Rethinking Strings

Mark Zeren

C++Now, May 16, 2017

vmware®

1 **Inspiration**

2 Experience

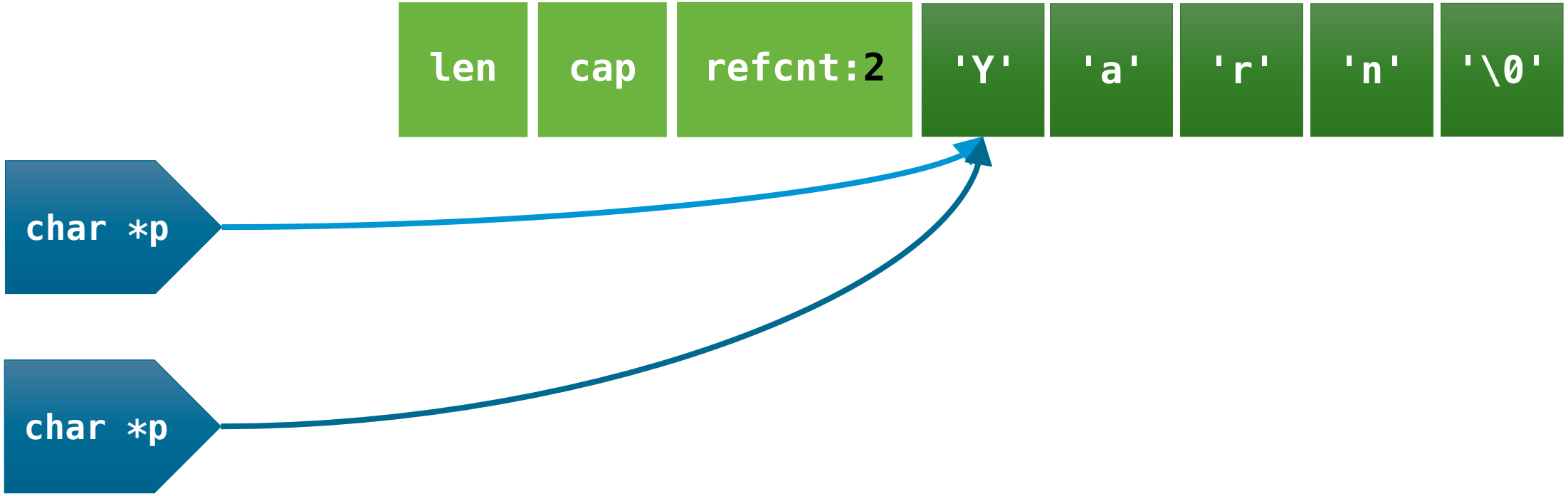
3 Rethinking

4 Code

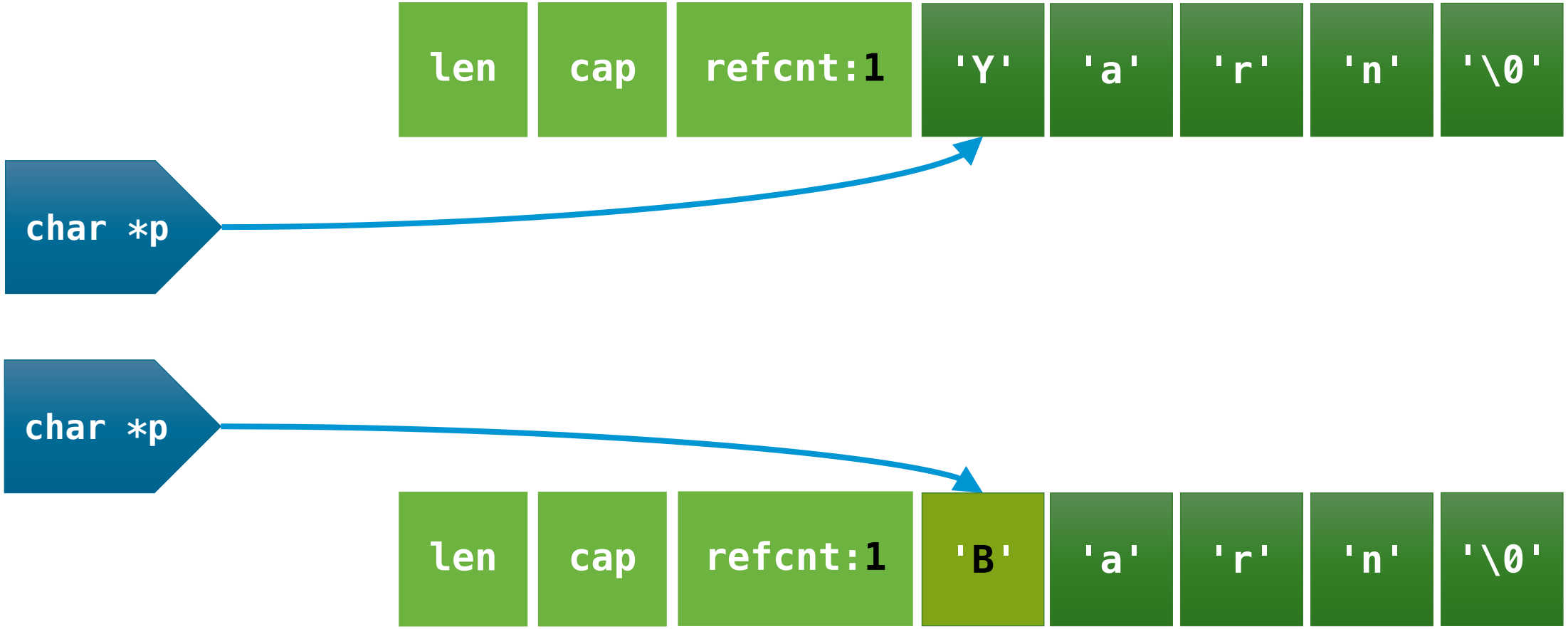
The COW is Dead

Copy on write strings are going away

The COW is Dead



The COW is Dead



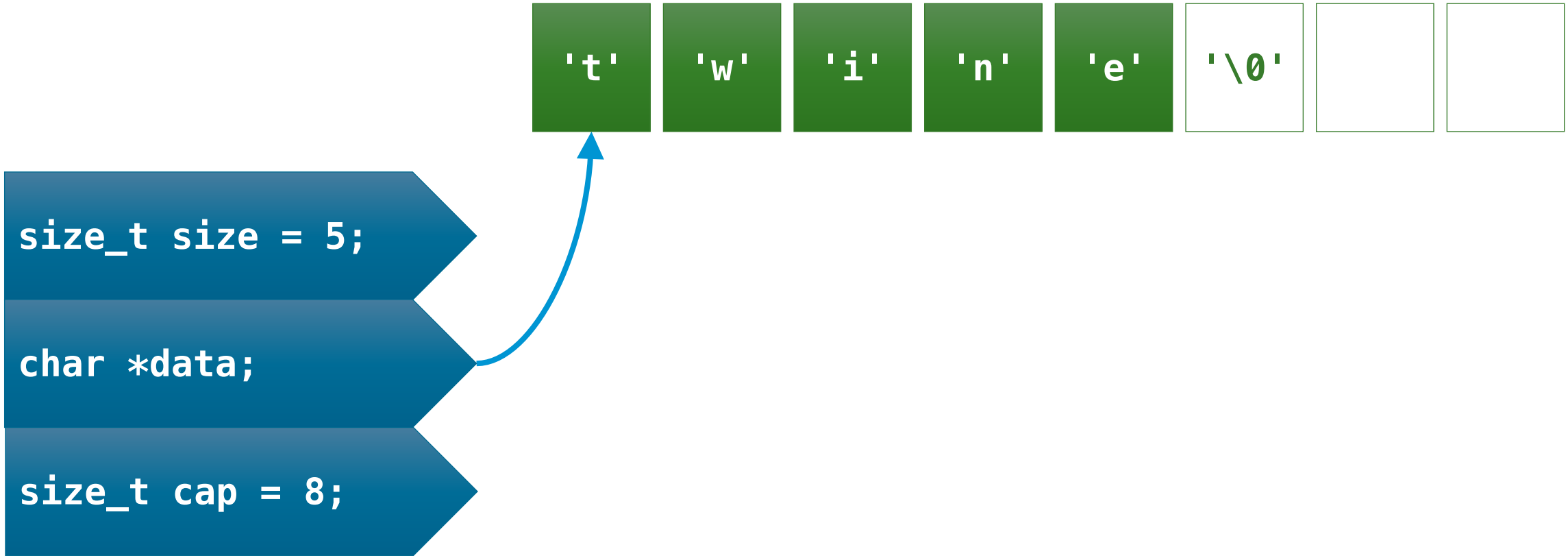
The COW is Dead

Copy on write strings are going away

SSO Is Here To Stay

Small string optimized strings have won

SSO Is Here To Stay



SSO Is Here To Stay

```
size_t size = 5;
char *end = 0x00000065'6e697774;
size_t cap;
```

"twine"

At What Cost?

- Switch to C++11 ABI in GCC:
- **8.6**% increase in average heap block size
- **12**% increase in number of used heap allocations
- **21**% increase in used heap bytes
