# NOT GETTING LOST IN TRANSLATION

Daniela Engert - Meeting C++ 2024

## Outline

- The beginnings
- Let's talk
- A new library
- A new world
- What now
- An experiment
- Move on

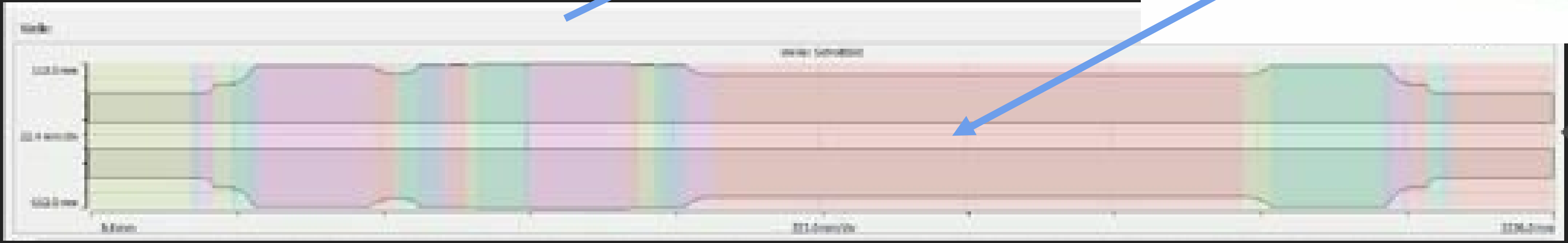
GMH Prüftechnik
GmbH · ND-Testing – Systems – Services

## ABOUT ME

- Electrical engineer
- I  build computers and create software for more than 40 years
- I develop hardware and software in the field of applied digital signal processing for more than 30 years

- I'm a member of the C++ committee (learning novice) for 4 years (EWG, SG15

ICE
DB



END

ICE
DB

FEUERWEHR

ICE
DB

# TIMELINE

2009

Boost::format

German only

## BOOST::FORMAT

Why?

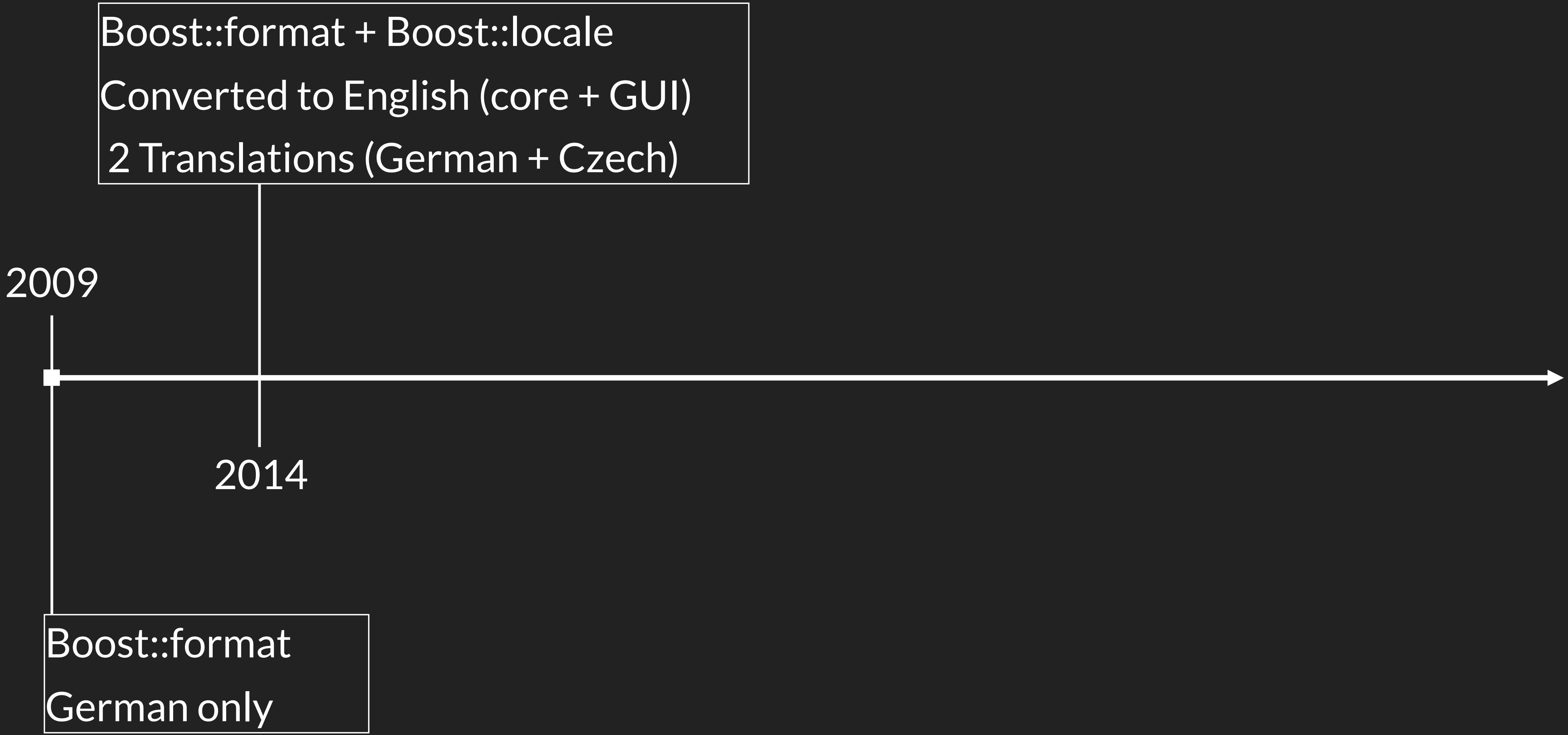
- it's compatible to C's printf() family
- it's different from C's printf() family
- it supports advanced formatting specifiers
- it "creatively" uses operators to separate the formatting specification from the arguments
- it supports formatting of user-defined types
- it supports stream manipulators

Example:

```
#include <boost/format.hpp>

std::cout << boost::format("formatting specifier") % argument1 % argument2;
```

## let's talk

# TIMELINE


Boost::format + Boost::locale
Converted to English (core + GUI)
2 Translations (German + Czech)
2009
2014
Boost::format
German only

## STRING TRANSLATION

Questions

## STRING TRANSLATION

Questions

- which translation system?
- is it mature?
- does it have an ecosystem?
  - editors
  - tools
  - build system
- does it fit? Does it feel "natural"?

## STRING TRANSLATION

can it be used by end users?

- domain experts
- not even remotely firm in IT

## TRANSLATION FORMAT

~~Binary~~

~~XML~~

~~JSON~~

Text

## GETTEXT

pretty standard, available on all major platforms

two major implementations

- GNU gettext
- Boost::locale (since Boost 1.48)

both rely on C locales (or their C++ equivalents) and message catalogs

# GETTEXT

~~pretty~~ standard, available on all major platforms

two major implementations

- GNU gettext (new: POSIX 1103.1-2024, ISO/IEC 9985-1:2024 🎉)
- Boost::locale (since Boost 1.48)

both rely on C locales (or their C++ equivalents) and message catalogs

## TOOLS

step 1: mark translatable text

## TOOLS

step 1: mark translatable text

```
#include <boost/format.hpp>

std::cout << boost::format("formatting specifier") % argument1 % argument2;
```

```
#include <boost/format.hpp>
#include <boost/locale/message.hpp>
using boost::message::translate

std::cout << boost::format(translate("formatting specifier")) % argument1 % argument2;
```

## TOOLS

step 2: scan source files

# TOOLS

step 2: scan source files,

using 'xgettext' for all marked texts

```
$(Keyword) is e.g. "translate"

xgettext --keyword --keyword=$(Keyword):1,1t --keyword=$(Keyword):1c,2,2t
--keyword=$(Keyword):1,2,3t --keyword=$(Keyword):1c,2,3,4t
a.cpp b.cpp c.hpp ...
```

Result: a so-called PO template file (.pot) with all found texts

- language agnostic
- transient

## MO' TOOLS

step 3: translate the strings

## MO' TOOLS

step 3: translate the strings,

create the translation files, one for each country, region, and company specialties

- msguniq
- msgmerge
- msgattrib

The result is a collection of PO (portable object) files

- specific to language, country, region, company
- stable
- checked into repositories

you can have multiple of them, e.g. one for each subsystem

## PO FILE CONTENT

```
#
msgid ""
msgstr ""
"Project-Id-Version: Example 1.0\n"
"Report-Msgid-Bugs-To: somebody@example.com\n"
"POT-Creation-Date: \n"
"PO-Revision-Date: 2024-04-30 11:38+0200\n"
"Last-Translator: EMAIL@ADDRESS\n"
"Language-Team: Czech\n"
"Language: cs\n"
"MIME-Version: 1.0\n"
"Content-Type: text/plain; charset=UTF-8\n"
"Content-Transfer-Encoding: 8bit\n"
"Plural-Forms: nplurals=3; plural=(n==1 ? 0 : n>=2 && n<=4 ? 1 : 2);\n"
"X-Generator: Poedit 3.4.2\n"

# This is only a example
#: main.cpp
msgctxt "Disambiguation"
msgid "Singular"
msgid_plural "Plural"
msgstr[0] "Singulární"
msgstr[1] "Duální"
msgstr[2] "Plurál"
```

## EVEN MORE TOOLS

step 4: compile the target files

## EVEN MORE TOOLS

step 4: compile the target files

(optionally) combine multiple PO files, and put the output into the target directory

- msgmerge
- msgfmt

'msgfmt' checks the consistency and validity of format specifiers and placeholders!

```
             +------------------------------------------+
    byte  0  | magic number = 0x950412de                |
             |                                          |
          4  | file format revision = 0                 |
             |                                          |
          8  | number of strings                        |  == N
             |                                          |
         12  | offset of table with original strings    |  == O
             |                                          |
         16  | offset of table with translation strings |  == T
             |                                          |
         20  | size of hashing table                    |  == S
             |                                          |
         24  | offset of hashing table                  |  == H
             |                                          |
             .                                          .
             .    (possibly more entries later)         .
             .                                          .
             |                                          |
          O  | length & offset 0th string  ----------------.
      O + 8  | length & offset 1st string  ------------------.
              ...                                    ...   | |
O + ((N-1)*8)| length & offset (N-1)th string           |  | |
             |                                          |  | |
          T  | length & offset 0th translation  ---------------.
      T + 8  | length & offset 1st translation  -----------------.
              ...                                    ...   | | | |
T + ((N-1)*8)| length & offset (N-1)th translation      |  | | | |
             |                                          |  | | | |
          H  | start hash table                         |  | | | |
              ...                                    ...   | | | |
  H + S * 4  | end hash table                           |  | | | |
             |                                          |  | | | |
             | NUL terminated 0th string  <----------------' | | |
             |                                          |    | | |
             | NUL terminated 1st string  <------------------' | |
             |                                          |      | |
              ...                                    ...       | |
             |                                          |      | |
             | NUL terminated 0th translation  <---------------' |
             |                                          |        |
             | NUL terminated 1st translation  <-----------------'
             |                                          |
              ...                                    ...
             |                                          |
             +------------------------------------------+
```

# MO FILE CONTENT

The result is a collection of MO (machine object) files

- specific to language, country, region, company
- binary
- optimized for consumption
- distributed to the customer's machine

