

Jak to się dzieje, że programy działają, czyli opowieść symbiozie sprzętu i oprogramowania

Roman Simiński
roman.siminski@us.edu.pl



UNIWERSYTET ŚLĄSKI
W KATOWICACH

Wydział Nauk Ścisłych i Technicznych

UNIWERSYTET ŚLĄSKI
WYDZIAŁ NAUK ŚCISŁYCH I TECHNICZNYCH

Europejskie
Mieście Nauki
Katowice 2024

XVIII ŚWIĘTO LICZBY PI

14–15 marca 2024 r.

swietoliczbypi
www.swietopi.us.edu.pl

50 Tygodni
w Mieście Nauki
**Tydzień
Liczb¹¹**

WSPÓLORGANIZATOR
Województwo
Śląskie

Funded by
the European Union

Komputery „oczywiste”



dreamstime.com



UNIWERSYTET ŚLĄSKI
W KATOWICACH

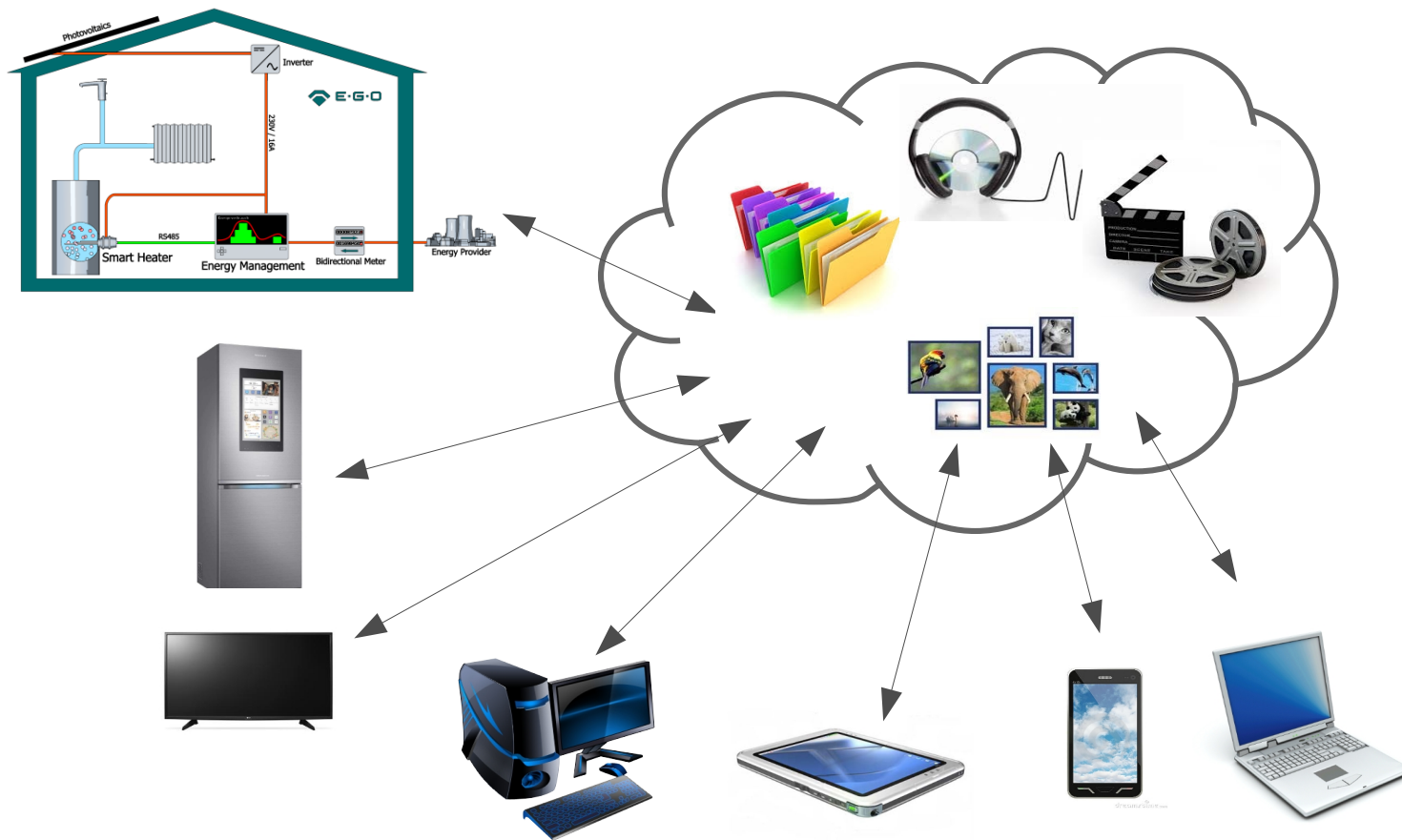
Komputery „nieoczywiste”



Źródło: www.geniusgadget.com

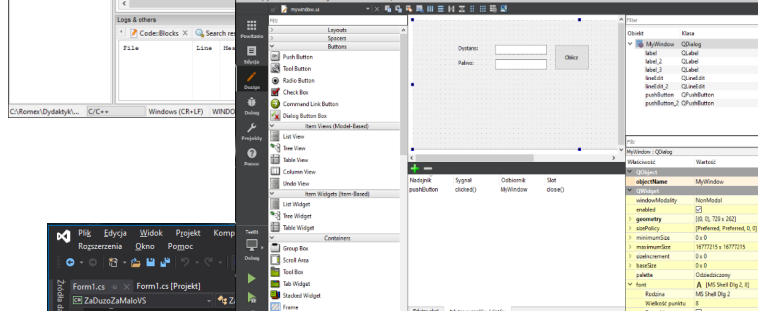


Komputery „wszędzie” - , IoT czyli Internet Rzeczy i Cloud Computing

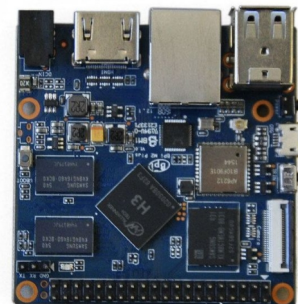
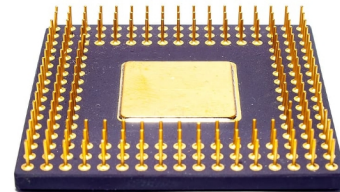
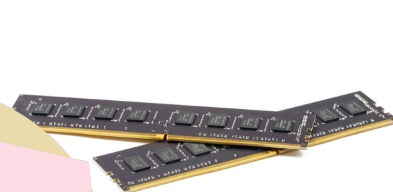
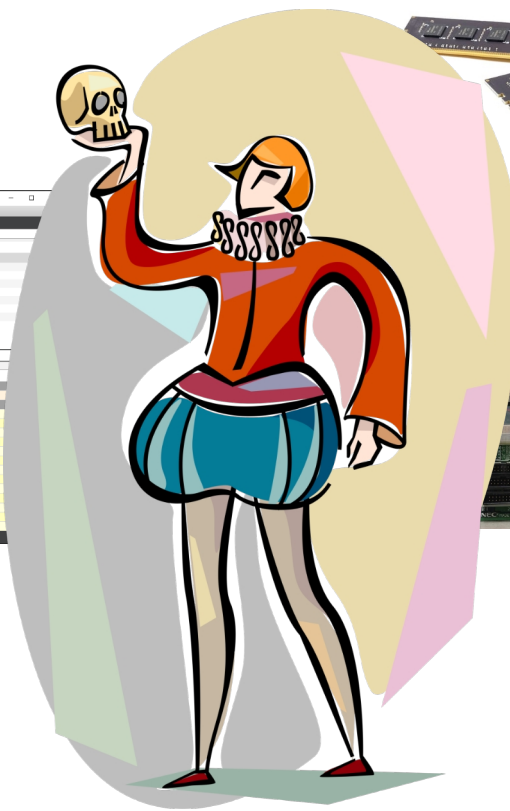


Sprzęt a oprogramowanie, oto jest pytanie...

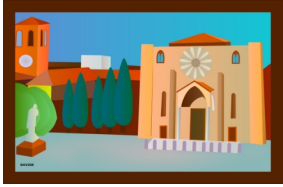
```
main.c(Autokom) - CodeBlocks2003
File Edit View Search Project Build Debug Fortran waSmith Tools Plugins DovyBlocks Settings Help
-:globab-
-:usun_pojazdy(void)
Management
Projects Files F5ymbol!
Workspace
Autokom
Sources
main.c
main.c
201 int nr;
202 if (lb_pojazdow == 0)
203     printf("Nwidencja jest pusta.");
204     for (nr = 0; nr < lb_pojazdow; nr++)
205     {
206         przewin_ekran();
207         printf("\nDane pojazdu nr: %d\n", nr + 1);
208         pokaz_info( kpojazdy[ nr ] );
209         printf("Wszysty chcecie susnaw ten pojazd? (t/n) ");
210         if ( tolower( getchar() ) == 't' )
211             {
212                 fflush( stdin );
213                 print
214             }
215     }
216 }
```



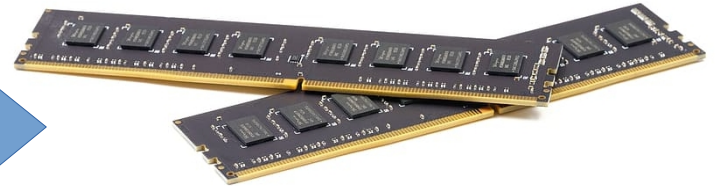
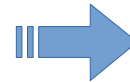
```
namespace ZaDuzoZaMalowS
{
    class Form1 : Form
    {
    public:
        Form1()
        {
            InitializeComponent();
            gen = new Random();
            wylosowana = 15; //gen.Next(100) + 1;
        }
    };
}
```



Ze świata analogowego do świata cyfrowego...

