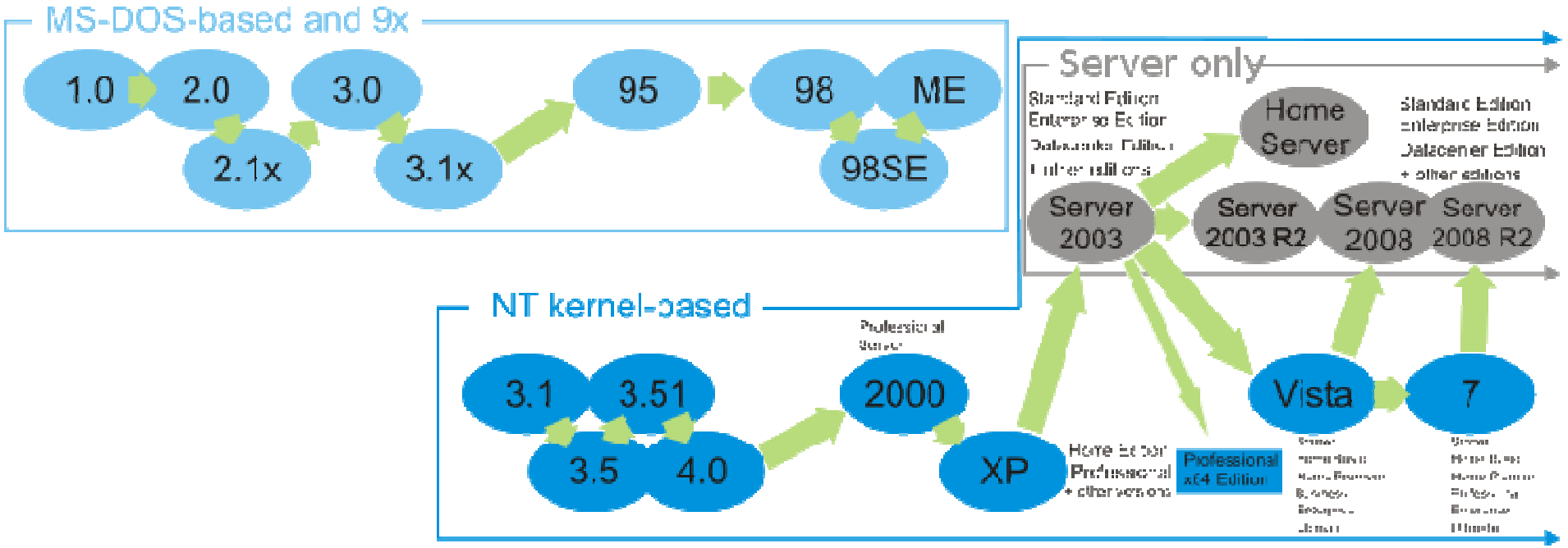
Arhitektura i način rada Windows OS-a

Maja Štula

Ak. God. 2011/2012

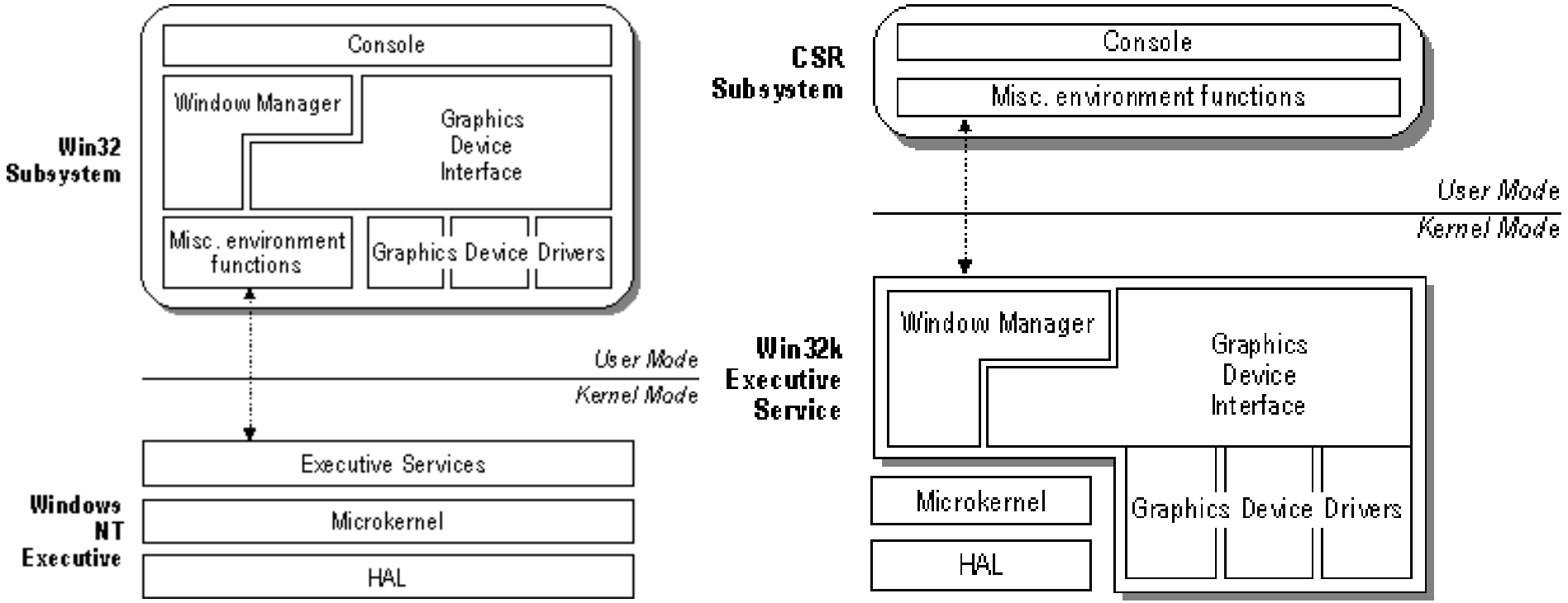
Microsoft Windows

family tree

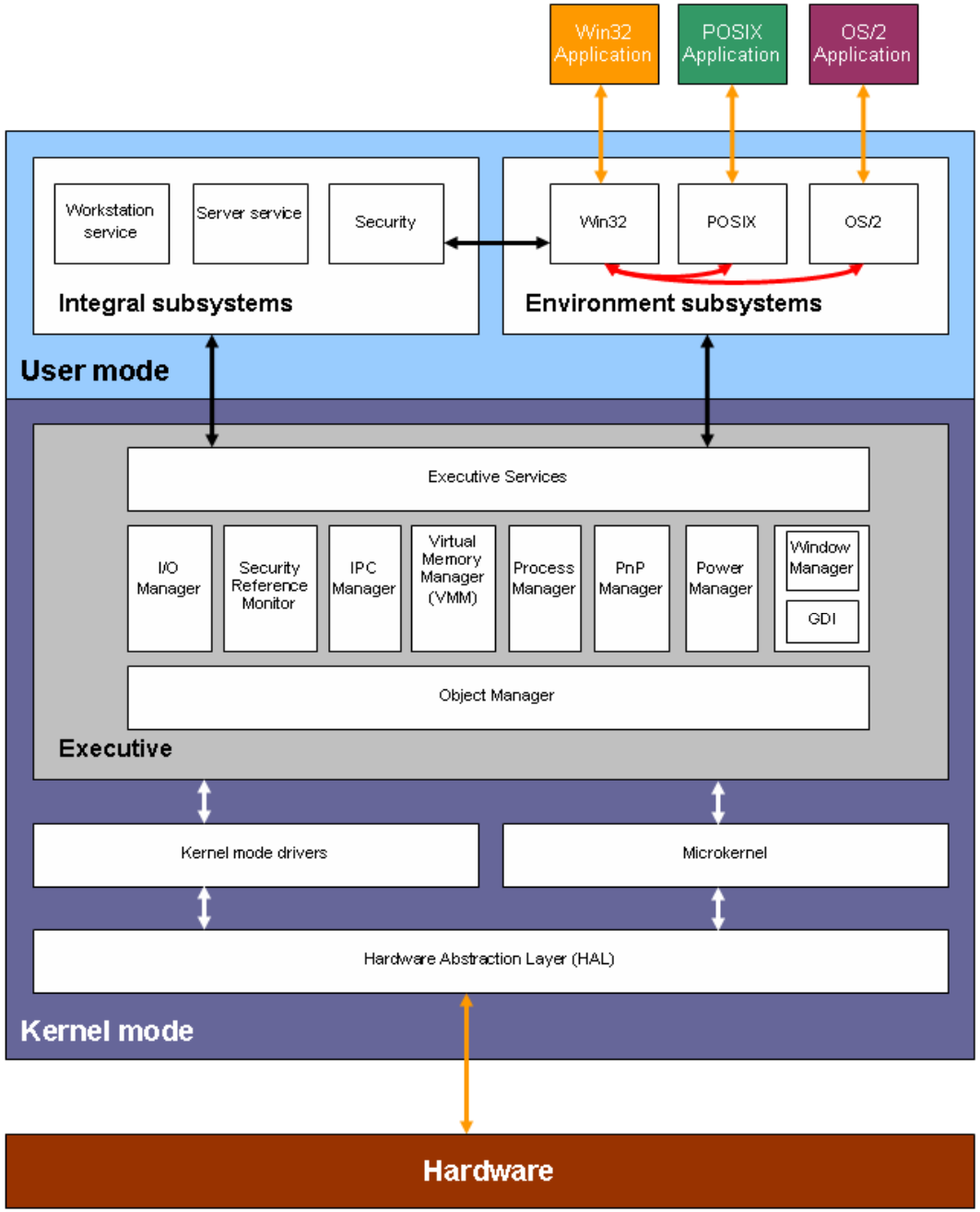


+ WindowCE (Windows Embedded CE)