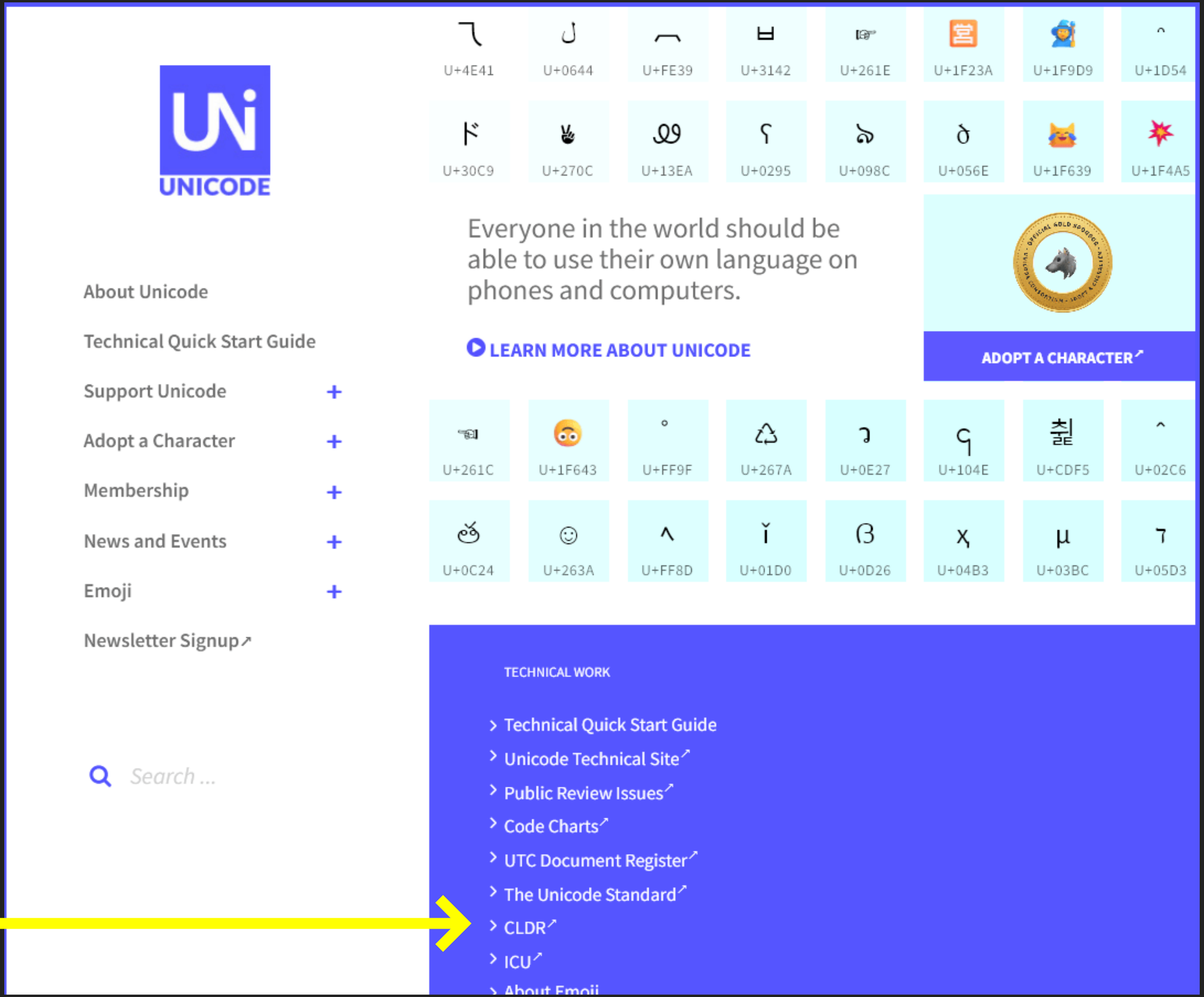
The format is described in the manual: https://www.gnu.org/software/gettext/manual/gettext.html#MO-Files

# UNICODE

Unicode covers more than glyphs, code points, and their encodings!

CLDR

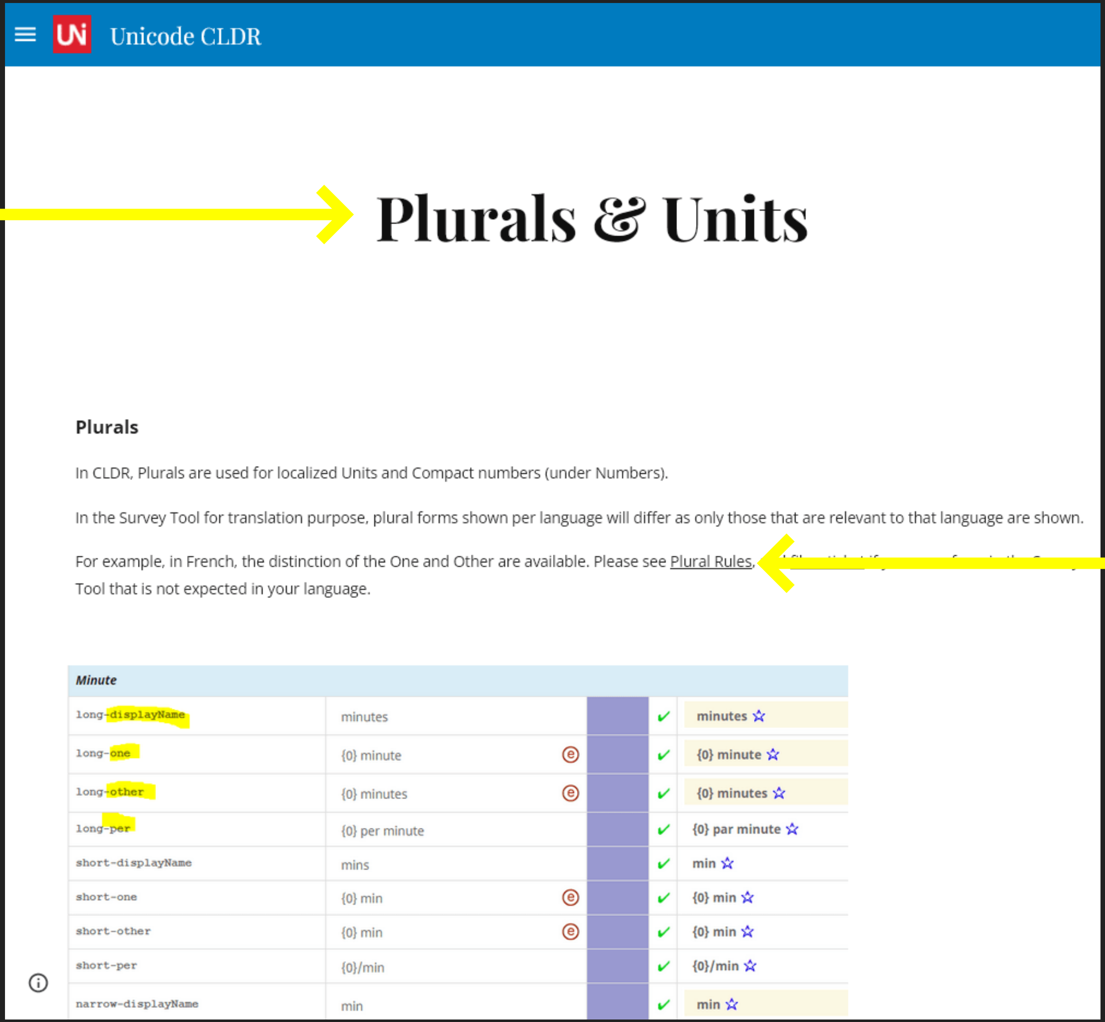
Common
Language
Data
Repository



Everyone in the world should be able to use their own language on phones and computers.

# CLDR

The CLDR defines machine readable rules for e.g. translation



Unicode CLDR

## Plurals & Units

### Plurals

In CLDR, Plurals are used for localized Units and Compact numbers (under Numbers).

In the Survey Tool for translation purpose, plural forms shown per language will differ as only those that are relevant to that language are shown.

For example, in French, the distinction of the One and Other are available. Please see Plural Rules, [illegible] Tool that is not expected in your language.

| *Minute* | | | | | |
|---|---|---|---|---|---|
| long-displayName | minutes | | | ✓ | minutes ☆ |
| long-one | {0} minute | ⓔ | | ✓ | {0} minute ☆ |
| long-other | {0} minutes | ⓔ | | ✓ | {0} minutes ☆ |
| long-per | {0} per minute | | | ✓ | {0} par minute ☆ |
| short-displayName | mins | | | ✓ | min ☆ |
| short-one | {0} min | ⓔ | | ✓ | {0} min ☆ |
| short-other | {0} min | ⓔ | | ✓ | {0} min ☆ |
| short-per | {0}/min | | | ✓ | {0}/min ☆ |
| narrow-displayName | min | | | ✓ | min ☆ |