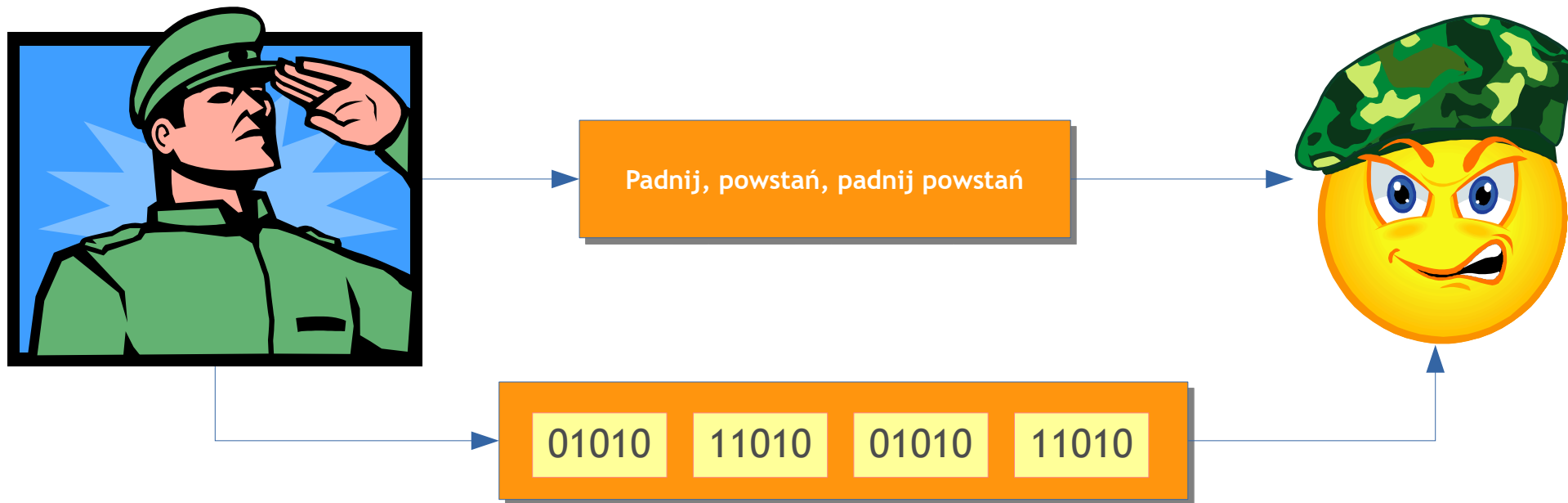


↓
Digitalizacja
↓



Jak cyfrowo reprezentować rozkazy?

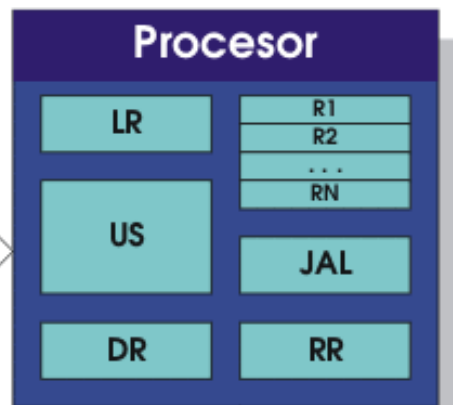
Rozkazy też mogą zostać „zdigitalizowane” poprzez ich ponumerowanie...



Architektura typowego komputera

Komputer jako środowisko wykonywania programu

Pamięć operacyjna	
Adresy	Komórki pamięci
00000000	10100101
00000001	00101101
00000002	11100010
00000003	11100001
...	...
...	...
...	...
01FFFFFF	11100001
20000000	01101000



LR - licznik rozkazów
RR - rejestr rozkazów
DR - dekodery rozkazów
US - układ sterujący
JAL - jedn. arytm.-logiczna
R1, R2, ..., RN - rejestry

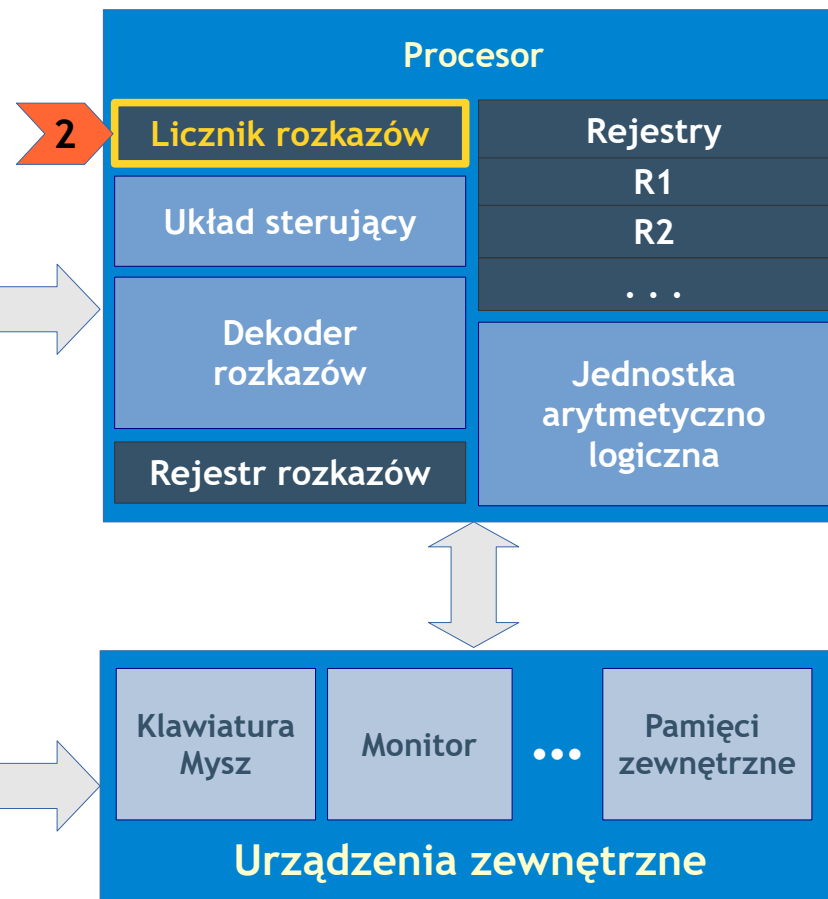


Cykl rozkazowy: przygotowanie do wykonania rozkazu

Licznik rozkazów zawiera adres rozkazu do pobrania

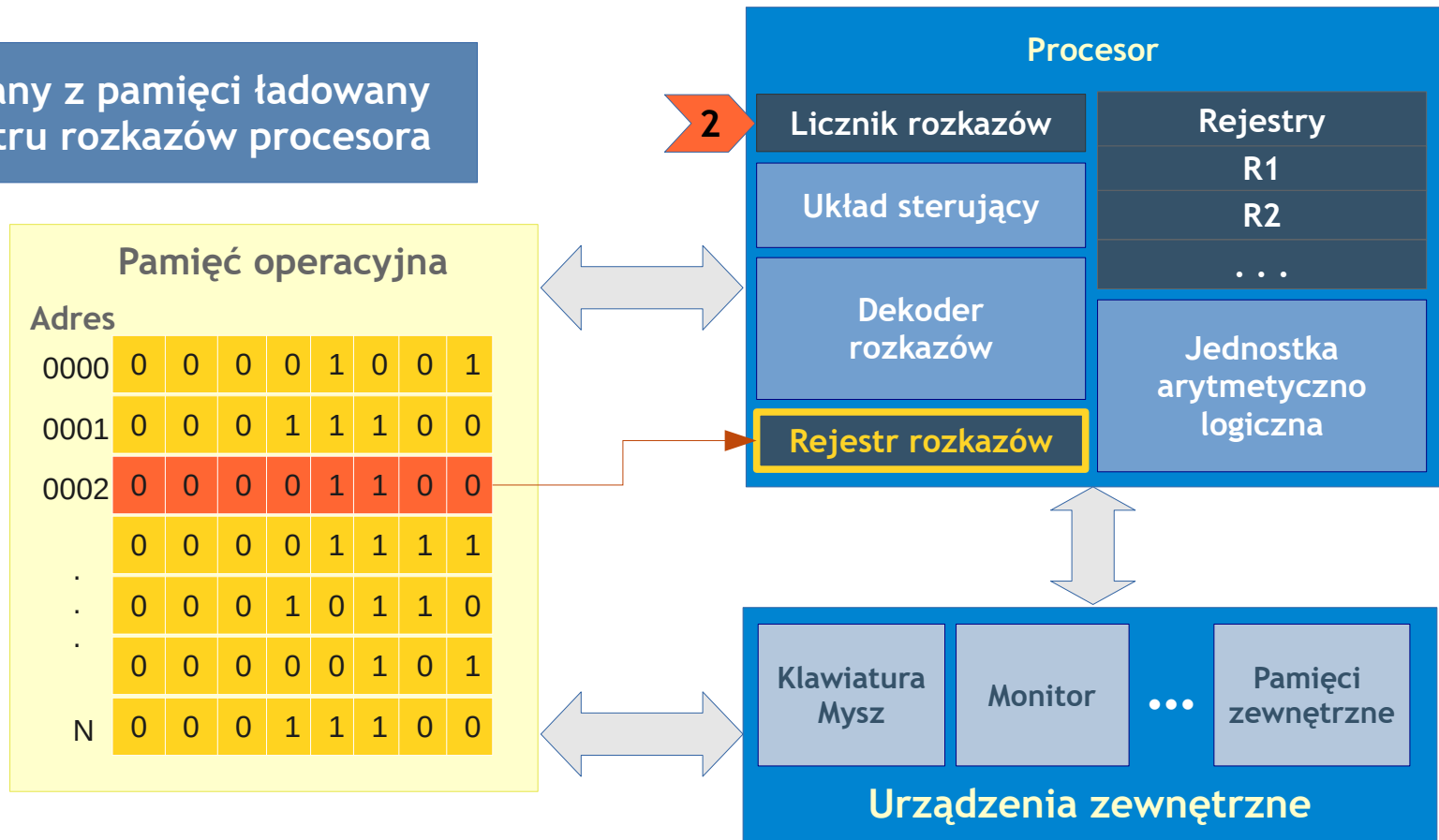
Pamięć operacyjna

Adres								
0000	0	0	0	0	1	0	0	1
0001	0	0	0	1	1	1	0	0
0002	0	0	0	0	1	1	0	0
	0	0	0	0	1	1	1	1
.	0	0	0	1	0	1	1	0
.	0	0	0	0	0	1	0	1
N	0	0	0	1	1	1	0	0



