Delphi+
czyli łączenie kodu

Zlot Progamistów Delphi
Mszczonów 2022

Tomasz Tyrakowski

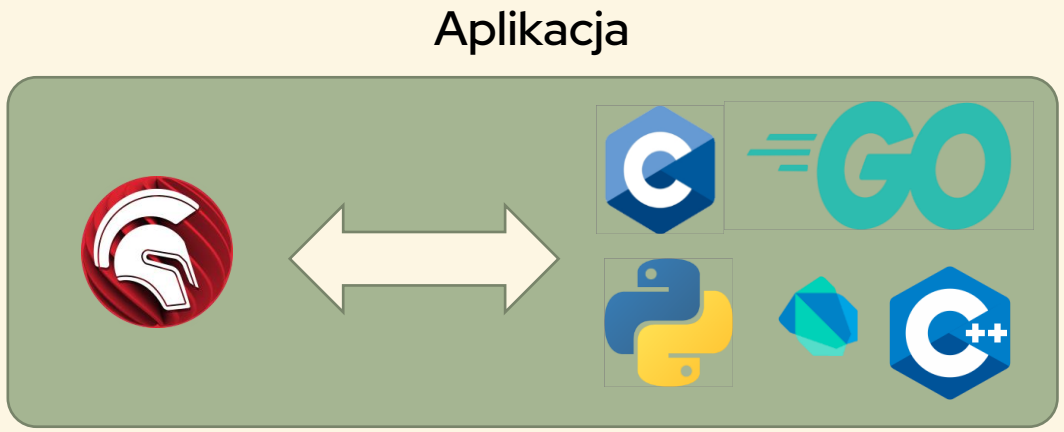
Plan

1. Po co łączyć kod?
2. Co mamy na myśli?
3. Jak łączyć kod?
4. W czym problem?
5. Przykład: C
6. Przykład: C++
7. Przykład: Go
8. Przykład: Python
9. Wnioski, pytania

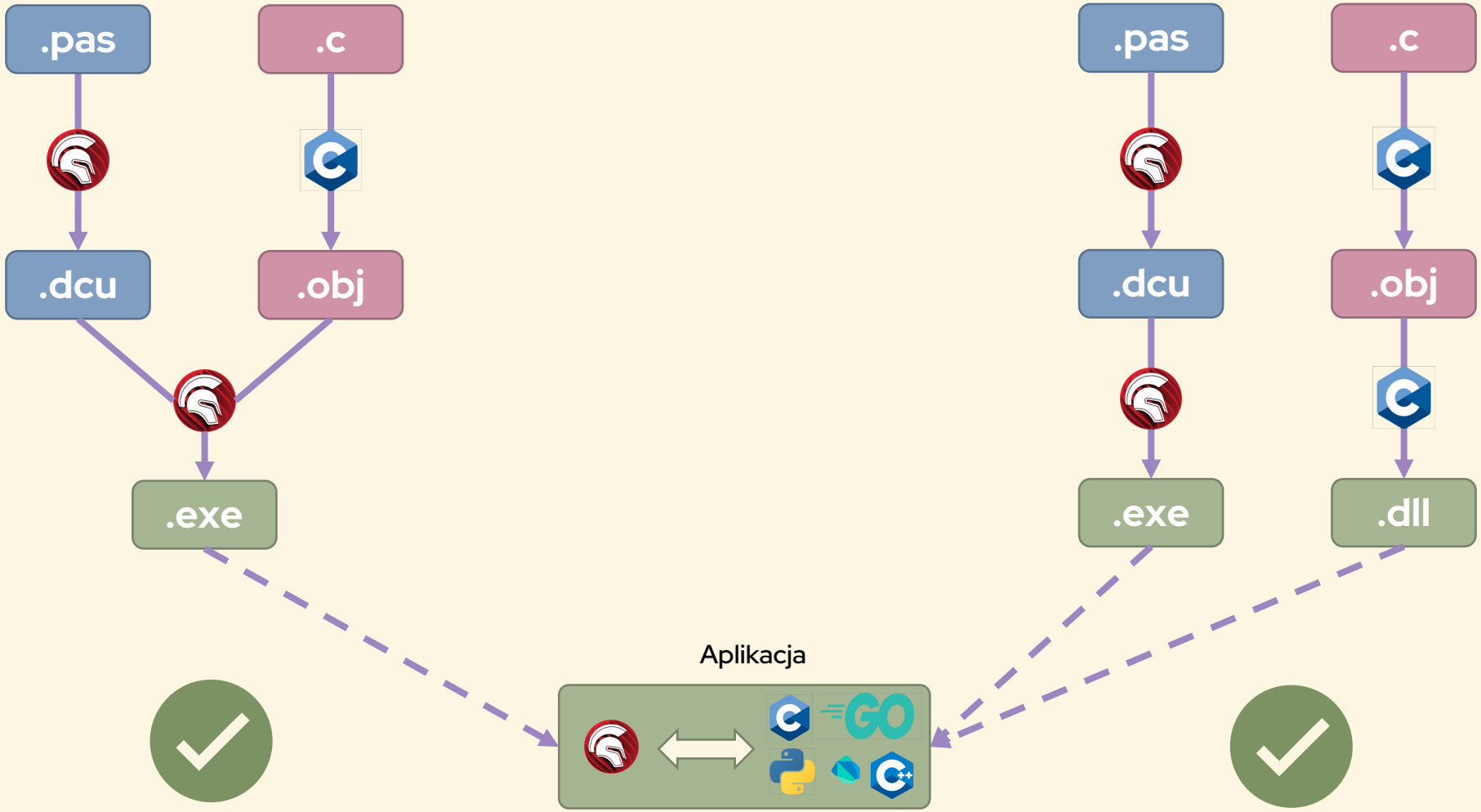
Po co łączyć kod?

1. Zewnętrzna biblioteka zawiera potrzebną nam funkcjonalność (reimplementacja kosztowna / czasochłonna).
2. Chcemy użyć bezpośrednio funkcji systemu operacyjnego.
3. Integrujemy nasze oprogramowanie z innymi systemami / implementujemy fragment większego systemu.
4. Inny język zawiera mechanizmy ekspresji lepiej nadające się do rozwiązania naszego problemu.
5. Nie mamy wystarczającej liczby programistów znających Delphi.