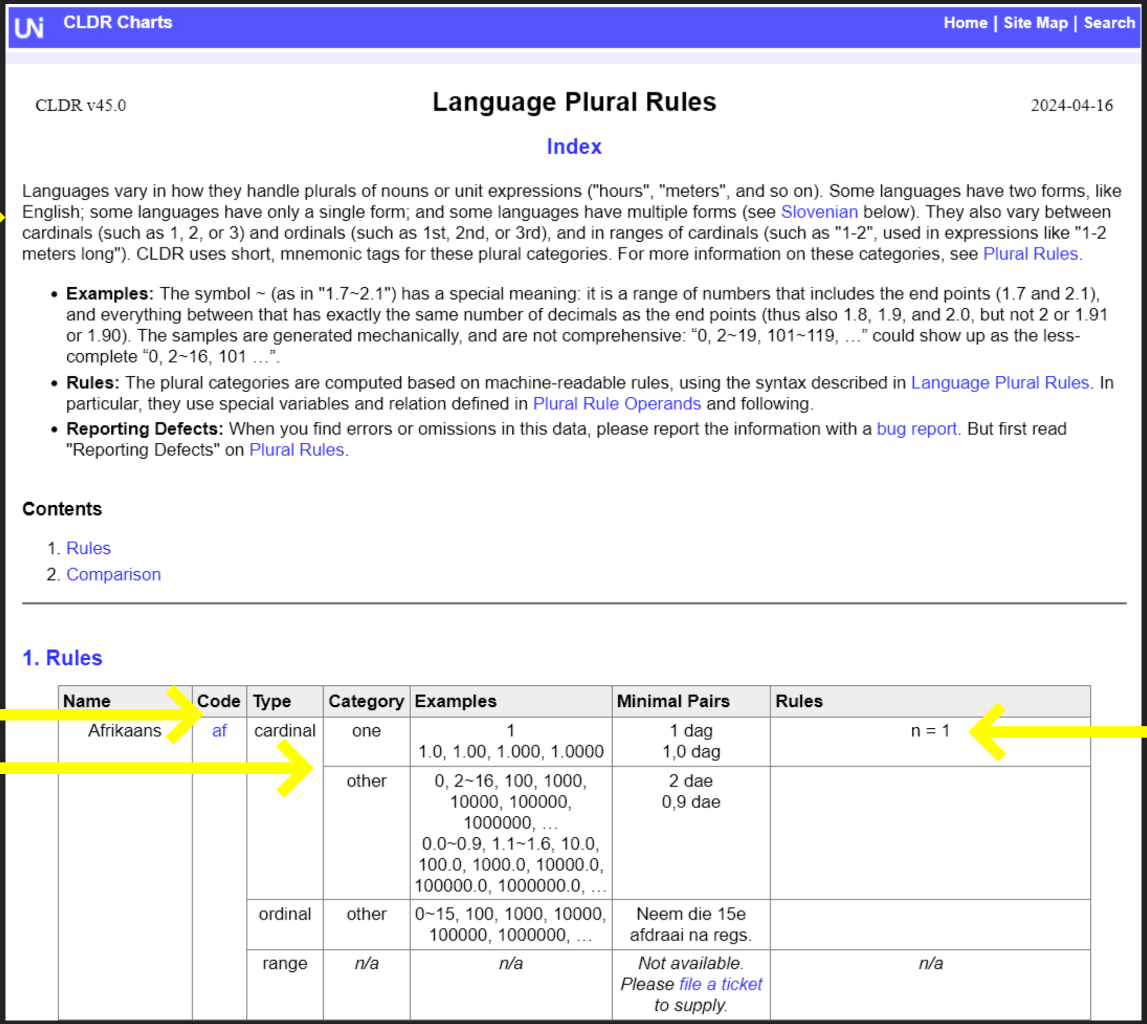
# LANGUAGE PLURAL RULES

Languages vary in plural forms

ISO 639

ISO 3166,

POSIX



CLDR Charts Home | Site Map | Search

CLDR v45.0 **Language Plural Rules** 2024-04-16

**Index**

Languages vary in how they handle plurals of nouns or unit expressions ("hours", "meters", and so on). Some languages have two forms, like English; some languages have only a single form; and some languages have multiple forms (see Slovenian below). They also vary between cardinals (such as 1, 2, or 3) and ordinals (such as 1st, 2nd, or 3rd), and in ranges of cardinals (such as "1-2", used in expressions like "1-2 meters long"). CLDR uses short, mnemonic tags for these plural categories. For more information on these categories, see Plural Rules.

- **Examples:** The symbol ~ (as in "1.7~2.1") has a special meaning: it is a range of numbers that includes the end points (1.7 and 2.1), and everything between that has exactly the same number of decimals as the end points (thus also 1.8, 1.9, and 2.0, but not 2 or 1.91 or 1.90). The samples are generated mechanically, and are not comprehensive: "0, 2~19, 101~119, …" could show up as the less-complete "0, 2~16, 101 …".
- **Rules:** The plural categories are computed based on machine-readable rules, using the syntax described in Language Plural Rules. In particular, they use special variables and relation defined in Plural Rule Operands and following.
- **Reporting Defects:** When you find errors or omissions in this data, please report the information with a bug report. But first read "Reporting Defects" on Plural Rules.

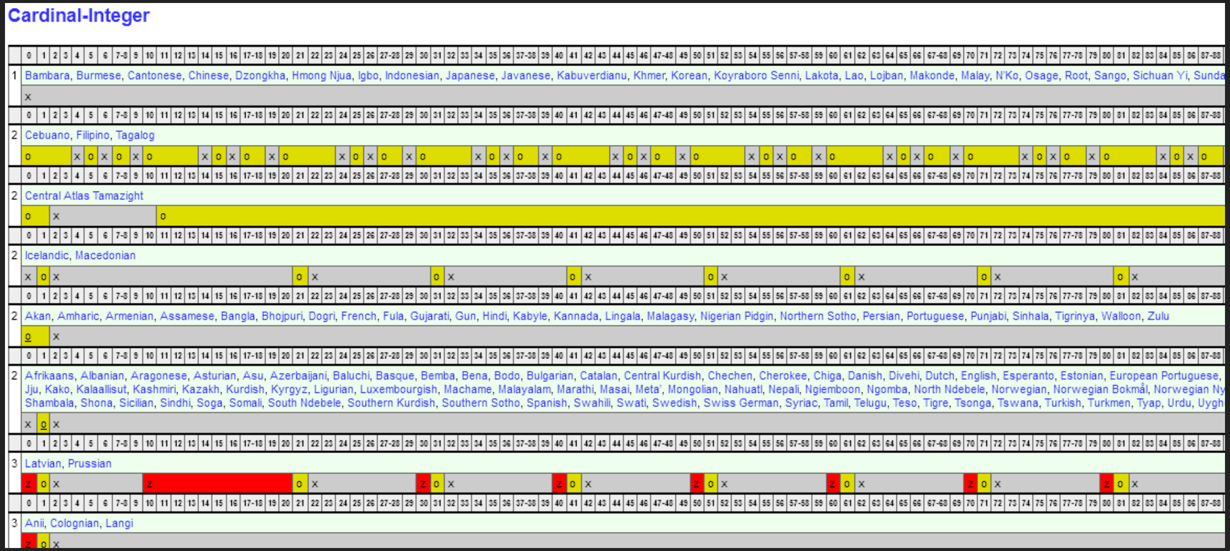
**Contents**

**1. Rules**

| Name | Code | Type | Category | Examples | Minimal Pairs | Rules |
|---|---|---|---|---|---|---|
| Afrikaans | af | cardinal | one | 1<br>1.0, 1.00, 1.000, 1.0000 | 1 dag<br>1,0 dag | n = 1 |
| | | | other | 0, 2~16, 100, 1000, 10000, 100000, 1000000, …<br>0.0~0.9, 1.1~1.6, 10.0, 100.0, 1000.0, 10000.0, 100000.0, 1000000.0, … | 2 dae<br>0,9 dae | |
| | | ordinal | other | 0~15, 100, 1000, 10000, 100000, 1000000, … | Neem die 15e afdraai na regs. | |
| | | range | *n/a* | *n/a* | *Not available. Please file a ticket to supply.* | *n/a* |

# PLURAL FORMS

from none, to simple, to complicated



**Cardinal-Integer**

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7-8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17-18 | 19 | 20 | … | 87-88 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

1 Bambara, Burmese, Cantonese, Chinese, Dzongkha, Hmong Njua, Igbo, Indonesian, Japanese, Javanese, Kabuverdianu, Khmer, Korean, Koyraboro Senni, Lakota, Lao, Lojban, Makonde, Malay, N'Ko, Osage, Root, Sango, Sichuan Yi, Sunda

x

2 Cebuano, Filipino, Tagalog

o x o x o x o x o x o x o x o x o x o x o x o x o x o x o x o x o x o x o x o x o x o x o x o x o x o x o x o x o

2 Central Atlas Tamazight

o x o

2 Icelandic, Macedonian

x o x o x o x o x o x o x o x o x o x

2 Akan, Amharic, Armenian, Assamese, Bangla, Bhojpuri, Dogri, French, Fula, Gujarati, Gun, Hindi, Kabyle, Kannada, Lingala, Malagasy, Nigerian Pidgin, Northern Sotho, Persian, Portuguese, Punjabi, Sinhala, Tigrinya, Walloon, Zulu

o x

2 Afrikaans, Albanian, Aragonese, Asturian, Asu, Azerbaijani, Baluchi, Basque, Bemba, Bena, Bodo, Bulgarian, Catalan, Central Kurdish, Chechen, Cherokee, Chiga, Danish, Divehi, Dutch, English, Esperanto, Estonian, European Portuguese,
Jju, Kako, Kalaallisut, Kashmiri, Kazakh, Kurdish, Kyrgyz, Ligurian, Luxembourgish, Machame, Malayalam, Marathi, Masai, Meta', Mongolian, Nahuatl, Nepali, Ngiemboon, Ngomba, North Ndebele, Norwegian, Norwegian Bokmål, Norwegian Ny
Shambala, Shona, Sicilian, Sindhi, Soga, Somali, South Ndebele, Southern Kurdish, Southern Sotho, Spanish, Swahili, Swati, Swedish, Swiss German, Syriac, Tamil, Telugu, Teso, Tigre, Tsonga, Tswana, Turkish, Turkmen, Tyap, Urdu, Uygh

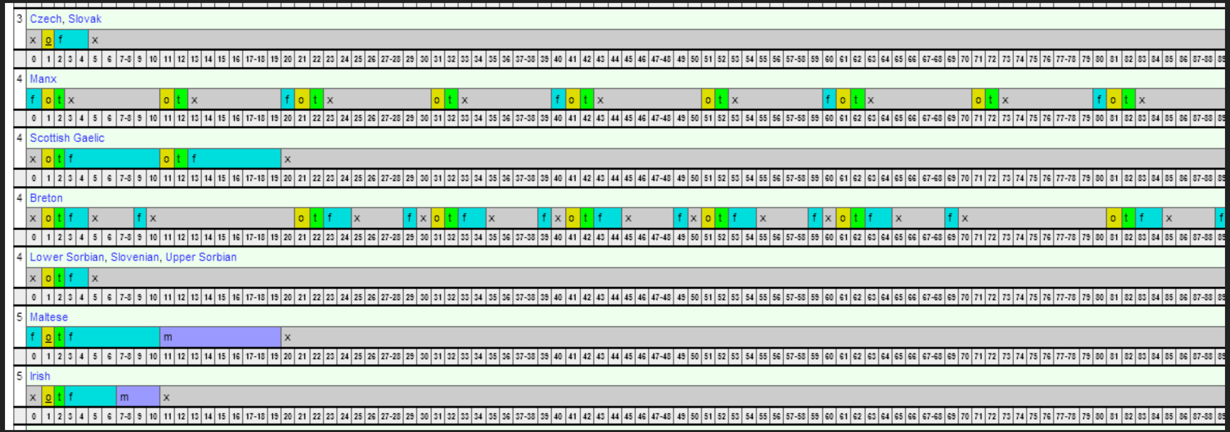
x o x

3 Latvian, Prussian

z o x z o x z o x z o x z o x z o x z o x z o x

3 Anii, Colognian, Langi

z o x



3 Czech, Slovak

x o f x

4 Manx

f o t x o t x f o t x o t x f o t x o t x f o t x o t x f o t x

4 Scottish Gaelic

x o t f o t f x

4 Breton

x o t f x f x o t f x f x o t f x f x o t f x f x o t f x f x o t f x f x o t f x f x o t f x f

4 Lower Sorbian, Slovenian, Upper Sorbian

x o t f x

5 Maltese

f o t f m x

5 Irish

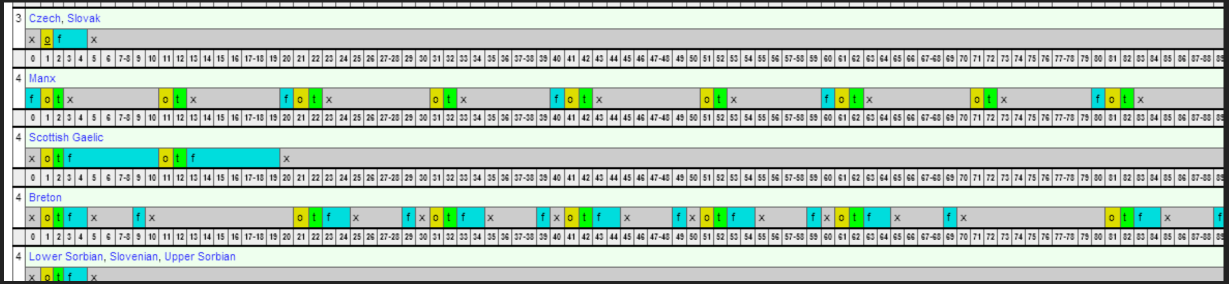
x o t f m x

# PLURAL FORMS

GNU gettext tries to cover these patterns in single-line C expressions.

