SIMPLITO
software R&D

NISKOPOZIOMOWE BIBLIOTEKI W NOWOCZESNYCH APLIKACJACH

STANDARDOWY PROGRAM

Program.cpp

```
#include <iostream>

long long fib(int n) {
    if (n == 0) return 0;
    if (n == 1) return 1;
    return fib(n-2) + fib(n-1);
}

int main() {
    int n;
    std::cin >> n;

    long long result = fib(n);

    std::cout << result << std::endl;

    return 0;
}
```

Program

PROGRAM PODZIELONY NA PLIKI

Fib.hpp

```
long long fib(int n);
```

Fib.cpp

```
#include ”Fib.hpp”

long long fib(int n) {
    if (n == 0) return 0;
    if (n == 1) return 1;
    return fib(n-2) + fib(n-1);
}
```

Program.cpp

```
#include <iostream>
#include ”Fib.hpp”

int main() {
    int n;
    std::cin >> n;

    long long result = fib(n);

    std::cout << result << std::endl;

    return 0;
}
```

Program

PROGRAM PODZIELONY NA ZUPEŁNIE OSOBNE MODUŁY

Fib.hpp

```
long long fib(int n);
```

Fib.cpp

```
#include ”Fib.hpp”

long long fib(int n) {
    if (n == 0) return 0;
    if (n == 1) return 1;
    return fib(n-2) + fib(n-1);
}
```