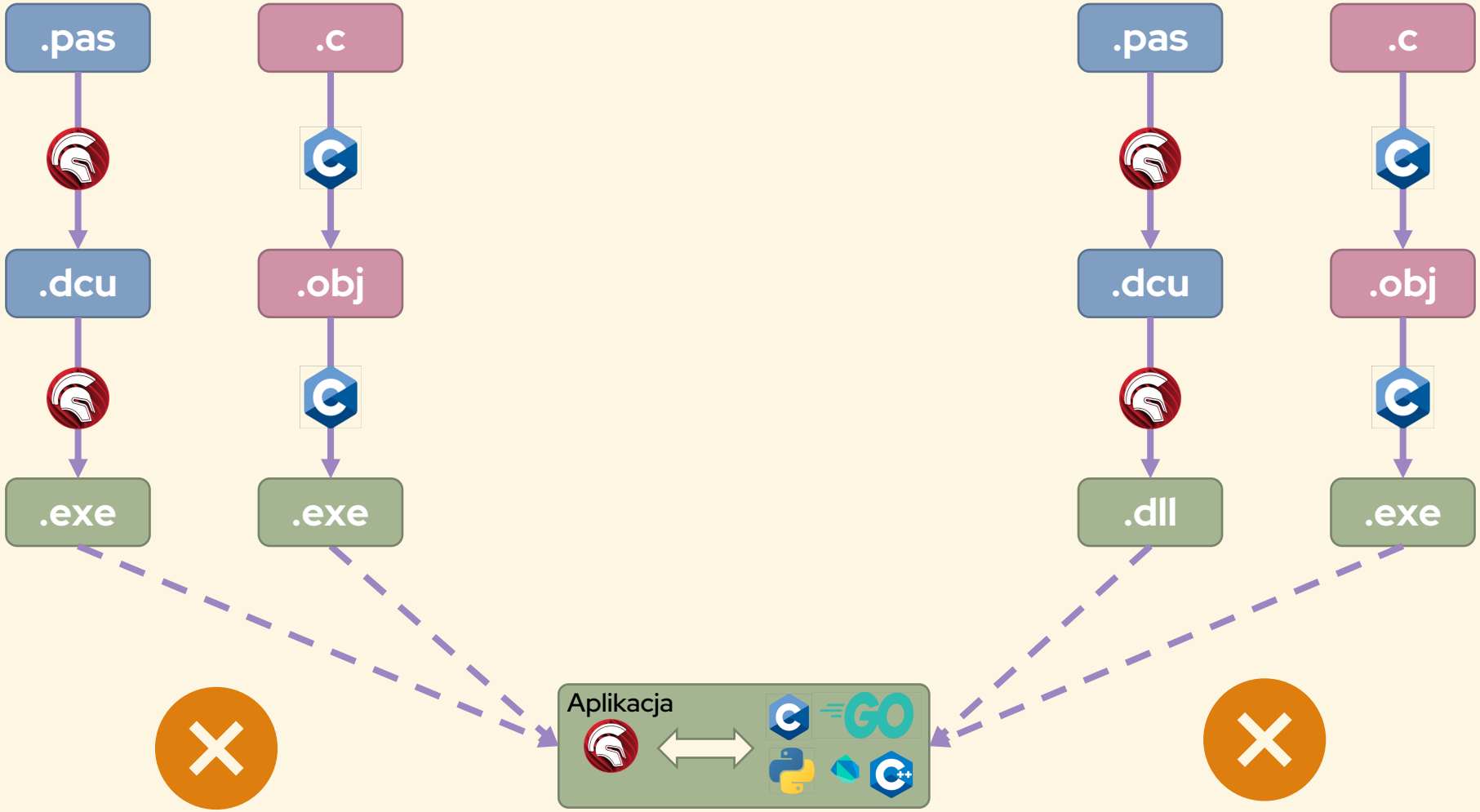
Co mamy na myśli?



Co mamy na myśli?



Czego nie mamy na myśli?



Co mamy na myśli?

- Aplikacja w Delphi, obcy kod dostarcza bibliotekę użytecznych funkcji
  - W postaci OBJ linkowanych z kodem w Delphi
  - W postaci DLL ładowanych w runtime przez kod w Delphi

- Aplikacja w innym języku, Delphi dostarcza bibliotekę funkcji

- Kod Delphi i obcy jako odrębne procesy
  - Komunikacja przez IPC
  - Komunikacja przez sieć

Jak łączyć kod?

- Na etapie kompilacji:
  - Kompilacja obcego kodu do postaci pośredniej (.obj)
  - Deklaracja funkcji w Delphi jako `external`
  - Konsolidacja do wynikowego pliku wykonywalnego `{$L *.obj}`
  - .obj w formacie OMF lub COFF (MSVC/Clang/GCC)
  - Problem: brak obcego runtime
    Rozwiązanie (częściowe) dla C (MSVCRT): `System.Win.Crtl`
- W runtime:
  - Kompilacja obcego kodu do postaci DLL (własny runtime)
  - Deklaracja funkcji w Delphi jako `external 'libxx.dll'`
  - Alternatywnie: `LoadLibrary`, `GetProcAddress`

Łączenie w czasie kompilacji

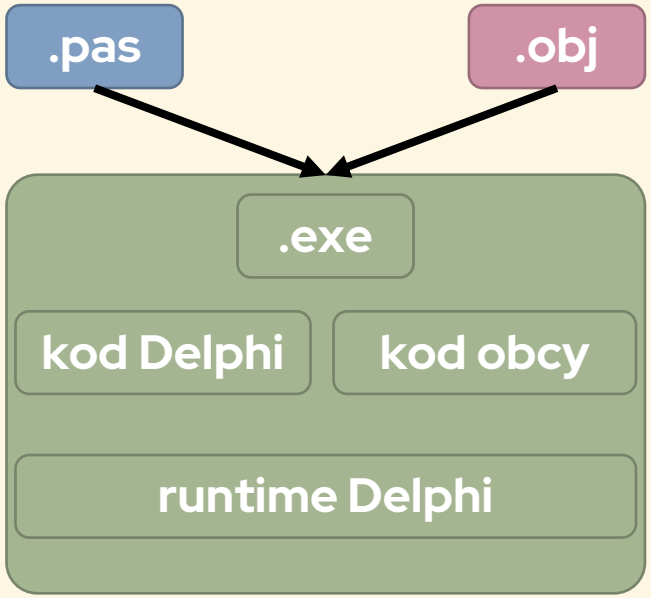
```
uses System.Win.Crtl;

// c_code.c: char* __cdecl dyn_greet() { … malloc() … }

function hello(): PAnsiChar; cdecl; external name '_dyn_greet';

{$L 'c_code.obj'}

begin
    var c: PAnsiChar := hello();
    writeln(c);
    free(c); // from System.Win.Crtl
end.
```