Most implementations follow the CLDR rules.

```
"Language: ja"
"Plural-Forms: nplurals=1; plural=0;"

"Language: de"
"Plural-Forms: nplurals=2; plural=(n != 1);"

"Language: cs"
"Plural-Forms: nplurals=3; plural=(n==1 ? 0 : n>=2 && n<=4 ? 1 : 2);"

"Language: br"
"Plural-Forms: nplurals=5; plural=(n%10==1 && n%100!=11 && n%100!=71 && n%100!=91 ? 0 : n%10==2 && n%100!=12 &&
n%100!=72 && n%100!=92 ? 1 : ((n%10>=3 && n%10<=4) || n%10==9) && (n%100<10 || n%100>19) && (n%100<70 || n%100>79)
&& (n%100<90 || n%100>99) ? 2 : n!=0 && n%1000000==0 ? 3 : 4);"
```



## BOOST.LOCALE

it does too much!

😭

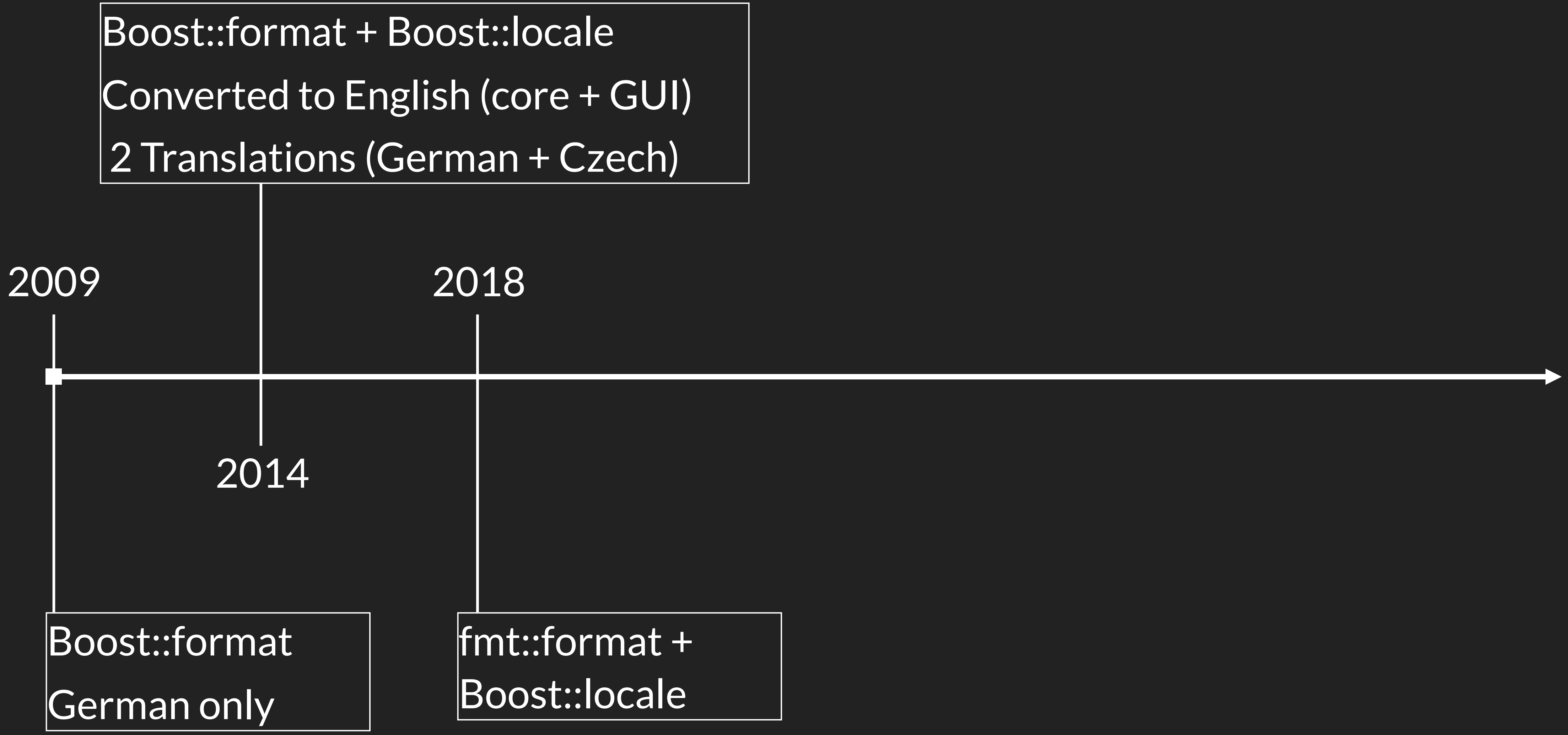
on every return from a translation call,

- it does a translation
- it does an encoding conversion

therefore it requires a string allocation

## a new library

## TIMELINE



Boost::format + Boost::locale
Converted to English (core + GUI)
2 Translations (German + Czech)
2009
2018
2014
Boost::format
German only
fmt::format +
Boost::locale

# {FMT}

```cpp
#include <boost/format.hpp>

std::cout << boost::format("formatting specifier") % argument1 % argument2;
```

↓

```cpp
#include <fmt/format.hpp>

std::cout << fmt::format("formatting specifier", argument1, argument2);
```

```cpp
template <typename Char>
format(const something-related-to-Char & FS) -> boost::basic_format<Char>;
         +
template <typename Char, typename T>
operator%(boost::basic_format<Char> &, const T &) -> boost::basic_format<Char>;
```

↓

```cpp
template <typename... Ts>
format(std::string_view FS, Ts &&... Args) -> std::string;

template <typename... Ts>
format(std::wstring_view FS, Ts &&... Args) -> std::wstring;
```

# {FMT} HAS TWO INTERFACES!

```
// Interface 1: typeful

template <typename... Types>
format(std::string_view, Types &&... Args) -> std::string;


// Interface 2: type-erased

vformat(std::string_view, format_args) -> std::string;


// the typeful interface is a thin wrapper around the type-erased interface

template <typename... Types>
format(std::string_view FS, Types &&... Args -> std::string {
    return vformat(FS, make_format_args(Args));
}
```

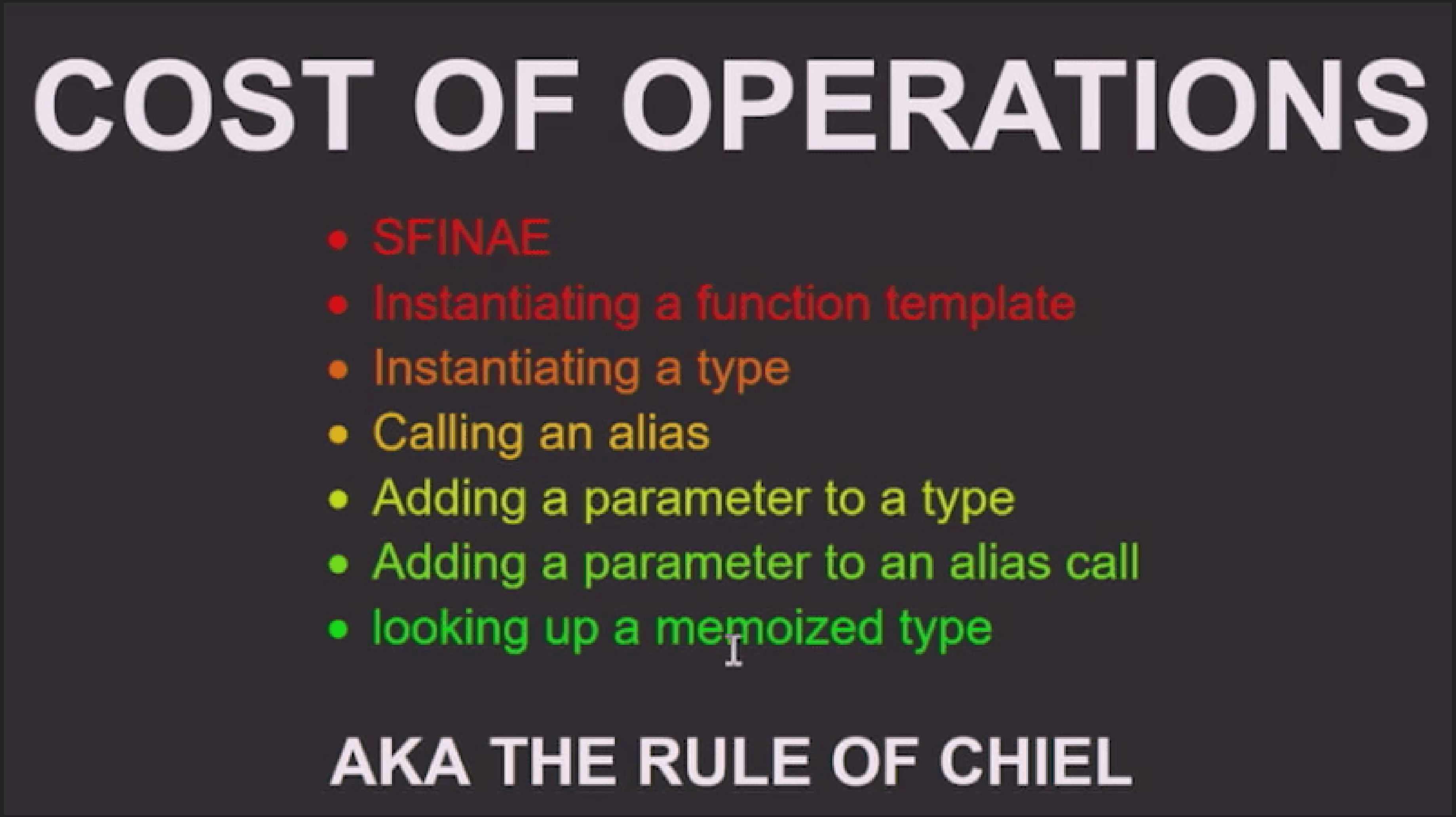
Type erasure, or better: type classification

# TYPE ERASURE

{fmt} is using all kinds of type-based metaprogramming to achieve its goals:

- templates (type calculations)
- partial template specializations (i.e. "pattern matching")
- overload resolution and conversion sequences of unimplemented functions (conversion-based metaprogramming)
- SFINAE ("substitution failure is not an error") to control overload sets and viability of functions