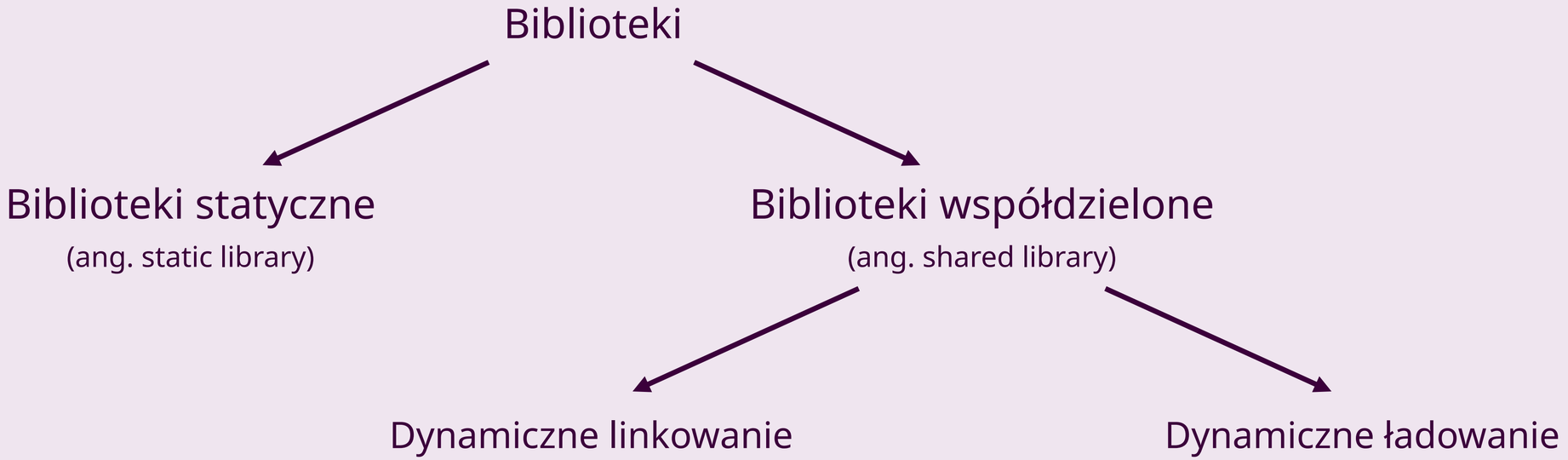
Program.cpp

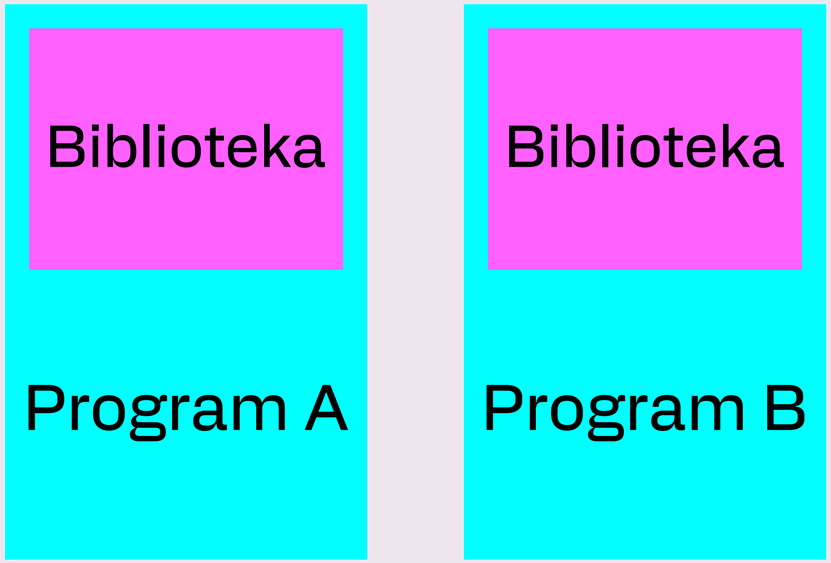
```
#include <iostream>
#include ”Fib.hpp”

int main() {
    int n;
    std::cin >> n;

    long long result = fib(n);

    std::cout << result << std::endl;

    return 0;
}
```

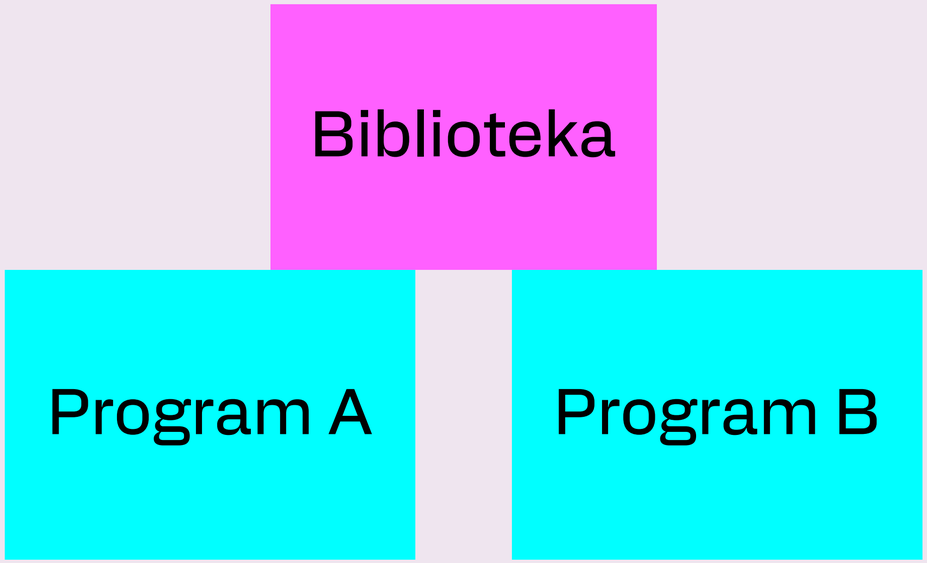
Biblioteka

Program

RODZAJE BIBLIOTEK







ROZSZERZENIA BIBLIOTEK, LOKALIZACJE I KONWENCJE NAZEWNICTWA

	Statyczne biblioteki	Dynamiczne biblioteki	Prefiks w nazwie	Przykłady
Windows	.lib	.dll	brak	Fib.dll, Fib.lib
Linux	.a	.so	lib	libFib.so, libFib.a
macOS	.a	.dylib	lib	libFib.dylib, libFib.a

Dodatkowo w systemie Windows występują też biblioteki importu .lib.

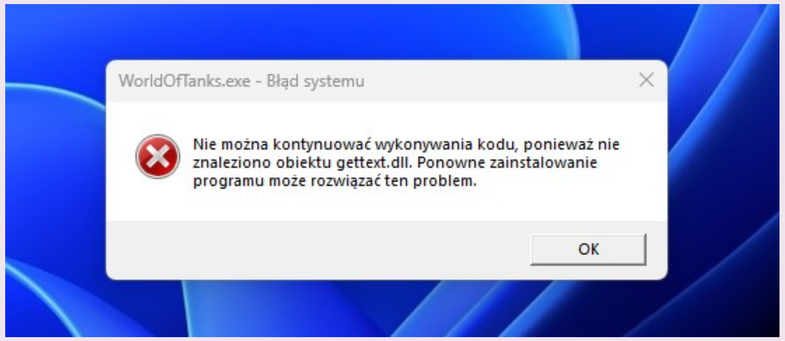
W systemie Linux biblioteki instalowane są w katalogach lib, np. /usr/local/lib.
W systemie Windows biblioteki są instalowane w tych samych katalogach co programy .exe.

