Fast and small C++ - When efficiency matters

Presentation Material

C++Online, online, 2025-02-27

Version: v1.0

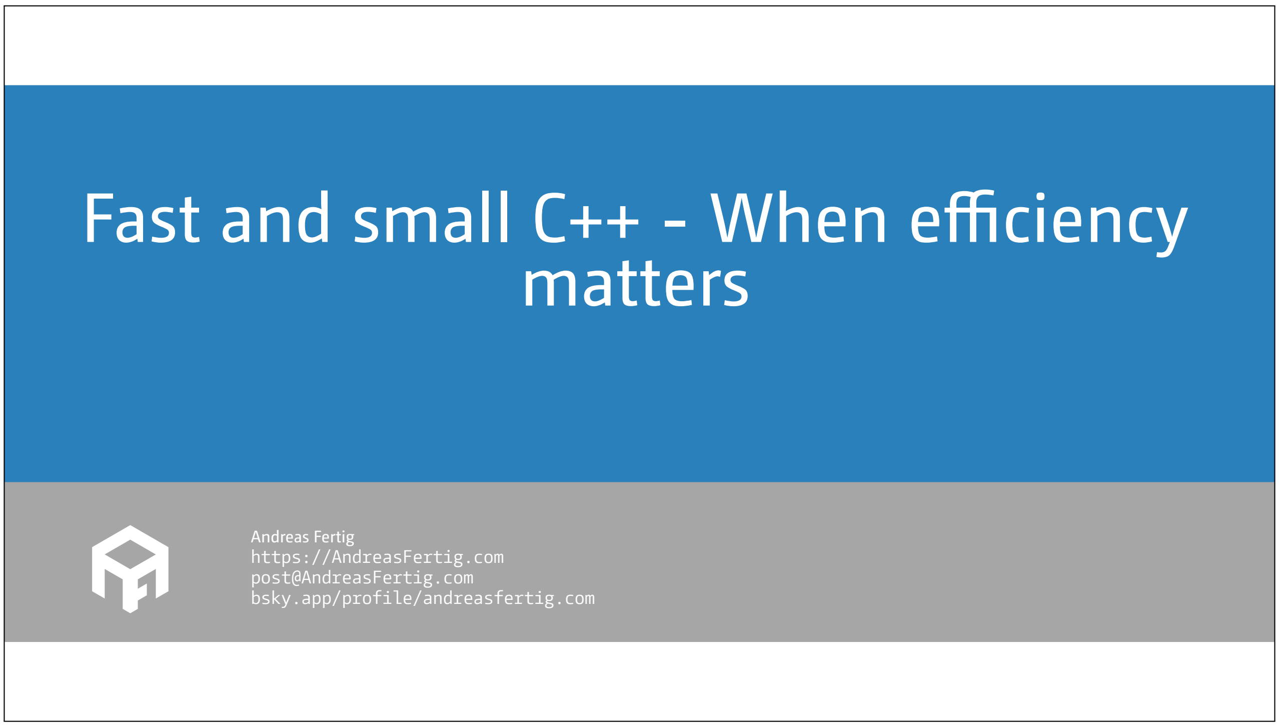
Planning, typesetting and cover design: Andreas Fertig
Cover art and illustrations: Franziska Panter https://franziskapanter.com
Production and publishing: Andreas Fertig

Style and conventions

The following shows the execution of a program. I used the Linux way here and skipped supplying the desired output name, resulting in `a.out` as the program name.

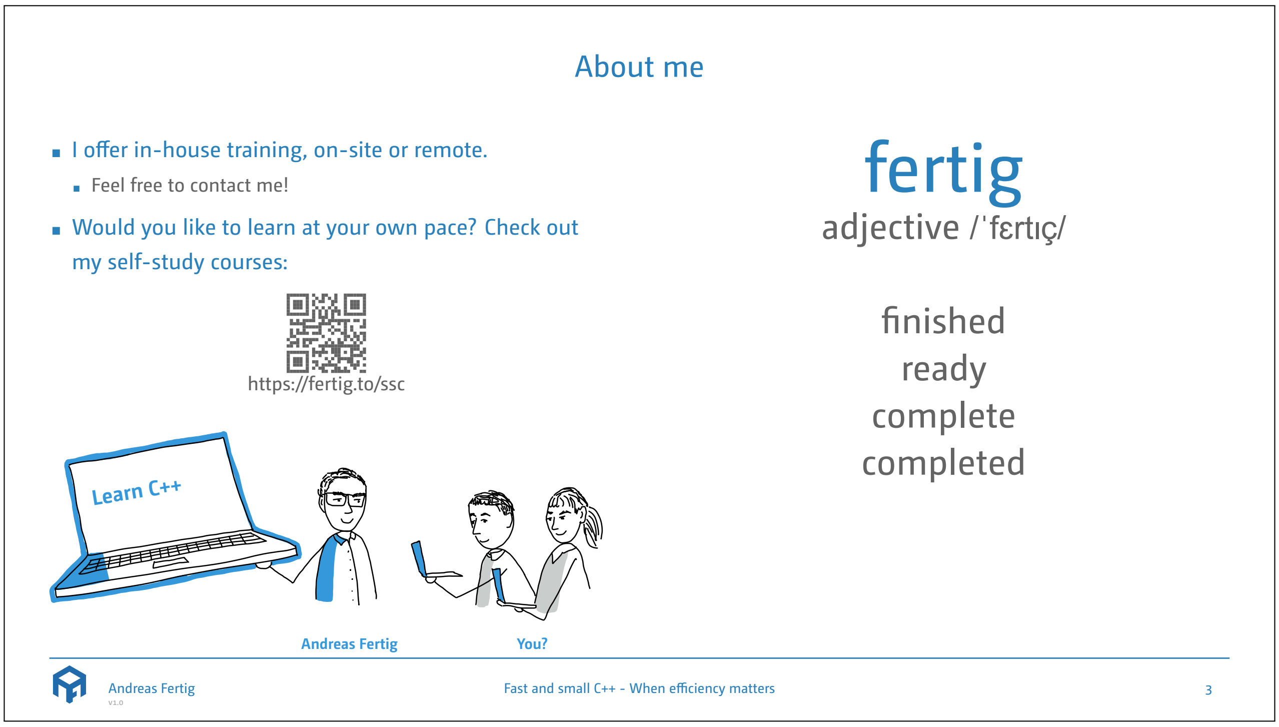
```
$ ./a.out
Hello, C++!
```

- `<string>` stands for a header file with the name `string`
- `[[xyz]]` marks a C++ attribute with the name `xyz`.



Fast and small C++ - When efficiency matters

Andreas Fertig
https://AndreasFertig.com
post@AndreasFertig.com
bsky.app/profile/andreasfertig.com



About me

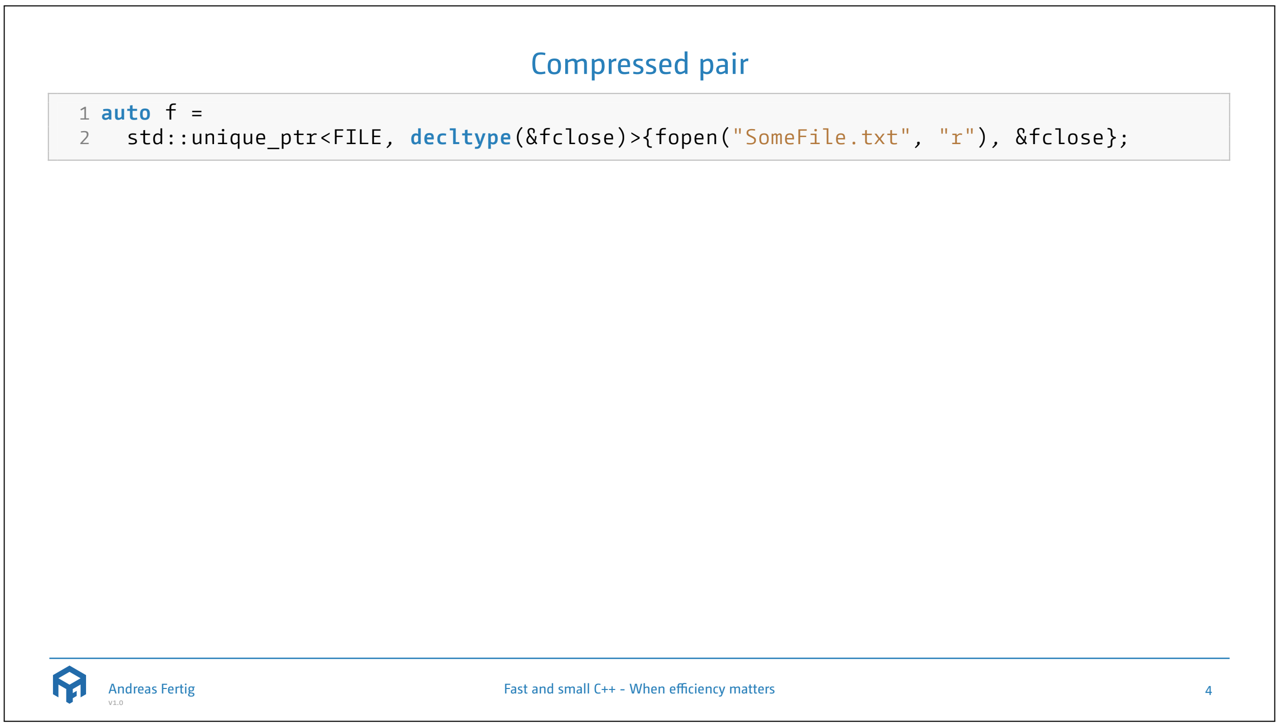
- I offer in-house training, on-site or remote.
  - Feel free to contact me!
- Would you like to learn at your own pace? Check out my self-study courses:

https://fertig.to/ssc

Andreas Fertig You?