## RULE OF Chiel Douwes 💜



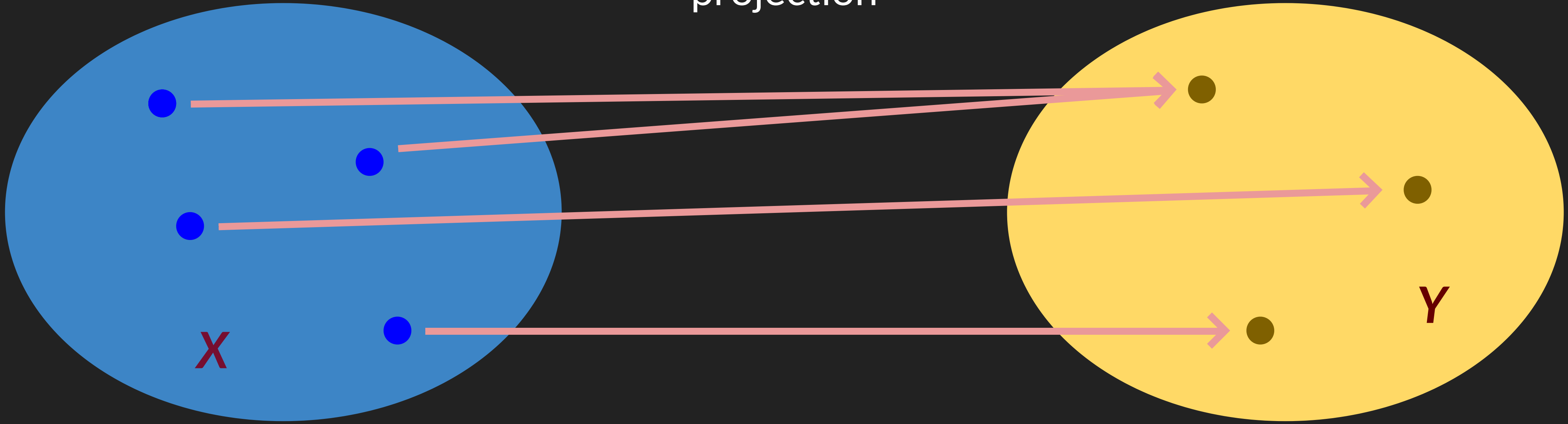
COST OF OPERATIONS

- SFINAE
- Instantiating a function template
- Instantiating a type
- Calling an alias
- Adding a parameter to a type
- Adding a parameter to an alias call
- looking up a memoized type

AKA THE RULE OF CHIEL

## WHAT IS A FUNCTION?

the operator *f* describes any kind of projection



anything,

not necessarily just one element

anything else,

necessarily exactly one element

*example f*: $x \longrightarrow \begin{cases} y_0 & \text{if } x \text{ matches predicate } 0 \\ y_1 & \text{if } x \text{ matches predicate } 1 \\ \dots & \text{if } x \text{ matches predicate } \dots \end{cases}$

# WHAT IS A FUNCTION?

the context determines in which parameters are valid

$$y = f(x)$$

generally a runtime parameter, but potentially valid at compile time

$$y = f\langle x_0 \rangle(x_1)$$

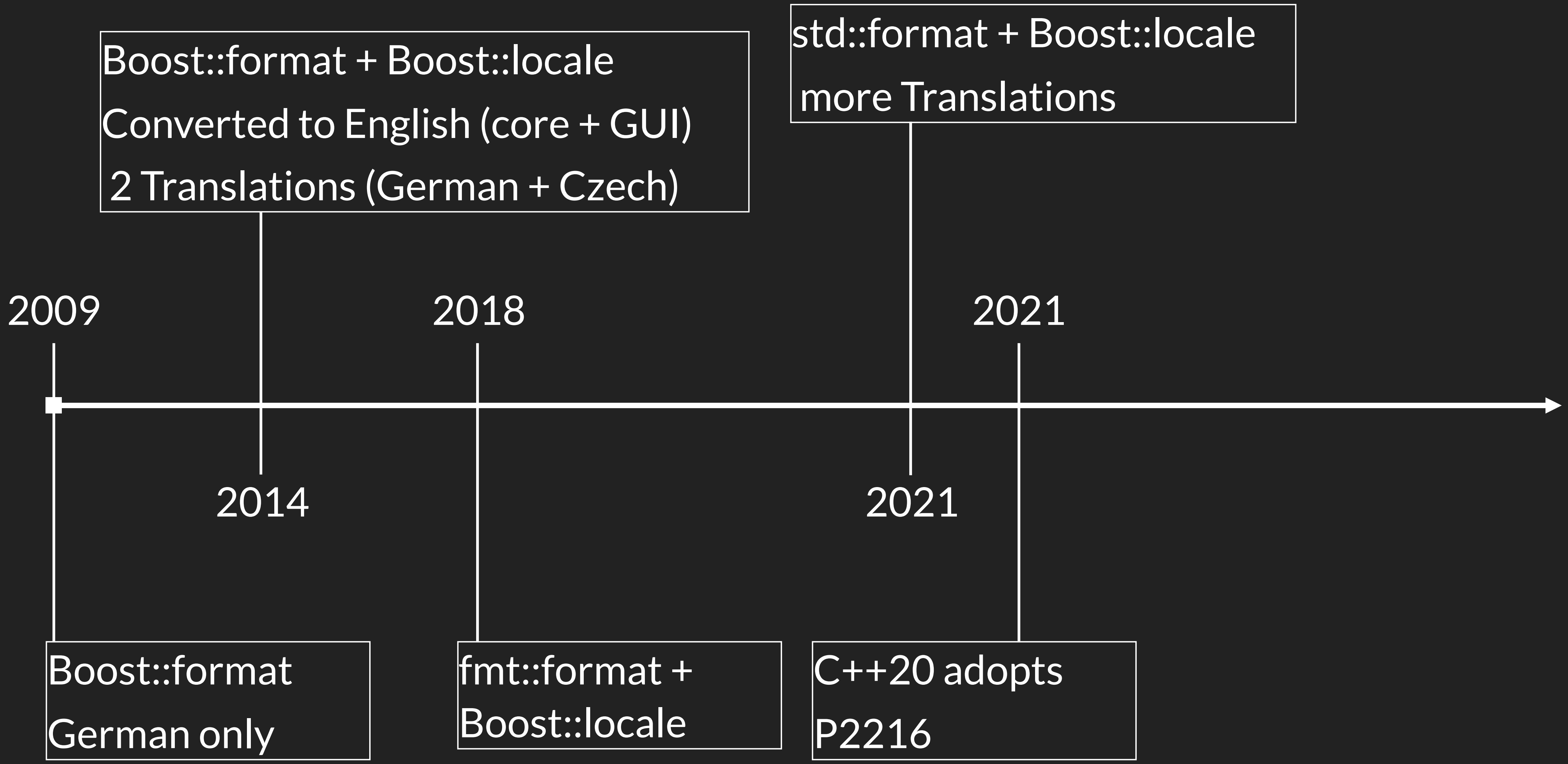
manifestly a compile time parameter

$$y = f\langle x \rangle()$$

# a new world

J. Mulder Sculp.
3

# TIMELINE


Boost::format + Boost::locale
Converted to English (core + GUI)
2 Translations (German + Czech)
std::format + Boost::locale
more Translations
2009
2018
2021
2014
2021
Boost::format
German only
fmt::format + Boost::locale
C++20 adopts P2216

## FROM C++20

Original C++20 (Prague 2020, N4860)

```cpp
template<typename... Types>
auto std::format(std::string_view Fmt, const Types &... Args) -> std::string {
    return std::vformat(Fmt, std::make_format_args(Args...));
}
```

# FROM C++20 TO C++20

## Original C++20 (Prague 2020, N4860)

```cpp
template<typename... Types>
auto std::format(std::string_view Fmt, const Types &... Args) -> std::string {
    return std::vformat(Fmt, std::make_format_args(Args...));
}
```

## Contemporary C++20 (August 2024, N4988)

```cpp
template<typename... Types>
auto std::format(std::format_string<Types...> Fmt, Types &&... Args) -> std::string {
    return std::vformat(Fmt.get(), std::make_format_args(Args...));
}

template <typename... Types>
struct format_string {
    consteval format_string(std::string_view Str) : Str_(Str) {}

    constexpr std::string_view get() const noexcept { return Str_; }

private:
    std::string_view Str_;
};
```

# CONSTEVAL

## WHAT IS CONSTANT EVALUATION?

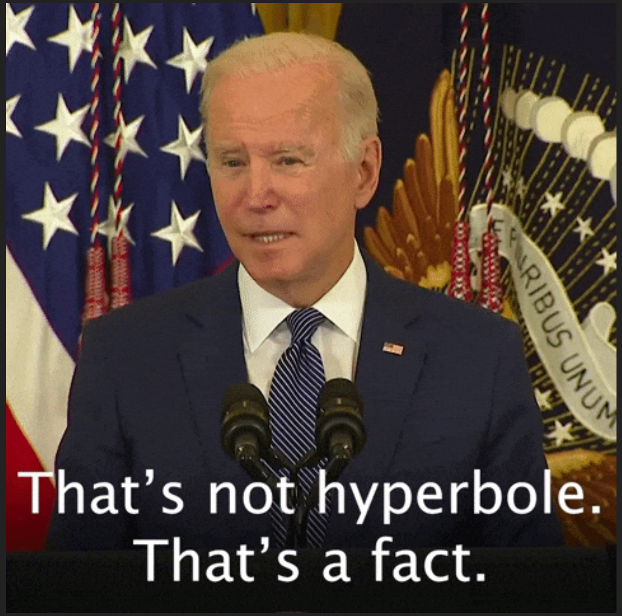
## WHAT IS CONSTANT EVALUATION?

During compilation, the compiler has to remember everything it has seen so far:

- identifiers
- entities
- declarations
- definitions
- ...
- templates
- all template instantiations so far

Pretty much everything.

BTW, this is why we have modules now 🎉



# WHAT IS CONSTANT EVALUATION?

What about these code lines?

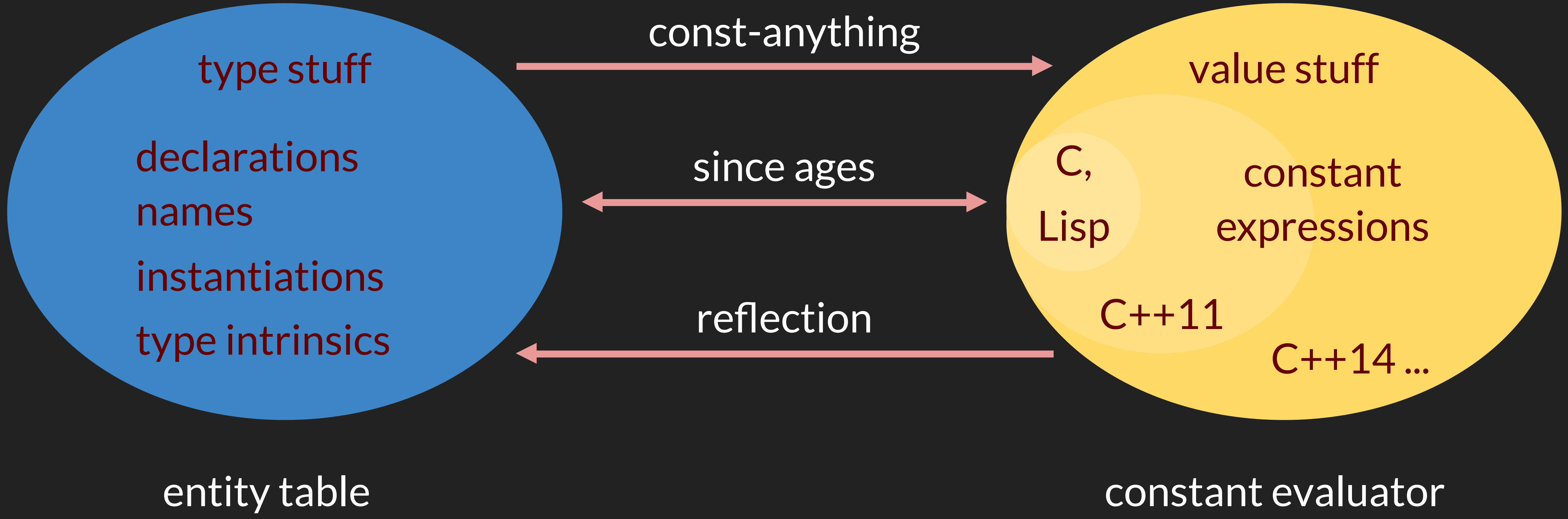
- int i = 1 + 2 + 3 / 4;
- bool b{ 42 == L"42" };
- char a[] = "huh?";
- enum e { none, anyone, couple };
- static_assert( none != anyone);
- const int c = couple;
- std::vector<int> v = { none * 2, anyone * sizeof(a), couple * c };
- ...