Cykl rozkazowy: pobranie rozkazu z pamięci operacyjnej

Rozkaz pobrany z pamięci ładowany jest do rejestru rozkazów procesora



Cykl rozkazowy: zdekodowanie rozkazu

Dekoder rozkazów zna każdy rozkaz i potrafi go rozpoznać

Pamięć operacyjna

Adres	0	0	0	0	1	0	0	1
0000	0	0	0	0	1	0	0	1
0001	0	0	0	1	1	1	0	0
0002	0	0	0	0	1	1	0	0
.	0	0	0	0	1	1	1	1
.	0	0	0	1	0	1	1	0
.	0	0	0	0	0	1	0	1
N	0	0	0	1	1	1	0	0



Cykl rozkazowy: wykonanie rozkazu

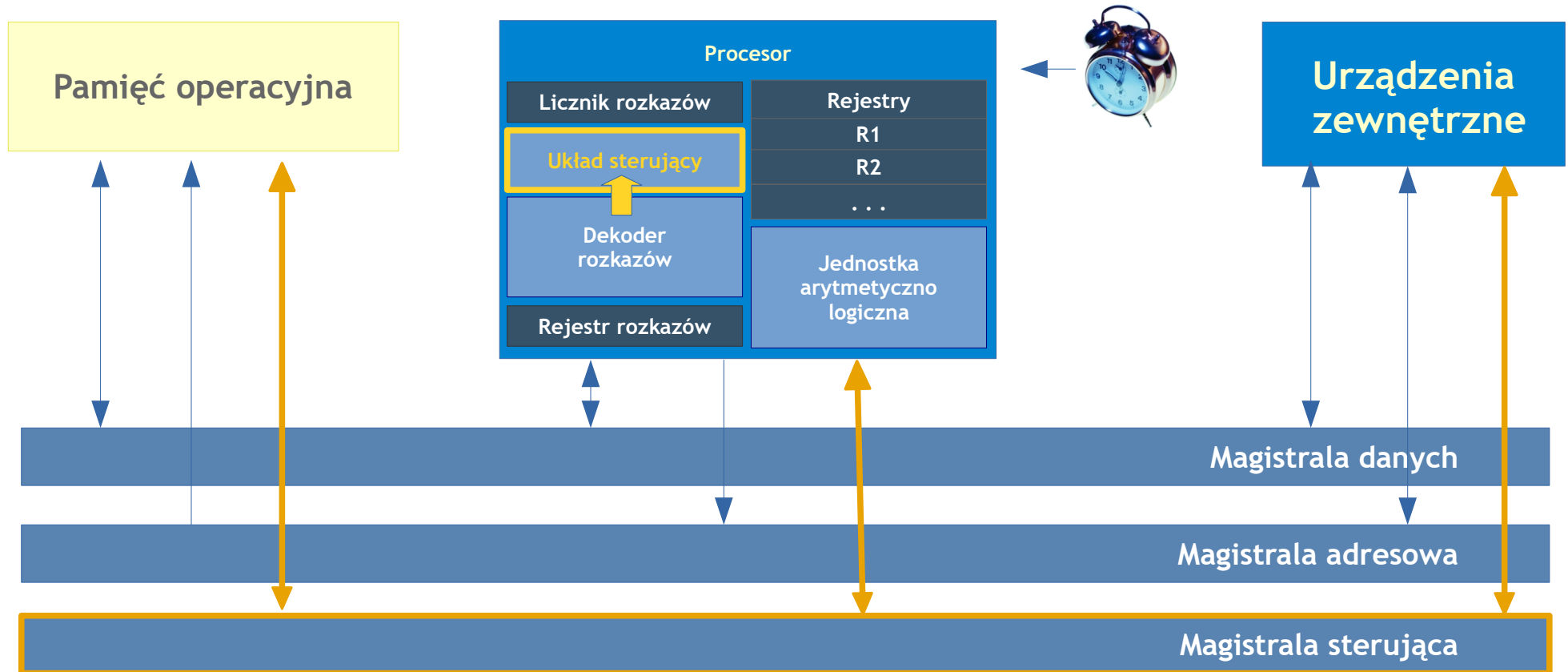
Układ sterujący za przepis na wykonanie każdego rozkazu

Pamięć operacyjna

Adres	0	0	0	0	1	0	0	1
0000	0	0	0	0	1	0	0	1
0001	0	0	0	1	1	1	0	0
0002	0	0	0	0	1	1	0	0
.	0	0	0	0	1	1	1	1
.	0	0	0	1	0	1	1	0
.	0	0	0	0	0	1	0	1
N	0	0	0	1	1	1	0	0



Wykonanie rozkazu: sekwencja mikrooperacji

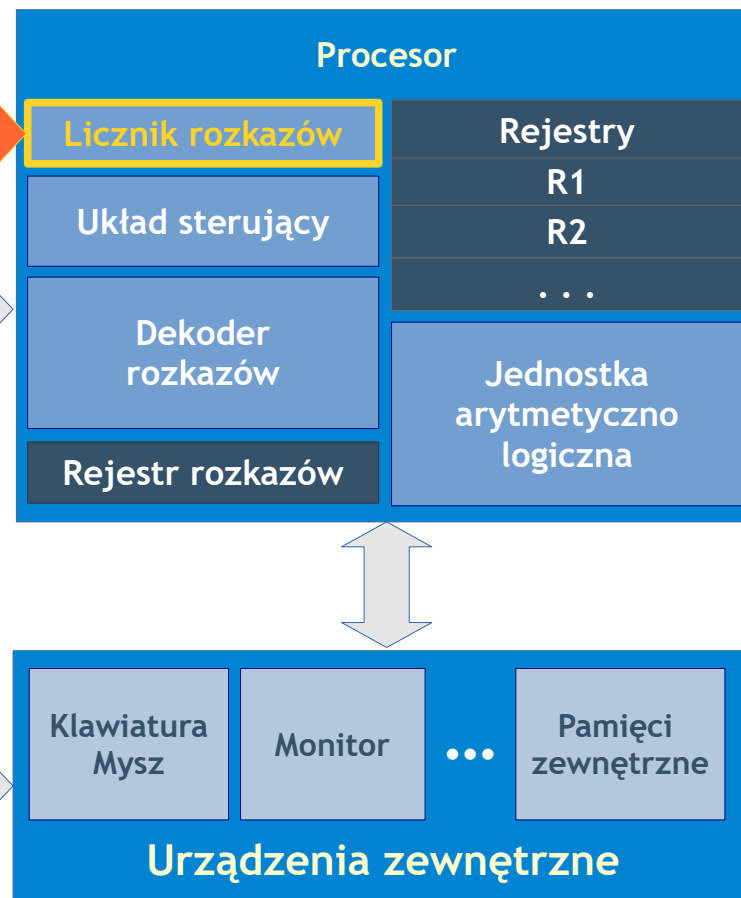


Cykl rozkazowy: ustalenie adresu rozkazu następnego

Trzeba ustalić, skąd w następnym cyklu zostanie pobrany rozkaz

Pamięć operacyjna

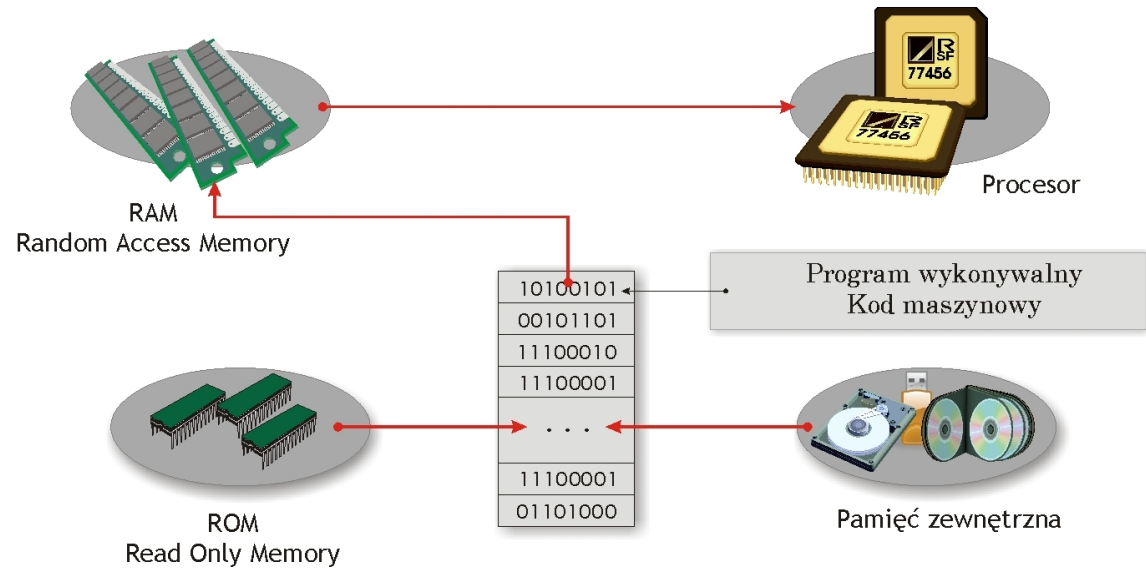
Adres	0	0	0	0	1	0	0	1
0000	0	0	0	0	1	0	0	1
0001	0	0	0	1	1	1	0	0
0002	0	0	0	0	1	1	0	0
.	0	0	0	0	1	1	1	1
.	0	0	0	1	0	1	1	0
.	0	0	0	0	0	1	0	1
N	0	0	0	1	1	1	0	0