fertig
adjective /ˈfɛrtɪç/

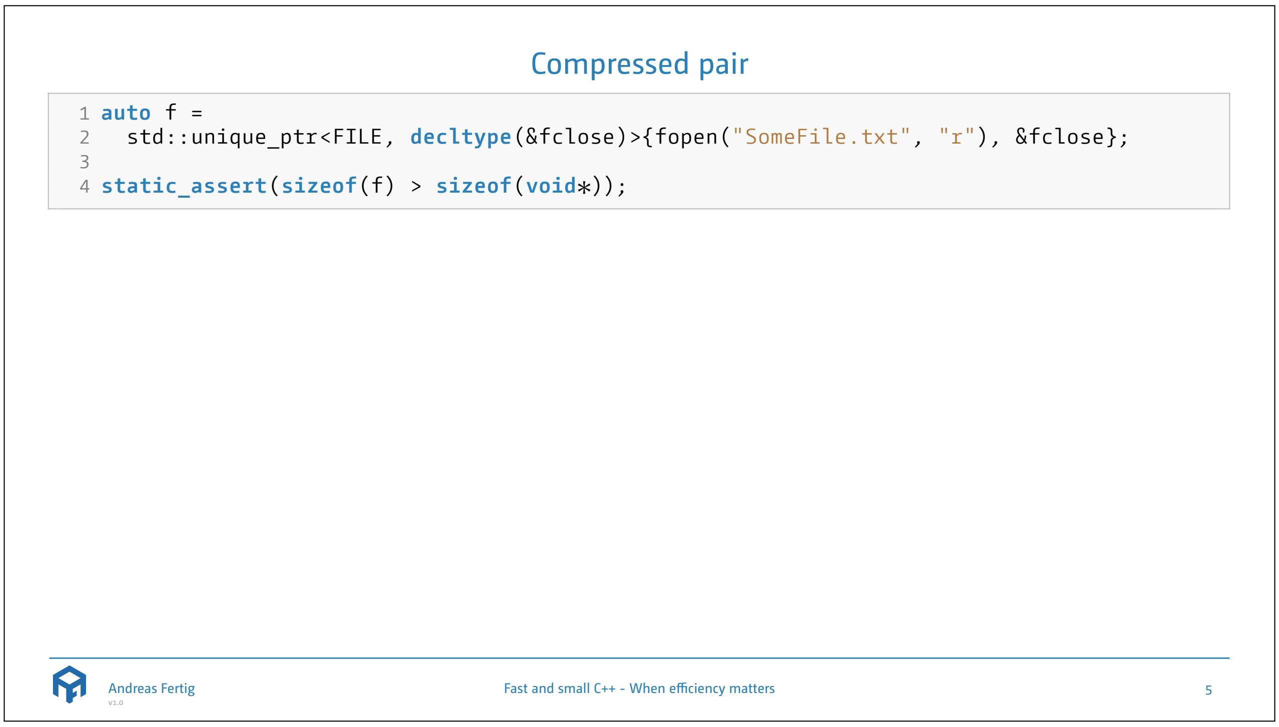
finished
ready
complete
completed



Compressed pair

```cpp
auto f =
  std::unique_ptr<FILE, decltype(&fclose)>{fopen("SomeFile.txt", "r"), &fclose};
```

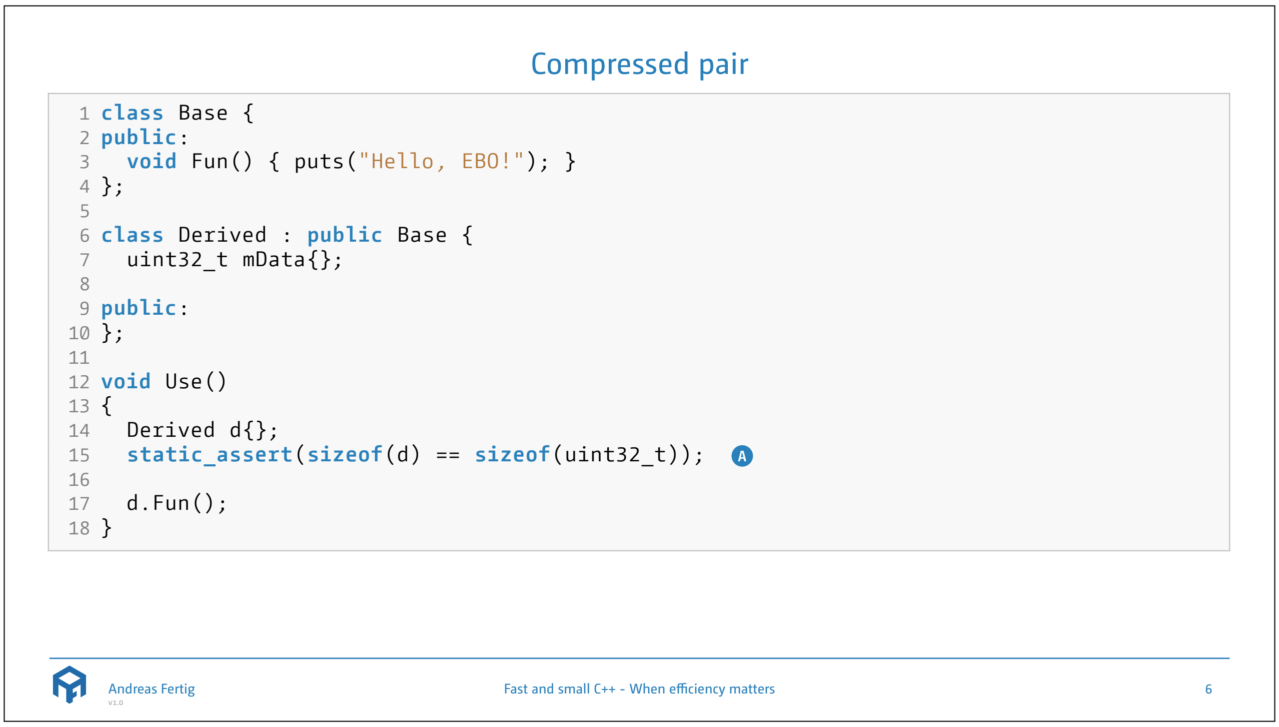
Andreas Fertig v1.0 Fast and small C++ - When efficiency matters 4



Compressed pair

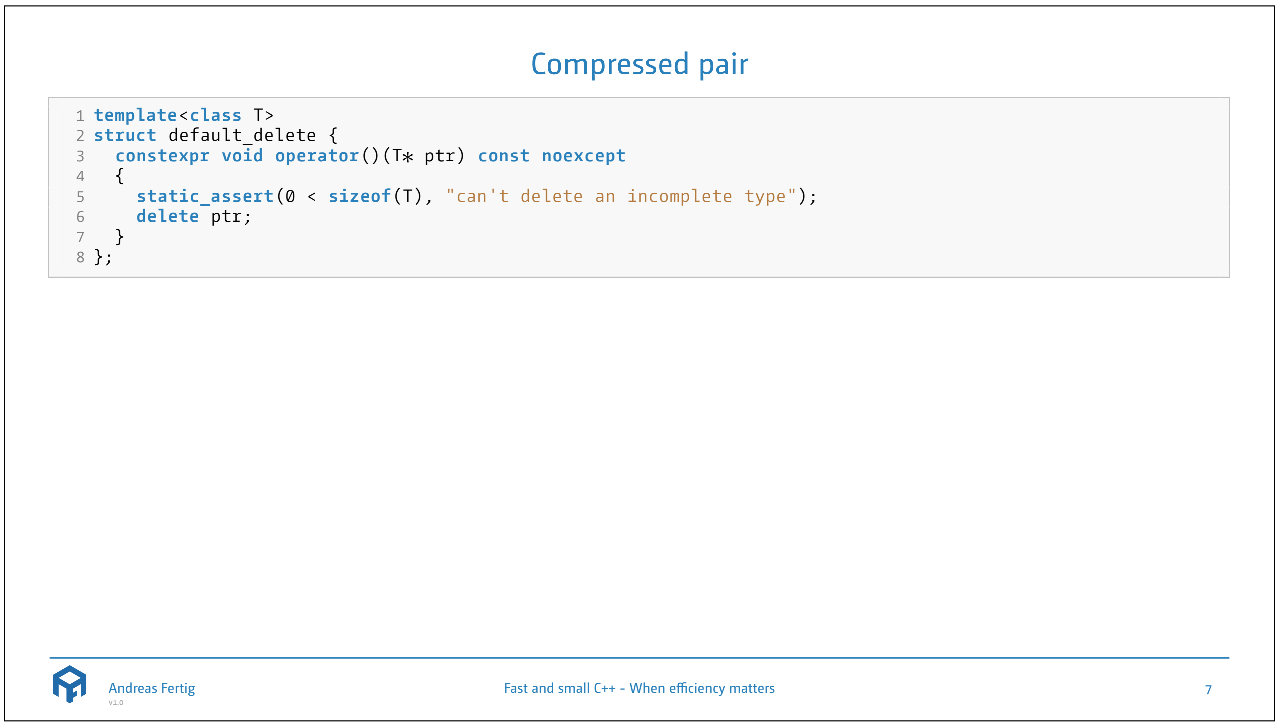
```cpp
auto f =
  std::unique_ptr<FILE, decltype(&fclose)>{fopen("SomeFile.txt", "r"), &fclose};

static_assert(sizeof(f) > sizeof(void*));
```