- Caveat: SSO is only a component of the increase
- Caveat: Performance difference unknown
- Caveat: Code is optimized for COW

The COW is Dead

Copy on write strings are going away

SSO Is Here To Stay

Small string optimized strings have won the day

Who will pay?

I will.

Within limits.

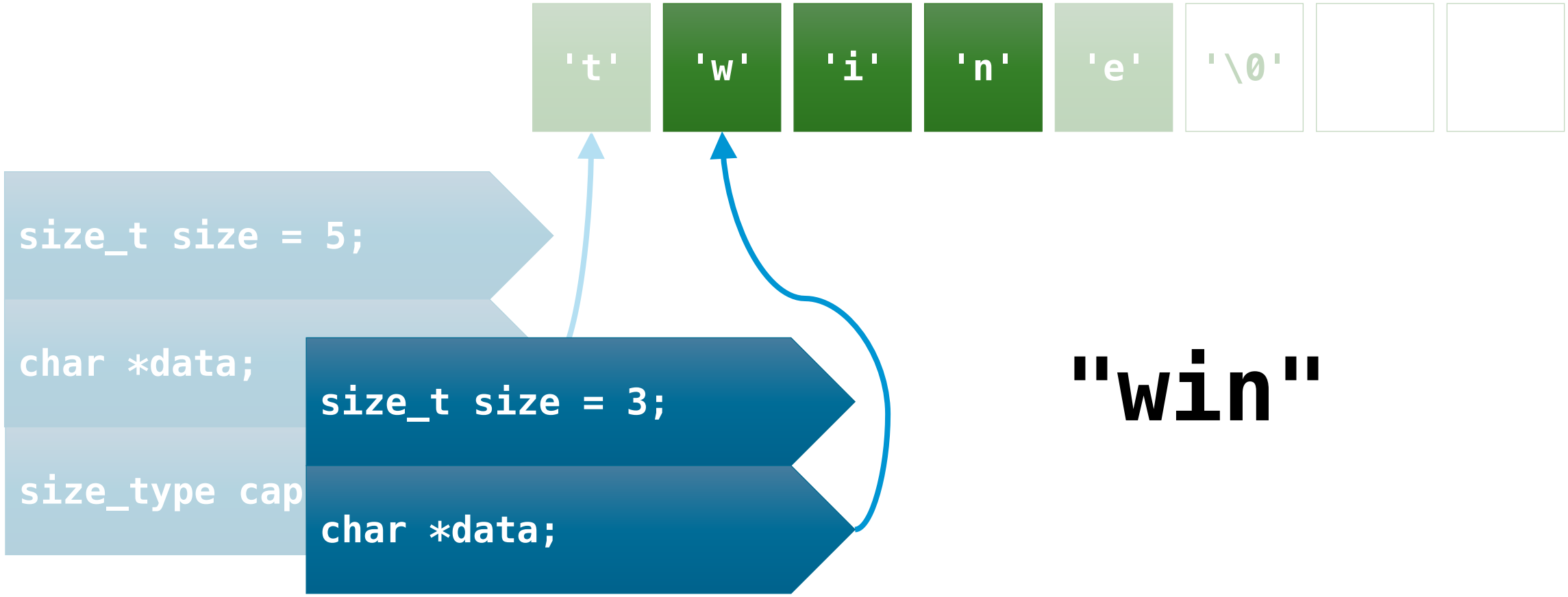
Can I Gain from the Pain?

- Just ask for more RAM?
- Atomic ops are not a bottleneck
- Can I "buy" or build a library solution?

The COW is Dead

`string_view`

string_view


't'
'w'
'i'
'n'
'e'
'\0'
size_t size = 5;
char *data;
size_type cap
size_t size = 3;
char *data;
"win"

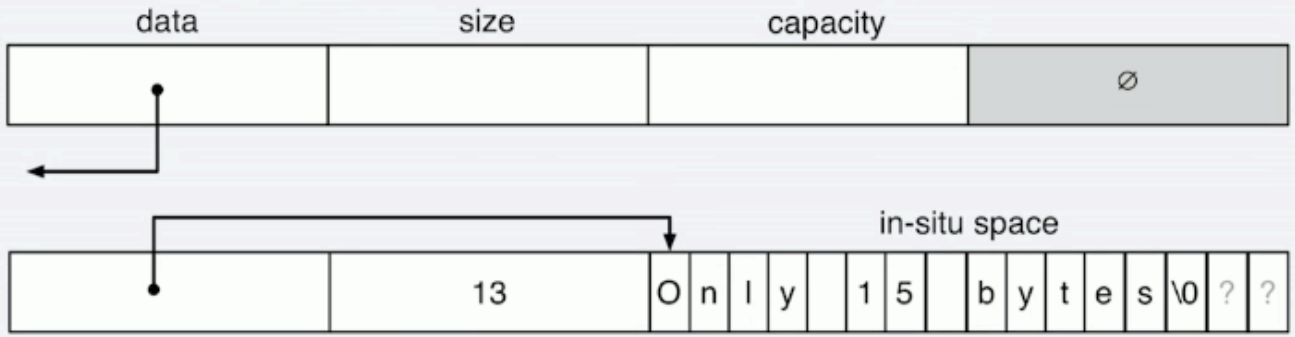
string_view

- Have limited use to date
- Will ripple virally
- If we're changing everything...
- Is it an opportunity to change everything?