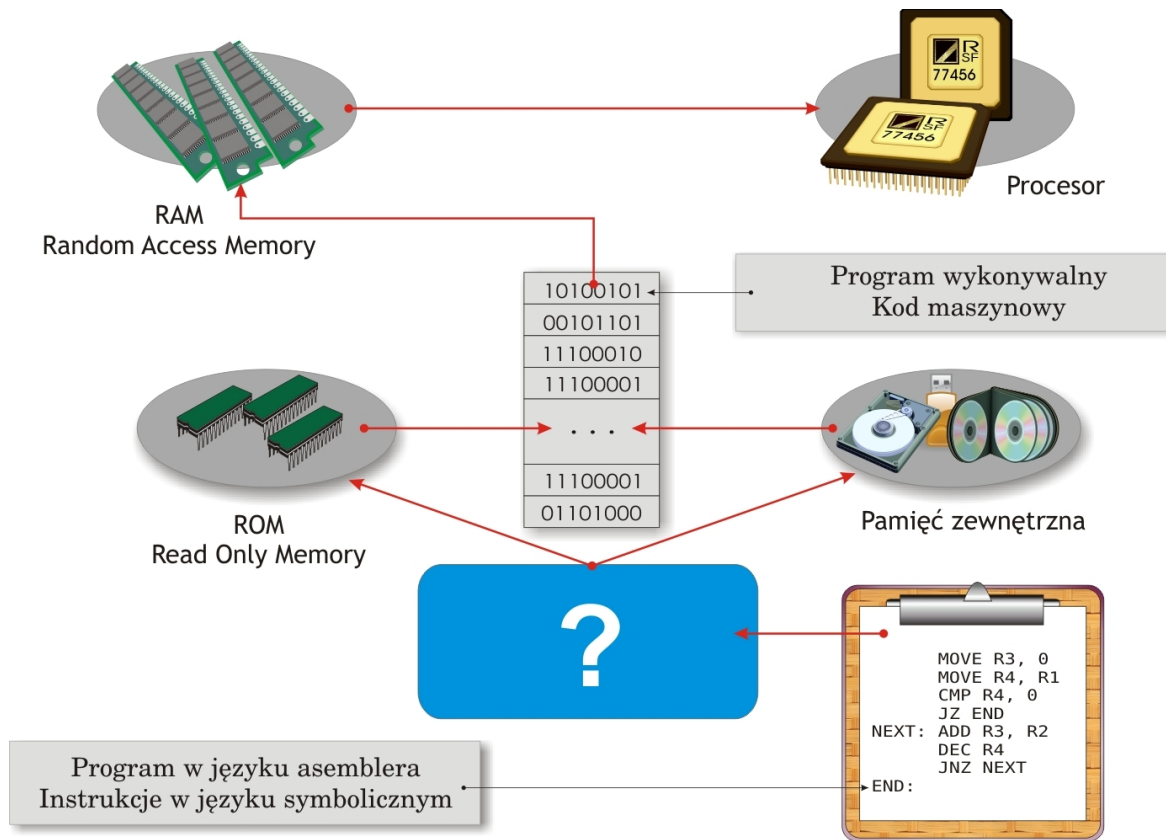
Jak powstaje program?



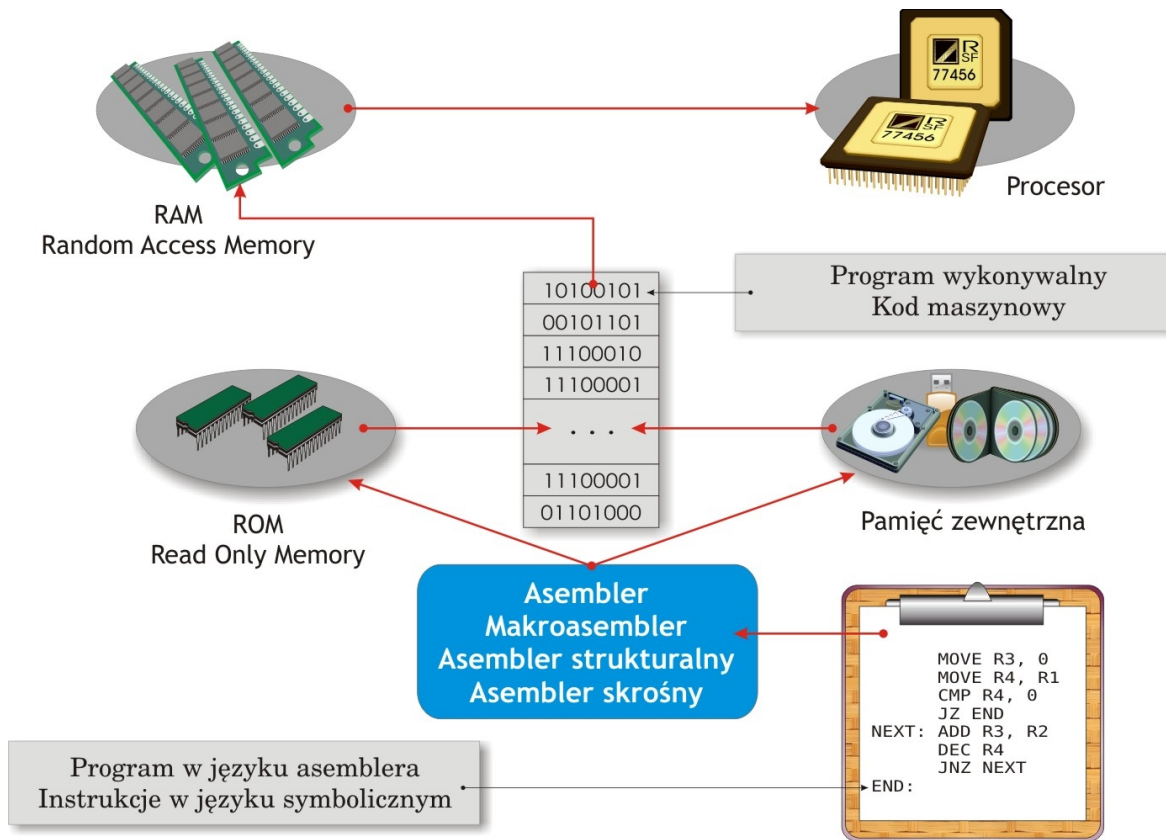
Procesor wykonuje rozkazy maszynowe



Programowanie w języku symbolicznym

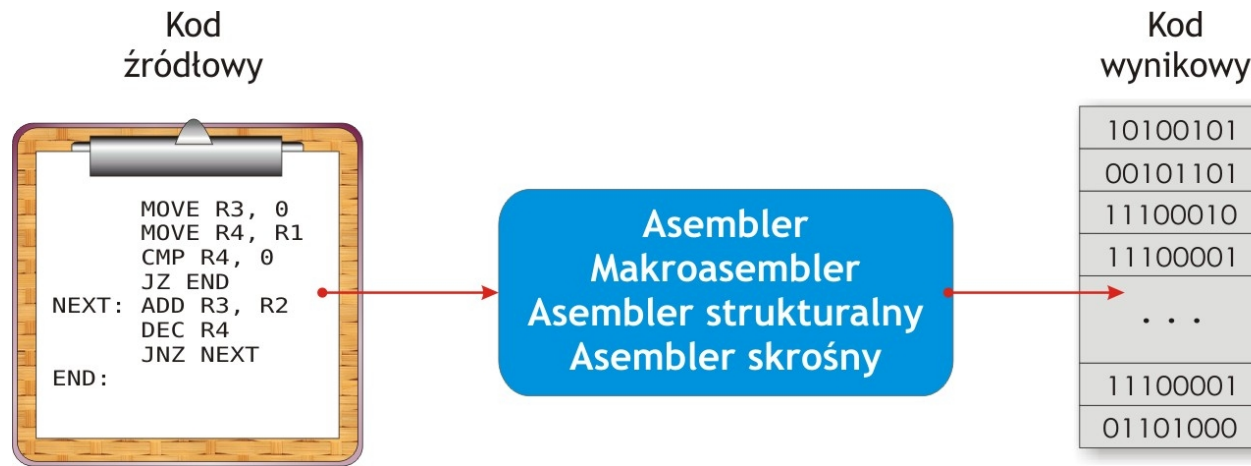


Programowanie w języku symbolicznym



Programowanie w języku symbolicznym

- **Język symboliczny** (język assemblera) umożliwia pisanie programów z wykorzystaniem symboli (mnemonik, mnemoników) przypisanych poszczególnym rozkazom procesora.
- **Mnemoniki** to symboliczne oznaczenia rozkazów i innych elementów, np. rejestrów – *mov, move, ld, add, sub, out, ax, bc, si, di, ss, ...*
- **Assembler** to program dokonujący tłumaczenia kodu źródłowego programu w języku symbolicznym na kod maszynowy – tzw. kod wynikowy.