Andreas Fertig v1.0 Fast and small C++ - When efficiency matters 5



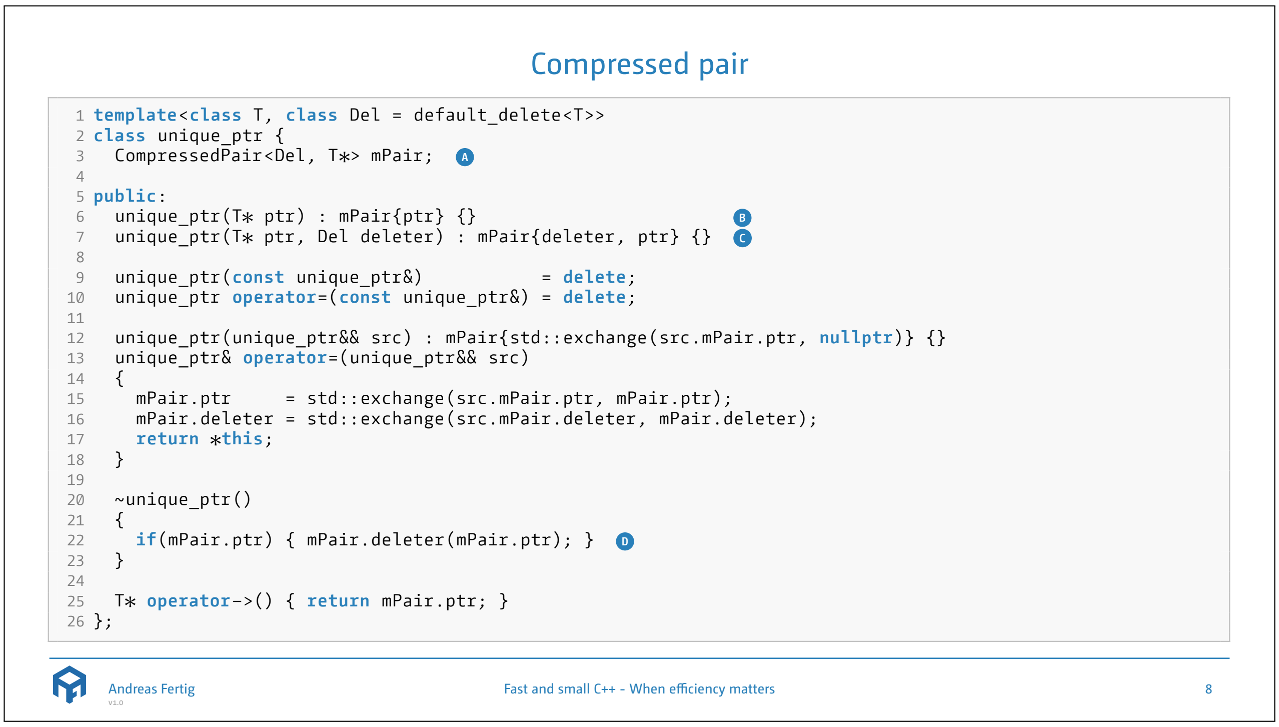
Compressed pair

```cpp
class Base {
public:
  void Fun() { puts("Hello, EBO!"); }
};

class Derived : public Base {
  uint32_t mData{};

public:
};

void Use()
{
  Derived d{};
  static_assert(sizeof(d) == sizeof(uint32_t));   A

  d.Fun();
}
```



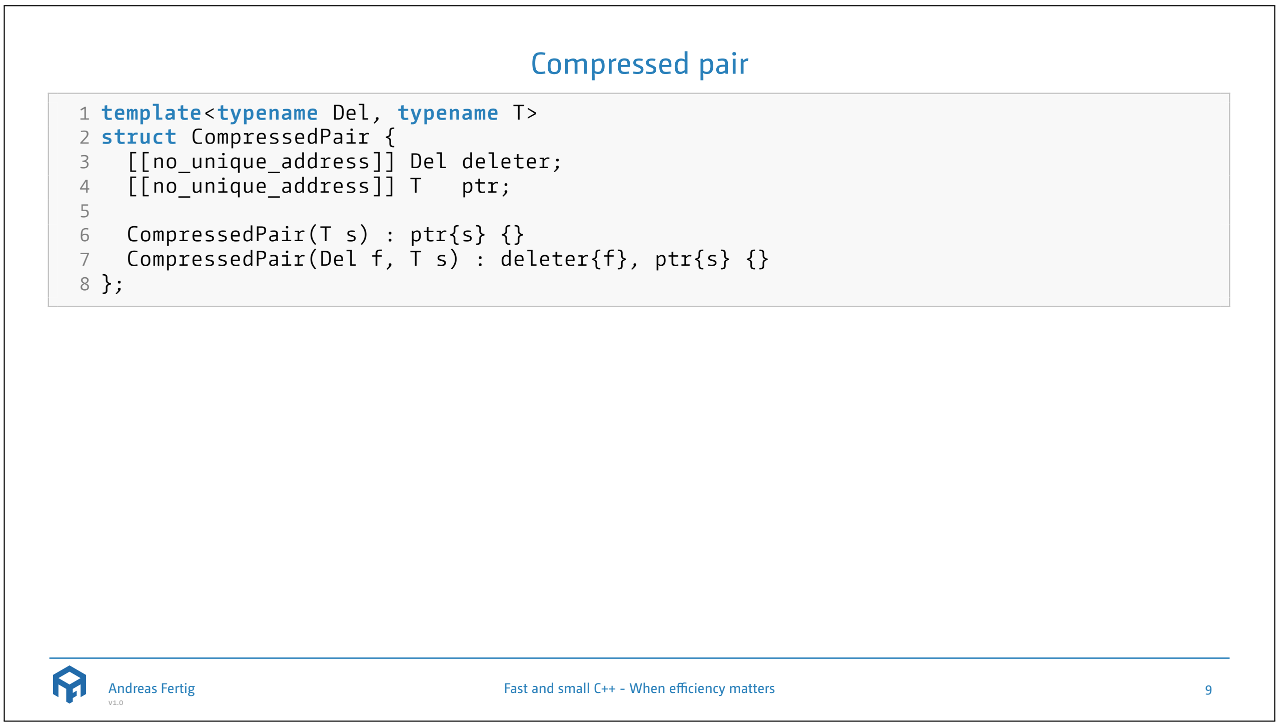
Compressed pair

```cpp
template<class T>
struct default_delete {
  constexpr void operator()(T* ptr) const noexcept
  {
    static_assert(0 < sizeof(T), "can't delete an incomplete type");
    delete ptr;
  }
};
```



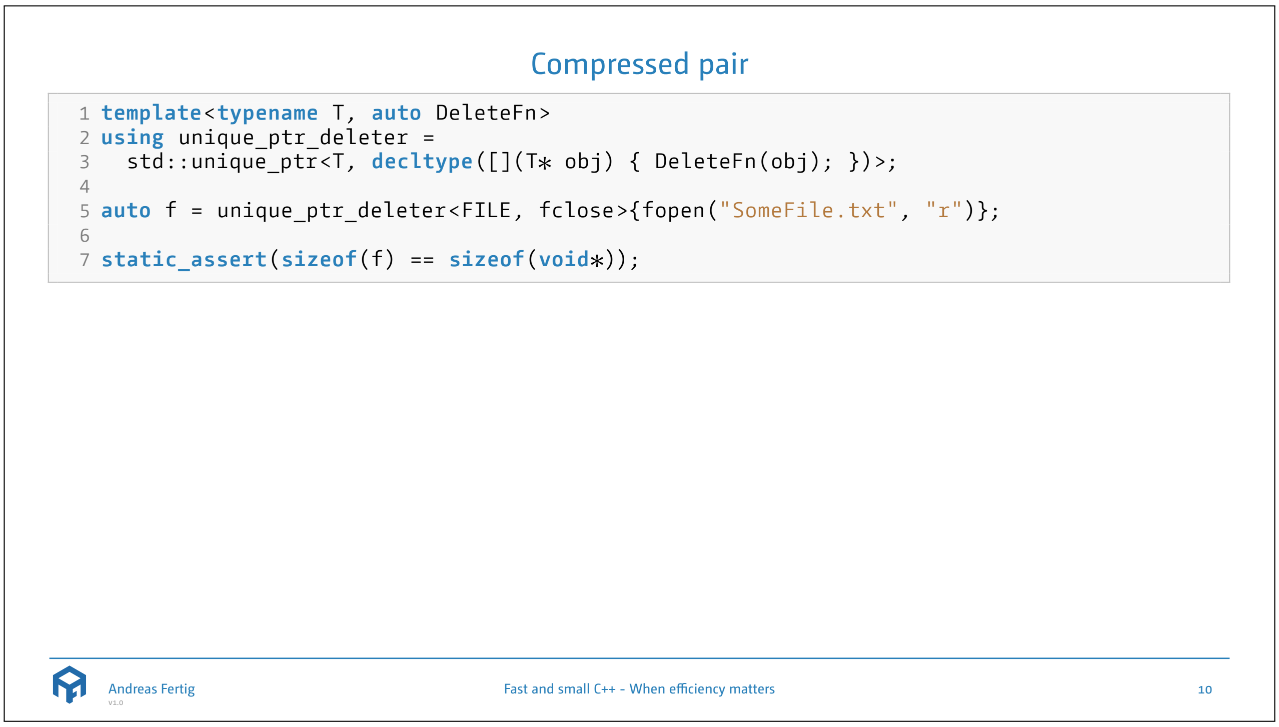
Compressed pair

```cpp
template<class T, class Del = default_delete<T>>
class unique_ptr {
  CompressedPair<Del, T*> mPair;  A

public:
  unique_ptr(T* ptr) : mPair{ptr} {}                          B
  unique_ptr(T* ptr, Del deleter) : mPair{deleter, ptr} {}   C

  unique_ptr(const unique_ptr&)           = delete;
  unique_ptr operator=(const unique_ptr&) = delete;

  unique_ptr(unique_ptr&& src) : mPair{std::exchange(src.mPair.ptr, nullptr)} {}
  unique_ptr& operator=(unique_ptr&& src)
  {
    mPair.ptr     = std::exchange(src.mPair.ptr, mPair.ptr);
    mPair.deleter = std::exchange(src.mPair.deleter, mPair.deleter);
    return *this;
  }

  ~unique_ptr()
  {
    if(mPair.ptr) { mPair.deleter(mPair.ptr); }  D
  }

  T* operator->() { return mPair.ptr; }
};
```