Arhitektura Microsoft Windows OS-ova



Pojednostavljena arhitektura Windows NT 3.51 i NT 4.0



Arhitektura Windowsa 2000

Arhitektura Windows OS-a

- Korisničke aplikacije izvode se u korisničkom načinu rada, a pristup resursima računala ide preko Izvršnog Servisa.
- Okolinski podsustavi omogućavaju izvođenje aplikacija pisanih za različite operacijske sustave (poput OS/2), a integralni podsustavi su zaduženi za sistemske funkcije.
- Kod procesora sa zaštitom memorije (*memory protection*), kernel mod ili privilegirani način rada služi za izvođenje operacijskog sustava koji je, pojednostavljeno gledajući, aplikacija. Aplikacija koja se izvodi u privilegiranom načinu rada ima neograničen pristup memoriji i vanjskim uređajima računala. Zaštita memorije štiti memoriju na način da onemogućava jednom procesu da ošteti memoriju drugog procesa koji se istovremeno izvodi.

WIN32 API biblioteke

- Kako bi olakšao i omogućio programiranje za svoje operacijske sustave Microsoft je razvio *Application Programming Interface* (API).
- Za 32-bitne platforme koristi se Microsoft® Win32® API.
- API sadrži skup funkcija, poruka (*messages*) i struktura koje su smještene u sljedećim dll-ovima (*dynamic link libraries*) tj. dinamički linkanim bibliotekama: KERNEL32.DLL, ADVAPI32.DLL, GDI32.DLL, USER32.DLL, i CRTDLL.DLL.

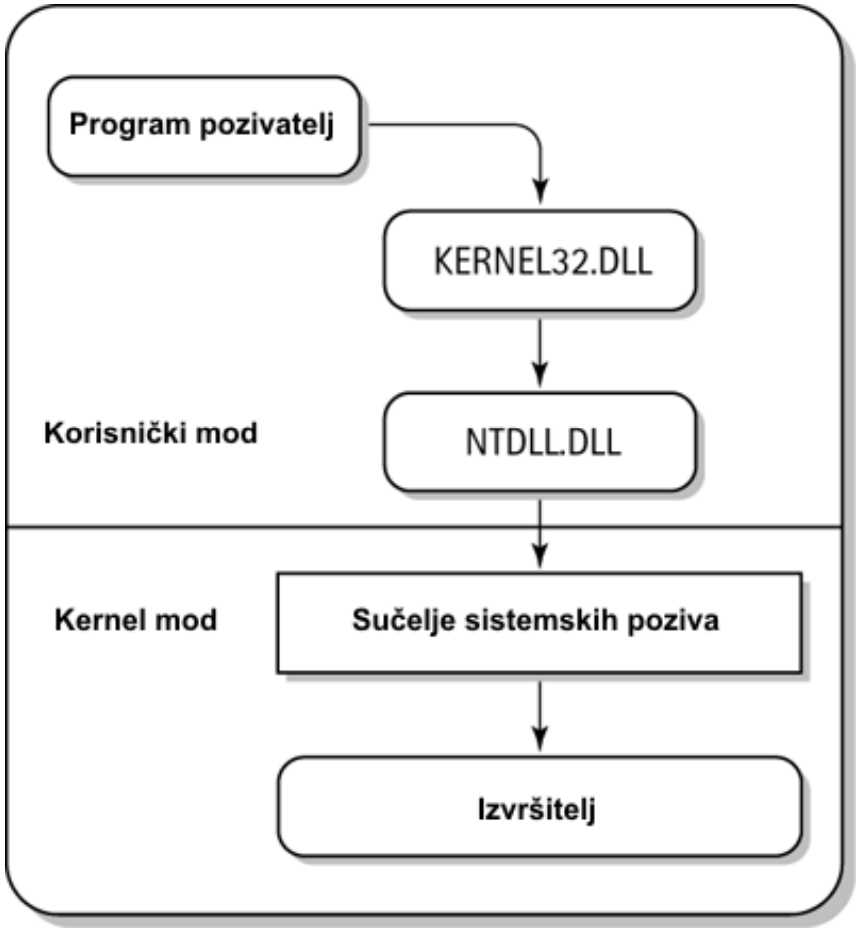
BIBLIOTEKE

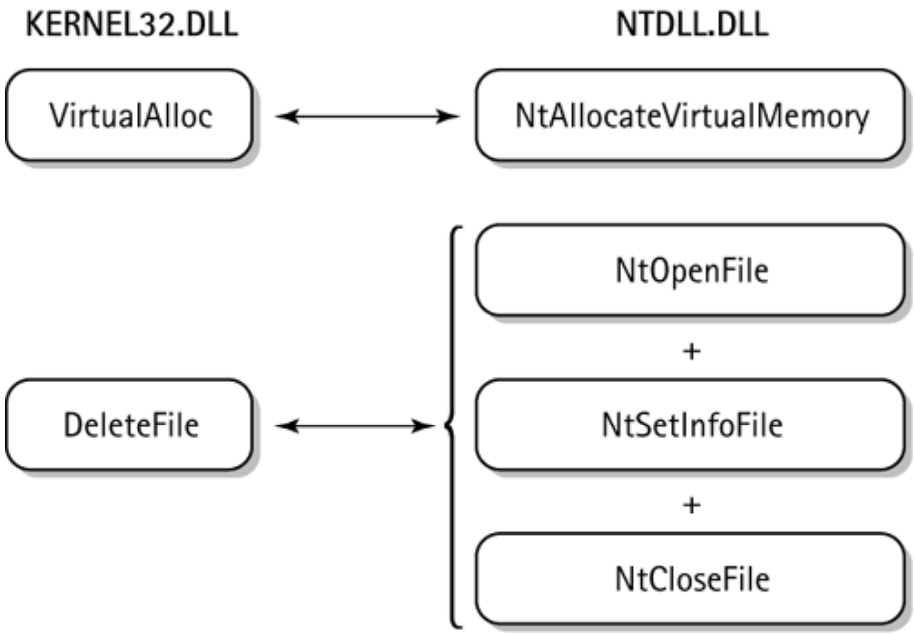
- **Statičko linkanje**
- Osim dinamički linkanih biblioteka postoje i statički linkane biblioteke s kakvima ste se već susretali. Kod statičkog linkanja biblioteka one postaju dio exe koda aplikacije. Svi pozivi funkcija se rješavaju prilikom linkanja pa se stoga to i zove statičko linkanje. (.lib)
- **Dinamičko linkanje**
- Dinamičkim linkanjem se aplikacija povezuje sa bibliotekom tijekom izvođenja (run time). Biblioteke su zasebni izvodivi file-ovi i ne moraju se kopirati u exe kod aplikacije kao kod statičkog linkanja. Te biblioteke nazivamo (dynamic-link libraries) DLL-ovi da bi naglasili da se aplikacija vezuje uz biblioteku prilikom učitavanja u memoriju i izvođenja, a ne prilikom linkanja. Kada aplikacija koristi funkciju iz nekog DLL, operacijski sustav učitava taj DLL u memoriju i rješava reference na funkcije koje aplikacija poziva, te briše DLL iz memorije kada više nije potreban. (.dll)

IZVORNI (NATIVE) API

- Izvorni (*native*) API su servisi koji su jezgra servisa operacijskog sustava na raspolaganju pogonskim programima uređaja i korisničkim aplikacijama.
- Win32 podsustav oslanja se na izvorni API.
- Kod NT-ja na raspolaganju je preko 200 sistemskih poziva kroz izvorne API, a samo 21 je dokumentiran od Microsofta.