CppCon 2016: Nicholas Ormrod "The strange details of std::string at Facebook"

cppcon | 2016
THE C++ CONFERENCE • BELLEVUE, WASHINGTON

gcc string (version >=5)



+Has SSO

+`data()`, `size()` very fast

+Size, 32, is power of 2

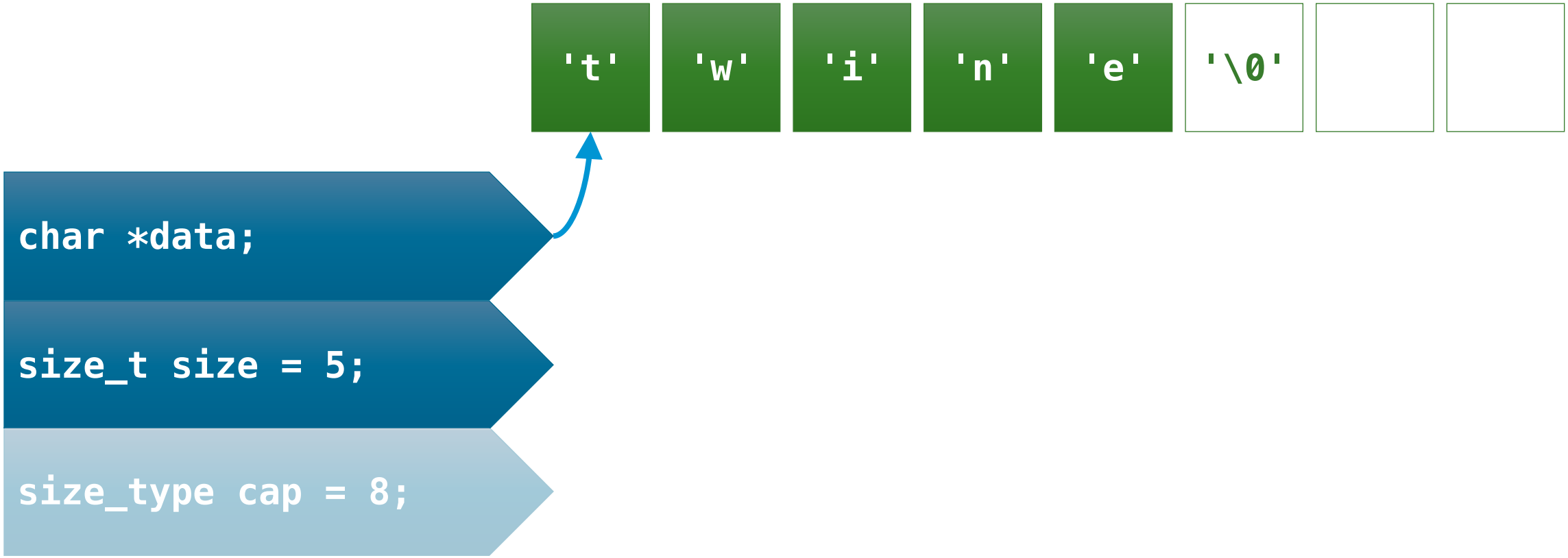
−Only 15-byte capacity

−Move is no longer memcopy

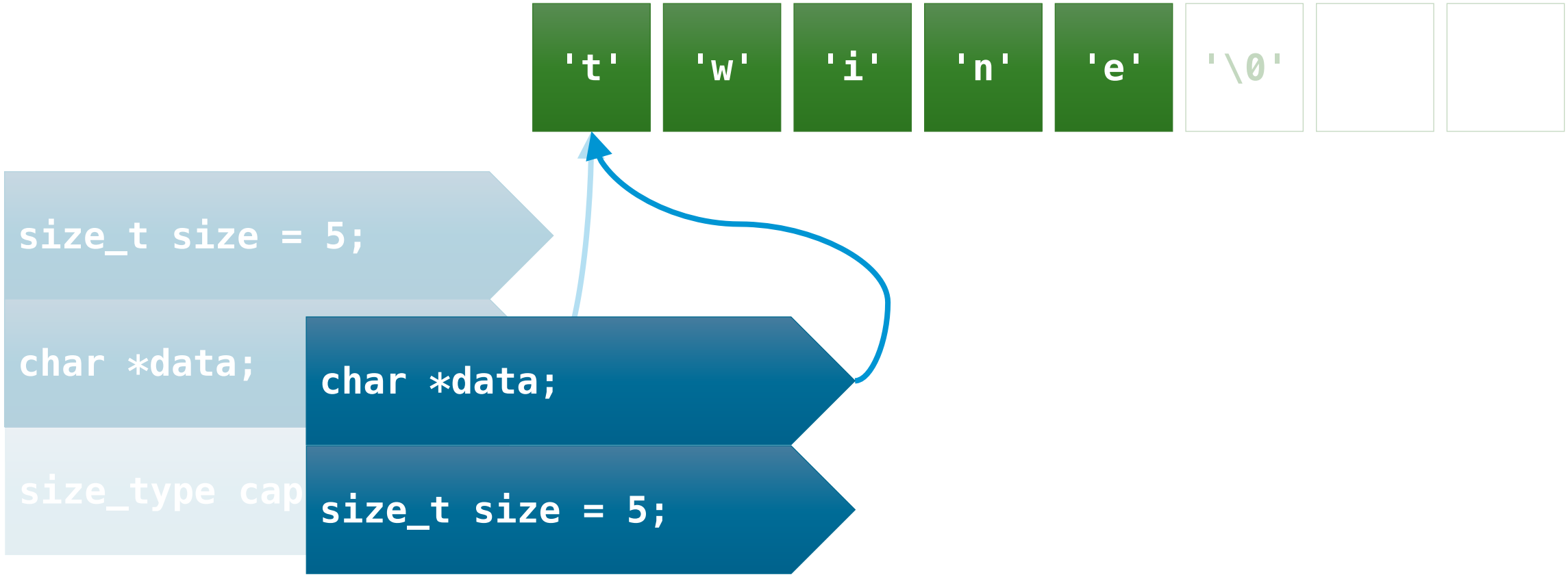
−Size is 33% larger than fbstring

The strange details of std::string at Facebook

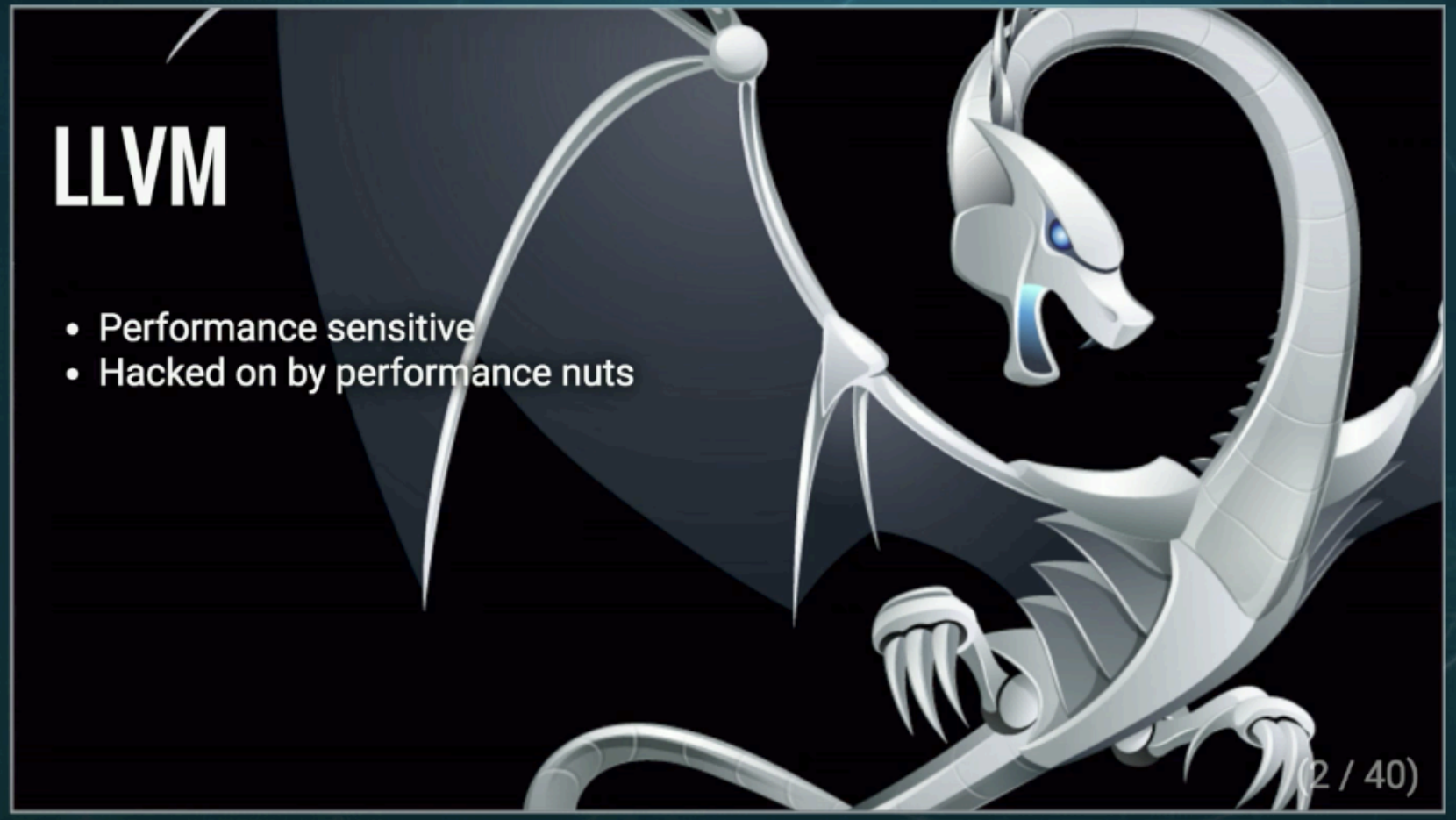
string_view



string_view


't'
'w'
'i'
'n'
'e'
'\0'
size_t size = 5;
char *data;
size_type cap
char *data;
size_t size = 5;

CppCon 2016: Chandler Carruth "High Performance Code 201: Hybrid Data Structures"



High Performance Code 201: Hybrid Data Structures

string_view

```
struct string_view {
  size_t len;
  char *data;
};
```

string

```
struct string {
  char *data;
  size_t len;
  size_t cap;
};
```

sso_string

```
class sso_string
{
  size_t len;
  char *data;
  size_t cap;
};
```

string_view

```
class string_view
{
  size_t len;
  char *data;
};
```

cow_string

```
class cow_string
{
  char *data;
};
struct cow_heap
{
  size_t len;
  size_t cap;
  int refcount;
  char data[];
};
```

c_string

```
class c_string
{
  (strlen)
  char *data;
};
```

But Why Now?

SSO Strings

string_view

constexpr

Unicode (UTF-8)

64-bit Addressing

STL2 (concepts, ranges)

Reflection and Metaprogramming

I'm not alone

- CsString (4:30pm @ Bethe)
- QString
- FBString
- fixedString
- QStringView
- ...

I'm not alone