Compressed pair

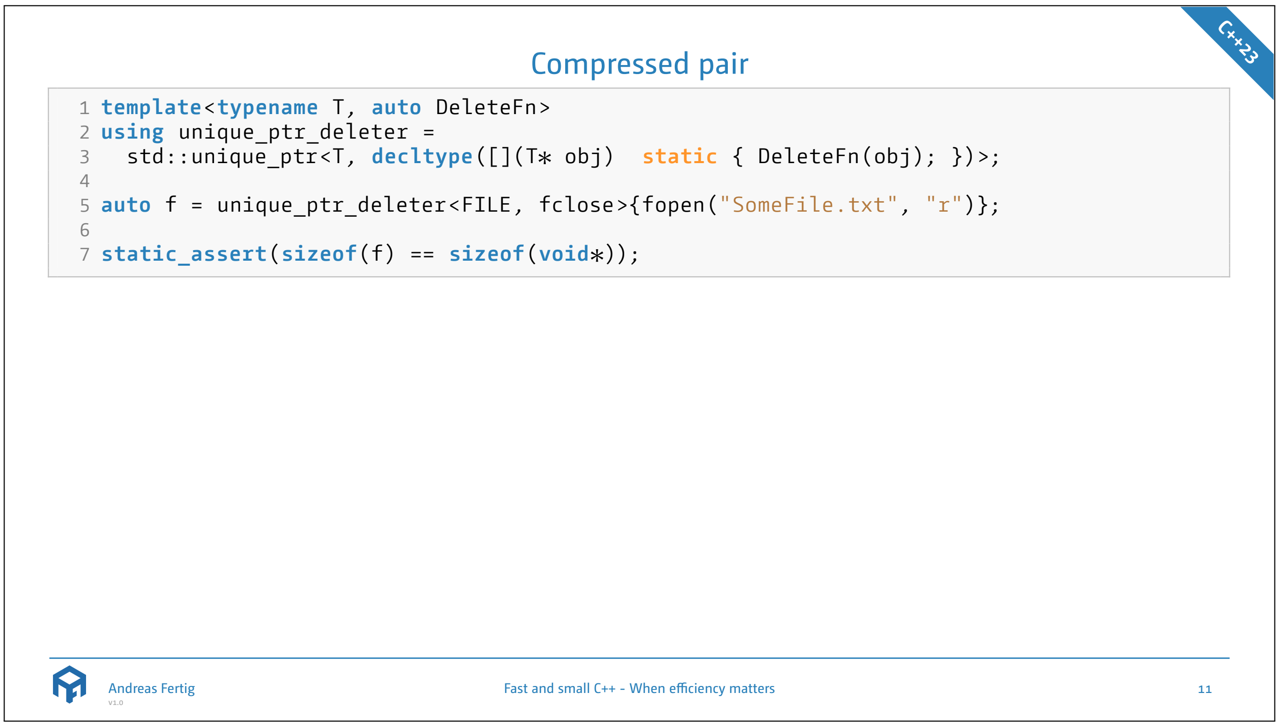
```cpp
template<typename Del, typename T>
struct CompressedPair {
  [[no_unique_address]] Del deleter;
  [[no_unique_address]] T   ptr;

  CompressedPair(T s) : ptr{s} {}
  CompressedPair(Del f, T s) : deleter{f}, ptr{s} {}
};
```



Compressed pair

```cpp
template<typename T, auto DeleteFn>
using unique_ptr_deleter =
  std::unique_ptr<T, decltype([](T* obj) { DeleteFn(obj); })>;

auto f = unique_ptr_deleter<FILE, fclose>{fopen("SomeFile.txt", "r")};

static_assert(sizeof(f) == sizeof(void*));
```

Andreas Fertig v1.0 Fast and small C++ - When efficiency matters 10

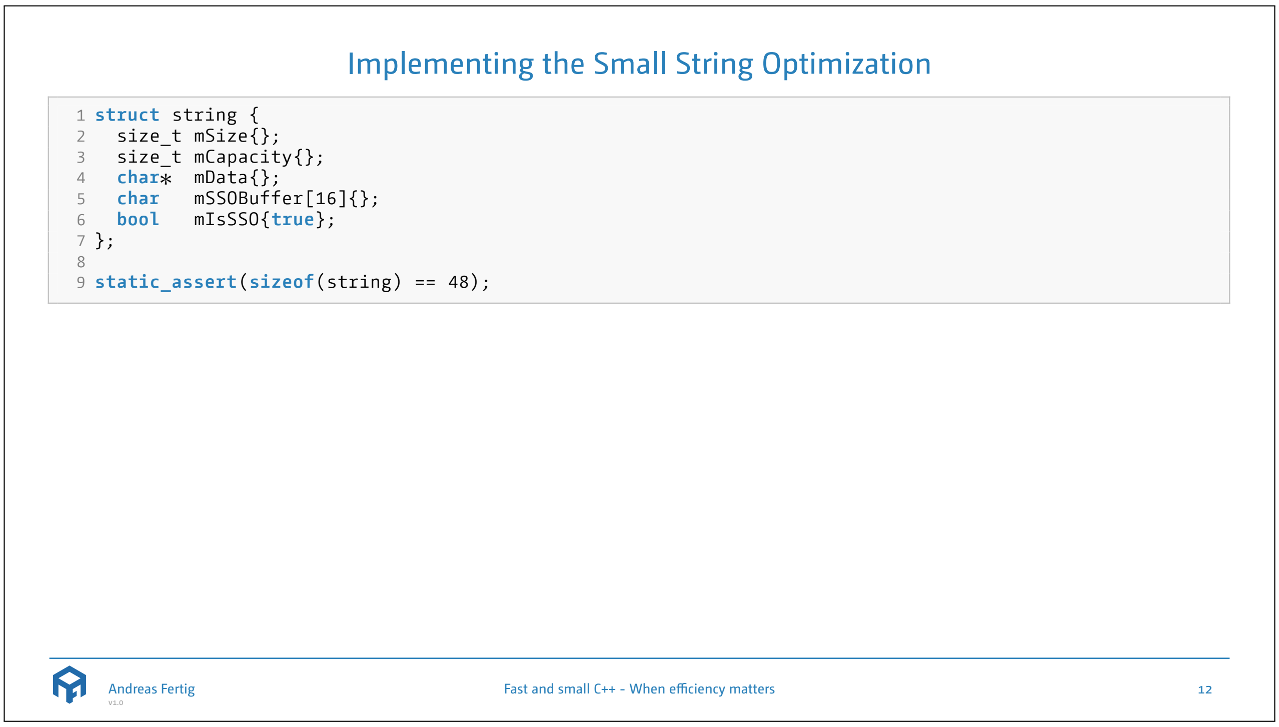


C++23

Compressed pair

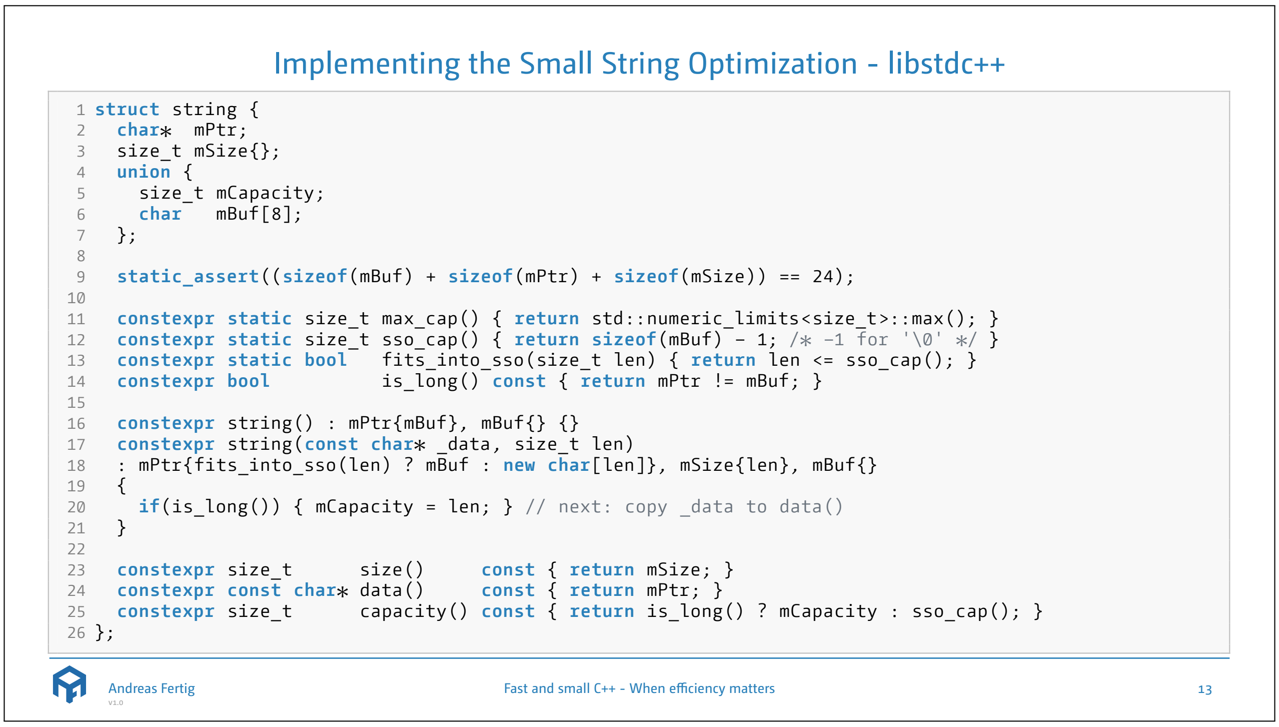
```cpp
template<typename T, auto DeleteFn>
using unique_ptr_deleter =
  std::unique_ptr<T, decltype([](T* obj)  static { DeleteFn(obj); })>;

auto f = unique_ptr_deleter<FILE, fclose>{fopen("SomeFile.txt", "r")};

static_assert(sizeof(f) == sizeof(void*));
```