PORÓWNANIE

	Statyczne biblioteki	Dynamiczne biblioteki
Łączenie	W procesie kompilacji przez linker	W czasie uruchomienia lub działania programu przez system operacyjny
Rozmiar wynikowego pliku wykonywalnego (programu)	Jest większy, ponieważ biblioteki są wbudowane w program; Usuwany jest kod bez odniesień	Jest mniejszy, ponieważ biblioteki są kopiowane dynamicznie w pamięci
Zmiana w zależnej bibliotece	Cały program musi zostać przebudowany	Nie jest wymagane przebudowanie programu
Kompatybilność	Zawsze kompatybilne, cały kod znajduje się w jednym pliku wykonywalnym	Program zależy od zewnętrznych bibliotek

PROBLEMY

- W systemie Linux dodać ścieżkę z bibliotekami do zmiennej środowiskowej LD_LIBRARY_PATH, w Windows – do PATH
- Przy kompilacji podać ścieżkę z opcją -L
- Dodać do nagłówka biblioteki RUNPATH




```
patryk@meetit-simplito:~/repos/cpp/example4$ ./program
./program: error while loading shared libraries: libfib.so:
 cannot open shared object file: No such file or directory
patryk@meetit-simplito:~/repos/cpp/example4$
```

```
patryk@meetit-simplito:~/repos/cpp/example4$ g++ program.cpp -o program
/usr/bin/ld: /tmp/ccsLcS1G.o: in function `main':
program.cpp:(.text+0x37): undefined reference to `fib(int)'
collect2: error: ld returned 1 exit status
patryk@meetit-simplito:~/repos/cpp/example4$
```

```
patryk@meetit-simplito:~/repos/cpp/example1$ g++ program.cpp -o program -lfake
/usr/bin/ld: cannot find -lfake: No such file or directory
collect2: error: ld returned 1 exit status
patryk@meetit-simplito:~/repos/cpp/example1$
```

DEKOROWANIE NAZW I EXTERN C

Fib.h

```
#ifdef __cplusplus
extern "C" {
#endif

long long fib(int n);

#ifdef __cplusplus
}
#endif
```

Fib.cpp

```
#include ”Fib.h”

long long fib(int n) {
    if (n == 0) return 0;
    if (n == 1) return 1;
    return fib(n-2) + fib(n-1);
}
```

W C++ zostało wprowadzone przeciążanie funkcji ze względu na typy przyjmowanych argumentów. Kompilator wykonuje zamianę deklaracji funkcji na unikalną nazwę, która nazywa się dekorowaniem nazw (ang. name mangling). Symbole te są zrozumiałe dla kompilatora i używane przez linker do łączenia ze sobą plików.

Problem 1: Różne kompilatory używają różnych konwencji.

Problem 2: Różne środowiska mogą wymagać funkcji w języku C.

Rozwiązanie: Eksportowanie funkcji w języku C++ jako funkcji w języku C.