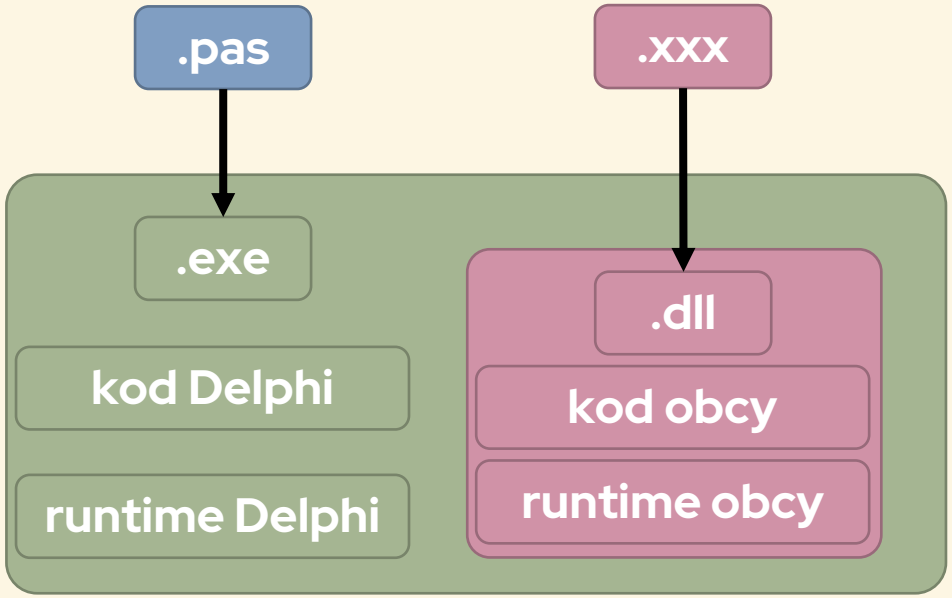
Łączenie w runtime

```
// c_code.c: __declspec(dllexport) char* __cdecl dyn_greet() { … malloc() … }
// c_code.c: __declspec(dllexport) void __cdecl lib_free(void* p) { … free(p) … }

function hello(): PAnsiChar;
cdecl; external 'c_code.dll' name 'dyn_greet';

procedure c_free(p: Pointer);
cdecl; external 'c_code.dll' name 'lib_free';

begin
    var c: PAnsiChar := hello();
    writeln(c);
    c_free(c); // from System.Win.Crtl
end.
```

+ delayed

Runtime Delphi i obcy





Powiemy to tylko raz...

Nie można łączyć binarnego kodu 32-bitowego z 64-bitowym.

Nie można łączyć binarnego kodu kompilowanego dla różnych platform (CPU/OS).

W czym problem? w ABI

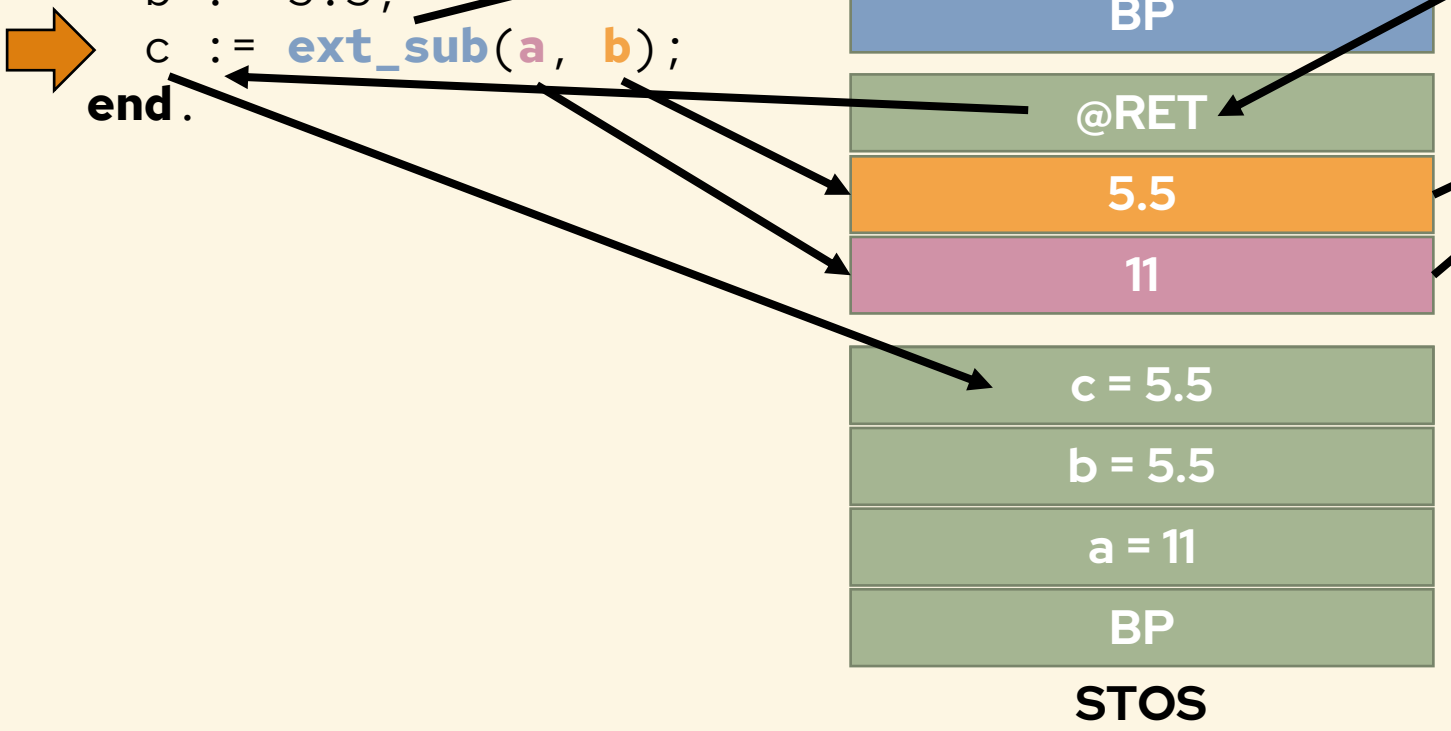
ABI – Application Binary Interface

1. Konwencja wywołań
   - stos / rejestry
   - kolejność parametrów
   - przekazanie wyniku
   - sprzątanie stosu
   - przekształcenia nazw funkcji (name mangling)
2. Binarna reprezentacja danych
   - liczby
   - tekst
   - tablice, struktury
3. Zarządzanie pamięcią dynamiczną
   - zwalnianie pamięci alokowanej przez obcy runtime

Wywołanie funkcji

```
var a: Integer;
var b,c: Double;
begin
  a := 11;
  b := 5.5;
  c := ext_sub(a, b);
end.
```