Andreas Fertig v1.0 Fast and small C++ - When efficiency matters 11



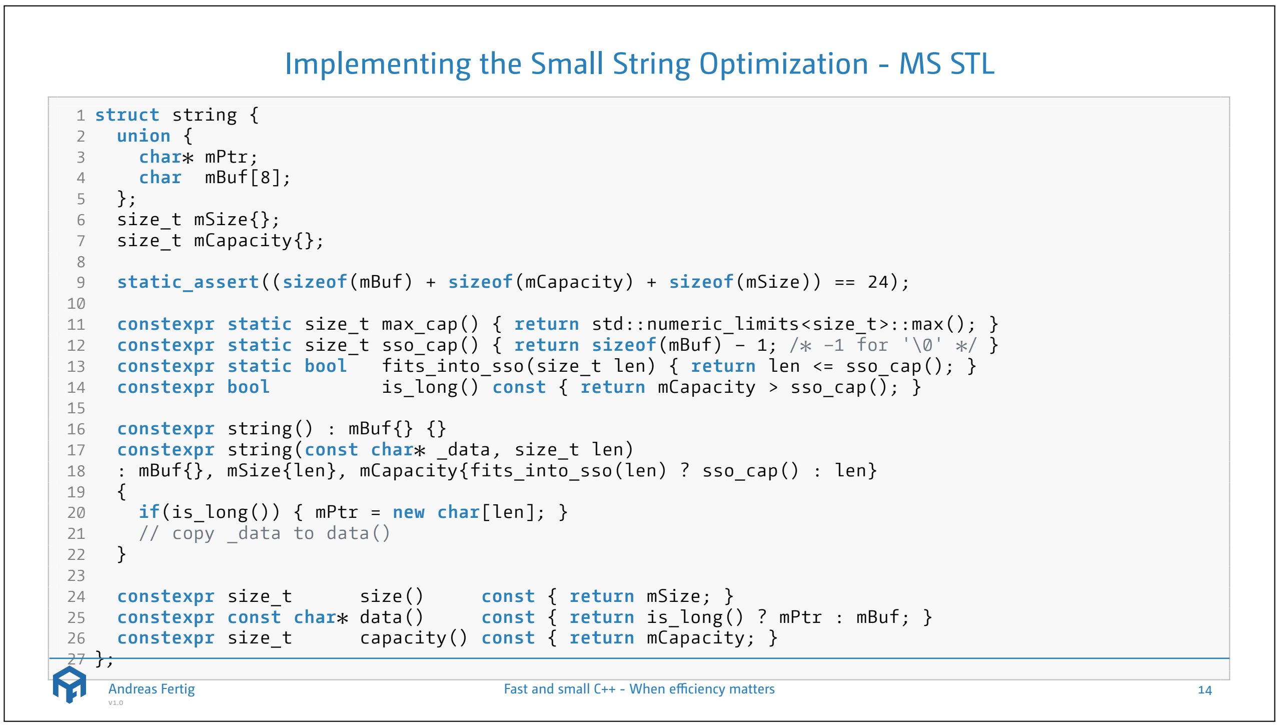
Implementing the Small String Optimization

```cpp
struct string {
  size_t mSize{};
  size_t mCapacity{};
  char*  mData{};
  char   mSSOBuffer[16]{};
  bool   mIsSSO{true};
};

static_assert(sizeof(string) == 48);
```



Implementing the Small String Optimization - libstdc++

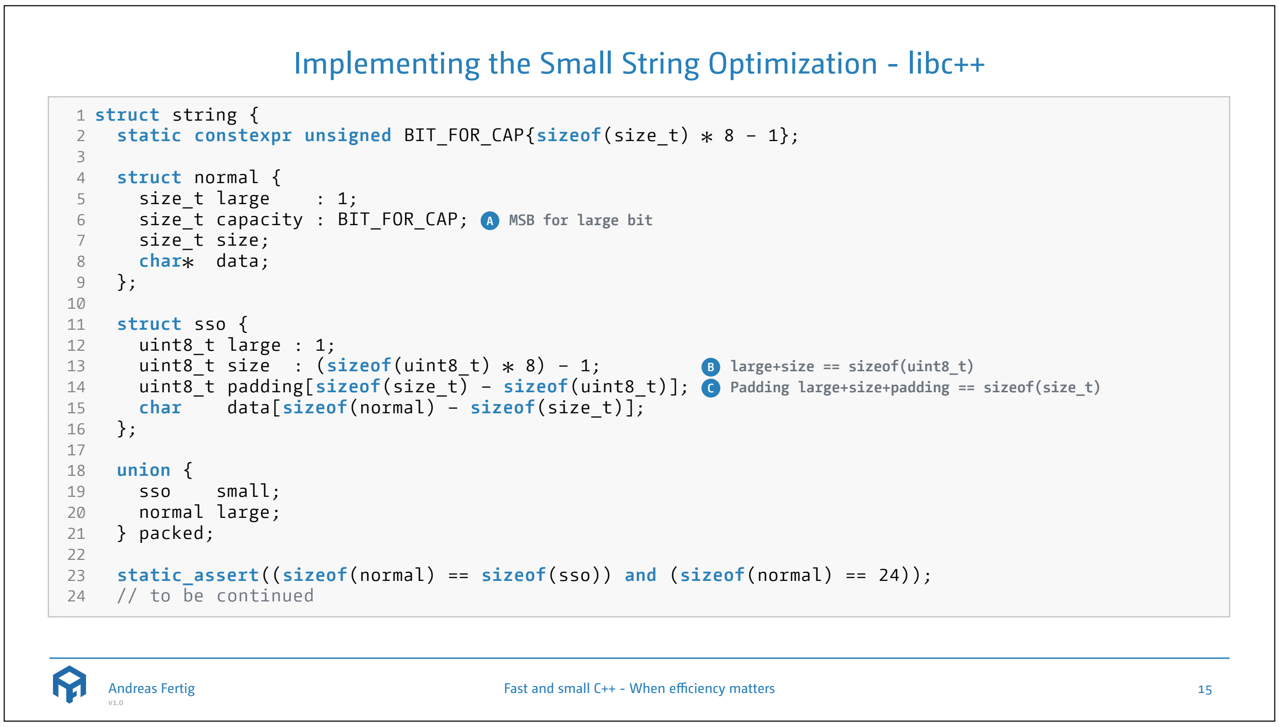
```cpp
struct string {
  char*  mPtr;
  size_t mSize{};
  union {
    size_t mCapacity;
    char   mBuf[8];
  };

  static_assert((sizeof(mBuf) + sizeof(mPtr) + sizeof(mSize)) == 24);

  constexpr static size_t max_cap() { return std::numeric_limits<size_t>::max(); }
  constexpr static size_t sso_cap() { return sizeof(mBuf) - 1; /* -1 for '\0' */ }
  constexpr static bool   fits_into_sso(size_t len) { return len <= sso_cap(); }
  constexpr bool          is_long() const { return mPtr != mBuf; }

  constexpr string() : mPtr{mBuf}, mBuf{} {}
  constexpr string(const char* _data, size_t len)
  : mPtr{fits_into_sso(len) ? mBuf : new char[len]}, mSize{len}, mBuf{}
  {
    if(is_long()) { mCapacity = len; } // next: copy _data to data()
  }

  constexpr size_t      size()     const { return mSize; }
  constexpr const char* data()     const { return mPtr; }
  constexpr size_t      capacity() const { return is_long() ? mCapacity : sso_cap(); }
};
```



Implementing the Small String Optimization - MS STL

```cpp
struct string {
  union {
    char* mPtr;
    char  mBuf[8];
  };
  size_t mSize{};
  size_t mCapacity{};

  static_assert((sizeof(mBuf) + sizeof(mCapacity) + sizeof(mSize)) == 24);

  constexpr static size_t max_cap() { return std::numeric_limits<size_t>::max(); }
  constexpr static size_t sso_cap() { return sizeof(mBuf) - 1; /* -1 for '\0' */ }
  constexpr static bool   fits_into_sso(size_t len) { return len <= sso_cap(); }
  constexpr bool          is_long() const { return mCapacity > sso_cap(); }

  constexpr string() : mBuf{} {}
  constexpr string(const char* _data, size_t len)
  : mBuf{}, mSize{len}, mCapacity{fits_into_sso(len) ? sso_cap() : len}
  {
    if(is_long()) { mPtr = new char[len]; }
    // copy _data to data()
  }

  constexpr size_t      size()     const { return mSize; }
  constexpr const char* data()     const { return is_long() ? mPtr : mBuf; }
  constexpr size_t      capacity() const { return mCapacity; }
};
```

Andreas Fertig v1.0 Fast and small C++ - When efficiency matters 14



Implementing the Small String Optimization - libc++

```cpp
struct string {
  static constexpr unsigned BIT_FOR_CAP{sizeof(size_t) * 8 - 1};

  struct normal {
    size_t large    : 1;
    size_t capacity : BIT_FOR_CAP; (A) MSB for large bit
    size_t size;
    char*  data;
  };

  struct sso {
    uint8_t large : 1;
    uint8_t size  : (sizeof(uint8_t) * 8) - 1;           (B) large+size == sizeof(uint8_t)
    uint8_t padding[sizeof(size_t) - sizeof(uint8_t)];  (C) Padding large+size+padding == sizeof(size_t)
    char    data[sizeof(normal) - sizeof(size_t)];
  };

  union {
    sso    small;
    normal large;
  } packed;

  static_assert((sizeof(normal) == sizeof(sso)) and (sizeof(normal) == 24));
  // to be continued
```