- codecvt deprecated P0618 Alisdair Meredith
- text_view P0244 Tom Honermann
- char8_t P0482 Tom Honermann

P0372 Spencer / Italiano

Now

1 Inspiration

2 **Experience**

3 Rethinking

4 Code

`std::string` Everywhere

- Worked well for early, rapid evolution
- But measurable overhead
- Now we refactor strings everywhere

Refactoring std::string Everywhere

```
static string const kName = "FOO";
→
char const kName[] = "FOO";
```

- Removes static ctors and dtors, indirections to heap, smaller code.

Rethink

- Constants

Refactoring std::string Everywhere

```
char const kName[] = "FOO";

int f(string const&);

int main() {
  return f(kName); // Ouch! temporary
}
```

- Which we have everywhere.

Refactoring std::string Everywhere

```
int f(string const&);
→
int f(char const*);
```

- Viral
- Add overloads for both.
- strlen
- Breaks COW.

Refactoring std::string Everywhere

```
int f(string const&);
→
int f(string_view);
```

- Viral
- ~~Add overloads for both.~~
- ~~strlen~~
- Breaks COW.

Rethink

- Constants
- Parameters

Building std::string Everywhere

- What do you get when you give C++ to a Java or Python programmer?.

```
string a = ssl ? "https" : "http";
a = a + "://" + path + "/" + query;
```

- At least one extra malloc.
- Profiles found these in droves.

Building std::string Everywhere

- With builders:

```
string a = cat(
  ssl ? "https" : "http",
  "://", path, '/', query);
```

Building std::string Everywhere

- And type safe formatters :

```
string a = fmt("%1://%2/%3",
  ssl ? "https" : "http",
  path, query);
```