IZVORNI (NATIVE) API

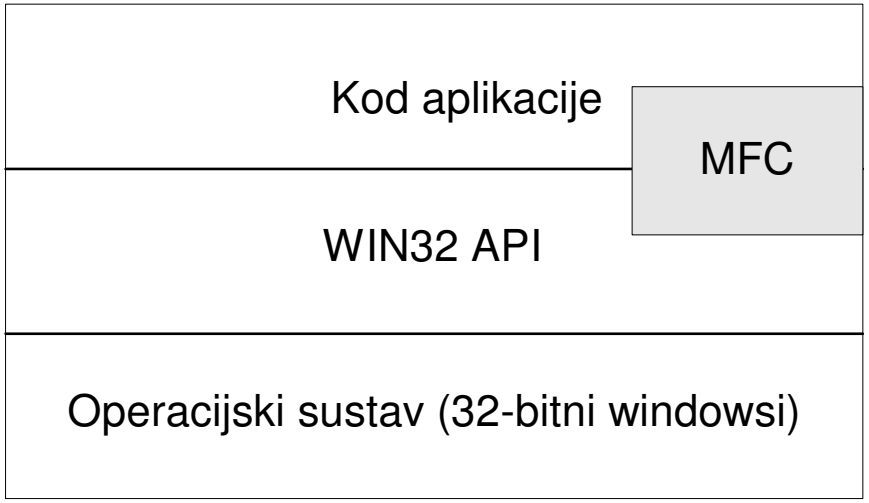




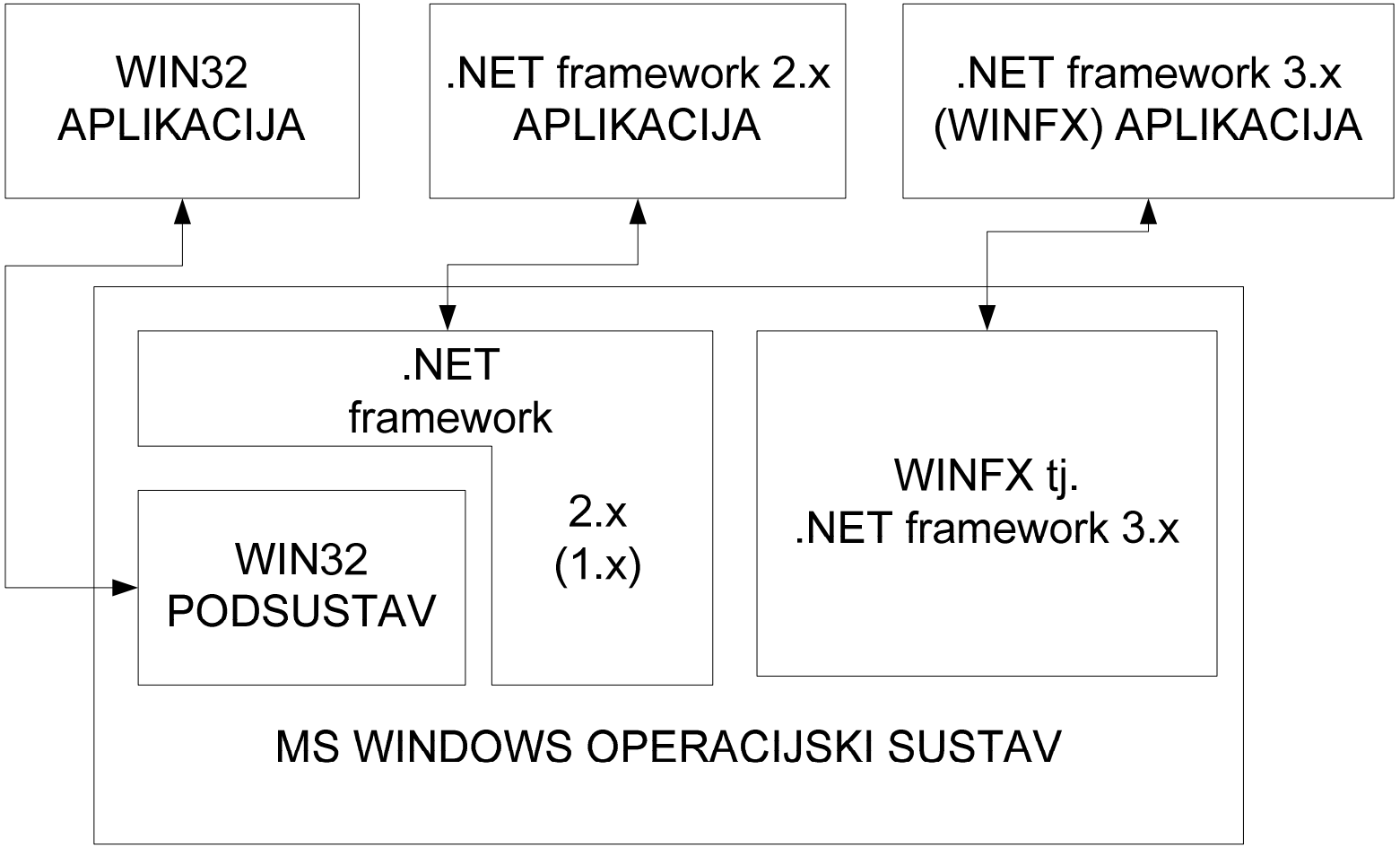
KERNEL32.DLL je dio WIN32 podsustava. Preko poziva WIN32 API funkcija koje su implementirane u nekom dll pozivaju se sistemske funkcije (*nativ* API).

MFC

- *Microsoft Foundation Class biblioteke* (MFC) klase koji su omotači (wrapper) WIN32 API funkcija.
- Općenito, MFC sadrži C++ klase koje opisuju objekte kao što su prozori, dijalog prozori i slični objekti važni za grafičko korisničko sučelje. MFC članske funkcije klasa pozivaju API funkcije i olakšavaju znatno kreiranje aplikacija u odnosu na direktno pozivanje API funkcija.



.NET


WIN32
APLIKACIJA
.NET framework 2.x
APLIKACIJA
.NET framework 3.x
(WINFX) APLIKACIJA
.NET
framework
2.x
(1.x)
WIN32
PODSUSTAV
WINFX tj.
.NET framework 3.x
MS WINDOWS OPERACIJSKI SUSTAV

Zajednička programska infrastruktura

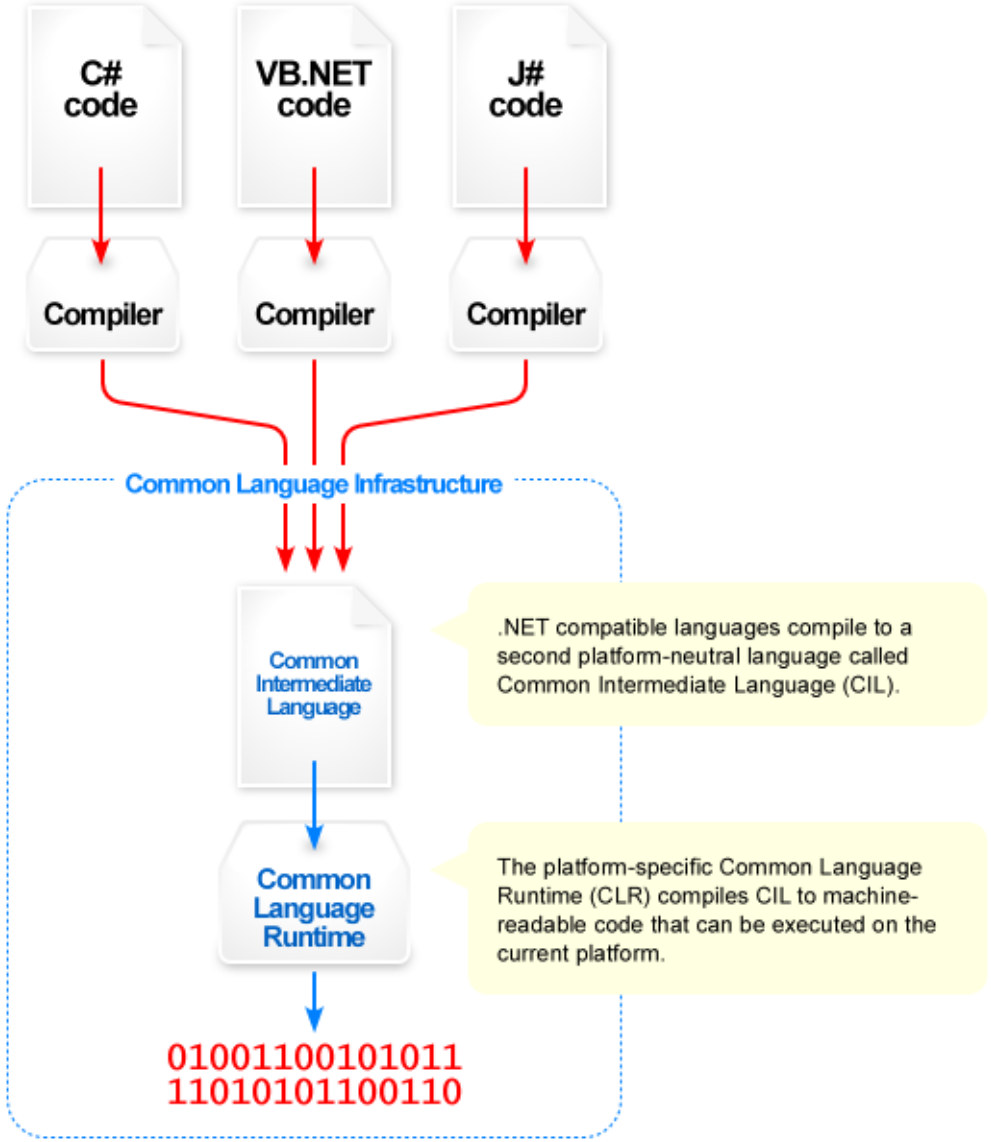
- Zajednička programska infrastruktura (*Common Language Infrastructure* - CLI) je javna specifikacija koju je razvio Microsoft koja opisuje izvršni kod i okolinu za izvršavanje kôda (*runtime environment*) što čini jezgru Microsoft .NET frameworka.
- Specifikacija opisuje okolinu koja omogućava programskim jezicima više razine korištenje na različitim platformama bez prilagođavanja kôda određenoj arhitekturi.
- CLI je specifikacija, a jedna od implementacija te specifikacije je *Common Language Runtime* **-** CLR.