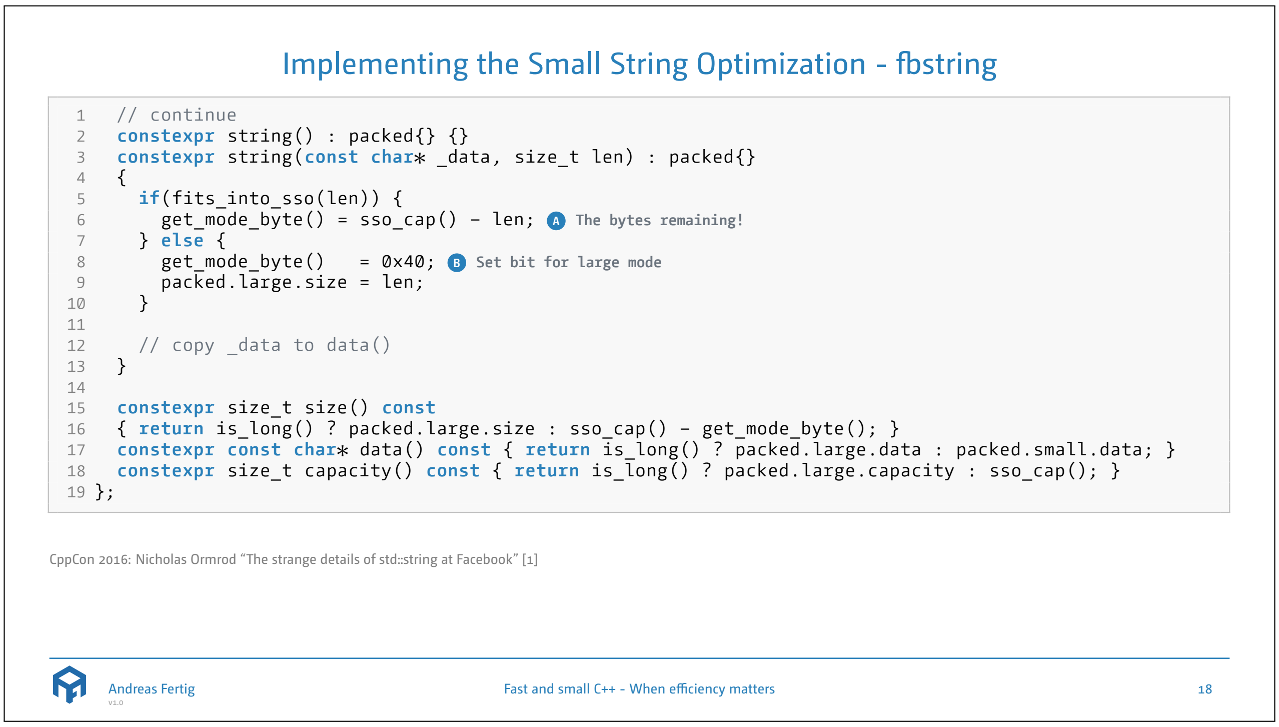
Andreas Fertig v1.0 Fast and small C++ - When efficiency matters 15

Implementing the Small String Optimization - libc++

```cpp
  // continue
  static size_t           max_cap() { return std::pow(2, BIT_FOR_CAP); }
  constexpr static size_t sso_cap() { return sizeof(sso::data) - 1; /* -1 for '\0' */ }
  constexpr static bool   fits_into_sso(size_t len) { return len <= sso_cap(); }
  constexpr bool          is_long() const { return packed.small.large; }

  constexpr string() : packed{} {}
  constexpr string(const char* _data, size_t len) : packed{}
  {
    if(fits_into_sso(len)) {
      packed.small.size = len;
    } else {
      packed.large.large = true;
      packed.large.size  = len;
    }

    // copy _data to data()
  }

  constexpr size_t      size() const { return is_long() ? packed.large.size : packed.small.size; }
  constexpr const char* data() const { return is_long() ? packed.large.data : packed.small.data; }
  constexpr size_t      capacity() const { return is_long() ? packed.large.capacity : sso_cap(); }
};
```

Implementing the Small String Optimization - fbstring

```cpp
struct string {
  struct normal {
    char*  data;
    size_t size;
    size_t capacity;  // virtually reduced by one byte
  };

  struct sso {
    char data[sizeof(normal)];  // MSB for long string mode indicator
  };

  union {
    sso    small;
    normal large;
  } packed;

  static_assert((sizeof(normal) == sizeof(sso)) and (sizeof(normal) == 24));

  static size_t           max_cap() { return std::pow(2, sizeof(normal::capacity) * 8 - 8); }
  constexpr static size_t sso_cap() { return sizeof(sso) - 1; }
  constexpr char&         get_mode_byte() { return packed.small.data[sizeof(normal) - 1]; }
  constexpr char          get_mode_byte() const { return packed.small.data[sizeof(normal) - 1]; }
  constexpr static bool   fits_into_sso(size_t len) { return len <= sso_cap(); }
  constexpr bool          is_long() const { return get_mode_byte() & 0x40; /* 0b100'0000 */ }
  // to be continued
```

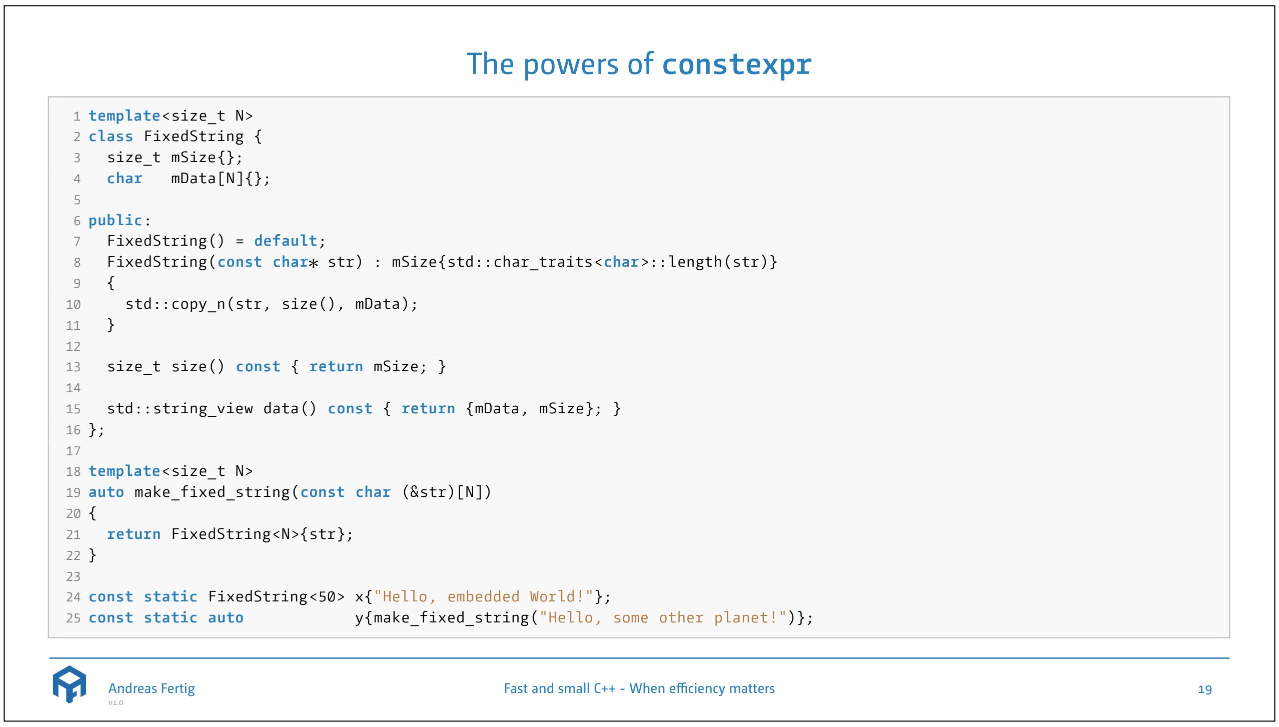


Implementing the Small String Optimization - fbstring

```
  // continue
  constexpr string() : packed{} {}
  constexpr string(const char* _data, size_t len) : packed{}
  {
    if(fits_into_sso(len)) {
      get_mode_byte() = sso_cap() - len; A The bytes remaining!
    } else {
      get_mode_byte()    = 0x40; B Set bit for large mode
      packed.large.size = len;
    }

    // copy _data to data()
  }

  constexpr size_t size() const
  { return is_long() ? packed.large.size : sso_cap() - get_mode_byte(); }
  constexpr const char* data() const { return is_long() ? packed.large.data : packed.small.data; }
  constexpr size_t capacity() const { return is_long() ? packed.large.capacity : sso_cap(); }
};
```

CppCon 2016: Nicholas Ormrod “The strange details of std::string at Facebook” [1]

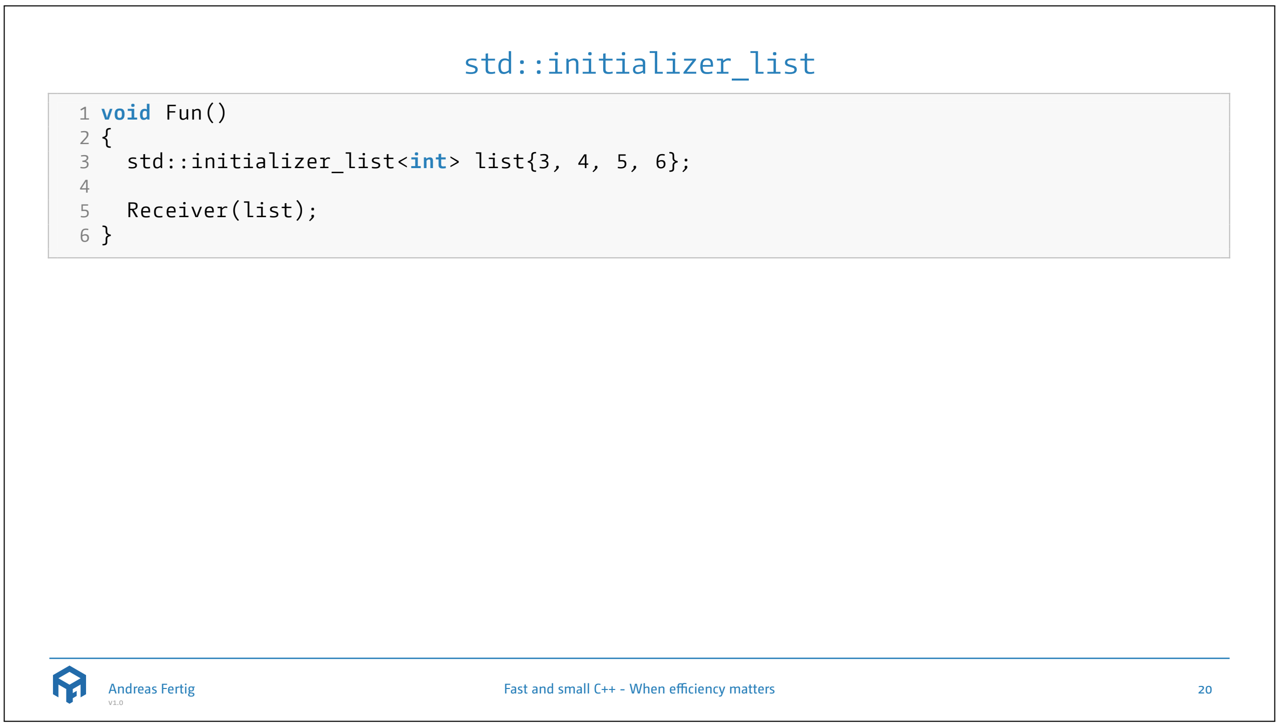
Andreas Fertig v1.0 Fast and small C++ - When efficiency matters 18