Rethink

- Constants
- Parameters
- Builders

Rethink

- Constants
- Parameters
- Builders
- Values

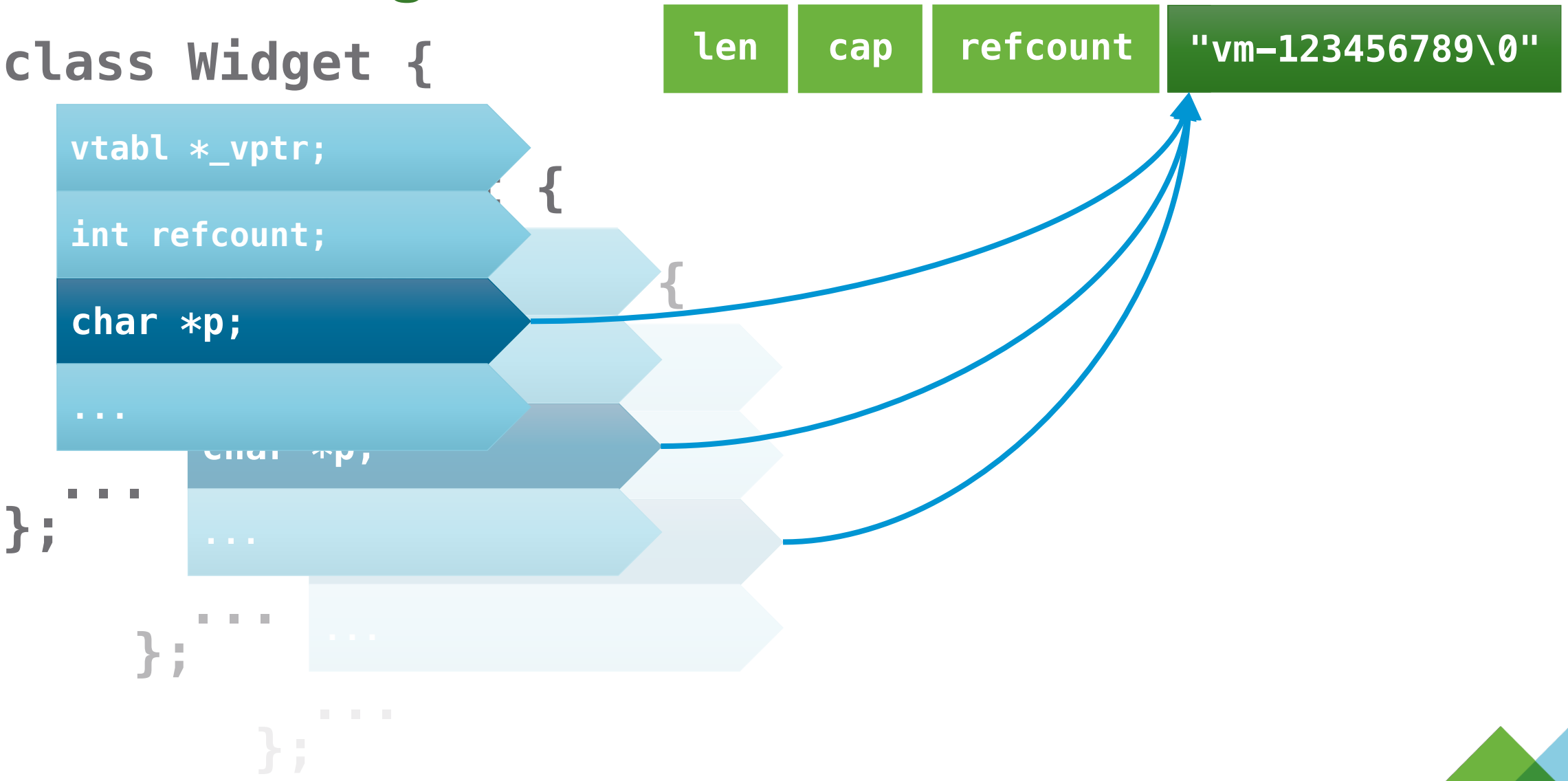
SSO Locality

```
class Widget {
    vtabl *_vptr;
    int refcount;
    size_t size = 12;
    char *end =  0x35343332'312d6d76;   "vm-123456789"
    size_t cap = 0x00000000'39383736;
    ...
...
};
```

Rethink

- Constants
- Parameters
- Builders
- Values
- Locality

COW Sharing


class Widget {
vtabl *_vptr;
int refcount;
char *p;
...
...
};
len
cap
refcount
"vm-123456789\0"

Rethink

- Constants
- Parameters
- Builders
- Values
- Locality
- Sharing

Mutable COW Disease

Bug 21334 - Lack of Posix compliant thread safety in std::basic_string

Status: SUSPENDED

Reported: 2005-05-02 11:45 UTC by James Kanze

. . .

Jonathan Wakely 2015-03-23 13:01:23 UTC **Comment 51**

This is no longer an issue when using the new non-reference-counted std::string implementation in GCC 5.

Rethink

- Constants
- Parameters
- Builders
- Values
- Locality
- Sharing
- Immutability

COW Size

```
class Optional<string> {
    bool isSet =  0x01;
    char* data =  0x00000000'0abcde0;
};
```

COW Size

```
class Optional<string> {
    bool isSet =  0x00;
    char* data =  0x00000000'00000000;
};
```

SSO Size

```
class Optional<string> {
    bool isSet =  0x00;
    size_t size = 0x00000000'00000000;
    char *data =  0x00000000'00000000;
    size_t cap =  0x00000000'00000000;
    void *ext =   0x00000000'00000000;
};
```

SSO Size

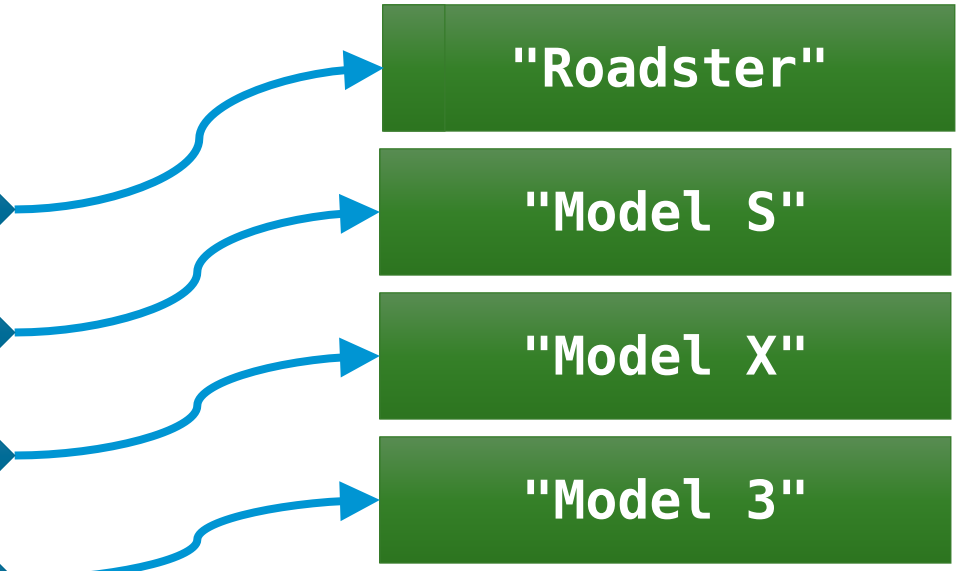
```
class Optional<string> {
    string *p = 0x00000000'00abcde8;
};
```

```
"some unfortunately long str
```

```
size_t size = 0x00000000'00000001e;
char *data =  0x00000000'00bcdef0;
size_t cap =  0x00000000'0000001f;
void *ext =   0x00000000'00000000;
```