PROCES KOMPILACJI

Preprocesor

Kompilator

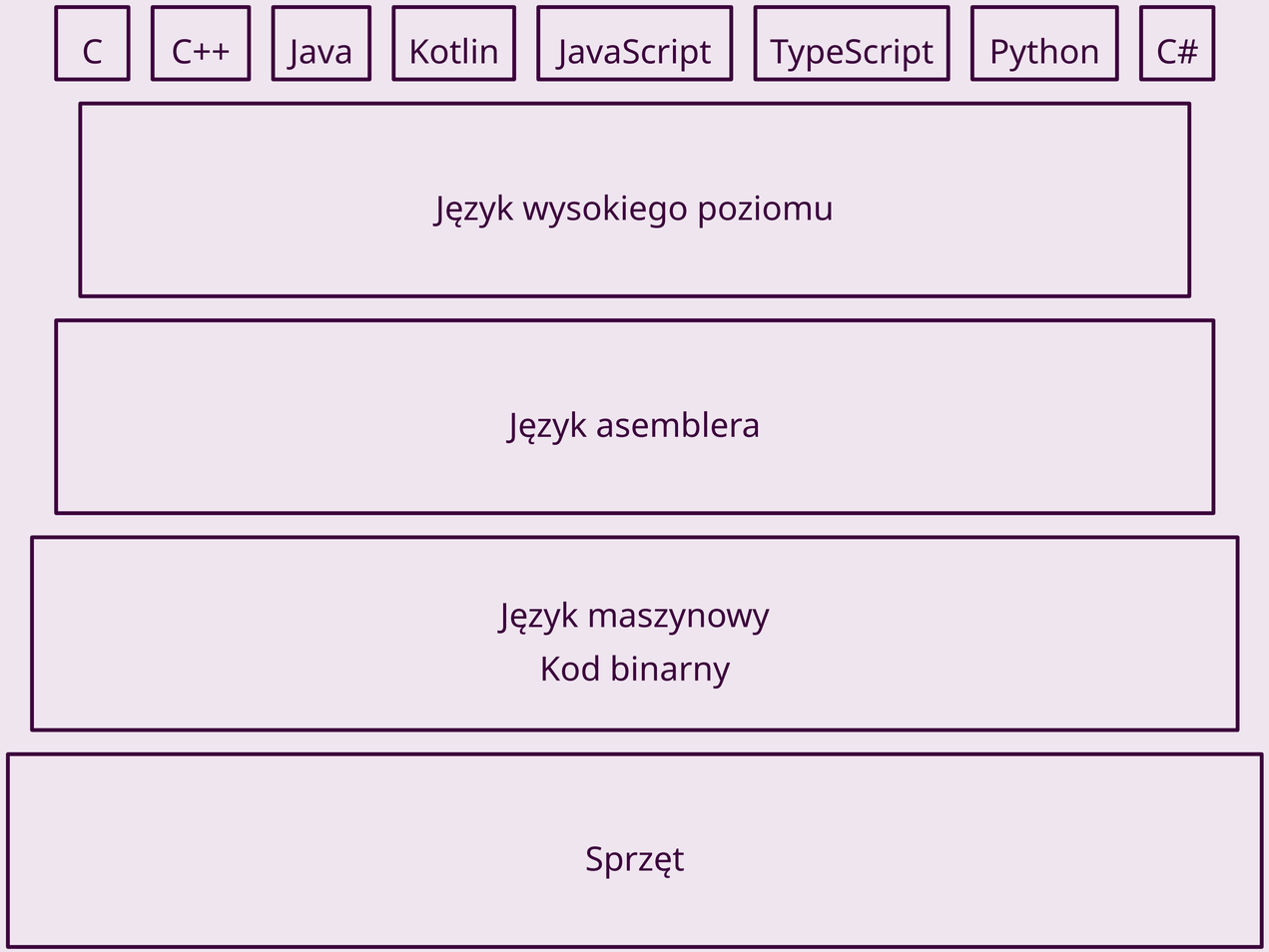
Asembler

(ang. assembler)

Konsolidator

(ang. linker)