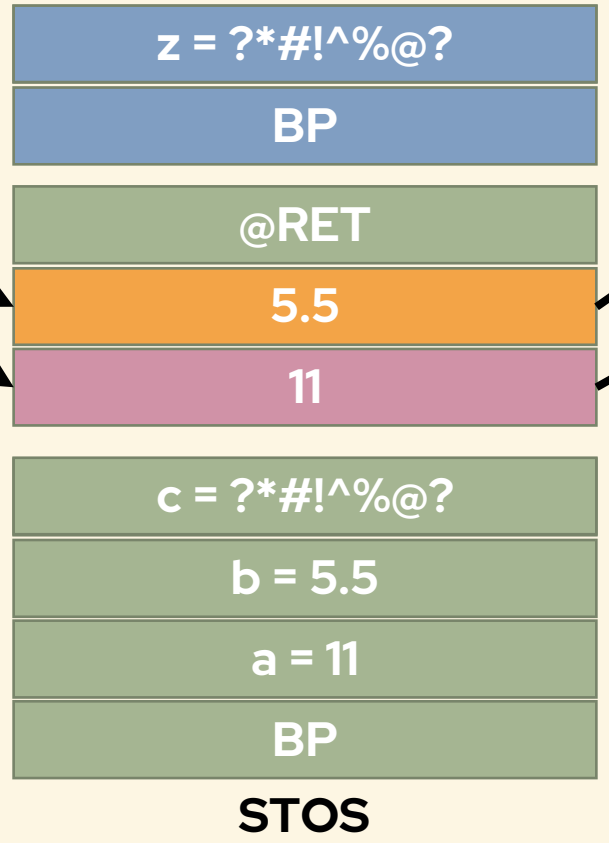
```
double ext_sub(int x, double y) {
  double z = x - y;
  return z;
}
```



Wywołanie funkcji – kolejność

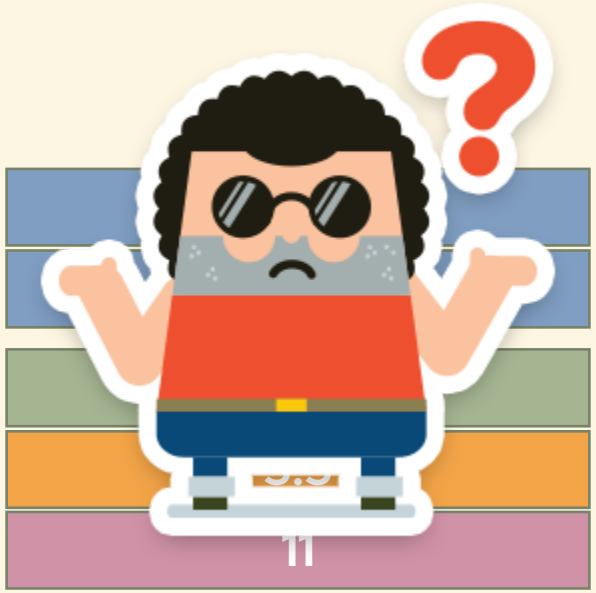
```
var a: Integer;
var b,c: Double;
begin
  a := 11;
  b := 5.5;
  c := ext_sub(a, b);
end.
```

```
double ext_sub(int x, double y) {
  double z = x - y;
  return z;
}
```

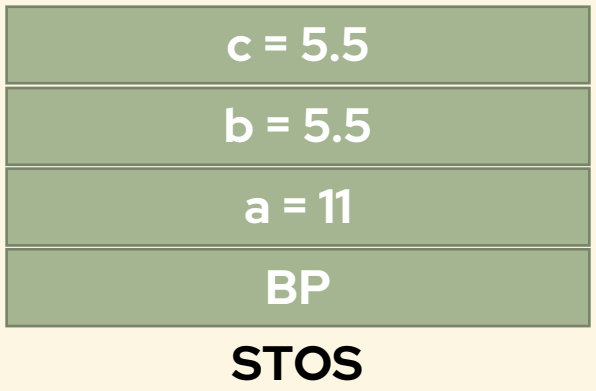


Wywołanie funkcji – sprzątanie

```
var a: Integer;
var b,c: Double;
begin
  a := 11;
  b := 5.5;
  c := ext_sub(a, b);
end.
```




```
double ext_sub(int x, double y) {
  double z = x - y;
  return z;
}
```



STOS

Konwencje wywołań

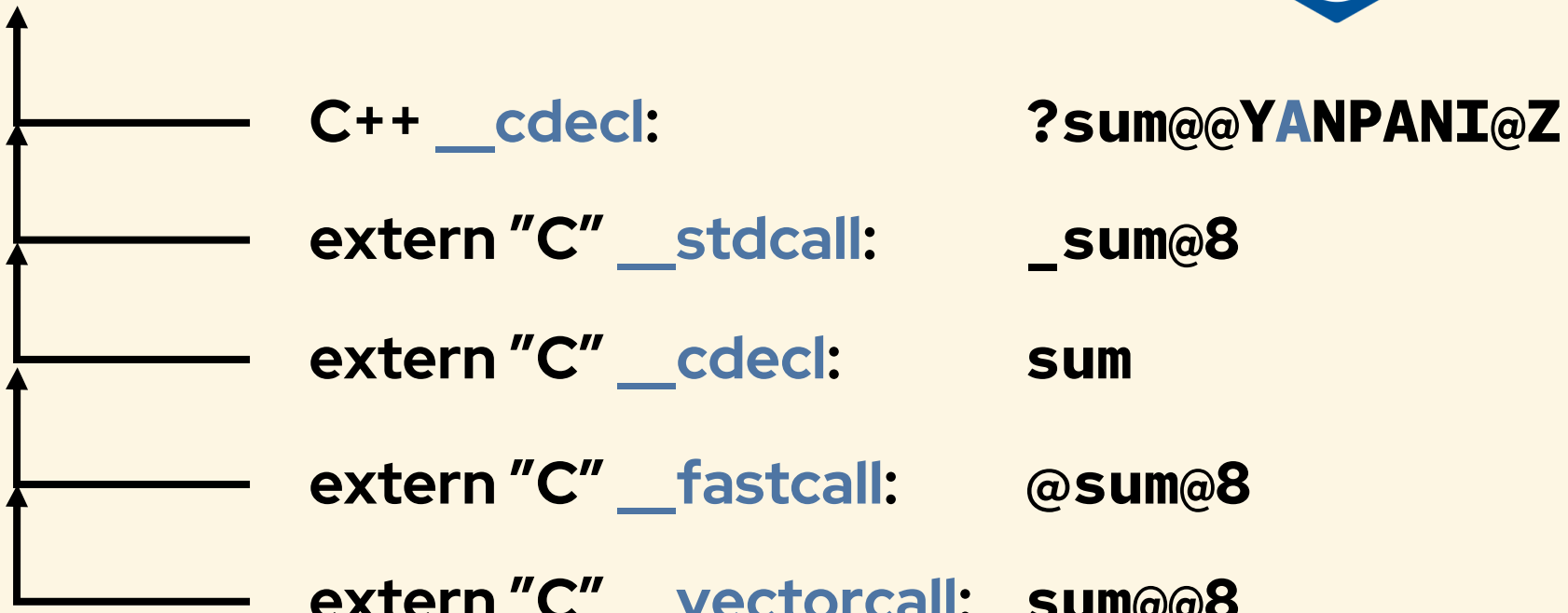
Konwencja	Sprząta	Kolejność parametrów
`cdecl`	wywołujący	od prawej do lewej
`stdcall`	wywoływany	od prawej do lewej
`fastcall`	wywoływany	przez rejestry, nadmiarowe przez stos
`vectorcall`	wywoływany	przez rejestry (SSE2), nadmiarowe przez stos
`thiscall`	wywoływany	dodatkowo `this` w rejestrze ECX/RCX

https://learn.microsoft.com/en-us/cpp/cpp/argument-passing-and-naming-conventions

- **API Win32: `stdcall`**
- **API Win64: `cdecl`**

Nazwy funkcji (name mangling)

```
double sum(double *a, unsigned int size);
```



Plik .DEF w czasie linkowania DLL rozwiązuje ten problem.