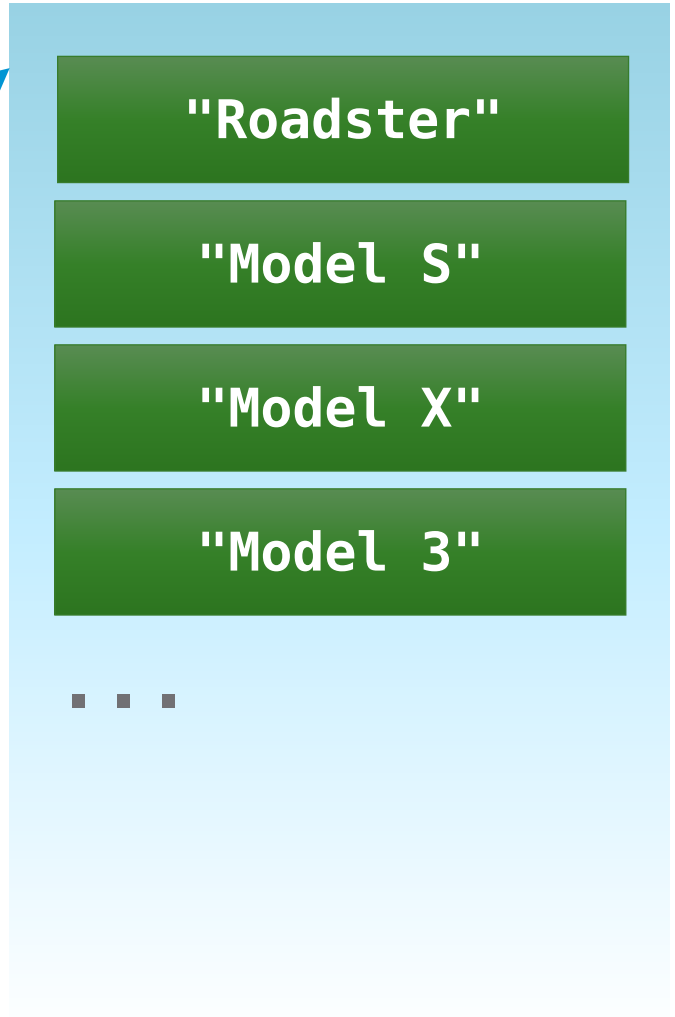
Beyond COW Sharing

```
class Widget {
  const char* s1 = 0x00000000'00abcde8;
  const char* s1 = 0x00000000'00abcdf8;
  const char* s1 = 0x00000000'00abce08;
  const char* s1 = 0x00000000'00abce18;
  ...
};
```



Beyond COW Sharing

```
class Widget {
  static const char* s1 = 0x00000000'00abcde8;
  short s1 = 0x0000;
  short s2 = 0x0001;
  short s3 = 0x0002;
  short s4 = 0x0003;
  ...
};
```



Rethink

- Constants
- Parameters
- Builders
- Values
- Locality
- Sharing
- Immutability
- Size

Optimizing Optional

```
class Optional<string> {
    bool isSet =  0x00; 000'00000000;
    char* data =  0x00000000'00000000;
};
```

Optimizing Optional

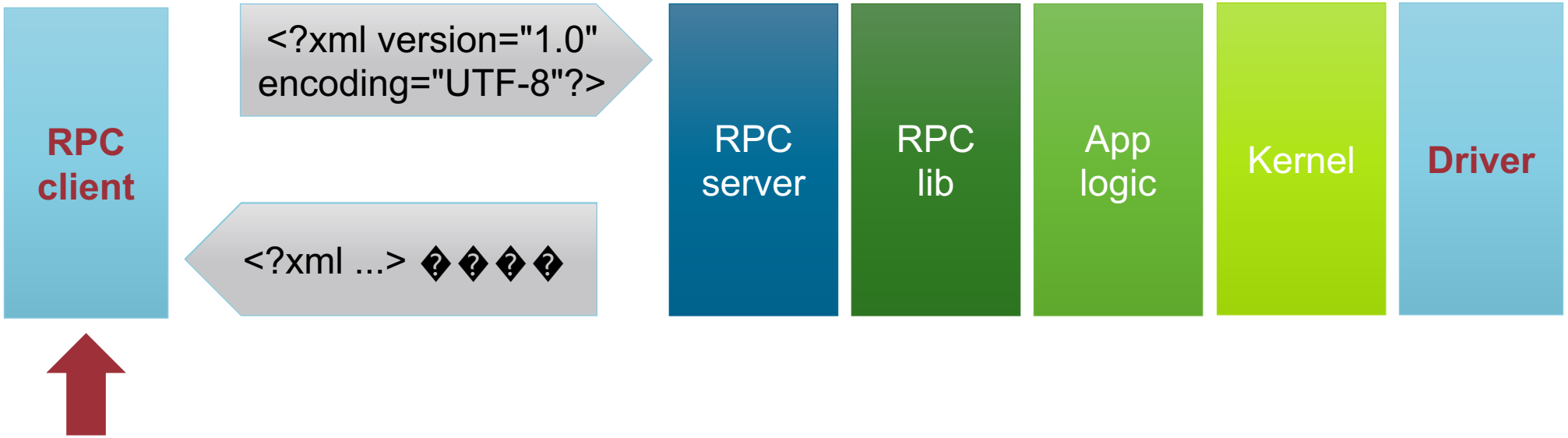
```
class Optional<string> {
    char* data =  0x00000000'00000000;
};
```

- **0 – 1.3**% decrease in used heap bytes

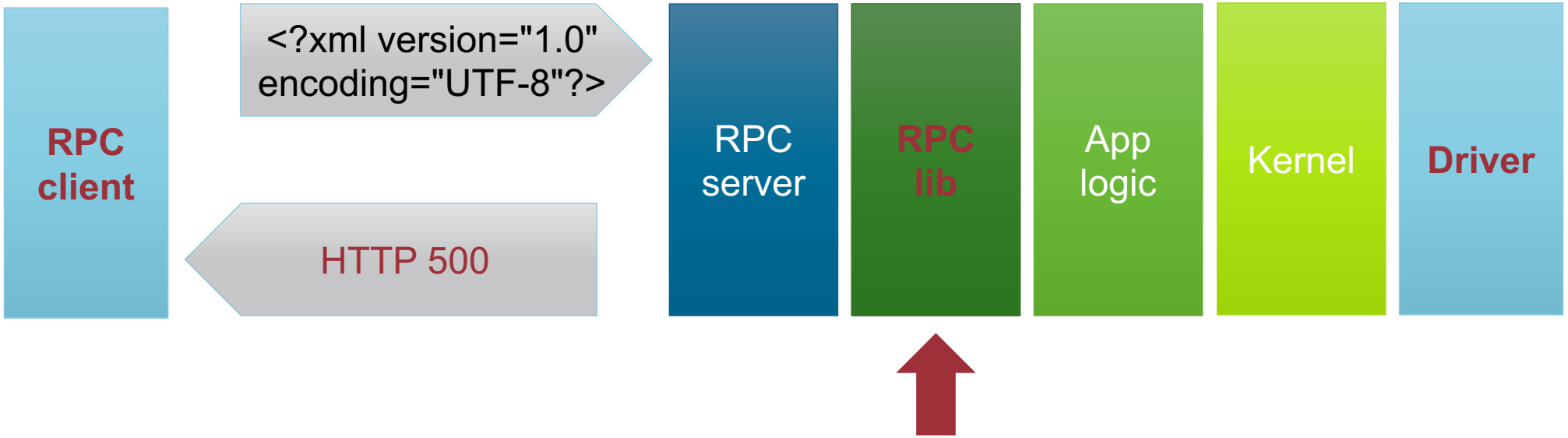
Rethink

- Constants
- Parameters
- Builders
- Values
- Locality
- Sharing
- Immutability
- Nullability