Zajednička programska infrastruktura

- CLI specifikacija definira 4 osnovna područja:
  1. Sustav zajedničkih tipova (*Common Type System* - CTS) - skup tipova i operacija zajedničkih većini programskih jezika.
  2. Metapodatke (*Metadata*) - podaci o strukturi programa neovisni o jeziku, tako da se mogu povezati s različitim jezicima i alatima.
  3. Zajedničku programsku specifikaciju (*Common Language Specification* - CLS) – skup zajedničkih pravila koja treba poštivati programski jezik kako bi bio interoperabilan s drugim CLS usuglašenim jezicima.
  4. Sustav virtualnog izvršavanja (*Virtual Execution System* - VES) — VES učitava i izvršava CLI kompatibilne programe, koristeći metapodatke kombinira zasebne dijelove kôda u cjelinu prilikom izvršavanja.

Zajednička programska infrastruktura

- Svi CLI kompatibilni jezici se kompajliraju u zajednički prijelazni jezik (*Common Intermediate Language* - CIL). To je prijelazni jezik izveden iz hardverske platforme na kojoj se nalazi .NET framework.
- Kada se program izvršava, VES, koji je ovisan o platformi i različit za različite platforme, će kôd kompajlirati iz CIL u strojni jezik u skladu sa hardverom.
- U kolovozu 2000, Microsoft, Hewlett-Packard, Intel, i drugi standardiziraju CLI. U prosincu 2001, standardizacija je potvrđena od strane ECMA, a u travnju 2003 i od strane ISO, a svi patenti koji su bitni za implementaciju CLI su dostupni pod "royalty-free and otherwise RAND (Reasonable & Non-Discriminatory)" uvjetima.


C# code
VB.NET code
J# code
Compiler
Compiler
Compiler
Common Language Infrastructure
Common Intermediate Language
.NET compatible languages compile to a second platform-neutral language called Common Intermediate Language (CIL).
Common Language Runtime
The platform-specific Common Language Runtime (CLR) compiles CIL to machine-readable code that can be executed on the current platform.
01001100101011
11010101100110

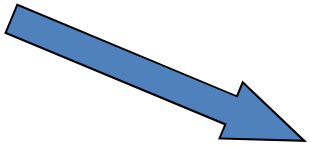
Zajednički prijelazni jezik

- Zajednički prijelazni jezik (*Common Intermediate Language* - CIL) je programski jezik niske razine, ali još uvijek *human-readable* u zajedničkoj programskoj infrastrukturi i .NET frameworku.
- Programi u programskim jezicima na .NET platformi se kompajliraju i sklapaju u "bytecode" ili PE (*portable executable*) datoteku (.exe, .dll). Bytecode još nije strojni kôd. To je također prijelazni kôd, ali u binarnom zapisu koji CLR prilikom izvođenja pretvara u strojni kôd. CIL nalikuje objektno-orijentiranom jeziku.
- U početku se CIL zvao *Microsoft Intermediate Language* - MSIL.

Zajednički prijelazni jezik

C#

```
namespace primjer_CLI_kod
{
    class Program
    {
        static void Main()
        {
            System.Console.WriteLine("Hello, world!");
        }
    }
}
```



CIL

```
.method public static void Main() cil managed
{
.entrypoint
.maxstack 1
ldstr "Hello, world!"
call void [mscorlib]System.Console::WriteLine(string)
ret
}
```

C:\Program Files\Microsoft SDKs\Windows\v7.0A\bin\ildasm

PE FORMAT

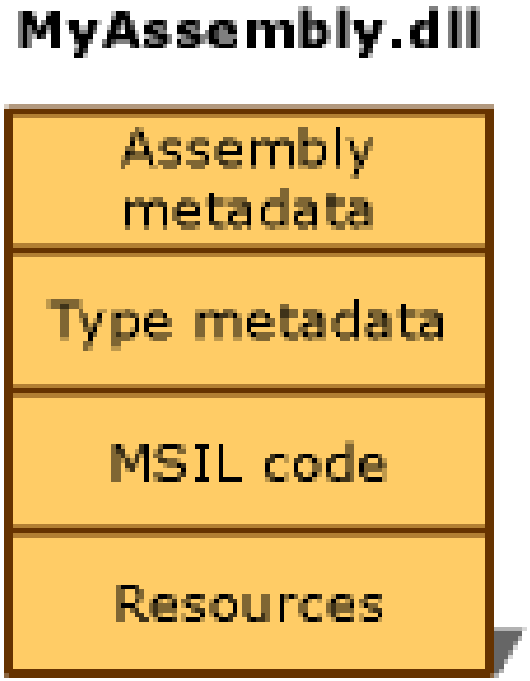
- Izvršna datoteka je datoteka čiji sadržaj računalo interpretira kao program koji se izvršava. Najčešće sadrži binarni zapis strojnih instrukcija, ali sadržaj može biti zapisan i u obliku nekog međuformata koji se ne izvršava direktno na procesoru.
- PE (*portable executable*) format je format izvršnih datoteka na Windows OS (proizišao iz *Common Object File Format* (COFF) s VAX/VMS od *Digital Equipment Corporation*) (exe, dll, ocx). "Portable" se odnosi na to da sve Windows platforme i svi procesori mogu prepoznati program.
- Na 64-bitnim Windowsima koristi se malo izmijenjena verzija PE koja se označava s PE32+.
- PE format je podatkovna struktura koja kaže Windows *loader-u* (Ntdll.dll je specijalna sistemska biblioteka, sadrži između ostalog *loader* izvršnih datoteka kao funkcije sa prefiksom Ldr) kako će rukovati s izvršnim kôdom sadržanim u datoteci.

PE FORMAT

- Od uvođenja .NET platforme postoje dva različita mehanizma učitavanja izvršnih modula (EXE ili DLL):
  - jedan za Windows, koji se koristi za neupravljive module (*unmanaged modules*),
  - i jedan za .NET Common Language Runtime (CLR) koji učitava .NET skup (*assembly*) tj. PE datoteku. (Assembly - the unit of deployment that allows the CLR to fully understand the contents of an application and to enforce the versioning and dependency rules defined by the application.)
- Kada se pokrene aplikacija, bilo da se radi o upravljivom ili neupravljivom kôdu, poziva se Windows loader koji je zadužen za učitavanje i izvršavanje aplikacije. Windows loader iz PE zaglavlja provjerava da li se radi o upravljivom kôdu. U slučaju upravljivog kôda Windows loader pokreće CLR koji je u tom slučaju zadužen za učitavanje i izvršavanje aplikacije.

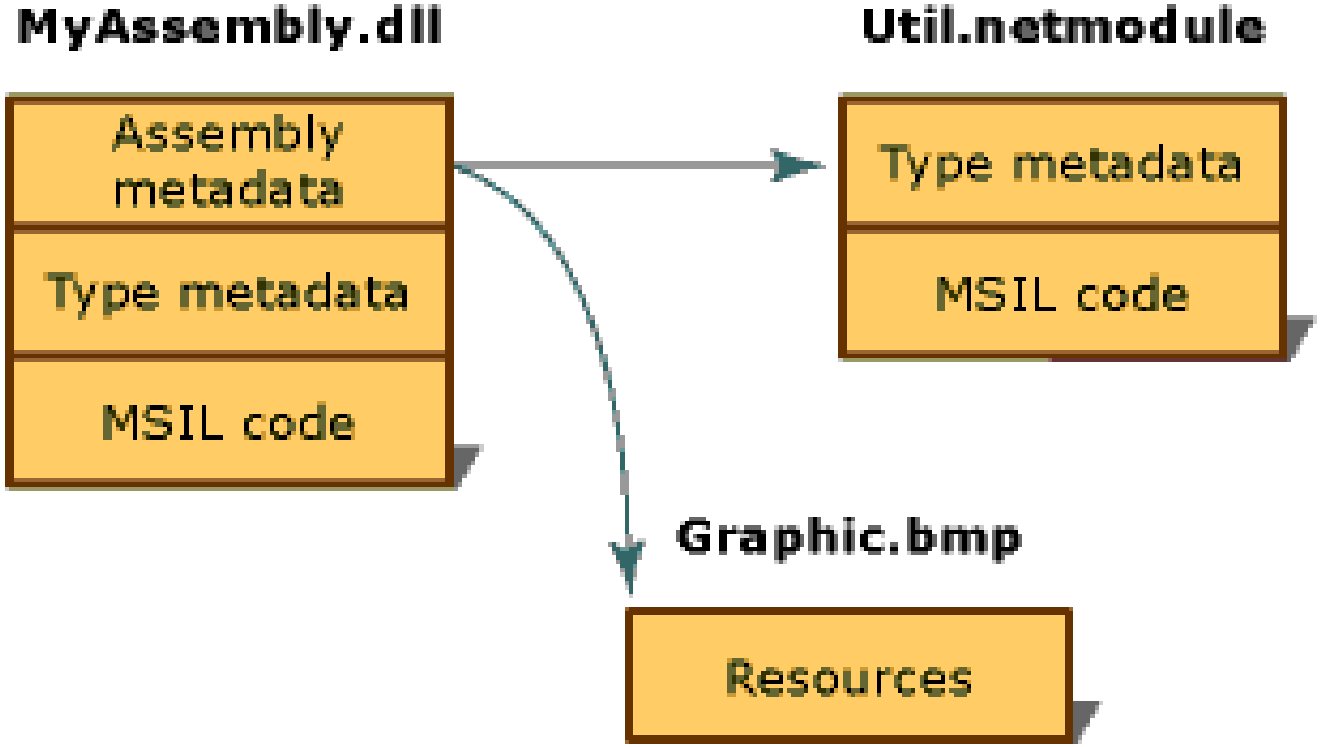
.Net skup (.NET *Assembly*)

- .Net skup može sadržavati četiri dijela:
  - Manifest skupa (*assembly manifest*) koji sadrži određene metapodatke skupa.
  - Metapodataci tipova
  - IL kôd koji implementira tipove.
  - Skup resursa.
- Manifest je obavezan dio.