Reprezentacja danych - liczby

Delphi	C
ShortInt	char
Byte	unsigned char
Integer	int
Int64	long long
Word	unsigned short
Cardinal	unsigned int
NativeInt	int, __int64
NativeUint	size_t

Delphi	C
Single	float
Real (=Double)	double
Double	double
Extended	long double
Pointer	void*
PAnsiChar	char*
PWideChar	wchar_t*
WideChar	wchar_t

https://docwiki.embarcadero.com/RADStudio/Alexandria/en/Delphi_to_C++_types_mapping

Reprezentacja danych - tekst

`ShortString`

`UnicodeString, AnsiString`

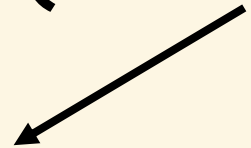
`WideString`

https://docwiki.embarcadero.com/RADStudio/Alexandria/en/Internal_Data_Formats_(Delphi)

`char*, wchar_t*`
wskaźnik na pierwszy znak

Reprezentacja danych - tekst

```
var s: string := 'very important text';

c_func_a(PAnsiChar(s));

c_func_w(PWideChar(s));

c_func_utf(PAnsiChar(UTF8Encode(s)));
```



RawByteString

```
void c_func_a(char *arg)
```

```
void c_func_w(wchar_t *arg)
```

```
void c_func_utf(char *arg)
```

Reprezentacja danych - tablice

```
array [x..y] of some_type
```


```
array of some_type
```

https://docwiki.embarcadero.com/RADStudio/Alexandria/en/Internal_Data_Formats_(Delphi)

```
some_type*
// wskaźnik na pierwszy element
```

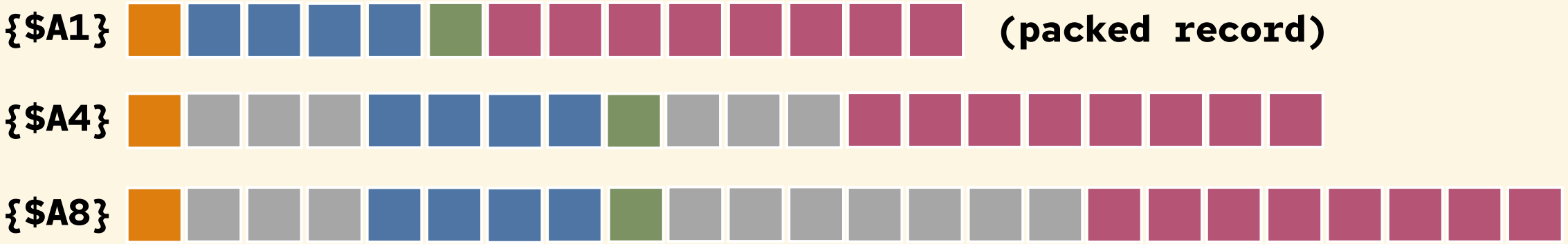

```
var a: array of Integer;
…
c_func(@a[0], Length(a));
```

```
void c_func(int* a, int len)
```

Reprezentacja danych - struktury

```
type structure = record
  b_field1: Byte;
  i_field:  Integer;
  b_field2: Byte;
  d_field:  Double;
end;
```

```
typedef struct {
  unsigned char   b_field1;
  int             i_field;
  unsigned char   b_field2;
  double          d_field;
} structure;
```

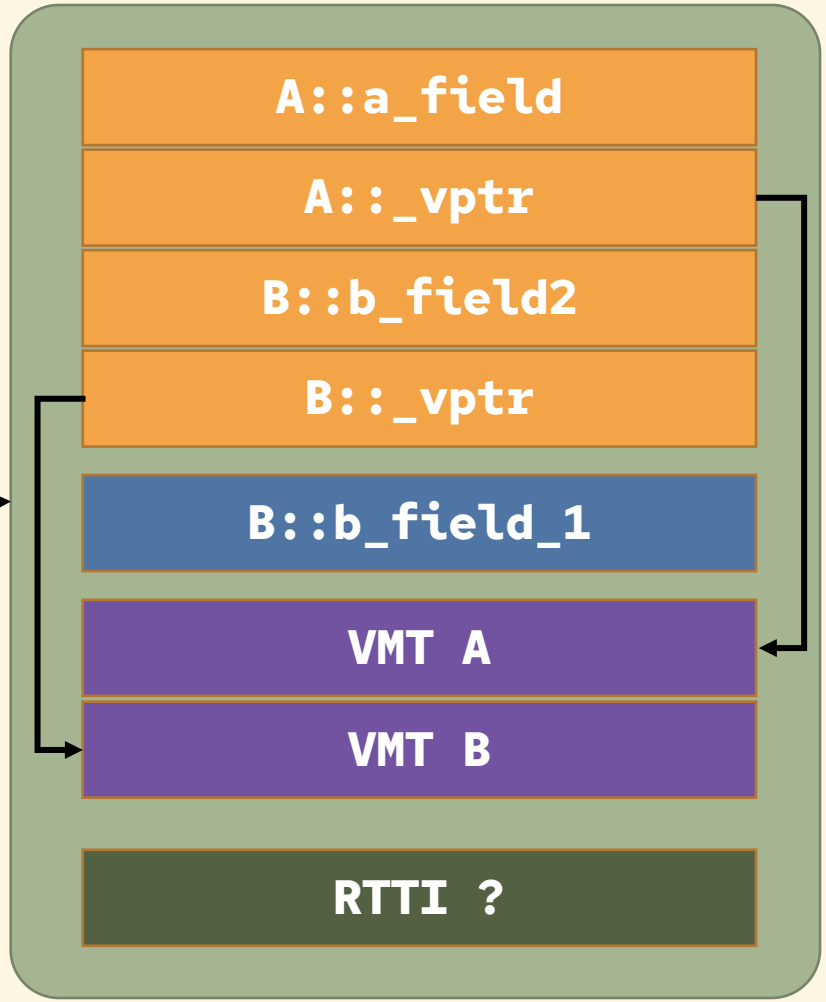


https://docwiki.embarcadero.com/RADStudio/Alexandria/en/Align_fields_(Delphi)
https://en.wikipedia.org/wiki/Data_structure_alignment

Reprezentacja danych – klasy

```
class A {
   int a_field;
   virtual void method1();
}

class B : A {
   static int b_field1;
   int b_field2;

   virtual void method1();
}
```

B obj;