Języki wysokiego poziomu

- *Język wysokiego poziomu* – niezależny od platformy sprzętowej i systemowej język programowania, pozwalający programiście skoncentrować się na logice rozwiązywanego problem.
- Notacja typowych języków wysokiego poziomu to połączenie elementów języka angielskiego z notacją wywodzącą się z matematyki.
- Jedna instrukcja programu w języku wysokiego poziomu odpowiada zwykle wielu rozkazom maszynowym.

Program w języku symbolicznym

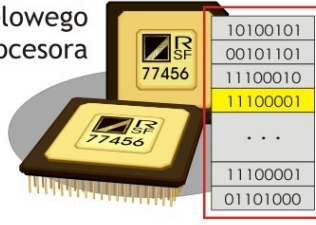
```
MOVE R3, 0
MOVE R4, R1
CMP R4, 0
JZ END
NEXT: ADD R3, R2
      DEC R4
      JNZ NEXT
END:
```

Program w języku wysokiego poziomu

```
z = 0;
for( i = x; i >= 0; --i )
  z += i + y;
```

Translacja programu w języku wysokiego poziomu

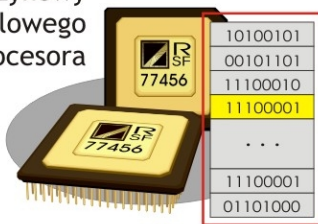
Kod maszynowy
docelowego
procesora



```
z = 0;  
for( i = x; i >= 0; --i )  
  z += i + y;
```

Translacja programu w języku wysokiego poziomu: kompilator

Kod maszynowy
docelowego
procesora



Kompilator

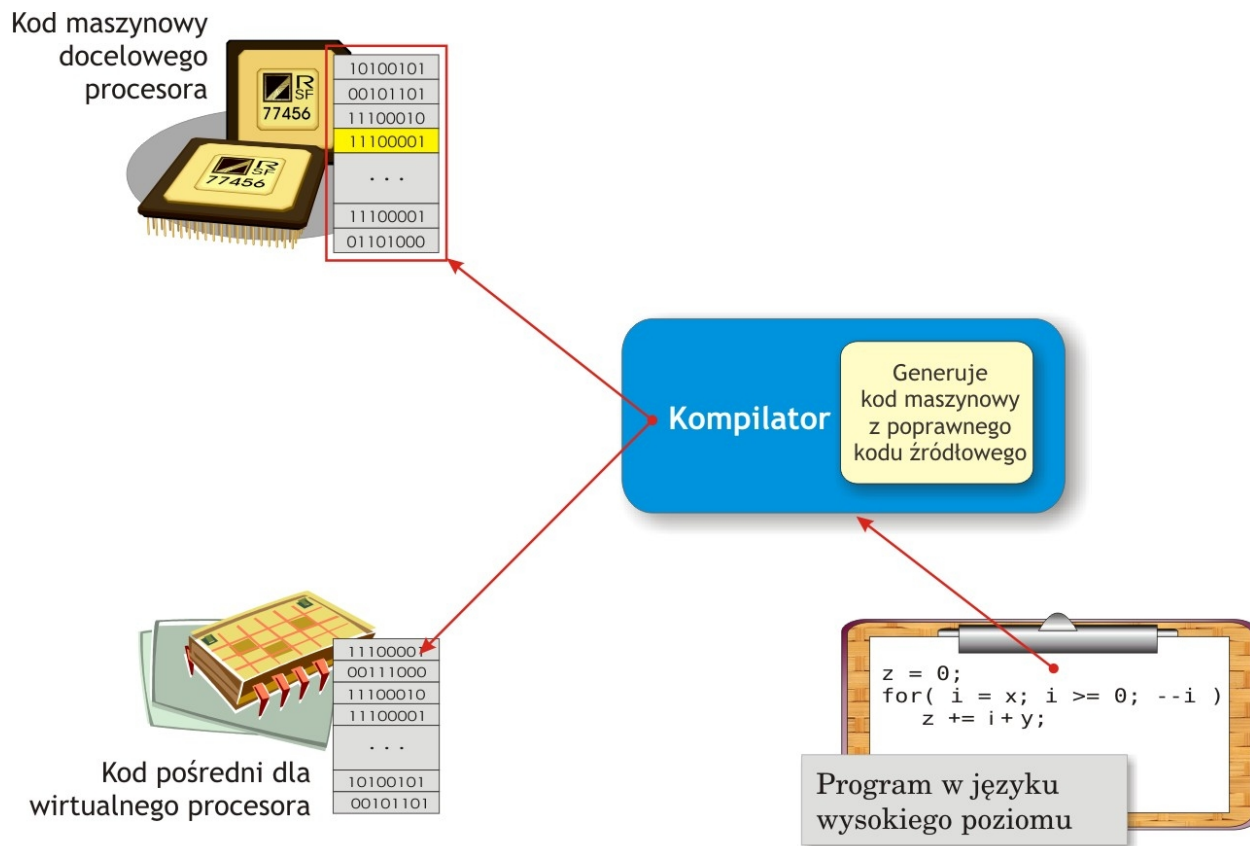
Generuje
kod maszynowy
z poprawnego
kodu źródłowego

```
z = 0;  
for( i = x; i >= 0; --i )  
  z += i+y;
```