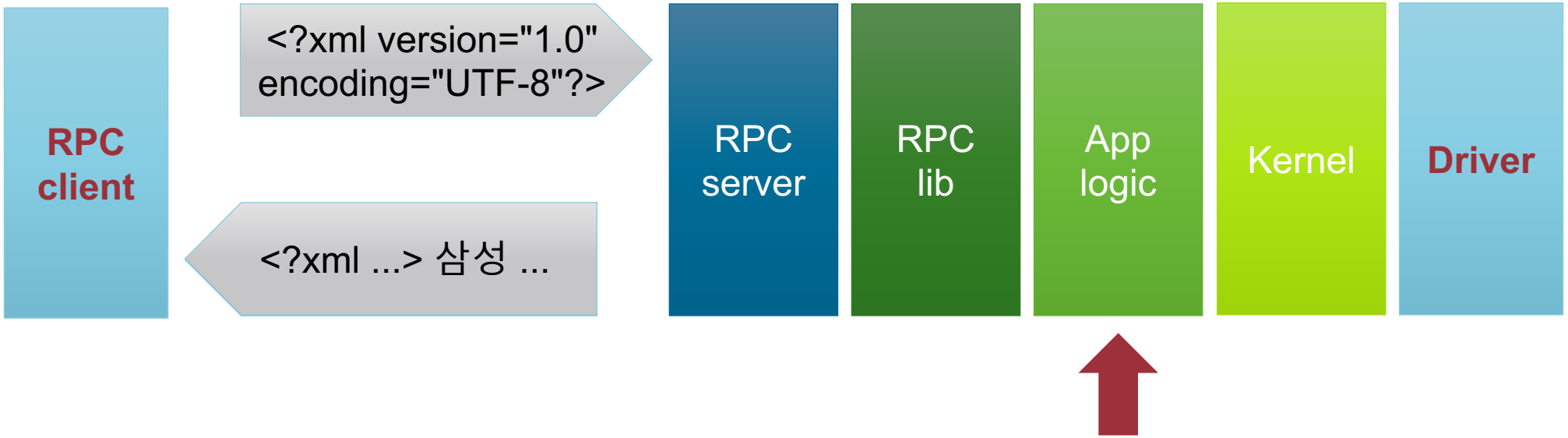
Encoding


RPC client
<?xml version="1.0" encoding="UTF-8"?>
<?xml ...> ????
RPC server
RPC lib
App logic
Kernel
Driver

Encoding


<?xml version="1.0" encoding="UTF-8"?>
RPC client
HTTP 500
RPC server
RPC lib
App logic
Kernel
Driver

Encoding


RPC client
<?xml version="1.0" encoding="UTF-8"?>
<?xml ...> 삼성 ...
RPC server
RPC lib
App logic
Kernel
Driver

Rethink

- Constants
- Parameters
- Builders
- Values
- Locality
- Sharing
- Immutability
- Nullability
- Encoding

1 Inspiration

2 Experience

3 **Rethinking**

4 Code

Intermission

Caveats

- I have as many questions as answers
- If you think I'm wrong,
- You're probably right

Mindset

- Zero cost
- Driven by my workloads and metrics
- 64-bit addressing
- Unicode

Mindset

- Not standards track
- Assume operations (find_first_not_of)
- No new compile time meta programming
- No library unloading (static duration is simple)

Mindset

- Failure is an option
- Default to `string` and `string_view`

Rethink

- Constants
- Parameters
- Builders
- Values

- Locality
- Sharing
- Immutability
- Nullability
- Encoding

Traits

- Data (Encoding)
- Data Size
- Ownership / Mutability
- Storage Duration
- Nullability

Data

`char*`

Data

- One or more **code units**
- Makeup a **code point**
- Which names a **character**

Data

```
code_unit*
```

Code Units

- `char` Execution
- `wchar` Execution wide
- `char16_t` UTF-16
- `char32_t` UTF-32
- `char` UTF-8

Code Units

- **`char`** Execution
- `wchar` Execution wide
- `char16_t` UTF-16
- `char32_t` UTF-32
- **`char`** UTF-8

Code Units (P0482)

- `char` Execution
- `wchar` Execution wide
- `char16_t` UTF-16
- `char32_t` UTF-32
- `char8_t` UTF-8

} × Private Use Areas

P0482R0: [Evolution, Library Evolution] char8_t: A type for UTF-8 characters and strings (by Tom Honermann) (2016-10-17) https://wg21.link/p0482r0

Data

```
code_unit<Encoding>*
```

Code Units

- Why not just `char`? (execution == UTF-8)
- In reality execution == WTF-8*
- `char16_t`? too hard to integrate
- `char8_t`?
- `type_safe`?

*Not Wobbly Text Format https://simonsapin.github.io/wtf-8/

Zero overhead utilities for preventing bugs at compile time http://type_safe.foonathan.net

Code Units

- `char*` for now

Traits

- `data(s) -> encoding::code_unit*`
- `decltype(s)::encoding`
- `decltype(c)::encoding` (wish list)

Data Size

- Signed
- `int` (`int_least32_t`)
- Static
- Dynamic
- Null terminated

P0122R4: span: bounds-safe views for sequences of objects
2017-02-06 Neil MacIntosh
https://wg21.link/p0122r4

Traits

- `size(s) -> int`
- `decltype(s)::size` (dynamic = -1)

Ownership / Mutability

- View
- Shared owner
- Unique owner

} Immutable

Mutable

Mutability Traits

- `decltype(s)::is_mutable_v`

Ownership Transfer

- Unique -> Shared
- Mutable -> Immutable

Storage Duration

- Static `constexpr char*`
- Automatic `char[3]`
- Dynamic `unique_ptr<char[]>` (Table)

Storage Duration Traits

- None?