.Net skup (.NET *Assembly*)

- Elementi *assemblyja* mogu se smjestiti u jednu datoteku ili se organizirati u više zasebnih datoteka.
- Zasebne datoteke mogu biti biblioteke (dll), moduli kompajliranog kôda (IL kôda) (.netmodule), resursi (npr. .bmp ili .jpg slike), ili neke druge datoteke.



Netmodule

- .Net skup organiziran u više datoteka koristi se kada postoje dijelovi kôda pisani u različitim jezicima pa se jedani dio kompajlira u jedan modul, a drugi u drugi, itd.
- Ili za optimiranje izvođenja aplikacije tako da se dijelovi kôda koji se rijetko koriste stavljaju u zasebne datoteke jer .NET framework učitava modul tek kada se pozove.
- Ili za razvoj aplikacije između više programera.

Netmodule

- Netmodule je kompajlirani kod.
- Ne može se izvršavati sam nego treba biti povezan unutar nekog *assemblyja*. Ne sadržava manifest nego samo metadpodatke i izvršni kod.
- Netmodul se dobiva kada se kod kompajlira sa opcijom /addmodule, ali to se treba raditi ručno iz konzole, a ne iz Visual Studia.

.Net skup (.NET *Assembly*)

- Smještaj skupa određuje može li CLR učitati određeni program (može li ga dohvatiti) i da li se skup može dijeliti s drugim skupovima. Skup se može smjestiti na sljedećim lokacijama:
  - U direktoriju ili poddirektoriju aplikacije. To je uobičajena lokacija.
  - U globalnom kešu *assembly-ja* (*global assembly cache*). Taj se globalni keš instalira na računalu prilikom instalacije CLR-a. Ukoliko je potrebno skup dijeliti s drugim aplikacijama treba ga smjestiti u globalnom kešu skupova. (Gacutil.exe - *Global Assembly Cache* alat je jedan od načina da smjestite skup u globalnom kešu *assembly-ja*. - cmd /C "C:/Program Files/Microsoft Visual Studio 8/SDK/v2.0/Bin/gacutil.exe" /l)
  - Na FTP serveru.

Hosting proces (vshost.exe)

- *Hosting* proces je svojstvo Visual Studio 2005 koje poboljšava postupak debugiranja, omogućava debugiranje djelomično pouzdanih aplikacija (*partial trust debugging*), i evaluaciju izraza prilikom izrade aplikacije (*design time expression evaluation*).
- *Hosting* proces datoteka sadrži u imenu vshost (get_save_file_name_c_sharp.vshost.exe) i smješta se u izlazni direktorij projekta.
- Project->Properties->Debug->Enable the Visual Studio hosting process

SUSTAV ZAJEDNIČKIH TIPOVA

Usporedba osnovnih tipova podataka

Content and size	Visual Basic	Visual J#	C++	C#	JScript	Visual FoxPro
Unknown data	n/a	**Object**	**VARIANT**	n/a	**Object**	**Variant**
Decimal	Decimal (.NET Framework structure)	n/a	**DECIMAL**	decimal	**decimal**	n/a
Date	Date (.NET Framework structure)	**java.util.Date** DateTime	**DATE**	**DateTime**	**DateTime** **Date** object	**Date** **DateTime**
SBCS character (1 byte)	n/a	n/a	signed char __int8	n/a	**sbyte**	**Character**
Unicode character (2 bytes)	Char (.NET Framework structure)	**char**	**wchar_t**	char	**char**	n/a
Unicode character sequence	String (.NET Framework class)	**java.lang.String** String	**wchar_t***	string	**String**	**VarChar**
Boolean (platform dependent)	Boolean (.NET Framework structure)	**boolean**	**VARIANT_BOOL**	bool	**boolean**	**Logical**
1 byte	SByte Data Type (Visual Basic) (.NET Framework structure)	n/a	**signed char**	sbyte	n/a	n/a
2 bytes	Short (.NET Framework structure)	**short**	**signed short int** __int16	short	**short**	n/a
4 bytes	Integer (.NET Framework structure)	**int**	long, (long int, signed long int)	int	**int**	**Integer**
8 bytes	Long (.NET Framework structure)	**long**	__int64	long	**long**	**Float**
1 byte unsigned	Byte (.NET Framework structure)	**byte**	**BYTE** bool	byte	**byte**	**Integer**
2 bytes unsigned	UShort Data Type (Visual Basic) (.NET Framework structure)	n/a	**unsigned short**	ushort	n/a	n/a
4 bytes unsigned	UInteger Data Type (.NET Framework structure)	n/a	**unsigned int** and **unsigned long**	uint	n/a	n/a
8 bytes unsigned	ULong Data Type (Visual Basic) (.NET Framework structure)	n/a	**unsigned __int64**	ulong	n/a	n/a
4 bytes floating point	Single (.NET Framework structure)	**float**	float	float	**float**	**Float**
8 bytes floating point	Double (.NET Framework structure)	**double**	double	double	**Double**	**Double**

Imena osnovnih podatkovnih tipova varijabli u različitim jezicima i CLR ime

C# type name	Visual Basic type name	JScript type name	Visual C++ type name	Ilasm.exe representation	CLR type name
sbyte	SByte	sByte	char	int8	SByte
byte	Byte	byte	unsigned char	unsigned int8	Byte
short	Short	short	short	int16	Int16
ushort	UInt16	ushort	unsigned short	unsigned int16	UInt16
int	Integer	int	int	int32	Int32
uint	UInt32	uint	unsigned int	unsigned int32	UInt32
long	Long	long	__int64	int64	Int64
ulong	UInt64	ulong	unsigned __int64	unsigned int64	UInt64
float	Single	float	float	float32	Single
double	Double	double	double	float64	Double
bool	Boolean	boolean	bool	bool	Boolean
char	Char	char	wchar_t	char	Char
string	String	string	String	string	String
object	Object	object	Object	object	Object

IntPtr tip

- IntPtr tip je ovisan o platformi (*platform-specific type*).
- Služi za predstavljanje pokazivača ili handle-a. Sadrži cjelobrojnu vrijednost čija veličina ovisi o platformi.
- Ukoliko je 32-bitna platforma onda je i vrijednost 32-bitna, ukoliko je platforma 64-bitna onda je i vrijednost 64-bitna.

IMENOVANJE VARIJABLI .NET

- .NET Framework (radno okruženje) ne koristi Mađarsku notaciju.
- Čak .NET Framework smjernice (Guidelines) savjetuju programere da ne koriste tu notaciju.
- http://msdn2.microsoft.com/en-us/library/ms229045.aspx
  - Word Choice
    - Do choose easily readable identifier names. For example, a property named HorizontalAlignment is more readable in English than AlignmentHorizontal.
    - Do favor readability over brevity. The property name CanScrollHorizontally is better than ScrollableX (an obscure reference to the X-axis).
    - Do not use underscores, hyphens, or any other nonalphanumeric characters.
    - Do not use Hungarian notation.
    - Hungarian notation is the practice of including a prefix in identifiers to encode some metadata about the parameter, such as the data type of the identifier.
    - Avoid using identifiers that conflict with keywords of widely used programming languages. While CLS-compliant languages must provide a way to use keywords as regular words, best practices dictate that you do not force developers to know how to do this. For most programming languages, the language reference documentation contains a list of the keywords used by the languages.

IMENOVANJE VARIJABLI .NET

- Abbreviations and Acronyms
  - In general, you should not use abbreviations or acronyms. These make your names less readable. Similarly, it is difficult to know when it is safe to assume that an acronym is widely recognized. Do not use abbreviations or contractions as parts of identifier names. For example, use OnButtonClick rather than OnBtnClick. Do not use any acronyms that are not widely accepted, and then only when necessary.
- Language-Specific Names
  - Do use semantically interesting names rather than language-specific keywords for type names. For example, GetLength is a better name than GetInt. Do use a generic common language runtime (CLR) type name, rather than a language-specific name, in the rare cases when an identifier has no semantic meaning beyond its type. For example, a method that converts data to Int16 should be named ToInt16, not ToShort because Short is the language-specific type name for Int16.