constant required,

"constant expression"

# SPLIT BRAIN

two subsystems for compile-time entities



# P2216

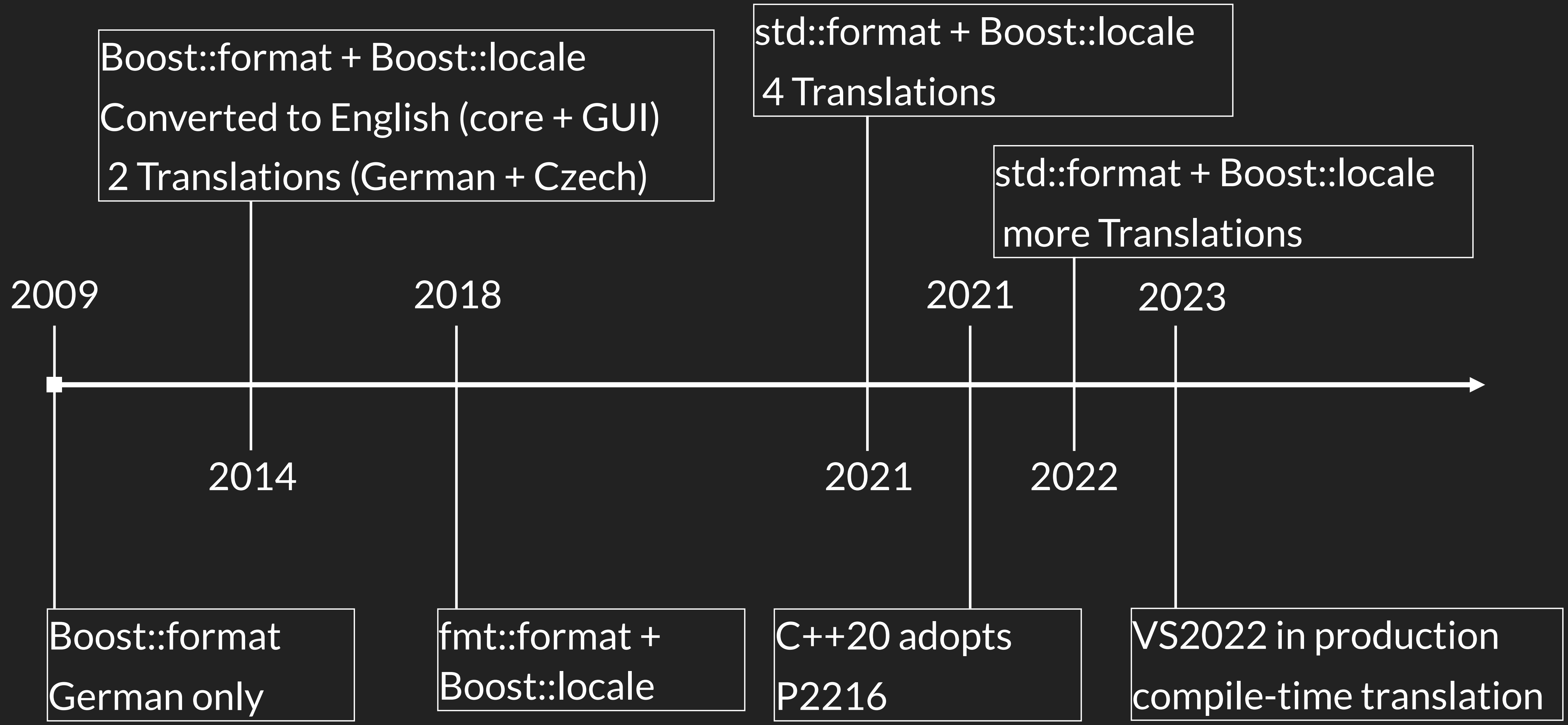
```
template<typename... Types>
auto std::format(std::format_string<Types...> Fmt, Args &&...) -> std::string;

template <typename... Types>
struct format_string {
    consteval format_string(std::string_view Str) {
        constexpr size_t num_args = sizeof...(Types);
        constexpr basic_format_arg_type arg_types[num_args > 0 ? num_args : 1] = {
            std::get_format_arg_type<Types>()...
        };

        parse_format_string(Str, format_checker<std::remove_cvref_t<Types>...>{Str, arg_types});
    }
};
```

format string syntax and argument type checking can - and will ! - be done at compile time

no more exceptions at runtime!

## what now?

## TIMELINE


2009
Boost::format
German only
2014
Boost::format + Boost::locale
Converted to English (core + GUI)
2 Translations (German + Czech)
2018
fmt::format + Boost::locale
2021
std::format + Boost::locale
4 Translations
2021
C++20 adopts P2216
2022
std::format + Boost::locale
more Translations
2023
VS2022 in production
compile-time translation

## P2216 FALLOUT

~~Boost.Locale~~

😢

## WHAT NOW?

```
unsigned amount = 3;
format(translate("she got an apple", "she got {} apples"), plural{ amount });

// "hun fikk 3 epler"      (no)
// "dostala 3 jablka"      (cs)
// "dobila je 3 jabolke"   (si)
```

# WHAT NOW?

```cpp
unsigned amount = 3;
format(translate("she got an apple", "she got {} apples"), plural{ amount });

// "hun fikk 3 epler"       (no)
// "dostala 3 jablka"       (cs)
// "dobila je 3 jabolke"    (si)
```

```cpp
namespace std {

struct format_string {
    consteval format_string(const convertible_to<string_view> & Str) : Str_(Str) { ... }

    constexpr std::string_view get() const noexcept { return Str_; }
};

}
```

## MO' OVERLOADS

```cpp
template <typename... Types>
auto format(const format_string_translator<Types...> XFmt, Types &&... Args) -> std::string {
    if constexpr (sizeof...(Args) > 0) {
        const auto Quantity = XFmt.Quantity( ... something with Args );
        return std::vformat(XFmt.get(Quantity),
                            make_format_args(
                                wrapped<std::remove_cvref_t<Types>>::translate(Args)...
                            ));
    } else {
        return std::vformat(XFmt.get(), {});
    }
}
```

# FORMAT_STRING_TRANSLATOR

```cpp
template <typename... Types>
auto format(const format_string_translator<Types...>, Types &&...) -> std::string;

template <typename... Types>
struct format_string_translator : basic_translator {
    using base = format_string<Types...>;

    consteval format_string_translator(const tTranslate & Tr)      // almost all work is done here
    : basic_translator(Tr) {
        base{ Tr.Singular() }, base{ Tr.Plural() }, ...;
    }
};
```

# FORMAT_STRING_TRANSLATOR

```cpp
template <typename... Types>
auto format(const format_string_translator<Types...>, Types &&...) -> std::string;

template <typename... Types>
struct format_string_translator : basic_translator {
    using base = format_string<Types...>;

    static constexpr auto numPluralArguments = (isMarkedAsPlural<Types> + ... + 0);
    static_assert(numPluralArguments <= 1, "Oops, more than one plural argument was found!");

    consteval format_string_translator(const tTranslate & Tr)       // almost all work is done here
    : basic_translator(Tr) {
        base{ Tr.Singular() }, base{ Tr.Plural() }, checkPlural(Tr.lenPlural_, numPluralArguments);
    }
};

consteval void checkPlural(std::size_t gotPluralFormat, std::size_t havePluralArguments) {
    if (gotPluralFormat and not havePluralArguments)
        throw "Sorry, a plural format string is present, but no plural argument was found!";
    if (havePluralArguments and not gotPluralFormat)
        throw "Sorry, a plural argument was found, but no plural format string is present!";
}

template <typename T>
constexpr bool isMarkedAsPlural = std::is_same_v<plural, T>;
```

# BASIC_TRANSLATOR

```cpp
template <typename... Types>
auto format(const format_string_translator<Types...>, Types &&...) -> std::string;

template <typename... Types>
struct format_string_translator : basic_translator;  // knows argument types, does syntax checking

struct basic_translator {   // knows only string digest, and how to translate from actual cardinal
    consteval basic_translator(const tTranslate & Tr)
    : Translator_{ markPluralsPresent(Tr.Digest_, Tr.lenPlural_ > 0) } {}

private:
    tBaseTranslate Translator_;                        // strips off all strings, knows only the digest
};
```

# BASIC_TRANSLATOR

```cpp
template <typename... Types>
auto format(const format_string_translator<Types...>, Types &&...) -> std::string;

template <typename... Types>
struct format_string_translator : basic_translator;  // knows argument types, does syntax checking

struct basic_translator {   // knows only string digest, and how to translate from actual cardinal
    consteval basic_translator(const tTranslate & Tr)
    : Translator_{ markPluralsPresent(Tr.Digest_, Tr.lenPlural_ > 0) } {}

    std::string_view get(plural::type N) const noexcept {
        return havePlurals(Translator_.Digest_) ? Translator_.multiple(N) : Translator_.single();
    }

    std::string_view get() const noexcept { return Translator_.single(); }

private:
    tBaseTranslate Translator_;                          // strips off all strings, knows only the digest
};

constexpr bool havePlurals(const uint64_t Digest) { ... }

consteval uint64_t markPluralsPresent(const uint64_t Digest, const bool havePlural) { ... }
```

## BASETRANSLATE

```cpp
template <typename... Types>
auto format(const format_string_translator<Types...>, Types &&...) -> std::string;

template <typename... Types>
struct format_string_translator : basic_translator;  // knows argument types, does syntax checking

struct basic_translator;    // knows only string digest, and how to translate from actual cardinal

struct tBaseTranslate {

    std::uint64_t Digest_;    // constructed by frontend at compile-time
};
```

the only

data member

# BASETRANSLATE

```cpp
template <typename... Types>
auto format(const format_string_translator<Types...>, Types &&...) -> std::string;

template <typename... Types>
struct format_string_translator : basic_translator;  // knows argument types, does syntax checking

struct basic_translator;    // knows only string digest, and how to translate from actual cardinal

struct tBaseTranslate {                                    // perform translation from compile-time digest
    constexpr auto single() const noexcept -> std::string_view {
        const auto & Maps = ...           // from backend;
        return lookup(Maps, Digest_);    // middle: wed frontend and backend □
    }

    constexpr auto multiple(plural::type N) const noexcept -> std::string_view {
        const auto & Maps = ...                              // from backend;
        return lookup(Maps, Digest_, std::uint64_t{ N });  // middle: wed frontend and backend □
    }

    ... more overloads of 'single()' and 'multiple(N)'

    std::uint64_t Digest_;   // constructed by frontend at compile-time
};
```

the only runtime code

## 3 QUESTIONS

```
template <typename... Types>
auto format(const format_string_translator<Types...> XFmt, Types &&... Args) -> std::string {
    const auto Quantity = XFmt.Quantity( ... something with Args );    1️⃣,2️⃣
    return std::vformat(XFmt.get(Quantity),
                        make_format_args(
                             ... somehow translate Args when necessary    3️⃣
                        );
}
```

1️⃣ which argument holds the cardinal that determines the language form?