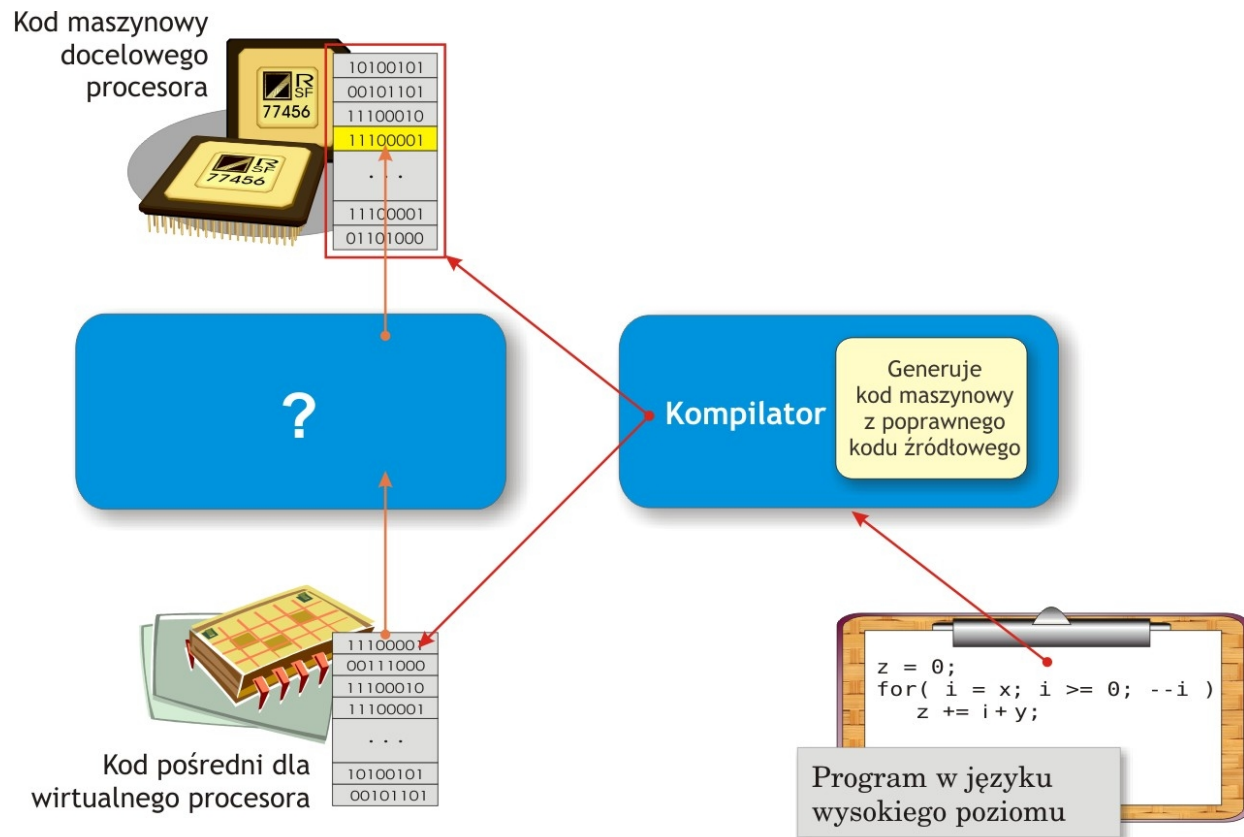
Program w języku
wysokiego poziomu



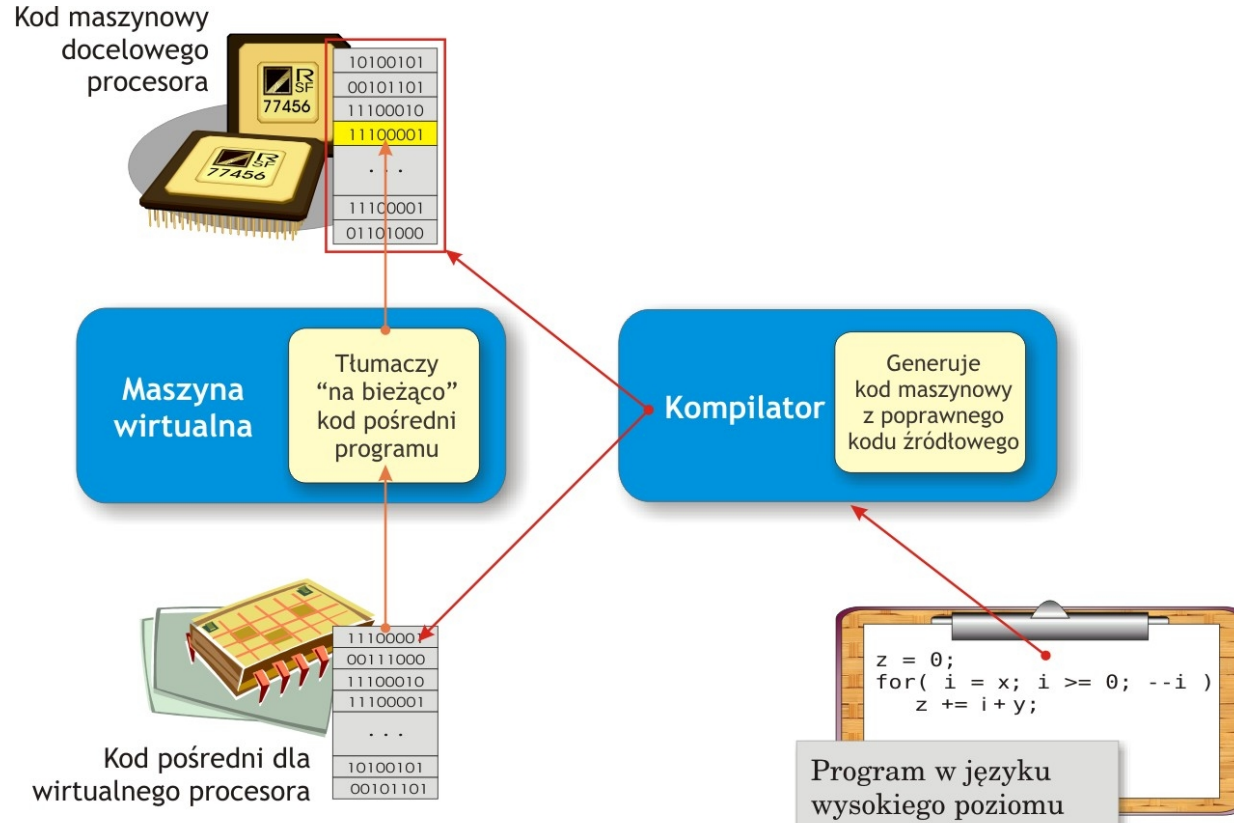
Hybrydowe podejście do kompilacji - kod pośredni



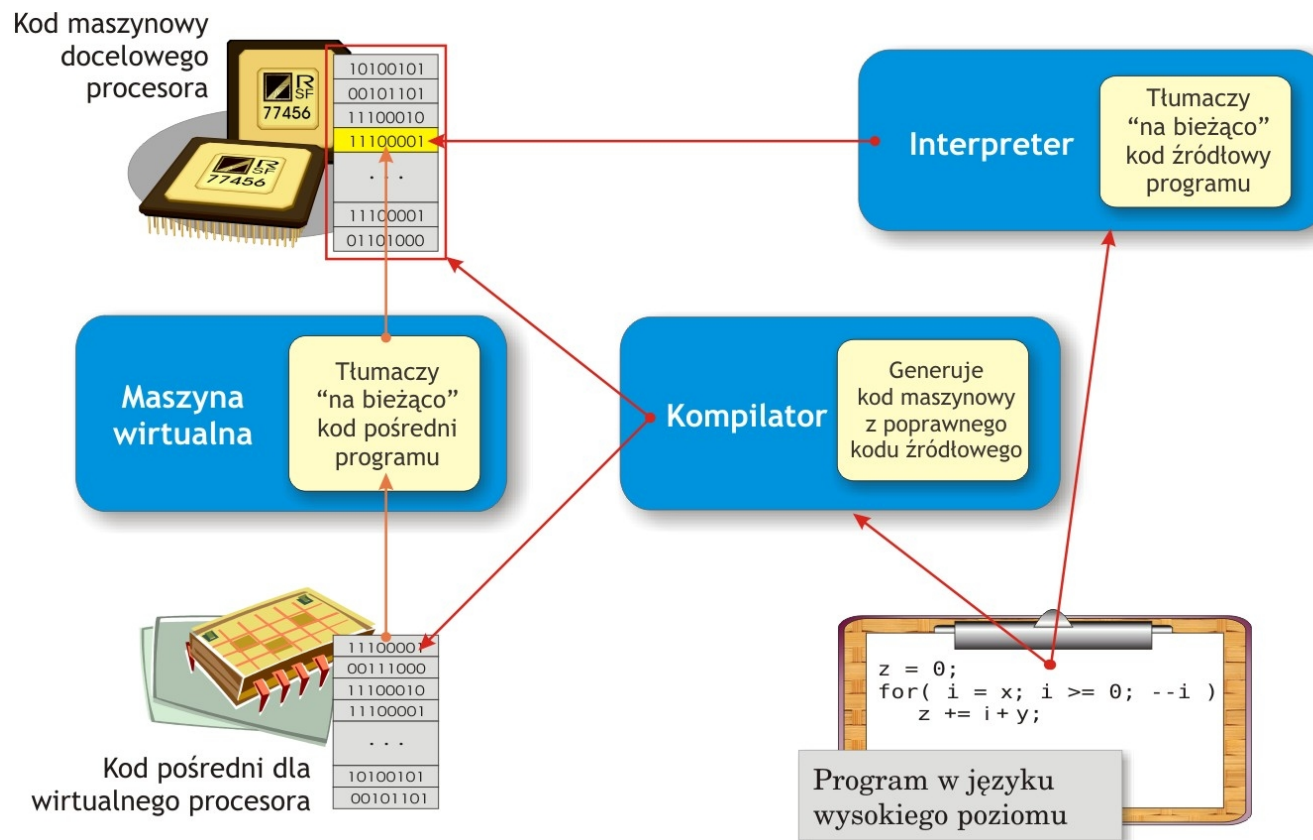
Kod pośredni wymaga kolejnej translacji



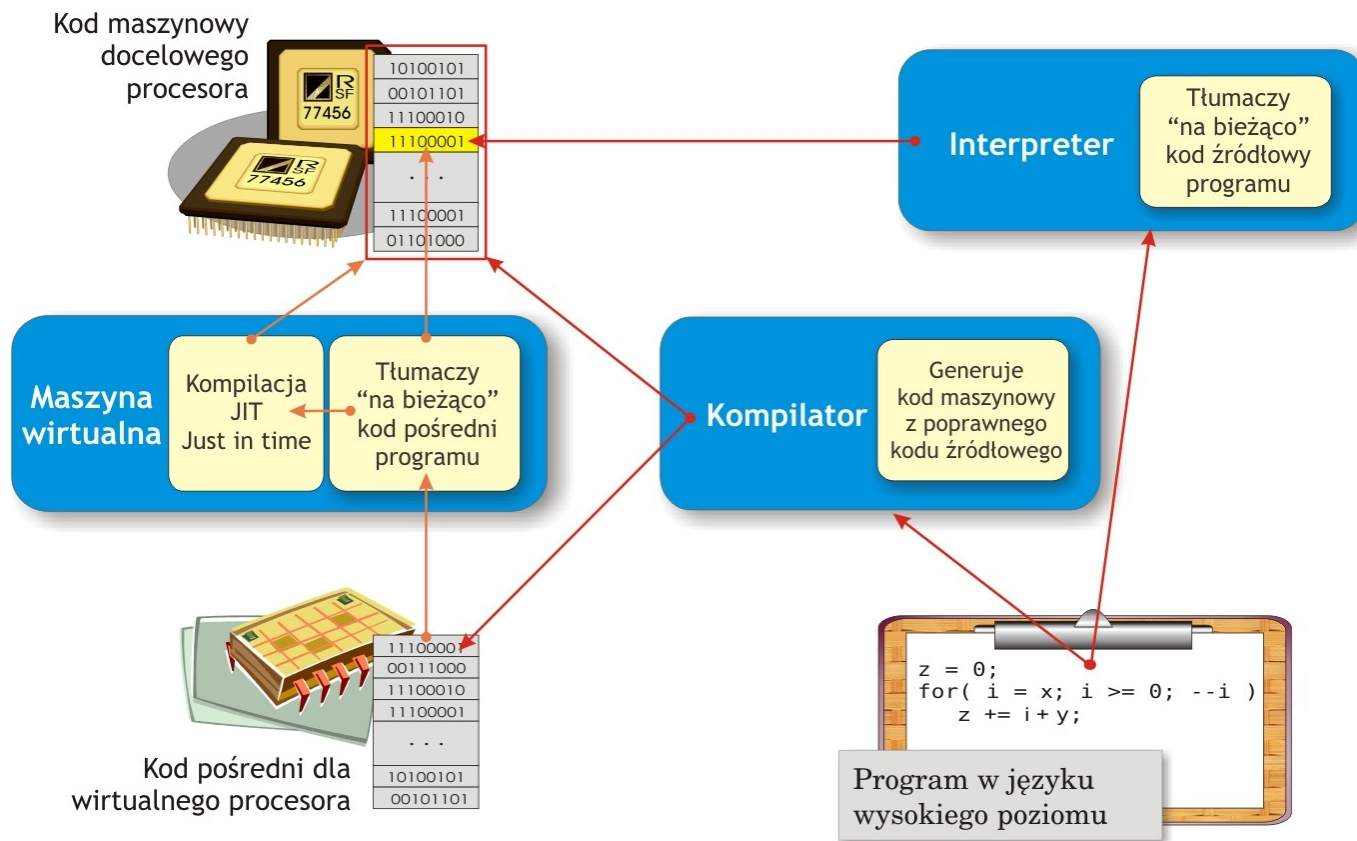
Maszyna wirtualna jako translator kodu pośredniego