TRZY POZIOMY JĘZYKÓW PROGRAMOWANIA



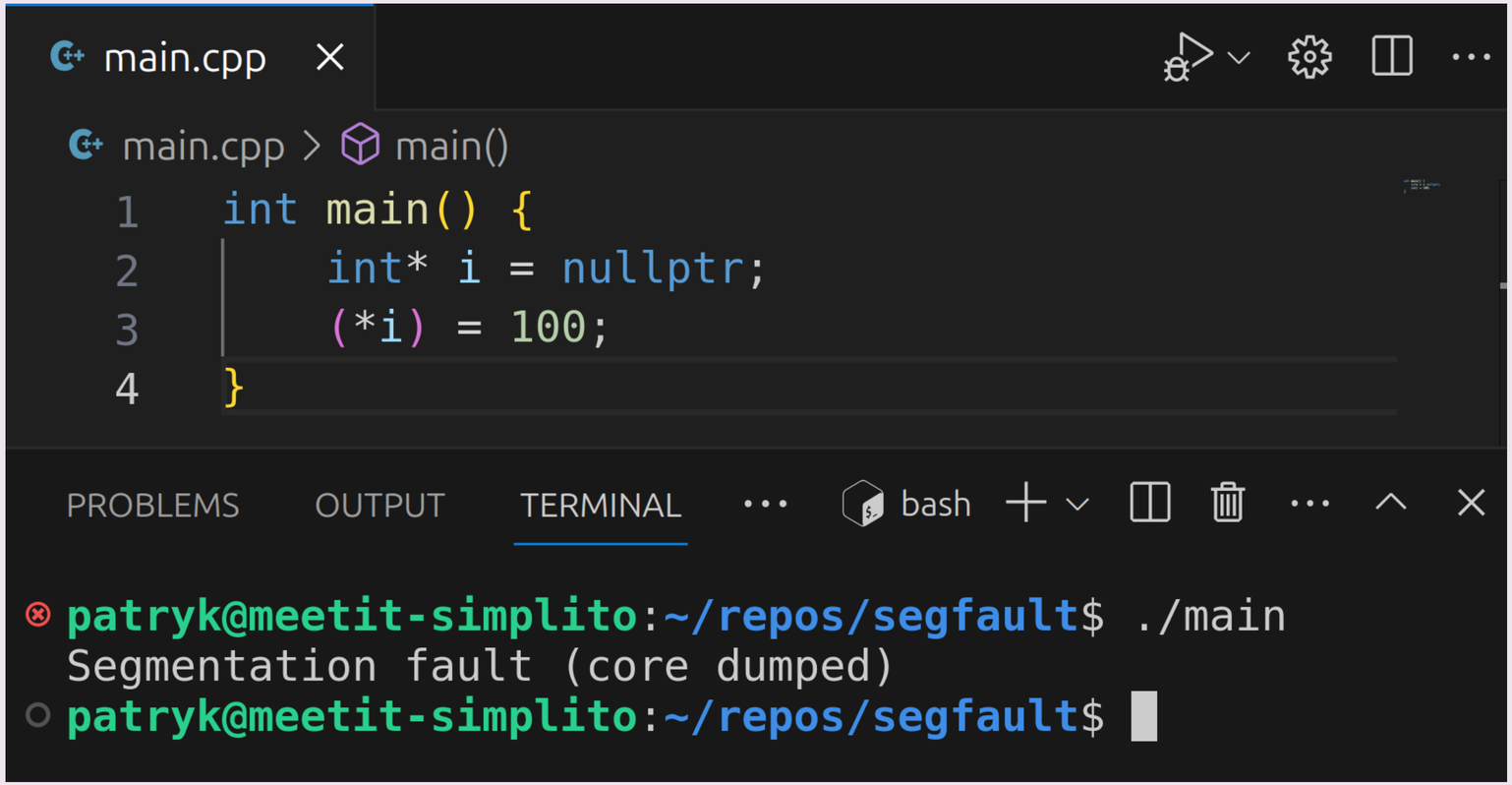
OPTYMALIZACJA NISKOPOZIOMOWYCH BIBLIOTEK

Rozwijanie i użycie bibliotek pozwala na optymalizowanie oprogramowania pod kątem m.in. zarządzania pamięcią, użytych instrukcji procesora i zmniejszenia ich liczby wykonań.

1. Użycie optymalnego algorytmu
2. Optymalizacja ręczna pamięci
3. Użycie opcji optymalizacyjnych podczas budowania (-O1, -O2, -O3)
4. Dodanie wstawek asemblerowych (inline assembly) lub pisanie całego modułu w asemblerze
5. Użycie gotowych bibliotek
6. Użycie systemowych bibliotek

PROBLEMY

- Przenośność (Różne architektury procesora → różne kompilacje biblioteki)
- Abstrakcja i wiedza o sprzęcie
- Obsługa błędów
- Kompilacja skrośna
(ang. Cross-compilation)