2️⃣ how to access its current value?

3️⃣ how to figure out how a given argument is translated? Or is it at all?

# 3 QUESTIONS

```
template <typename... Types>
auto format(const format_string_translator<Types...> XFmt, Types &&... Args) -> std::string {
    const auto Quantity = XFmt.Quantity( ... something with Args );    1️⃣,2️⃣
    return std::vformat(XFmt.get(Quantity),
                        make_format_args(
                            wrapped<std::remove_cvref_t<Types>>::translate(Args)...)    3️⃣
                        );
}
```

1️⃣ which argument holds the cardinal that determines the language form?

2️⃣ how to access its current value?

3️⃣ how to figure out how a given argument is translated? Or is it at all?

# P2663 (PACK INDEXING) + P2996 (REFLECTION)

```
template <typename... Types>
auto format(const format_string_translator<Types...> XFmt, Types &&... Args) -> std::string {
    constexpr auto Index = std::ranges::find_if({ Args^... }, isPluralType) - &Args...[0]^;
    return std::vformat(XFmt.get(Args...[Index]),  2️⃣                          1️⃣
                              make_format_args( [... something with Args^... ...]));
}
                                                   3️⃣
```

1️⃣ which argument holds the cardinal that determines the language form?

2️⃣ how to access its corrent value?

3️⃣ how to figure out how a given type is translated? Or is it at all?

Unfortunately, this is 2024. C++26 is not a thing yet 🤯

## HETEROGENEOUS SEQUENCES

## HETEROGENEOUS SEQUENCES

✨

Folds!

🚀

## HETEROGENEOUS SEQUENCES

A tutorial

on the universality

and expressiveness of fold

Graham Hutton, Journal of Functional Programming, 1999

# SOME EXAMPLES

| homogeneous<br>e.g. vector v of values | heterogeneous<br>e.g. type list <typename... Ts> |
|---|---|
| count(v) → size_t | size...(Ts) → size_t |
| count_if(v, pred) → size_t | (pred<Ts> + ... + 0) → size_t |
| count_if(v, pred, proj) → size_t | (pred<proj<Ts>> + ... + 0) → size_t |
| for_each(v, func) → void | (func<Ts>, ...) → void |
| for_each(v, func, proj) → void | (func<proj<Ts>>, ...) → void |
| all(v, pred) → bool | (pred<Ts> and ...) → bool |
| any(v, pred) → bool | (pred<Ts> or ...) → bool |
| none(v, pred) → bool | ((not pred<Ts>) and ...) → bool |

# 1️⃣ FIND_IF

Something like 'std::ranges::find_if' , but on a type list

```cpp
static constexpr auto noIndex = std::size_t{ 0 } - 1;

template <typename T>                                    // the predicate,  T -> bool
struct isMarkedAsPlural {                                // needs to be wrapped in a class for reasons
    constexpr operator bool() const noexcept { return std::is_same_v<plural, T>; }
};

template <typename... Types>
struct format_string_translator {
    static constexpr auto numPluralArguments = (isMarkedAsPlural<Types>{} + ... + 0);
    static_assert(numPluralArguments <= 1, "Oops, more than one plural argument was found!");

    static constexpr auto PluralIndex = findFirstIndex<isMarkedAsPlural, Types...>();
    static_assert(numPluralArguments > 0 ? PluralIndex < sizeof...(Types) : PluralIndex == noIndex);
};
```

🙄

## FINDFIRSTINDEX

```cpp
static constexpr auto noIndex = std::size_t{ 0 } - 1;

template <typename T>
struct isMarkedAsPlural {
    constexpr operator bool() const noexcept { return std::is_same_v<plural, T>; }
};


constexpr std::size_t findFirstIndex() {
    std::size_t Result = noIndex;

    return Result;
}

static_assert(findFirstIndex() == noIndex);   ✅
```

## FINDFIRSTINDEX

```cpp
static constexpr auto noIndex = std::size_t{ 0 } - 1;

template <typename T>
struct isMarkedAsPlural {
    constexpr operator bool() const noexcept { return std::is_same_v<plural, T>; }
};


constexpr std::size_t findFirstIndex() {
    std::size_t Result = noIndex + 1;

    return Result - 1;
}

static_assert(findFirstIndex() == noIndex);   ✅
```

# FINDFIRSTINDEX

```cpp
static constexpr auto noIndex = std::size_t{ 0 } - 1;

template <typename T>
struct isMarkedAsPlural {
    constexpr operator bool() const noexcept { return std::is_same_v<plural, T>; }
};

template <template <typename T> typename Predicate, typename... Types>
constexpr std::size_t findFirstIndex() {
    std::size_t Result = noIndex + 1;

    return (false or ... or Predicate<Types>{}),
           Result - 1;
}

static_assert(findFirstIndex<isMarkedAsPlural>()          ==  noIndex);        ✅
static_assert(findFirstIndex<isMarkedAsPlural, void>()  ==  noIndex);        ✅
static_assert(findFirstIndex<isMarkedAsPlural, plural, void>() == noIndex);  ❌
static_assert(findFirstIndex<isMarkedAsPlural, void, plural>() == noIndex);  ❌
```

must be a type because of http://eel.is/c++draft/temp.arg.template

# FINDFIRSTINDEX

```cpp
static constexpr auto noIndex = std::size_t{ 0 } - 1;

template <typename T>
struct isMarkedAsPlural {
    constexpr operator bool() const noexcept { return std::is_same_v<plural, T>; }
};

template <template <typename T> typename Predicate, typename... Types>
constexpr std::size_t findFirstIndex() {
    std::size_t Result = noIndex + 1;

    return (false or ... or (Predicate<Types>{} ? true : false)),
           Result - 1;
}

static_assert(findFirstIndex<isMarkedAsPlural>()         ==  noIndex);         ✅
static_assert(findFirstIndex<isMarkedAsPlural, void>()   ==  noIndex);         ✅
static_assert(findFirstIndex<isMarkedAsPlural, plural, void>() == noIndex);  ❌
static_assert(findFirstIndex<isMarkedAsPlural, void, plural>() == noIndex);  ❌
```

# FINDFIRSTINDEX

```cpp
static constexpr auto noIndex = std::size_t{ 0 } - 1;

template <typename T>
struct isMarkedAsPlural {
    constexpr operator bool() const noexcept { return std::is_same_v<plural, T>; }
};

template <template <typename T> typename Predicate, typename... Types>
constexpr std::size_t findFirstIndex() {
    std::size_t Result = noIndex + 1;
    std::size_t Index  = 0;

    return (false or ... or (++Index, Predicate<Types>{} ? true : false)),
           Result - 1;
}

static_assert(findFirstIndex<isMarkedAsPlural>()        ==  noIndex);        ✅
static_assert(findFirstIndex<isMarkedAsPlural, void>()  ==  noIndex);        ✅
static_assert(findFirstIndex<isMarkedAsPlural, plural, void>() == noIndex);  ❌
static_assert(findFirstIndex<isMarkedAsPlural, void, plural>() == noIndex);  ❌
```

# FINDFIRSTINDEX

```
static constexpr auto noIndex = std::size_t{ 0 } - 1;

template <typename T>
struct isMarkedAsPlural {
    constexpr operator bool() const noexcept { return std::is_same_v<plural, T>; }
};

template <template <typename T> typename Predicate, typename... Types>
constexpr std::size_t findFirstIndex() {
    std::size_t Result = noIndex + 1;
    std::size_t Index  = 0;

    return (false or ... or (++Index, Predicate<Types>{} ? static_cast<bool>(Result = Index) : false)),
           Result - 1;
}

static_assert(findFirstIndex<isMarkedAsPlural>()          ==  noIndex);  ✅
static_assert(findFirstIndex<isMarkedAsPlural, void>()  ==  noIndex);  ✅
static_assert(findFirstIndex<isMarkedAsPlural, plural, void>() == 0);  ✅
static_assert(findFirstIndex<isMarkedAsPlural, void, plural>() == 1);  ✅
```

## 2️⃣ INDEX ARGUMENT

```
template <typename... Types>
std::string format(const format_string_translator<Types...> XFmt, Types &&... Args) {
    const auto Quantity = XFmt.Quantity(
       { reinterpret_cast<uintptr_t>(std::addressof(Args))... }
    );
}

template <typename... Types>
struct format_string_translator {
    static constexpr auto PluralIndex = findFirstIndex<isMarkedAsPlural, Types...>();

    static plural::type Quantity(const uintptr_t (&pArgs)[]) noexcept {
        if constexpr (PluralIndex != noIndex)
            return *std::bit_cast<const plural *>(pArgs[PluralIndex]);
        else
            return 1;
    }
};
```

This is valid code, no undefined behaviour here! 😊

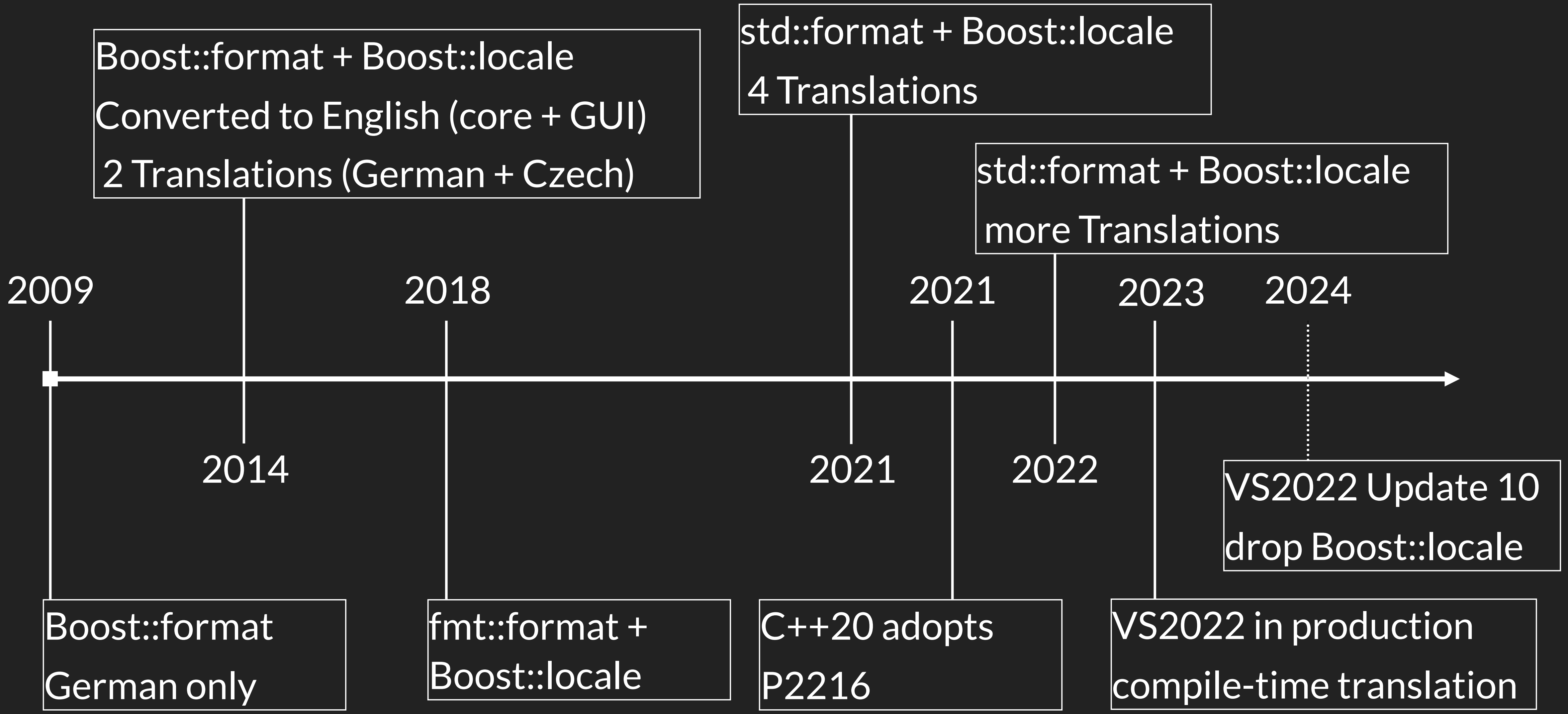
# 3️⃣ TYPE-BASED SELECTION

a.k.a. pattern-matching (P2688)

```cpp
template <typename T>
struct wrapped {
    using type = T;
    static constexpr const type & translate(const T & t) { return t; }
};

template <typename Char>
struct wrapped<tTranslate<Char>> {
    using type = std::basic_string_view<Char>;
    static constexpr type translate(const tBaseTranslate<Char> & t) { return t.single(); }
};

template <>
struct wrapped<plural> {
    using type = plural::type;
    static constexpr type translate(const plural & t) { return t; }
};
```