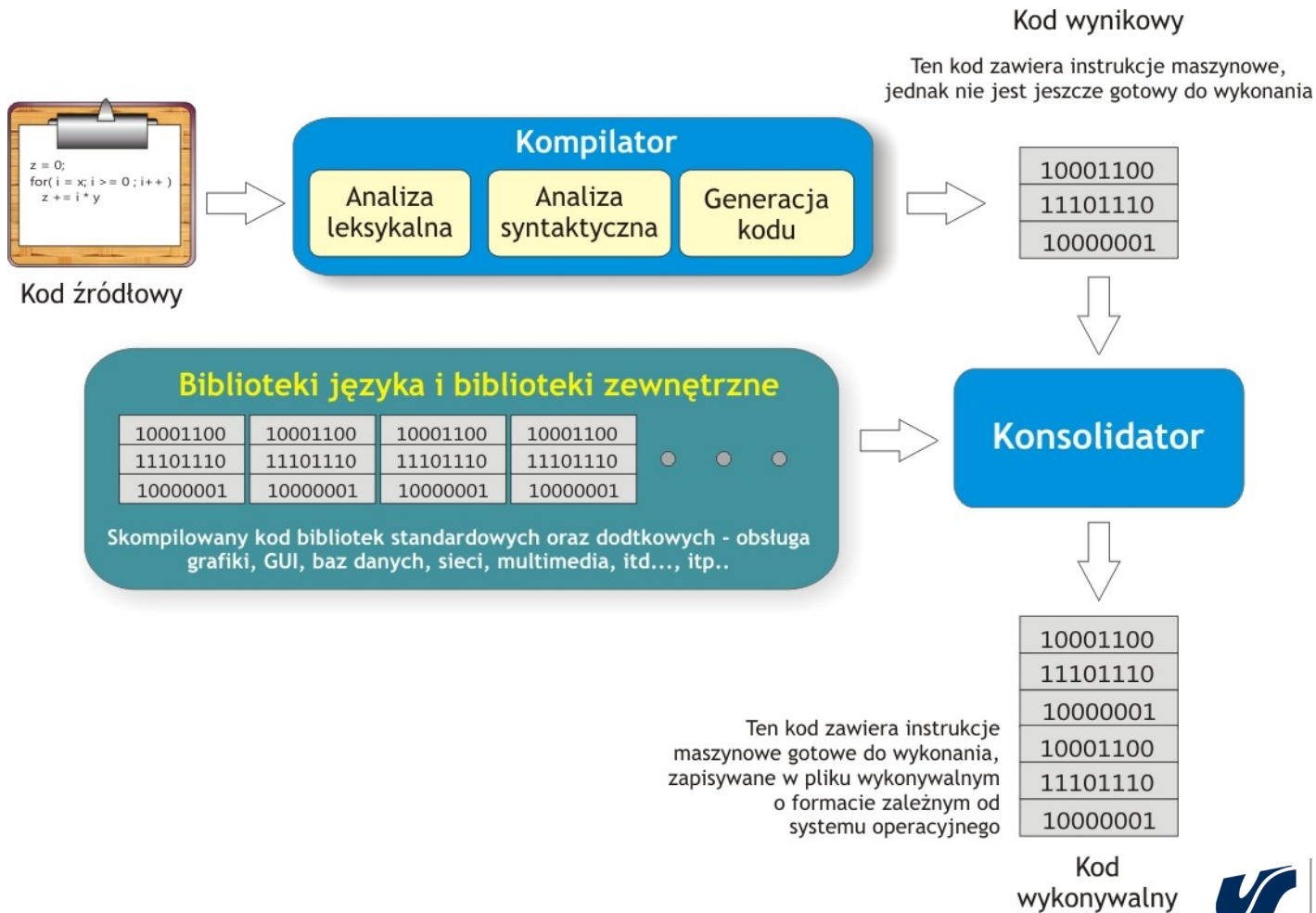
Interpreter - tłumaczenie „symultaniczne”



Kompilacja: Just In Time



Typowy przebieg generowania kodu kompilowanego



Rodzaje aplikacji i języki ich programowania



Aplikacje klasy „desktop”

C/C++

Qt, VCL, MFC, WinForms,
WxWidgets, GTK/GTK+

C#

.Net, .Net Core, WinForms, WPF,
UWP

Java

JVM, AWT, Swing, FX

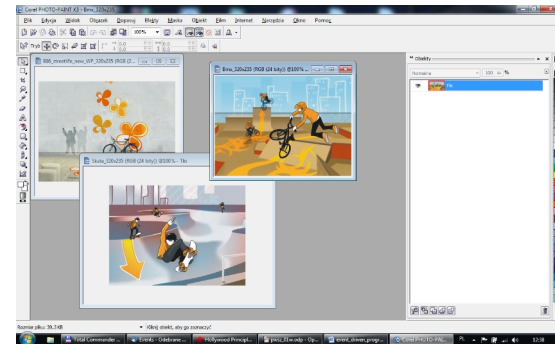
Python

QPyt, PyGTK, WxPython, Tkinter

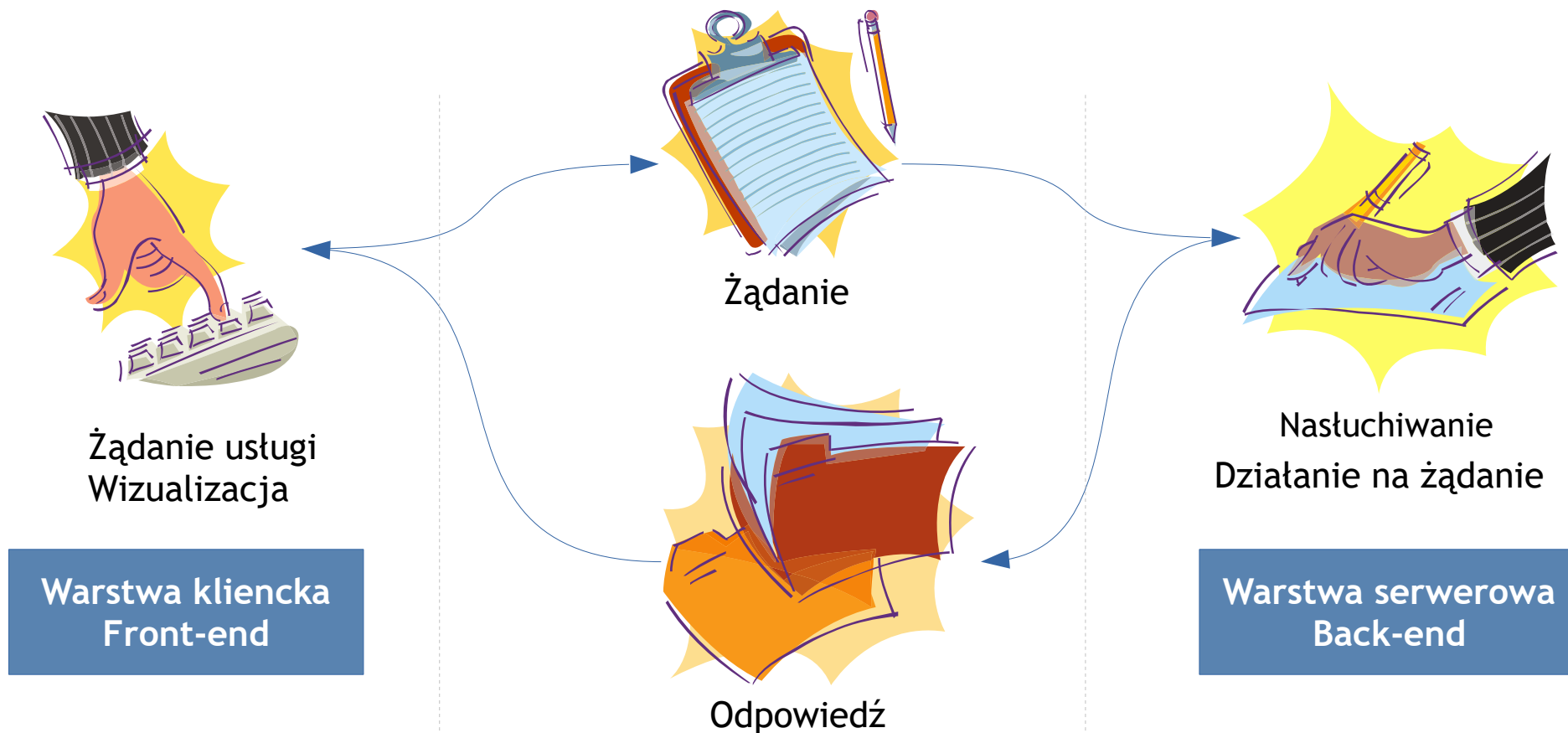
Swift

SwiftUI

Systemy operacyjne, oprogramowanie
narzędziowe, programy instalowane na
stacja roboczych



