weak_ptr

https://godbolt.org/z/M1rczeMjd

- Use lock() to get a shared_ptr
- Check to see if not null_ptr

std::shared_ptr

- Reference counted
  - Not garbage collected
- Thread safe initialization
  - Slow to construct
- Indicates shared ownership

std::unique_ptr

- Used to indicate ownership

  https://godbolt.org/z/Ecd1sx41P

Non-null managed pointers

- gsl::strict_not_null<std::unique_ptr<int>&> &
- gsl::strict_not_null<std::shared_ptr<int>> &

https://godbolt.org/z/Gs59Pzran

nullable pointers

- std::optional<gsl::strict_not_null<int*>> &

https://godbolt.org/z/1Y7roG44d

nullable pointers

```
using VN = std::variant<gsl::strict_not_null<int*>, std::nullptr_t>;
VN factory(gsl::strict_not_null<int*> b)
```

Use std::visit and you are guaranteed to handle both cases

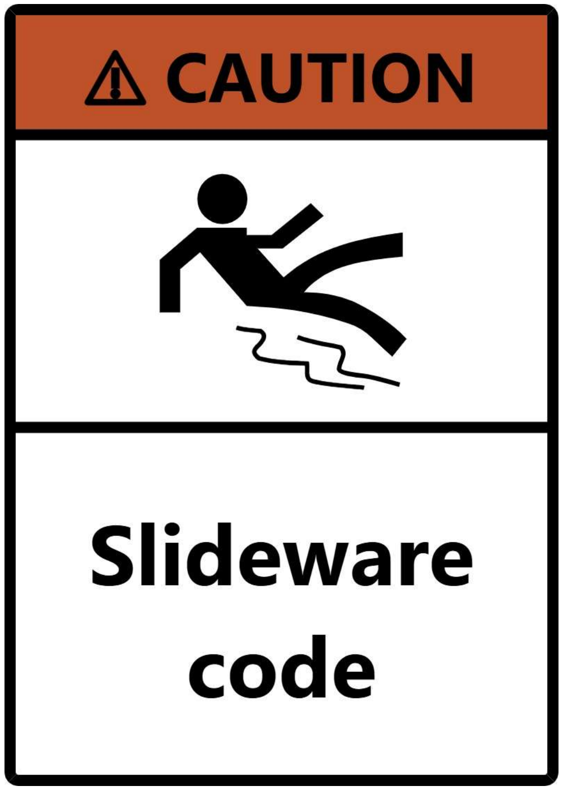
https://godbolt.org/z/oxxTMzf71

Backlog

- ~~Force nullable to be checked before use.~~
- ~~Managed resources~~
- **Cheaper checks on nullable**
- **Enforce safety owner = borrower**
- What is the binary representation of 0

One step beyond

Madness



Step 8: Compile time checking

Haskell

```
data OptionalInt = Some Int | None
    deriving (Show)

addOne :: OptionalInt -> OptionalInt
addOne None       = None
addOne (Some x) = Some (x + 1)
```

mp::strict_not_null

- copy gsl::strict_not_null
- Extra overloads to handle std::optional<strict_not_null<std::unique_ptr>>

mp::maybe_null

- Deprecates *, ->
- as_optional_not_null
- as_variant_not_null
- visit

ptr.hpp#L488

maybe_null_tests.cpp#L358

mp::borrower and mp::owner

- Owner cannot share ownership
- Owner can share usage with borrower
- Borrower can share usage with another borrower
- Borrower cannot give away ownership

mp::owner

- Cannot be assigned to another owner

```
auto fail()
{
    int p;
    mp::borrower<int*> ptr{&p};
    // ERROR: cannot create an owner from a borrower.
    mp::owner<int*> owner{ptr};
    return owner;
}
```

mp::owner

Use std::unique_ptr or std::shared_ptr instead

mp::owner

- as_borrower
  ```
  borrower<T>
  as_borrower() const {
    return borrower<T>(this->get());
  }
  ```

- Borrower explicit constructor
  ```
  template<Pointer U>
  explicit borrower(owner<U> const &other)
  ```

mp::borrower

- Compile time failure for delete
  - Can delete a pointer to a borrower
  - Can delete the pointer the borrower contains (BAD)
  - Cannot delete an object.
    mp::borrower<int*> p;
    delete p;

Backlog

- ~~Force check of maybe null~~
- What is the binary representation of 0

Nullptr binary representation

- conv.ptr
- basic.compound
- On most platforms, a null pointer is represented by a binary value of all zeros (e.g., 0x00000000 on a 32-bit system). However, this is not guaranteed by the standard. Some platforms might use a different representation for null pointers.

Backlog

- ~~What is the binary representation of 0~~

Summary int*

Three dimensions:

- Can be null?
- Collection?
- Ownership?

Summary Collection

Range checking with std::span

Summary int*

Nullptr is in-band encoding

Summary int*

Ownership is not specified

Summary int*

Wrappers give
strong type checking
at compile time

Thank you

marcpawl@gmail.com