



Vývoj aplikací v prostředí .NET

© Katedra řídicí techniky,
ČVUT-FEL Praha

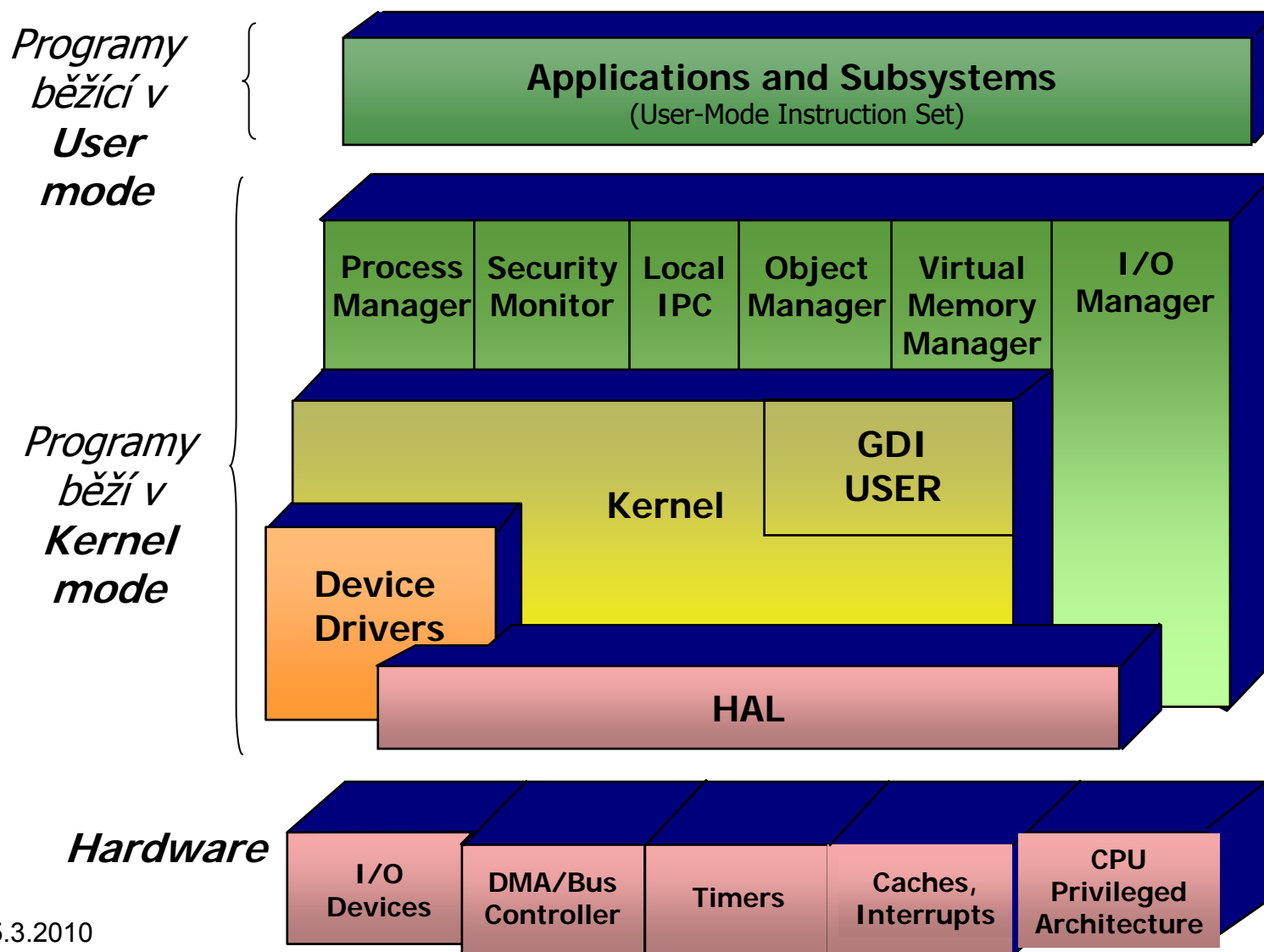
3. přednáška