Koncepcja aplikacji klient-serwer



Warstwa kliencka, front-end

JavaScript
TypeScript

Angular, React, Vue.js,
Backbone.js, Ember, Electron

HTML
CSS

Bootstrap, Foundation, YAML
LESS, Skeleton



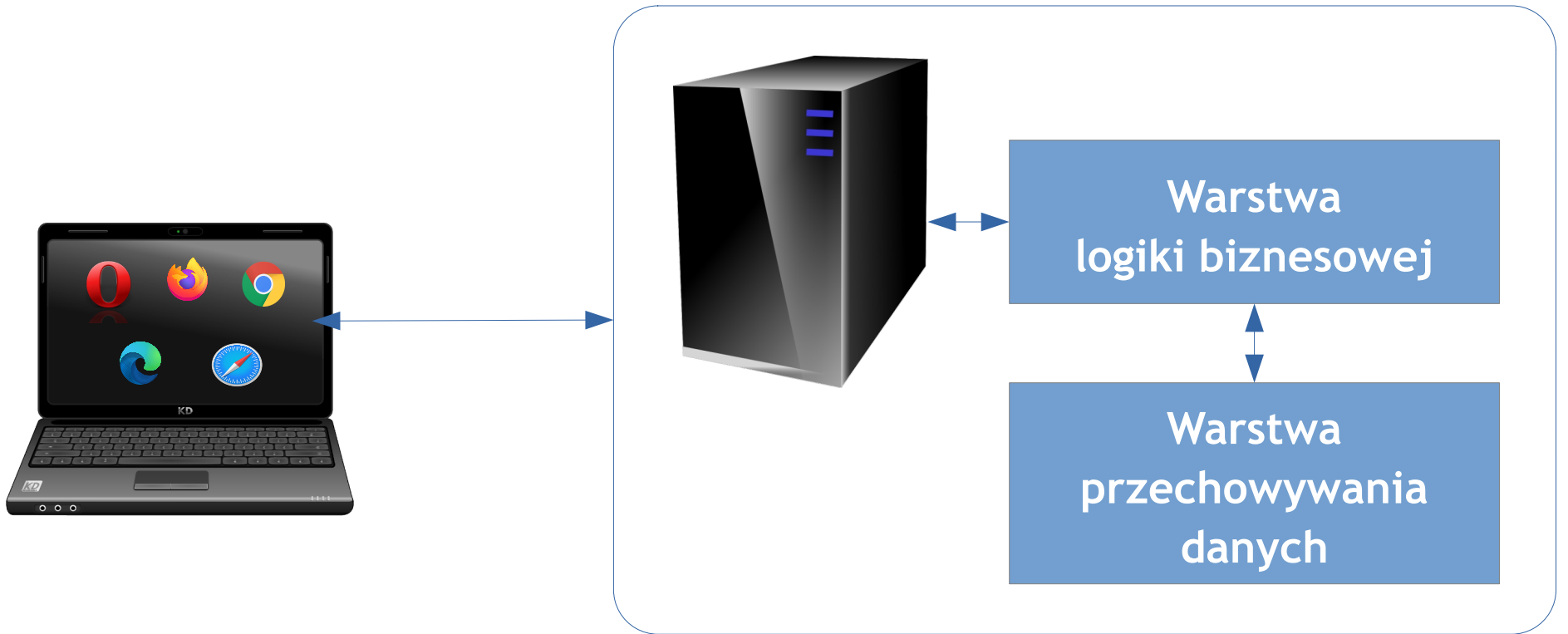
Front-end to różne aktywności, ale głównie programowanie

```
<script>function envFlush(a){function b(b){for(var c in a)b[c]=a[c]}window.requireLazy?wi
scroll":true},"isHeartbeatEnabled":true,"isArtilleryOn":false},"shouldLogCounters":true,"
0mhoj^q`xm`r","stack_trace_limit":30,"reliability_fixederrors_2018":true,"timesliceBuffer
{t_cstart>window._cstart},this.piggy_values={},this.bootloader_metrics={},this.resource_t
(d[a]=b);return this},CavalryLogger.prototype.getLastTtiValue=function(){return this.last
function(a){console.timeStamp(a)}:function(){}},CavalryLogger.prototype.addPiggyback=funct
{if(CavalryLogger.id===0){var b=CavalryLogger.getI
{}},CavalryLogger.prototype.profileEarlyResources=f
d=CavalryLogger.now();this.ongoing_watch[c]=d;"sta
(d.t_resource_download||(d.t_resource_download=0),
a),CavalryLogger.start_js=function(a){for(var b=0;
[1]?"js":"","_EF_")};CavalryLogger.getInstance().
```

```
14 </script><script nonce="1S0RACD6h13uv512joDYwg">
15 (function(){try{var n="Edge",ba="Opera",ca="complete",q="function",u="object",v="stri
ea(x.location.href);fa(d,"/mail/logstreamz");ha(d,"",void 0);var e=new FormData;e.app
{f(ja(g.target)),e)}}),la=function(){var b=new ea(x.location.href);switch(b.c.get("v
default:return"dot_new"==b.c.get("usp"?5:0)},oa=function(b,c,d){c=void 0===c?"":c;x.
e=0;5==b&&(e=1);x.top.GM_writeErrorPage(e,function(){x.top.GM_SLF();var f=x.performan
x.PerformanceResourceTiming&&"iframe"==g.initiatorType&&"+\\mail(\\u\\d+)?\\data(\\?
17 ui:x.GLOBALS[101]"hub":"refrito",error:c+
(d&&d.stack?"\n"+d.stack:""),debug:JSON.stringify({userAgent:x.GLOBALS[71],build:x.GL
BALS[61],reloadCount:x.localStorage.getItem("reload_count"),isCacheableHtml:x.GLOBALS
State:x.GM_DIRS||null,staleFlagError:x.GM_SFE||null}});x.top.document.getElementById
18 (x.top.document.getElementById("numeric_code").textContent=String(b))}}return new Pr
{done:!0}},B=function(b){var c="undefined"!=typeof Symbol&&Symbol.iterator&&b[Symbol
Object.defineProperty:function(b,c,d){if(b==Array.prototype||b==Object.prototype)retu
19 b,u==typeof window&&window,u==typeof self&&self,u==typeof global&&global};for(var c=0
{if(c){var d=sa;b=b.split(".");for(var e=0;e<b.length-1;e++){var f=b[e];f in d||(d[f]
{configurable:!0,writable:!0,value:c}}};C("Array.prototype.find",function(b){return
g=0;g<f;g++){var k=e[g];if(c.call(d,k,g,e)){c=k;break a}}c=void 0}return c}});C("Prom
20 {h(k)});if(b)return b;c.prototype.c=function(k){if(null==this.b){this.b=[];var h=this
{e(k,0)};c.prototype.i=function(){for(;this.b&&this.b.length;){var k=this.b;this.b=[
```



Warstwa serwerowa, back-end



Warstwa serwerowa, back-end

C/C++

Java

C#

Node.js

PHP

Ruby

Python

Golang

Warstwa
logiki biznesowej

Bazy SQL

Bazy NoSQL

Warstwa
przechowywania
danych



Programowanie urządzeń mobilnych

**C/C++
Java/Kotlin
Swift**

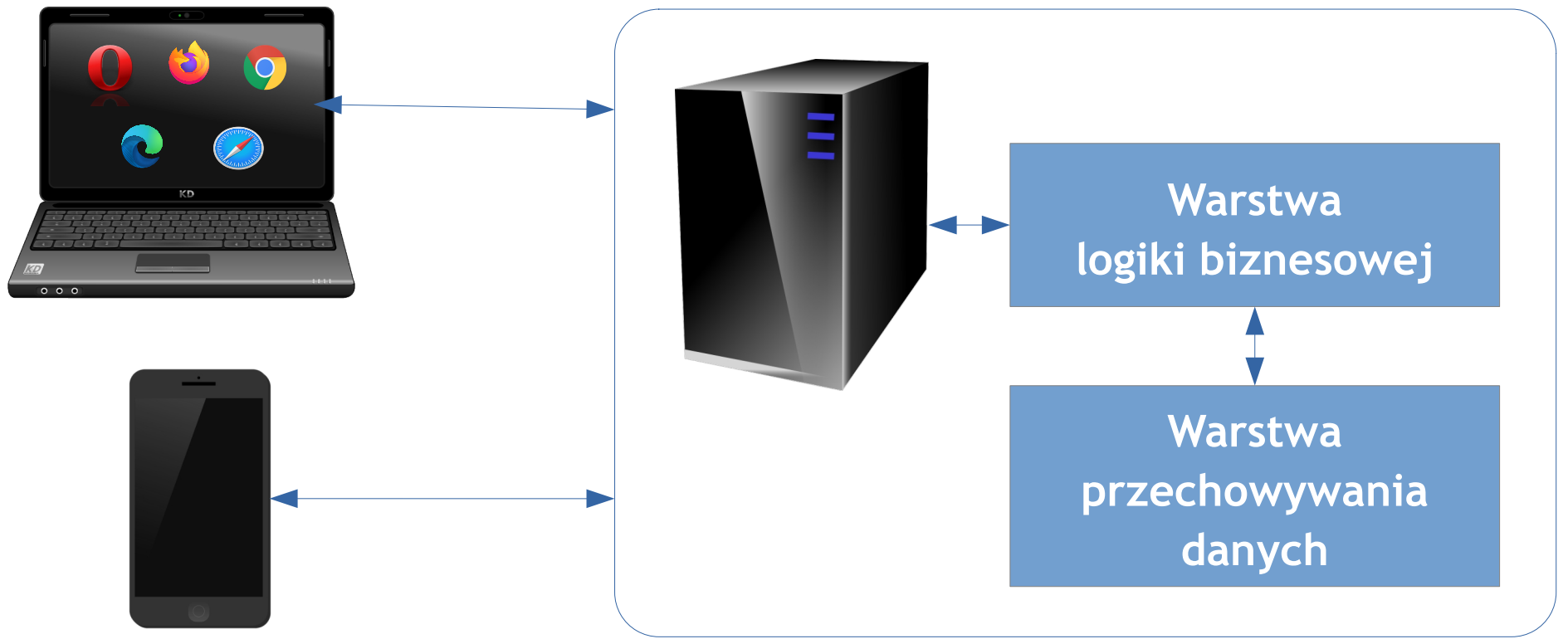
Podejście „natywne” osadzone
w kontekście systemu operacyjnego

**Progressive
Web Apps**

Wykorzystanie technologii
internetowych: JavaScript, HTML, CSS +
dodatkowe pakiety



Wszystko teraz związane jest z siecią Internet



Dziękuję za uwagę

roman.siminski@us.edu.pl