Reprezentacja danych - klasy 

- Reprezentacja obiektów w pamięci nie jest standaryzowana:
  - Różna w różnych językach
  - Różna w różnych kompilatorach tego samego języka
- Konieczny kod pośredni, konwertujący reprezentację obiektową na kombinacje wskaźników i funkcji, czyli na ABI C (np. qt4pas)
- Wyjątkiem jest C++ Builder: (CPPCast 16.04.2021, David Millington, Embarcadero) Specjalne modyfikacje w CLang / LLVM zapewniające kompatybilność klas z Delphi

Reprezentacja danych - klasy

Przykład: qt4pas / qcalendarwidget_c.cpp (+ drobne korekty)

Natywny C++:

```
QDate d = qcal.selectedDate();
```

```
extern "C"__declspec(dllexport)
void __cdecl QCalendarWidget_selectedDate(void* qcal, void* retval)
{
    *(QDate *)retval = ((QCalendarWidget *)qcal)->selectedDate();
}
```

Nieobiektowy C:

```
void* d = QDate_create(); // qdatetime_c.cpp
QCalendarWidget_selectedDate(qcal, d);
// do something with d
QDate_destroy(d);
```

Reprezentacja danych - klasy

Przykład: qt4pas / qcalendarwidget_c.cpp (+ drobne korekty)

Natywny C++:

```
QDate d = qcal.selectedDate();
```

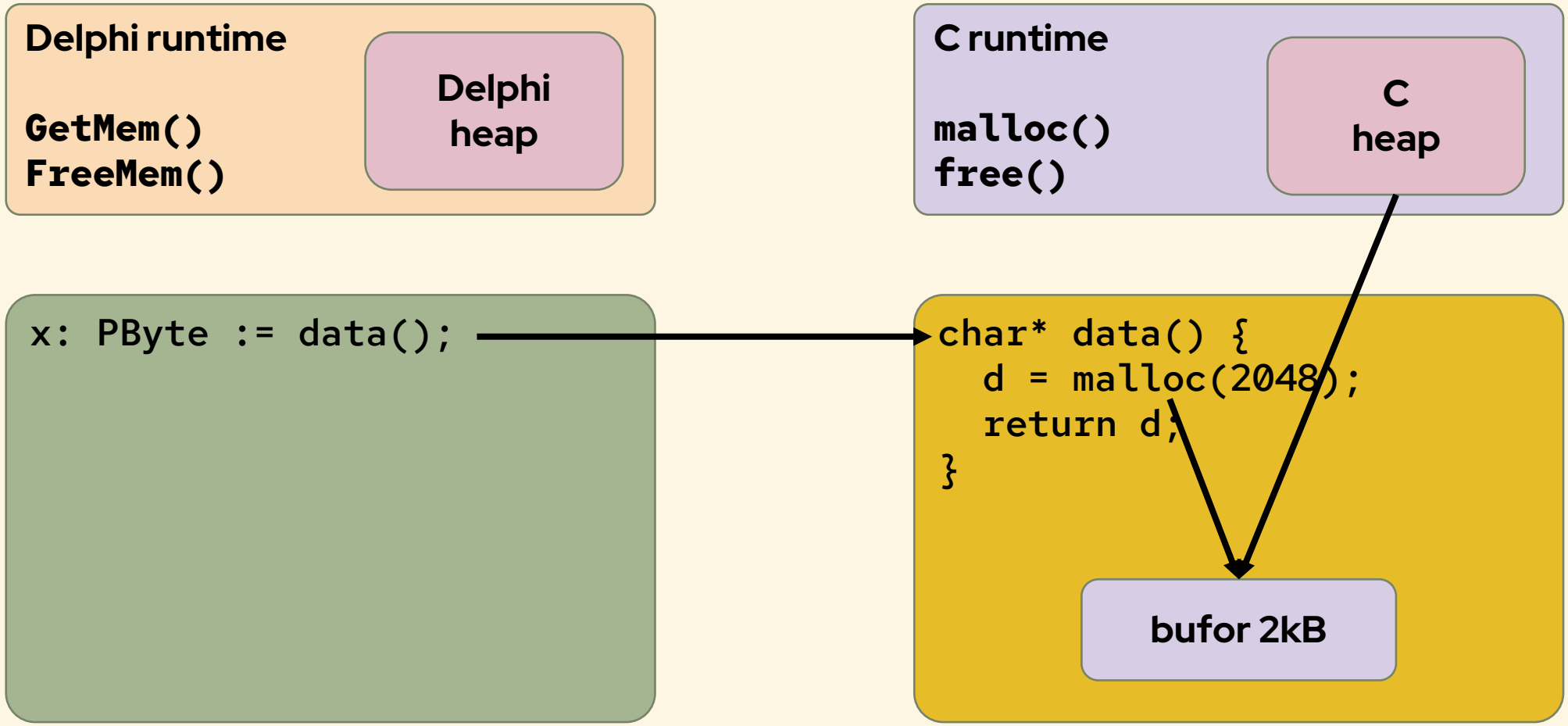
Nieobiektowy C:

```
void* d = QDate_create(); // qdatetime_c.cpp
QCalendarWidget_selectedDate(qcal, d);
// do something with d
QDate_destroy(d);
```