Referentni tipovi podataka u C#

- Varijable referentnog tipa sadrže referencu na stvarne podatke.
  - class
  - interface
  - delegate
- Ugrađeni referentni tipovi:
  - object
  - string

Sučelje

- Sučelje sadrži samo signature metoda, delegata ili događaja. Implementacija metoda nalazi se u klasi koja implementira sučelje.

```
interface ISampleInterface
{ void SampleMethod(); }

class ImplementationClass : ISampleInterface
{       public void ISampleInterface.SampleMethod()
        { // Method implementation. }
}
static void Main()
{ ISampleInterface obj = new ImplementationClass();
obj.SampleMethod(); }
```

Delegat

- Ključna riječ *delegate* se koristi za deklariranje referentnog tipa koji sadrži metodu.
- Delegati su slični funkcijskim pokazivačima u C ili C++, ali su sigurniji jer su to objekti pa omogućavaju provjeru tipa argumenata i povratne vrijednosti.
- Delegat omogućava prenošenje funkcije kao parametra; funkcija koja se prenosi mora biti istog tipa kao što je definirano u delegatu.
- Delegati se obično koriste za obradu događaja (*event handling*), *callback* funkcija i sl.

C sintaksa

```
// deklaracija pokazivača
int (*pt2Function)(float, char, char);
int DoMore(float a, char b, char c)
{ printf("DoMore\n"); return a-b+c; }
// povezivanje pokazivača i funkcije
pt2Function = &DoMore;
// poziv funkcije preko pokazivača
int result2 = (*pt2Function) (12, 'a', 'b');
 // prenošenje funkcije preko pokazivača u drugu funkciju
void PassPtr(int (*pt2Func)(float, char, char))
{
    int result = (*pt2Func)(12, 'a', 'b');
    printf(“%d”,result);
}
```

Delegat C#

```
using System;
delegate void SampleDelegate(string message);
class MainClass {
        static void SampleDelegateMethod(string message)
        {
                Console.WriteLine(message);
        }
        static void Main()
        {
                SampleDelegate d11 = new SampleDelegate (SampleDelegateMethod);
                SampleDelegate d1 = SampleDelegateMethod;
                SampleDelegate d2 = delegate(string message)
                {
                        Console.WriteLine(message);
                 };
                d1("Hello");
                d2(" World");
        }
    }
```

Događaj (Event)

- Događaji (event) se koriste da bi se objekt obavijestio o nekakvoj aktivnosti koja bi mogla utjecati na njegovo stanje.
- Dodavanje događaja u klasu radi se pomoću ključne riječi *event* uz koju ide tip delegata i ime događaja.
- Delegat koji je naveden u definiciji događaja se koristi za povezivanja metode sa događajem. Naime što će se desiti kada se dogodi neki događaj definira se nekom metodom.

Događaji

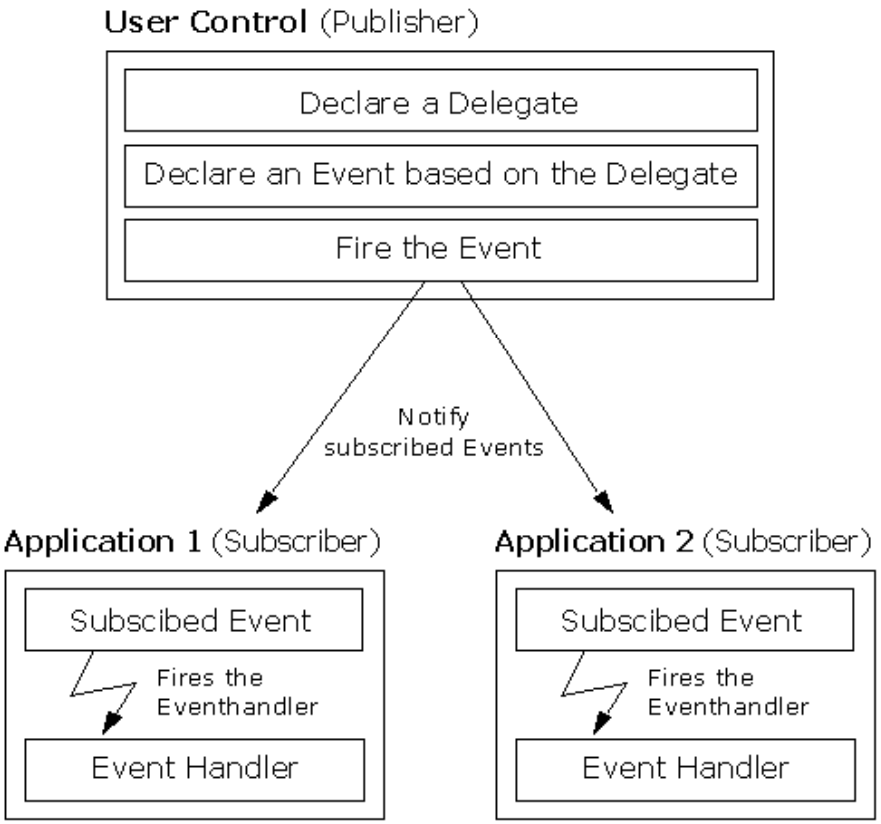
Ime događaja

```
public delegate void SampleEventDelegate(object
   sender, EventArgs e);

public class SampleEventSource
{
public event SampleEventDelegate SampleEvent;
}
```

Delegat uz koji se vezuje funkcija koja će pozvati kada se desi događaj.

Događaji



- Jedan objekt "podiže" događaj, a drugi objekti koji su "pretplaćeni" na taj događaj su obaviješteni o tom događaju.

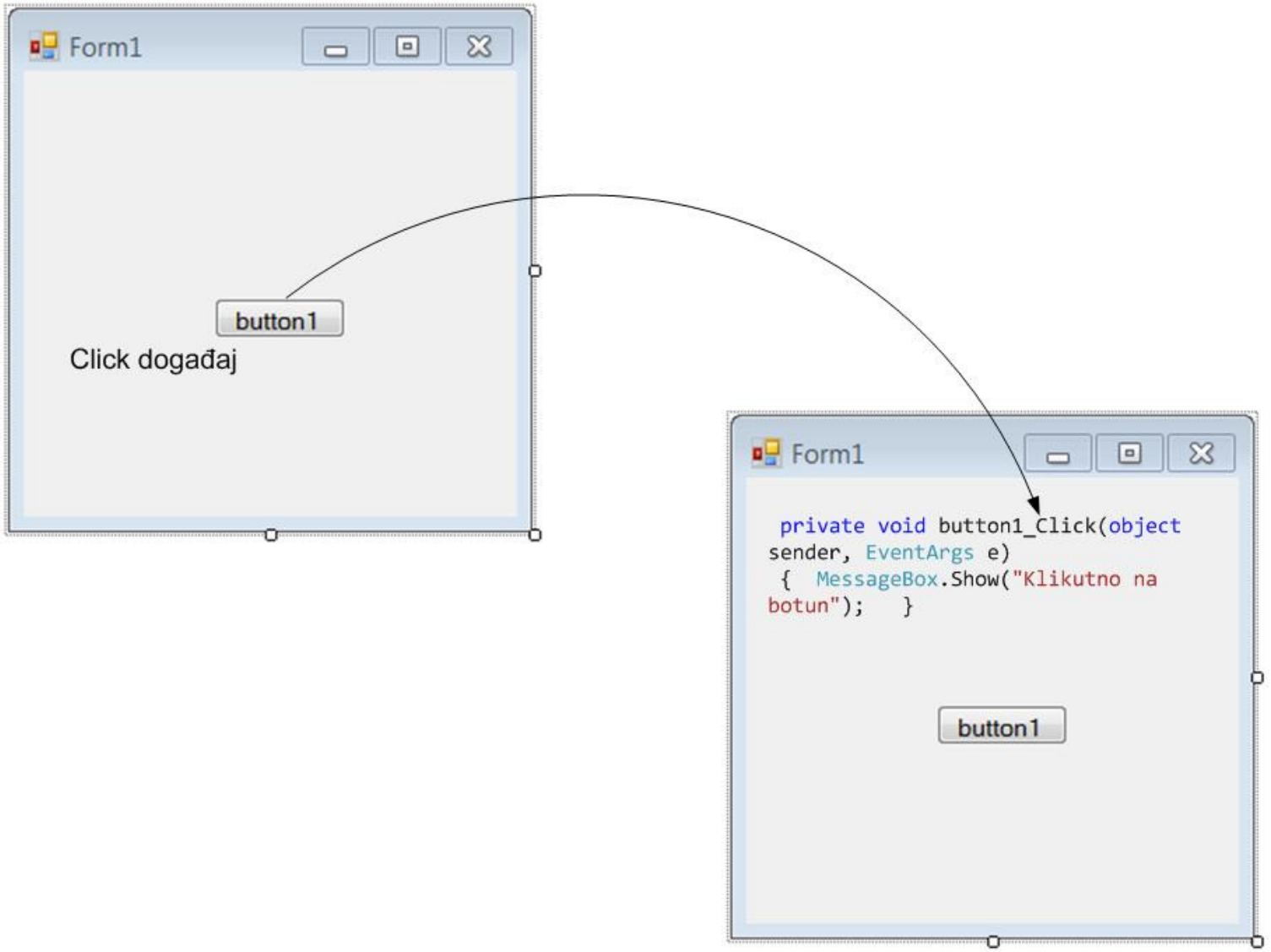
Događaji

- Događaj je akcija sustava na korisničku aktivnost (npr. klik miša, pritisak na tipku tipkovnice i sl.).
- Objekt u C# može publicirati *(publish)* događaje na koje se drugi objekti mogu pretplatiti (*subscribe*). Kada klasa koja je publicirala događaj taj događaj i izgenerira događaj, svi pretplaćeni objekti su obaviješteni o pojavi tog događaja.
- Događaji su svojstva (*properties*) klase koja je publicirala događaj.
- Ključna riječ prije *event* riječi kontrolira na koji način se događaju pristupa iz pretplaćenih klasa.