The powers of `constexpr`

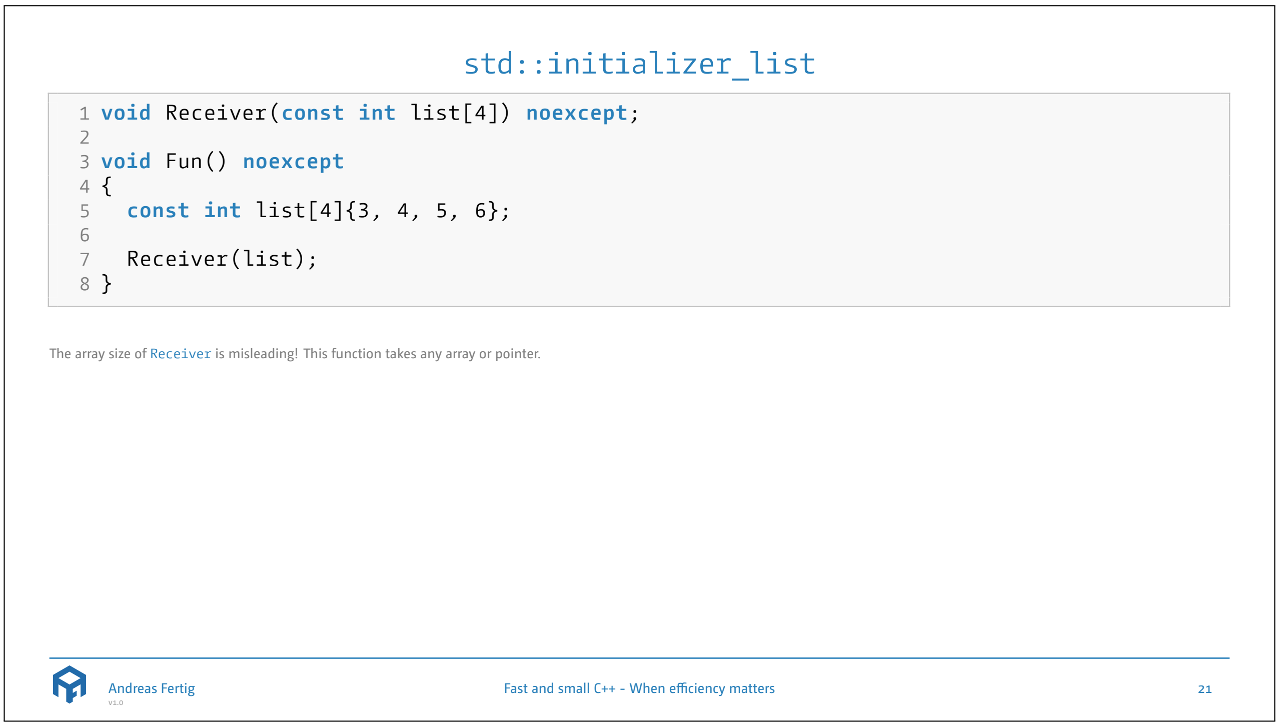
```
template<size_t N>
class FixedString {
  size_t mSize{};
  char   mData[N]{};

public:
  FixedString() = default;
  FixedString(const char* str) : mSize{std::char_traits<char>::length(str)}
  {
    std::copy_n(str, size(), mData);
  }

  size_t size() const { return mSize; }

  std::string_view data() const { return {mData, mSize}; }
};

template<size_t N>
auto make_fixed_string(const char (&str)[N])
{
  return FixedString<N>{str};
}

const static FixedString<50> x{"Hello, embedded World!"};
const static auto            y{make_fixed_string("Hello, some other planet!")};
```

Andreas Fertig v1.0 Fast and small C++ - When efficiency matters 19



std::initializer_list

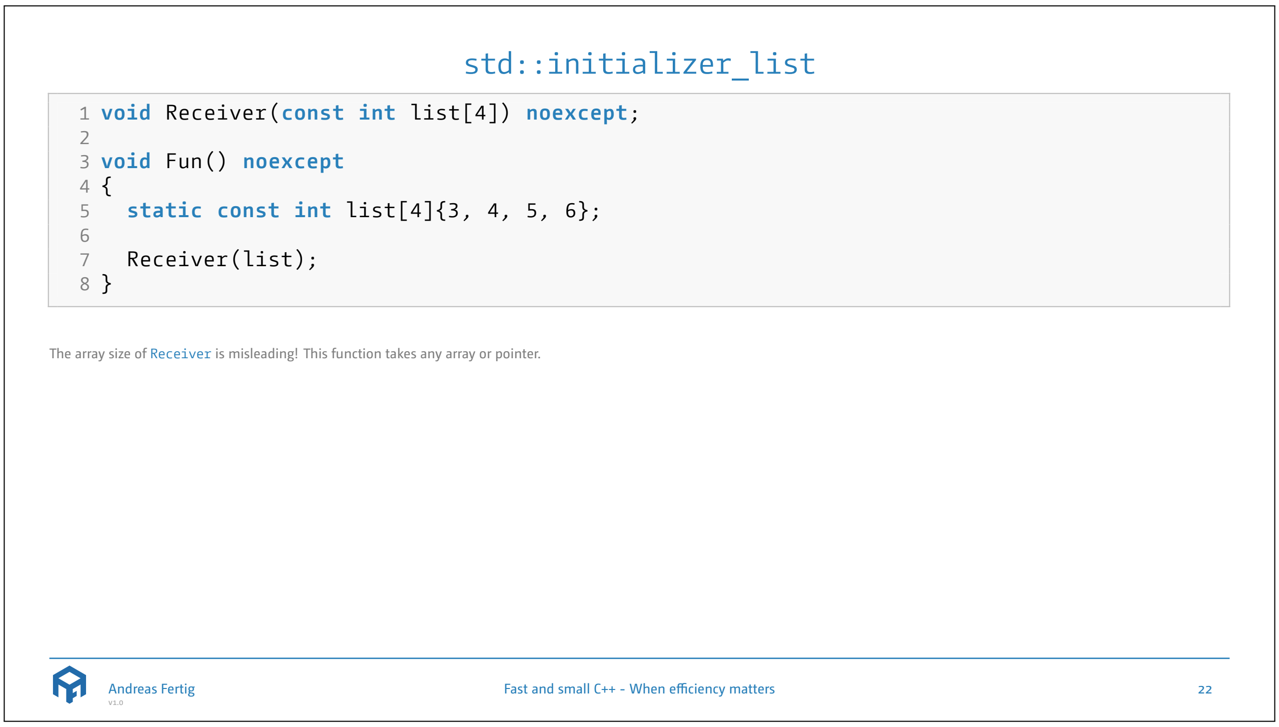
```cpp
void Fun()
{
  std::initializer_list<int> list{3, 4, 5, 6};

  Receiver(list);
}
```



std::initializer_list

```cpp
void Receiver(const int list[4]) noexcept;

void Fun() noexcept
{
  const int list[4]{3, 4, 5, 6};

  Receiver(list);
}
```

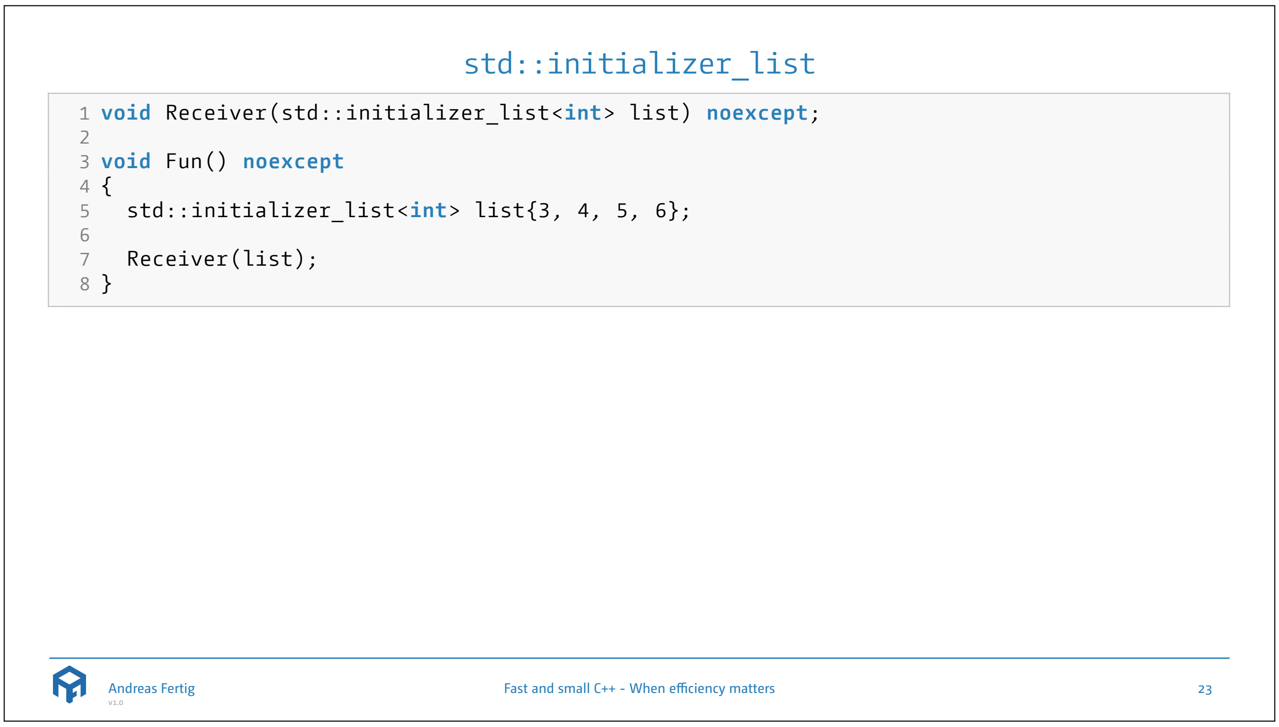
The array size of `Receiver` is misleading! This function takes any array or pointer.