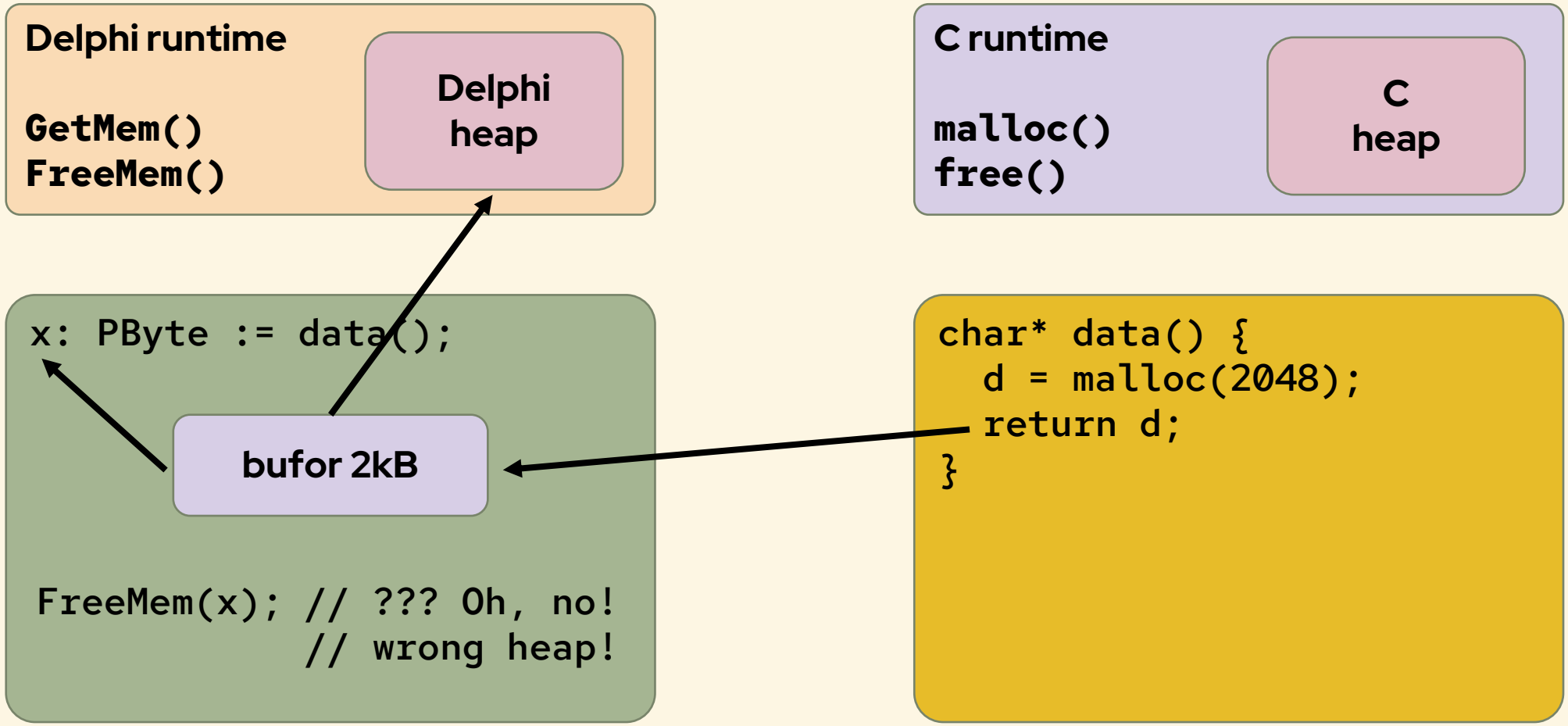
Delphi:

```
var d: Pointer := QDate_create();
QCalendarWidget_selectedDate(qcal, d);
// do something with d
QDate_destroy(d);
```

Zarządzanie pamięcią



Zarządzanie pamięcią



Zarządzanie pamięcią

Delphi runtime

GetMem()
FreeMem()

Delphi heap

C runtime

malloc()
free()

C heap

```
x: PByte := data();

        bufor 2kB

free_data(x);
```

```
char* data() {
  d = malloc(2048);
  return d;
}

void free_data(void* d) {
  free(d);
}
```

Zarządzanie pamięcią

Delphi runtime

GetMem()
FreeMem()

Delphi heap

C runtime

malloc()
free()

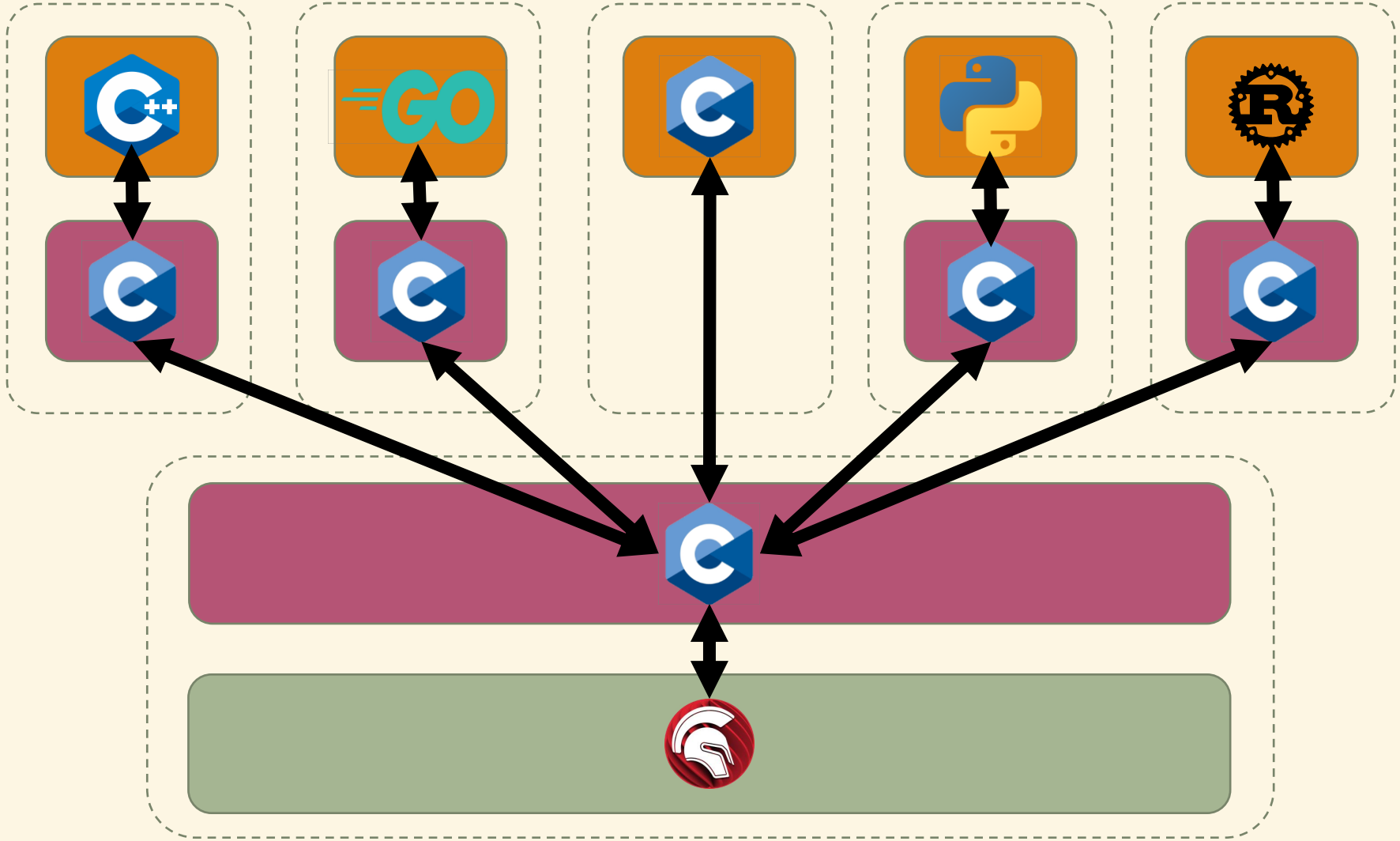
C heap

```
x: PByte := data();




free_data(x);
```