Događaji

- Funkcije za obradu događaja (koje se zovu *event handlers*) koji su definirani u .NET *frameworku* su povratnog tipa void, a primaju dva parametra.
- Prvi parametar je izvor događaja tj. objekt koji je publicirao događaj, a drugi parametar je objekt klase EventArgs (ili izvedene klase).
- Npr. private void button1_Click(object sender, EventArgs e) {}

Događaji


Form1
button1
Click događaj
Form1
private void button1_Click(object sender, EventArgs e)
{ MessageBox.Show("Klikutno na botun"); }
button1

Događaji - primjer

```
 public partial class Form2 : Form
{     // Definicija delegata
      public delegate void OtvoriDrugiProzor(object jedan, EventArgs dva);
      // Definicija Event baziranog na gornjem delegatu
      public event OtvoriDrugiProzor Otvori;
      // Generiranje eventa
      private void button1_Click(object sender, EventArgs e)
      {         if (Otvori != null)         {Otvori(sender, e);}}
}
PRETPLATA NA DOGAĐAJ
public partial class Form1 : Form
{
      public void Funkcija(object jedan, EventArgs dva)
      {         MessageBox.Show("AAAAAAA");       }
      private void button1_Click(object sender, EventArgs e)
      {         Form2 aaaa=new Form2();
            aaaa.Otvori += new Form2.OtvoriDrugiProzor(Funkcija);
            aaaa.Show();       }
}
```

String

- Tip *string* predstavlja sekvencu Unicode karaktera. To je drugo ime za klasu *String* u .NET frameworku.
- Operator konkatenacije:

```
string a = "good " + "morning";
```

- Indeksni operator:

```
string str = "test";
char x = str[2]; // x = 's';
```

PORUKE

- Windows aplikacije su pokretane događajima (*event-driven*). To znači da aplikacija ne poziva eksplicitno funkciju kojom se dobavlja ulaz (miš, tipkovnica,…) već čeka da sistem sam pošalje neki ulaz aplikaciji.
- Operacijski sustav šalje sve ulaze aplikacije na neki od prozora aplikacije. Svaki prozor ima funkciju koju poziva sustav svaki put kad ima ulaz za prozor. Ta funkcija obrađuje ulaz prozora i nakon toga vraća kontrolu sustavu.

PORUKE PROZORU

- Sustav šalje ulaz funkciji prozora u obliku poruka (*messages*).
- Poruke generira i operacijski sustav i aplikacija. Sustav generira poruku svaki put kad se desi neki događaj (pomicanje miša, pritisak na tipkovnicu). Sustav također generira poruke kao odgovor na promjene uzrokovane aplikacijom (npr. promjena veličine prozora).
- Aplikacija može generirati poruke da bi njezini vlastiti prozori izvršili neki zadatak ili da bi komunicirali sa prozorima drugih aplikacija.

SISTEMSKE PORUKE

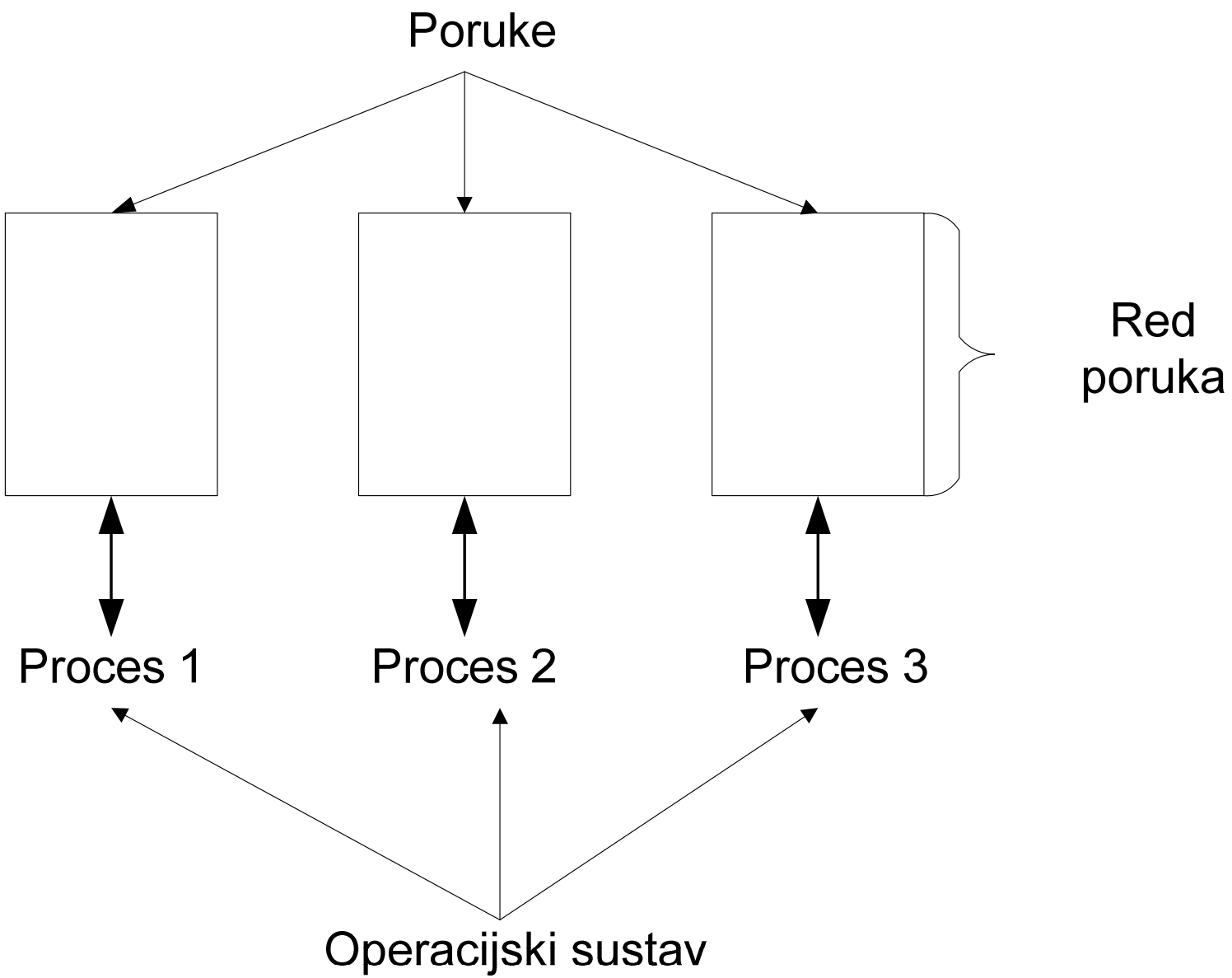
- Sistem (OS) šalje poruku kad komunicira sa aplikacijom.
- Porukama kontrolira rad aplikacije i šalje ulaze i druge informacije aplikaciji na obradu.
- Svaka sistemska poruka ima jedinstveni identifikator poruke i odgovarajuću simboličku konstantu (definiranu u WIN32 API SDK (*software development kit*) header datotekama) koji iznosi svrhu poruke.
- Npr. WM_PAINT konstanta zahtjeva da se prozor ponovo iscrta.

USMJERAVANJE PORUKA

- Dva su načina na koja sustav šalje poruke:

  1. smještanje poruka u FIFO red čekanja koji se zove red poruka (*message queue*). To je sistemski definiran objekt koji privremeno pohranjuje poruke.
  2. slanje poruke direktno prozoru.

- Svaki proces koji se izvodi ima svoj red čekanja poruka i sam je odgovoran za obrađivanje poruka iz svog reda čekanja.