Null-ability

- Not nullable `string_view`
- Nullable `char*`

Null-ability Traits

- `decltype(s)::is_nullable_v`
- `decltype(s)::write_null(void*)`
- `decltype(s)::is_null(void*)`

1 Inspiration

2 Experience

3 Rethinking

4 **Code**

Rethink

- Constants
- Parameters
- Builders
- **Values**

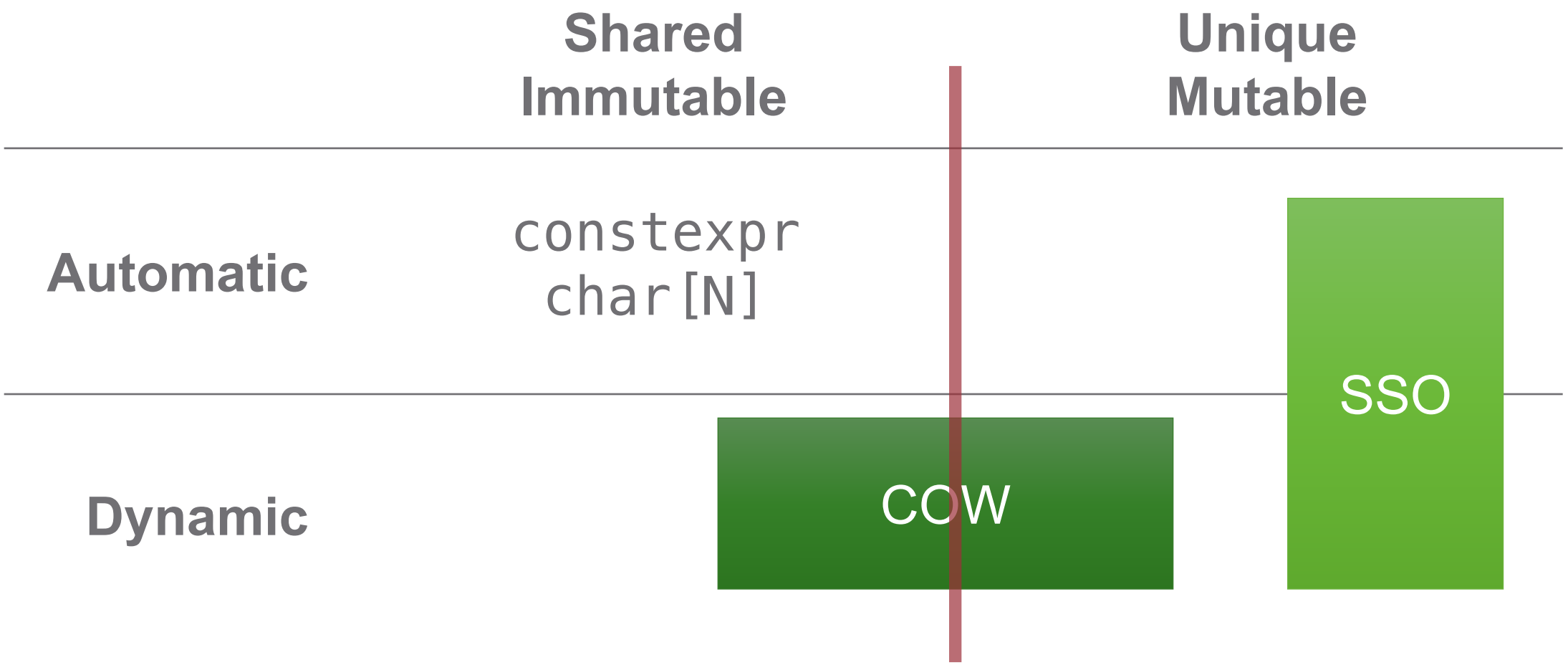
Traits

- Data (Encoding)
- Data Size
- Ownership / Mutability
- Storage Duration
- Nullability

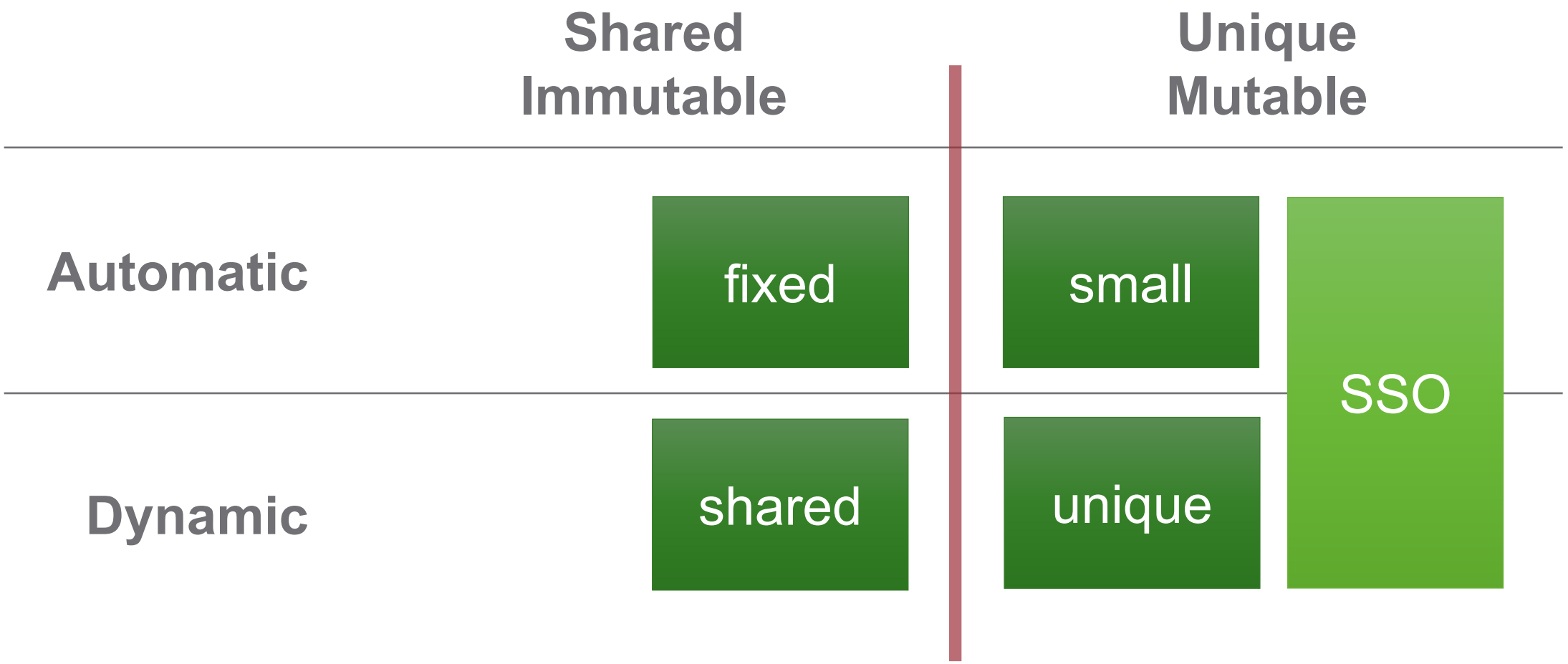
Traits

- Data (Encoding)
- Data Size
- Ownership / **Mutability**
- **Storage Duration**
- Nullability

	Shared Immutable	Unique Mutable
Automatic	`constexpr char[N]`	SSO
Dynamic	FBString	


Shared
Immutable
Unique
Mutable
Automatic
constexpr
char[N]
Dynamic
COW
SSO

	Shared Immutable	Unique Mutable	
Automatic	`constexpr char[N]`	small	SSO
Dynamic	shared	unique	


Shared
Immutable
Unique
Mutable
Automatic
Dynamic
fixed
shared
small
unique
SSO

	Shared Immutable	Unique Mutable	
Automatic	fixed	small	SSO
Dynamic	shared	unique	

https://github.com/vmware/rethinking-strings

Questions?

https://github.com/vmware/rethinking-strings

Mark Zeren
mzeren@vmware.com
cpplang.slack.com