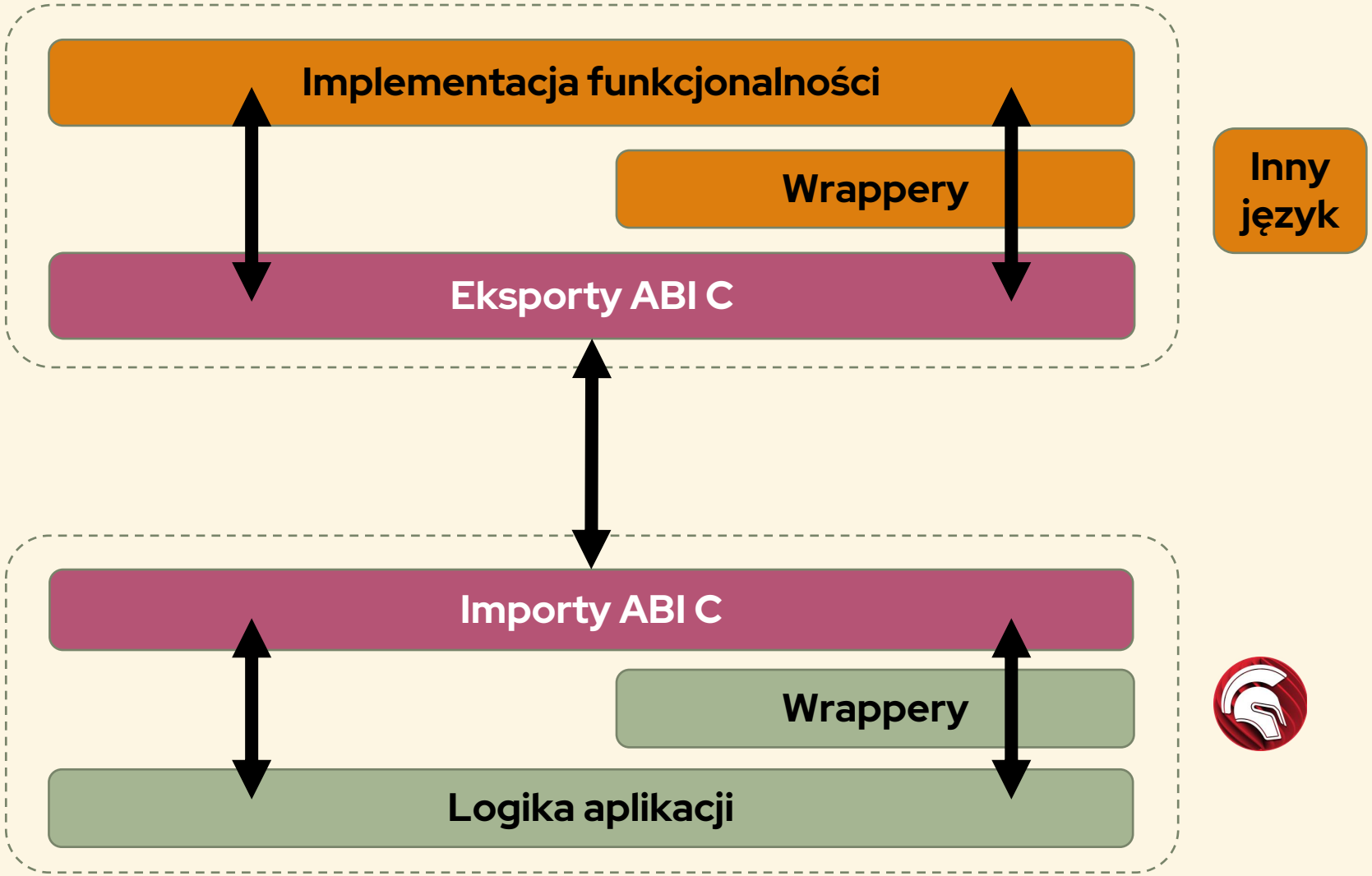
```
char* data() {
  d = malloc(2048);
  return d;
}

void free_data(void* d) {
  free(d);
}
```

bufor 2kB

ABI C jako lingua franca



ABI C jako lingua franca



Przykład: C

Zapisywanie grafiki w formacie WebP.

https://developers.google.com/speed/webp/

https://github.com/webmproject/libwebp/

Przykład: C++

Usuwanie duplikatów z tablicy.

Algorytm:

Idiom w Pythonie: `deduped_list = list(set(list_with_duplicates))`

Przykład: Go

Równoległe obliczenie sumy SHA-512 dla wszystkich plików w folderze.

- równoległość: goroutines, waitgroups
- wyliczanie hash'a: `crypto/sha512`
- metapakiet "C"
- komentarze eksportu funkcji: `//export func_name`
- `go build -buildmode=c-shared`
- automatycznie generowany plik nagłówkowy C
- runtime/cgo: handle, przekazywanie danych z/do Go
- C.CString, GoSlice, GoString

Go – więcej informacji

1. Vladimir Vivien: Calling Go Functions from Other Languages
   https://medium.com/learning-the-go-programming-language/calling-go-functions-from-other-languages-4c7d8bcc69bf

2. Go build modes
   https://pkg.go.dev/cmd/go#hdr-Build_modes

3. Dokumentacja CGo (przekazywanie wskaźników C-Go)
   https://pkg.go.dev/cmd/cgo#hdr-C_references_to_Go

Przykład: Python

1. Rozpoznawanie mowy: https://github.com/openai/whisper
2. Importy bezpośrednio z python3.dll (ABI C)
3. Py_IncRef / Py_DecRef: uwaga na pożyczone referencje (sprawdzać dokumentację Python C API)
4. Niektóre pakiety wymagają specjalnego przygotowania po stronie Delphi (np. numpy, pytorch – Math.SetExceptionMask)
5. Możliwe uruchamianie skryptów wprost (PyRun_SimpleString) lub interakcja niskopoziomowa (PyImport_Import, PyObject_GetAttrString, PyObject_Call, ...)
6. Python Windows embeddable package
7. https://docs.python.org/3/using/windows.html

Wnioski

1. ABI C to lingua franca w łączeniu kodu.
2. Czasami zachodzi konieczność pisania wrapperów / adapterów po obu stronach kodu (Delphi i obcego).
3. Znajomość obcego języka i środowiska bywa niezbędna.
4. Łączenie w runtime (DLL) jest wygodniejszym rozwiązaniem.
5. Musimy zapewnić zgodność reprezentacji danych na poziomie binarnym i odpowiednie konwersje.
6. Równolegle działające runtime'y powodują problemy z zarządzaniem pamięcią dynamiczną.
7. Im język „bardziej odległy" od C, tym więcej kodu pośredniego wymaga.
8. Twórcy języka X zwykle ułatwiają wywołania X → C, wywołania C → X bywają bardziej skomplikowane.

Pytania

- Inne języki:
  - Java – JNI / JNA (ABI C, w obie strony, sporo ekstra kodu)
  - Rust – ABI C (w obie strony)
  - .NET – (.NET → C działa, C → .NET - ?)
  - Dart – dart:ffi działa tylko w jedną stronę (Dart → C)
  - Kotlin/Native - ABI C (w obie strony)
  - Lua – podobnie jak Python (ABI C)
  - JavaScript – node.js native addons (JS → C++)
  - Ruby – ?
  - Perl – ?
- Źródła przykładów: https://github.com/ttyrakow/zlot22