## an experiment

# TIMELINE


2009
Boost::format
German only
2014
Boost::format + Boost::locale
Converted to English (core + GUI)
2 Translations (German + Czech)
2018
fmt::format +
Boost::locale
2021
std::format + Boost::locale
4 Translations
2021
C++20 adopts
P2216
2022
std::format + Boost::locale
more Translations
2023
VS2022 in production
compile-time translation
2024
VS2022 Update 10
drop Boost::locale

## INTEGRATION TEST

```
constexpr auto Test(
    const auto Source,
    const uint64_t Cardinal   = 1'000'000,
    const wstring_view Result = L"Language Form 3") {
    const auto Tr = tTranslator(TranslationDomain("Integration"))
                  .load(LanguageId("br"), Source);
    return Result ==
               translate("IntegrationContext", L"Singular", L"Plural")
               ._(Cardinal, Tr);
}
```

# INTEGRATION TEST

Compiler

```cpp
import Translate; // library implementation of the 'gettext' facilities
import utility;

static constexpr auto BretonConst = utility::embed(
    #include "br.bin"   // bring in the Breton translations (≈ 500)
);                      // from the compiled .mo

constexpr auto Test(
    const auto Source,
    const uint64_t Cardinal   = 1'000'000,
    const wstring_view Result = L"Language Form 3") {
    const auto Tr = tTranslator(TranslationDomain("Integration"))
                        .load(LanguageId("br"), Source);
    return Result ==
                translate("IntegrationContext", L"Singular", L"Plural")
                ._(Cardinal, Tr);
}

static_assert(Test(BretonConst));  // a single constant evaluation!

int main() {
//  auto BretonFile = "br.mo";
//  assert(Test(BretonFile));
}
```

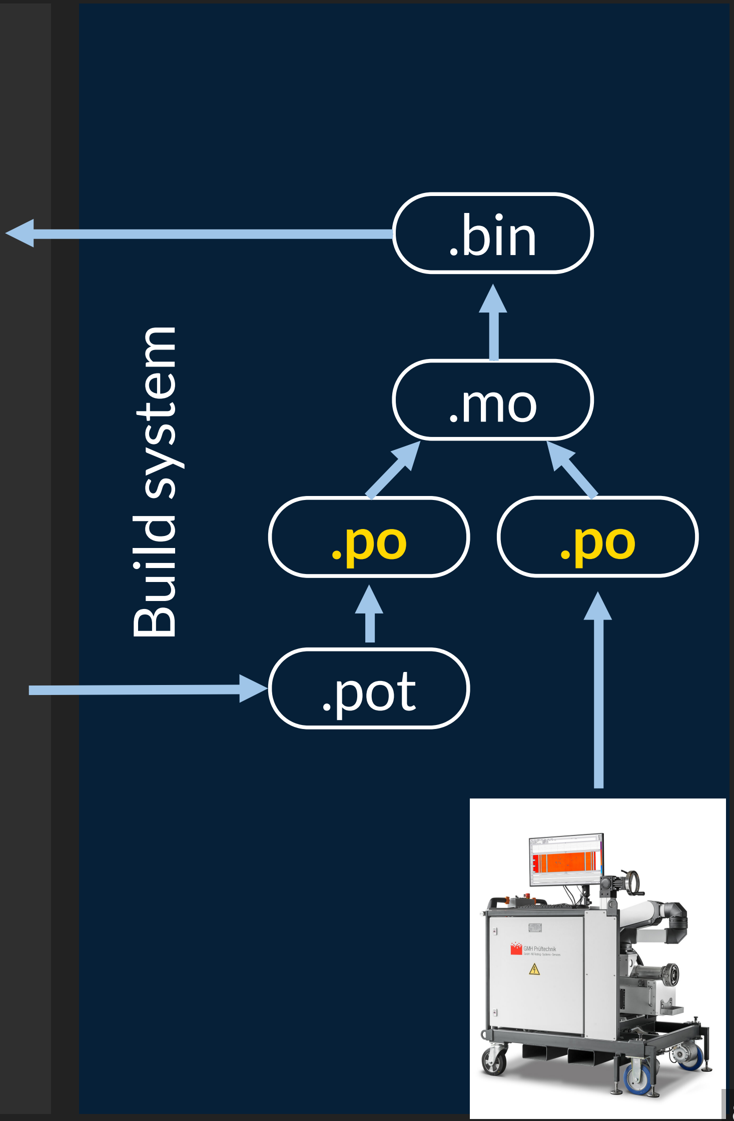
br.po

```
"Project-Id-Version: Integration"
"Language: br"
"Plural-Forms: nplurals=5; plural=
  (n%10==1 && n%100!=11 &&
   n%100!=71 && n%100!=91 ?  0 :
   n%10==2 && n%100!=12 &&
   n%100!=72 && n%100!=92 ?  1 :
  ((n%10>=3 && n%10<=4) ||
    n%10==9) &&
   (n%100<10 || n%100>19) &&
   (n%100<70 || n%100>79) &&
    (n%100<90 || n%100>99) ? 2 :
    n!=0 && n%1000000==0 ?   3 :
                             4);"
 …

#: main.cpp
msgctxt "IntegrationContext"
msgid "Singular"
msgid_plural "Plural"
msgstr[0] "Language Form 0"
msgstr[1] "Language Form 1"
msgstr[2] "Language Form 2"
msgstr[3] "Language Form 3"
msgstr[4] "Language Form 4"

 …
```

# TEST AT COMPILETIME

Compiler

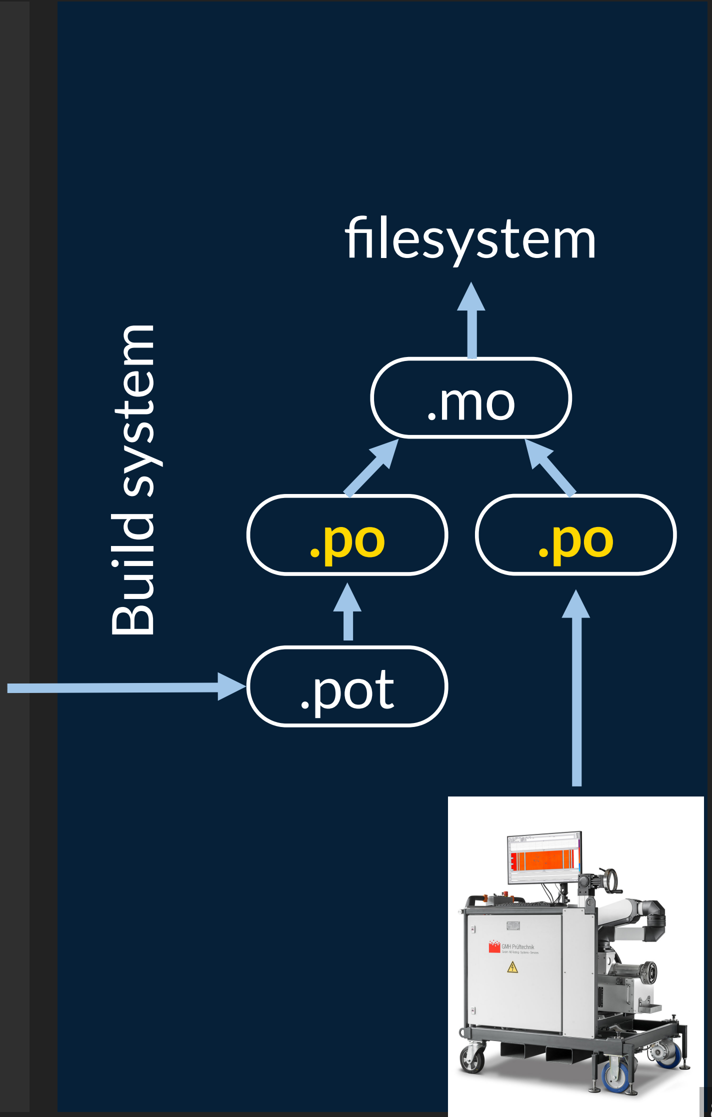
```cpp
import Translate; // library implementation of the 'gettext' facilities
import utility;

static constexpr auto BretonConst = utility::embed(
    #include "br.bin"   // bring in the Breton translations (≈ 500)
);                      // from the compiled .mo

constexpr auto Test(
    const auto Source,
    const uint64_t Cardinal   = 1'000'000,
    const wstring_view Result = L"Language Form 3") {
    const auto Tr = tTranslator(TranslationDomain("Integration"))
                  .load(LanguageId("br"), Source);
    return Result ==
               translate("IntegrationContext", L"Singular", L"Plural")
               ._(Cardinal, Tr);
}

static_assert(Test(BretonConst));  // a single constant evaluation!

int main() {
//  auto BretonFile = "br.mo";
//  assert(Test(BretonFile));
}
```



# TEST AT RUNTIME

Compiler

```cpp
import Translate; // library implementation of the 'gettext' facilities
import utility;

//  static constexpr auto BretonConst = utility::embed(
//      #include "br.bin"
//  );

constexpr auto Test(
    const auto Source,
    const uint64_t Cardinal   = 1'000'000,
    const wstring_view Result = L"Language Form 3") {
    const auto Tr = tTranslator(TranslationDomain("Integration"))
                  .load(LanguageId("br"), Source);
    return Result ==
                translate("IntegrationContext", L"Singular", L"Plural")
                ._(Cardinal, Tr);
}

// static_assert(Test(BretonConst));

int main() {
    auto BretonFile = "br.mo";
    assert(Test(BretonFile));  // map or load .mo
}
```



## move on

# RESOURCES

- Living, up-to-date C++ standard (currently at C++26)
- The Journal of Functional Programming, Cambridge University Press
- GNU gettext utilities
- Unicode CLDR

- Library code github.com/DanielaE/t.b.d.

Contact

dani@ngrt.de
@DanielaKEngert@hachyderm.io
DanielaE

Images: Maria Sibylla Merian (1705)

source: WikiMedia Commons, public domain , Unsplash & GIPHY

Ceterum censeo ABI esse frangendam