std::initializer_list

```cpp
void Receiver(const int list[4]) noexcept;

void Fun() noexcept
{
  static const int list[4]{3, 4, 5, 6};

  Receiver(list);
}
```

The array size of `Receiver` is misleading! This function takes any array or pointer.



std::initializer_list

```cpp
void Receiver(std::initializer_list<int> list) noexcept;

void Fun() noexcept
{
  std::initializer_list<int> list{3, 4, 5, 6};

  Receiver(list);
}
```

}

I am Fertig.

https://AndreasFertig.com

Available online:

https://AndreasFertig.com

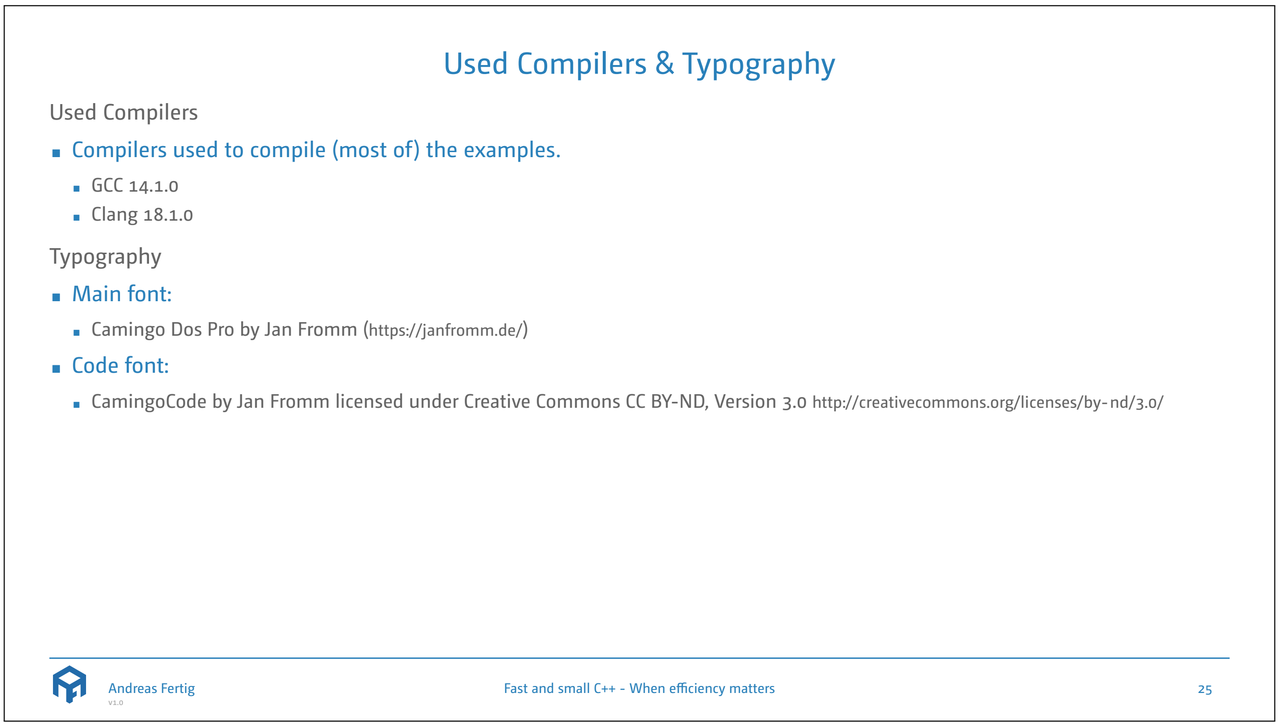
Images by Franziska Panter:

https://panther- concepts.de

Andreas Fertig
v1.0

Fast and small C++ - When efficiency matters

24



Used Compilers & Typography

Used Compilers

- Compilers used to compile (most of) the examples.
  - GCC 14.1.0
  - Clang 18.1.0

Typography

- Main font:
  - Camingo Dos Pro by Jan Fromm (https://janfromm.de/)
- Code font:
  - CamingoCode by Jan Fromm licensed under Creative Commons CC BY-ND, Version 3.0 http://creativecommons.org/licenses/by-nd/3.0/

Andreas Fertig
v1.0

Fast and small C++ - When efficiency matters

25

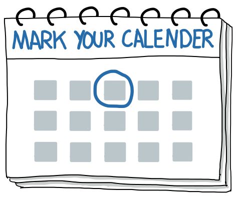
References

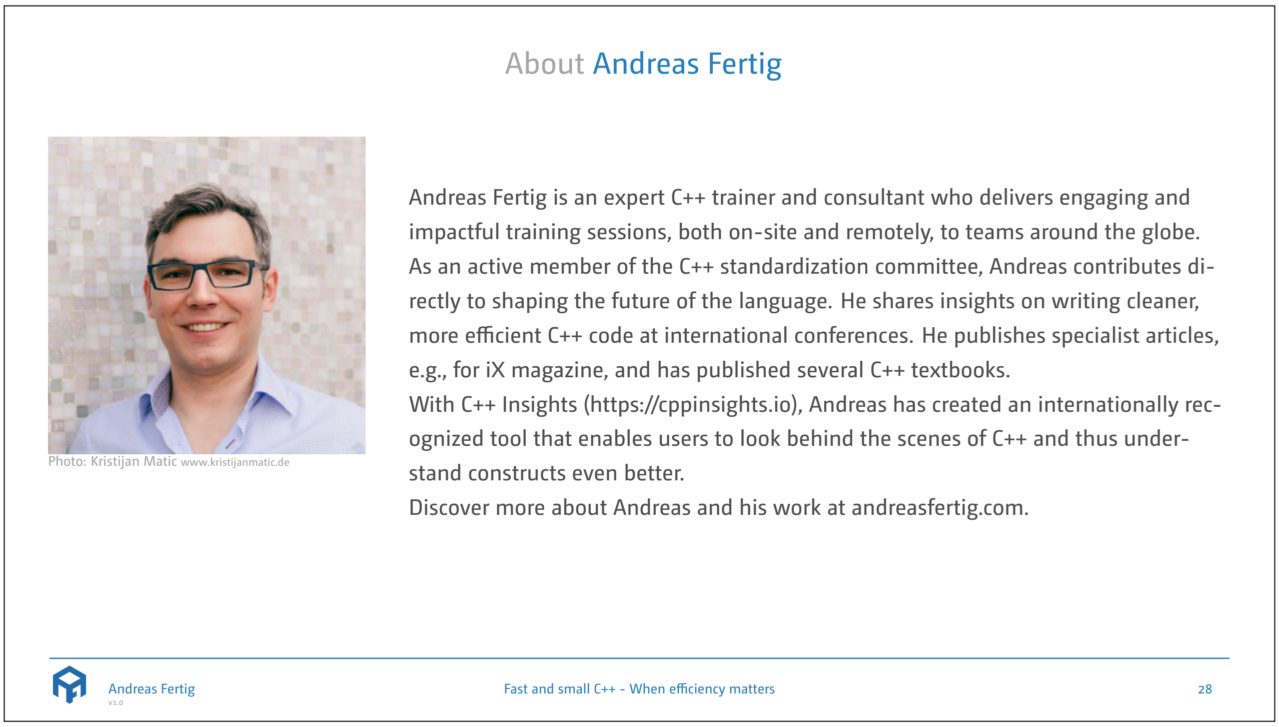
[1] Ormrod N., “Cppcon 2016: The strange details of std::string at facebook”. https://youtu.be/kPR8h4-qZdk

Images:
3: fran
27: Franziska Panter

Upcoming Events

For my upcoming talks you can check https://andreasfertig.com/talks/.
For my courses you can check https://andreasfertig.com/courses/.
Like to always be informed? Subscribe to my newsletter: https://andreasfertig.com/newsletter/.





About Andreas Fertig

Photo: Kristijan Matic www.kristijanmatic.de

Andreas Fertig is an expert C++ trainer and consultant who delivers engaging and impactful training sessions, both on-site and remotely, to teams around the globe. As an active member of the C++ standardization committee, Andreas contributes directly to shaping the future of the language. He shares insights on writing cleaner, more efficient C++ code at international conferences. He publishes specialist articles, e.g., for iX magazine, and has published several C++ textbooks.
With C++ Insights (https://cppinsights.io), Andreas has created an internationally recognized tool that enables users to look behind the scenes of C++ and thus understand constructs even better.
Discover more about Andreas and his work at andreasfertig.com.

Andreas Fertig v1.0 Fast and small C++ - When efficiency matters 28