PRZYKŁADY ZASTOSOWANIA

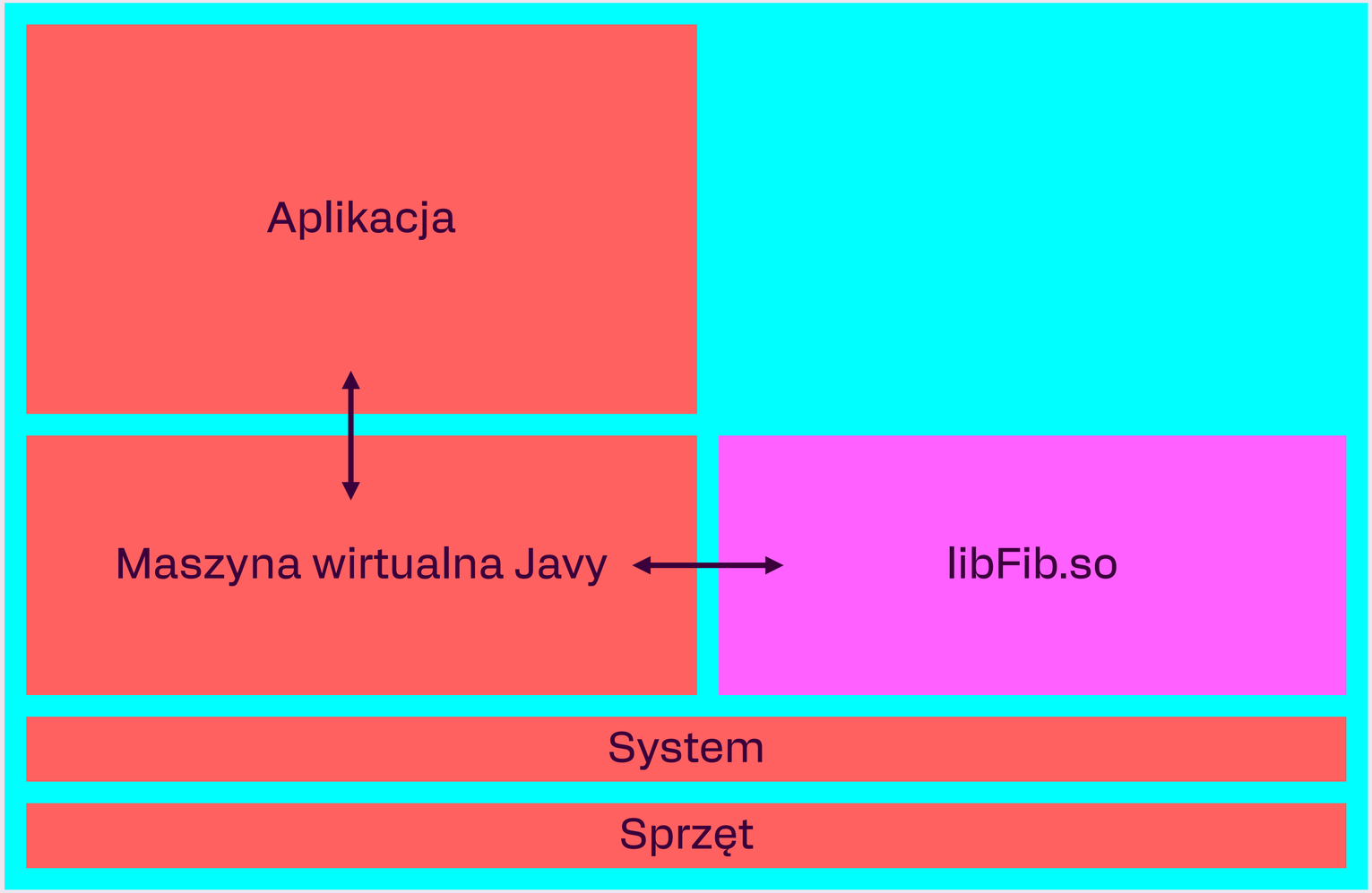
- Aplikacja mobilna Android: Java lub Kotlin

- Aplikacja mobilna iOS: Swift

- Aplikacja desktopowa w Electronie: JavaScript lub TypeScript

- Aplikacja internetowa w przeglądarce: JavaScript lub TypeScript

PRZYKŁAD INTEGRACJI Z JAVA



WRAPPERY

Wrappery to fragmenty kodu, które łączą ze sobą różne środowiska. Odbierają wywołania w języku programu, przekształcają na wywołanie w języku biblioteki i wynik przekształcają do odpowiedniego języka.

Wrappery wraz z bibliotekami można opakowywać w paczki. Końcowy programista aplikacji dostaje gotową paczkę z interfejsem w danym języku.

PROBLEMY NA STYKU JĘZYKÓW

- Typy wartościowe lub typy referencyjne, wskaźniki
- null i undefined
- Odśmiecanie pamięci (ang. garbage collection)
- Silne typowanie a dynamiczne typowanie
- Pętla zdarzeń (ang. event loop)
- Różne ograniczenia typów
- Bundlowanie na różne architektury

PODSUMOWANIE

1. Niskopoziomowe biblioteki można tworzyć samemu od zera lub używać gotowych.

2. Biblioteki można optymalizować pod kątem zarządzania pamięcią, użytych instrukcji procesora i zmniejszenia ich liczby wykonań.

3. Biblioteki można używać w różnych środowiskach i aplikacjach.

DZIĘKUJĘ ZA UWAGĘ

Kontakt

Patryk Kisielewski

pkisielewski@simplito.com

SIMPLITO
software R&D