Poruke
Red
poruka
Proces 1
Proces 2
Proces 3
Operacijski sustav

PORUKE

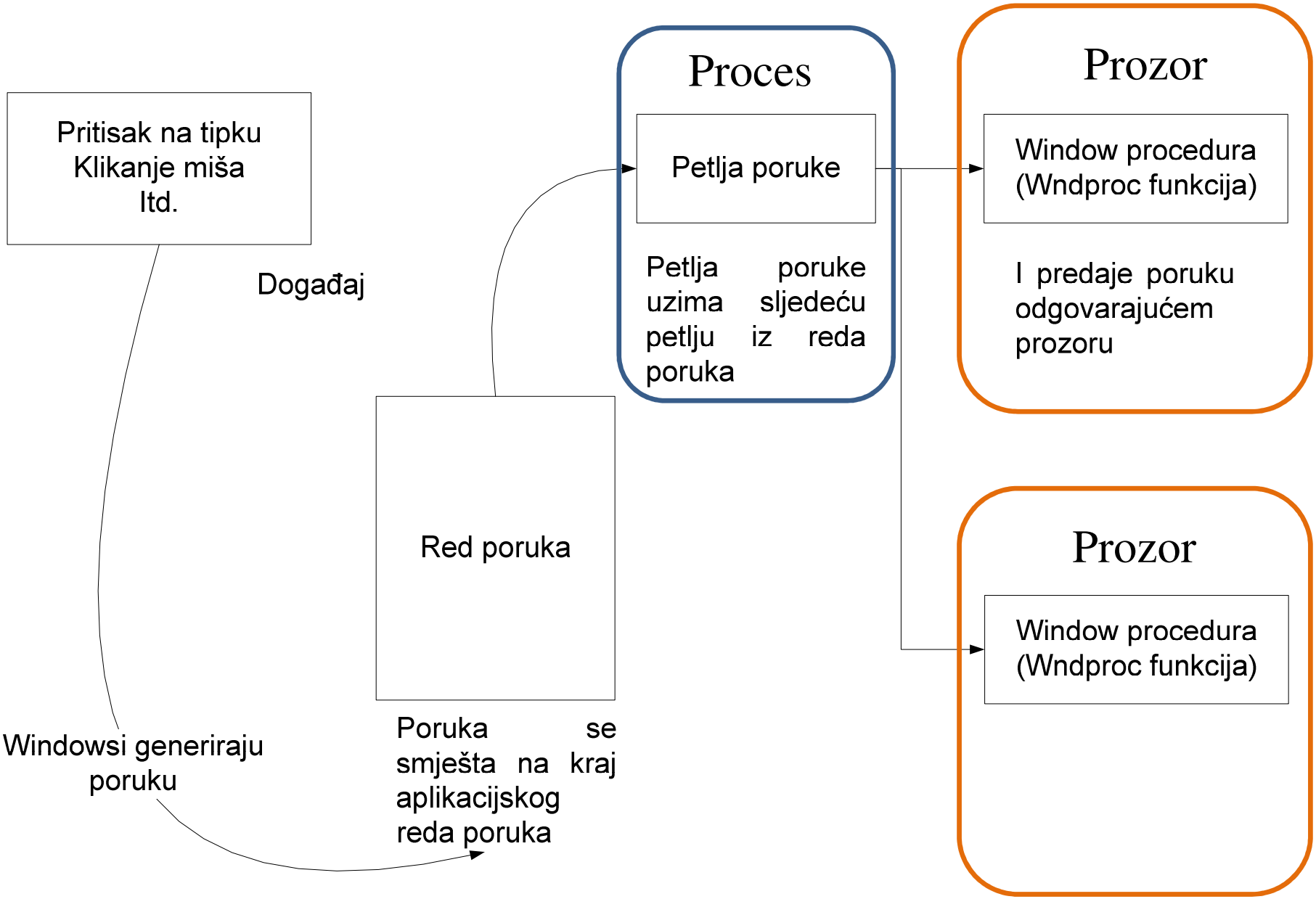
- Poruke koje se smještaju u red poruka zovu se *queued messages*.
- To su u prvom redu poruke s miša ili tipkovnice (WM_MOUSEMOVE, WM_LBUTTONDOWN, WM_KEYDOWN i sl.).
- To su također i poruke timera, poruke za iscrtavanje prozora ili prekid aplikacije (WM_TIMER, WM_PAINT, WM_QUIT).
- Većina ostalih poruka šalje se direktno proceduri prozora za obradu poruke (*nonqueued messages)*.

PORUKE

- Poruke koje se šalju direktno proceduri prozora za obradu poruke zaobilaze red poruka.
- Poruke koje se šalju direktno su obično poruke koje obavještavaju prozor o nekom događaju koji utječe na prozor.
- Npr. kada korisnik otvori novi prozor Windowsi prozoru šalju direktno niz poruka kao što su WM_ACTIVATE, WM_SETFOCUS i WM_SETCURSOR.
- Ove poruke obavještavaju prozor da je aktiviran, da je ulaz s tipkovnice usmjeren na prozor i da je kursor pomaknut unutar granica prozora.

OBRADA PORUKE

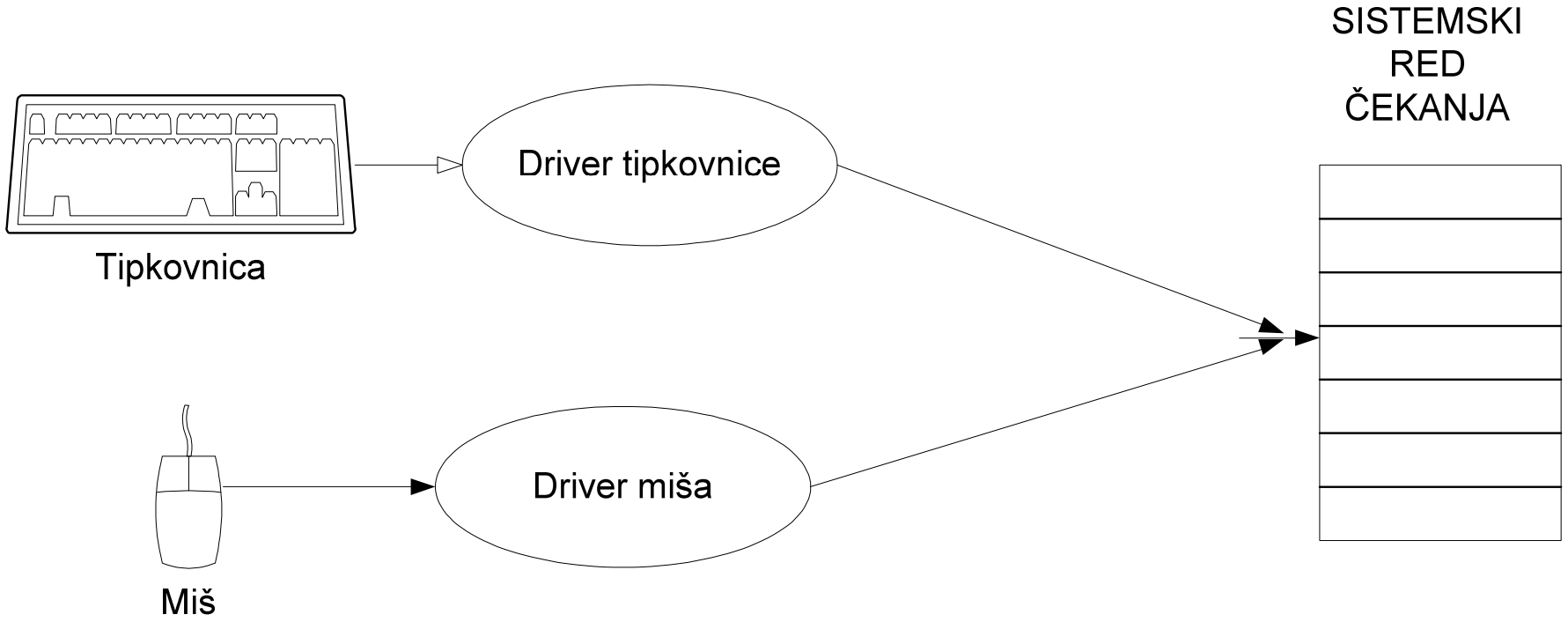
- Aplikacija mora obraditi poruke iz svog reda čekanja.
- Aplikacija za obradu poruka obično koristi tzv. petlju poruka (*message loop*) ili pumpu poruka (drugo ime za istu stvar) (*message pump*).
- Aplikacija s više niti (threadova) treba uključiti petlju poruka u svaku nit koja kreira prozor.


Pritisak na tipku
Klikanje miša
Itd.
Događaj
Windowsi generiraju poruku
Red poruka
Poruka se smješta na kraj aplikacijskog reda poruka
Proces
Petlja poruke
Petlja poruke uzima sljedeću petlju iz reda poruka
Prozor
Window procedura (Wndproc funkcija)
I predaje poruku odgovarajućem prozoru
Prozor
Window procedura (Wndproc funkcija)

REDOVI (QUEUES)

- Dva tipa reda čekanja koriste Windowsi za pohranu događaja i poruka:
  1. Sistemski red čekanja (*system queue*)
  2. Red čekanja poruka (*message queue*)

- Sistemski red čekanja primarno koriste driveri. Driver (pogonski program uređaja) je program za opsluživanje prekida koje generiraju uređaji (miš, tipkovnica,...). Driveri miša i tipkovnice prilikom prekida pozivaju ulazne točke (*entry points*) u programu da bi obavijestili operacijski sustav da se dogodio neki događaj (pritisak na tipku i sl.).



Sve trenutno aktivne aplikacije dijele zajednički sistemski red čekanja. Sistemski red čekanja nikada ne sadrži aplikacijske poruke. Njegova uloga je isključivo da bilježi hardverske događaje.

WIN32 API PETLJA PORUKA

- Jednostavna petlja poruka sastoji se od jednog poziva svake od sljedeće tri funkcije:
  - GetMessage
  - TranslateMessage
  - DispatchMessage
- GetMessage funkcija dohvaća poruku iz reda čekanja i kopira je u strukturu tipa MSG. Funkcija TranslateMessage pretvara poruku tipkovnice u poruku karaktera. DispatchMessage funkcija šalje poruku proceduri prozora za obradu poruka.

WIN32 API PETLJA PORUKA

```
MSG msg;
BOOL bRet;

while( (bRet = GetMessage( &msg, NULL, 0, 0 )) != 0)
{
   if (bRet == -1)
   {
      // obrada greške
   }
   else
   {
      TranslateMessage(&msg);
      DispatchMessage(&msg);
   }
}
```