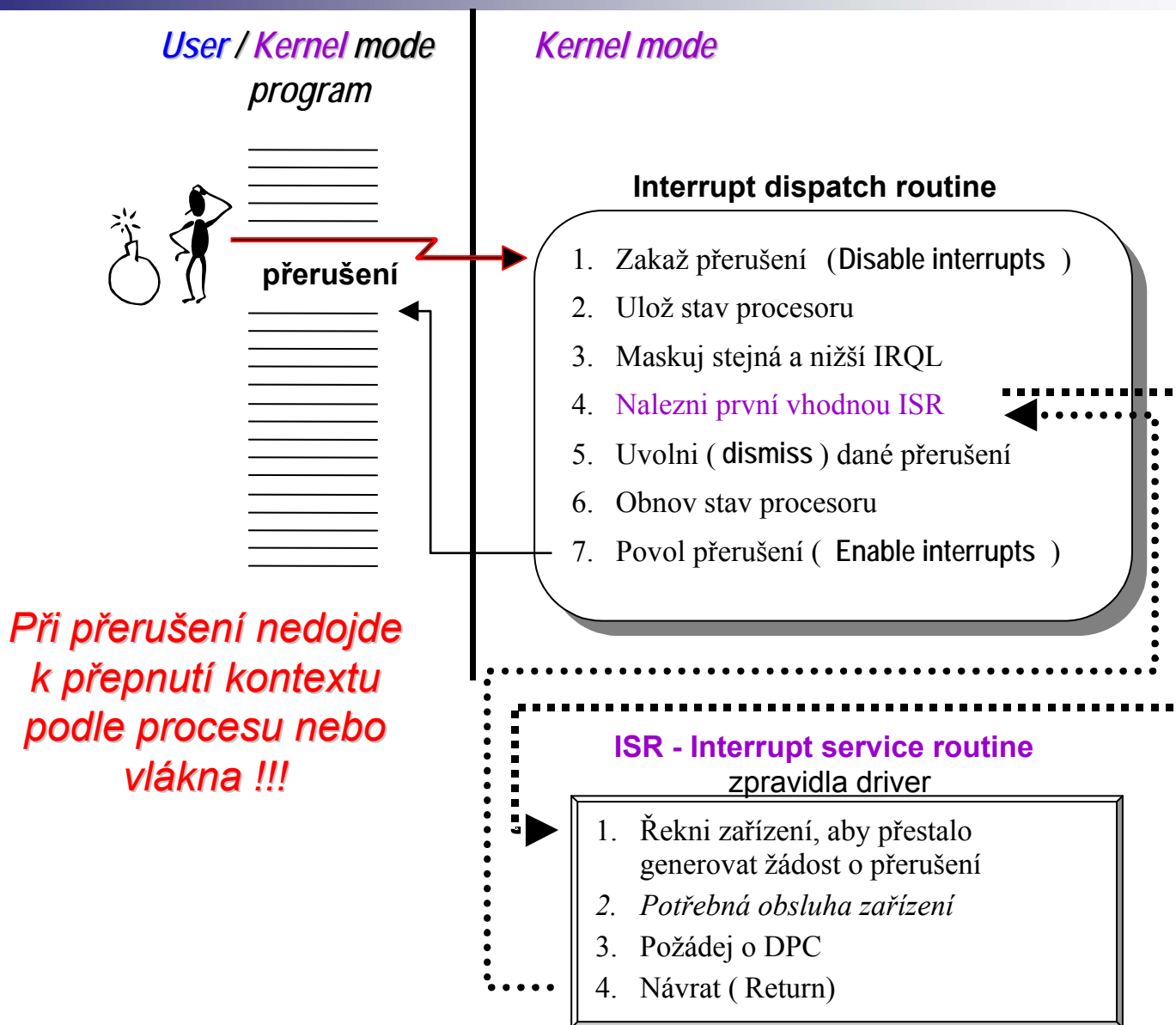
*Co počítač občas pilně dělá,
když vlastně nic nedělá?
Aneb, jak vznikají některé zprávy...*



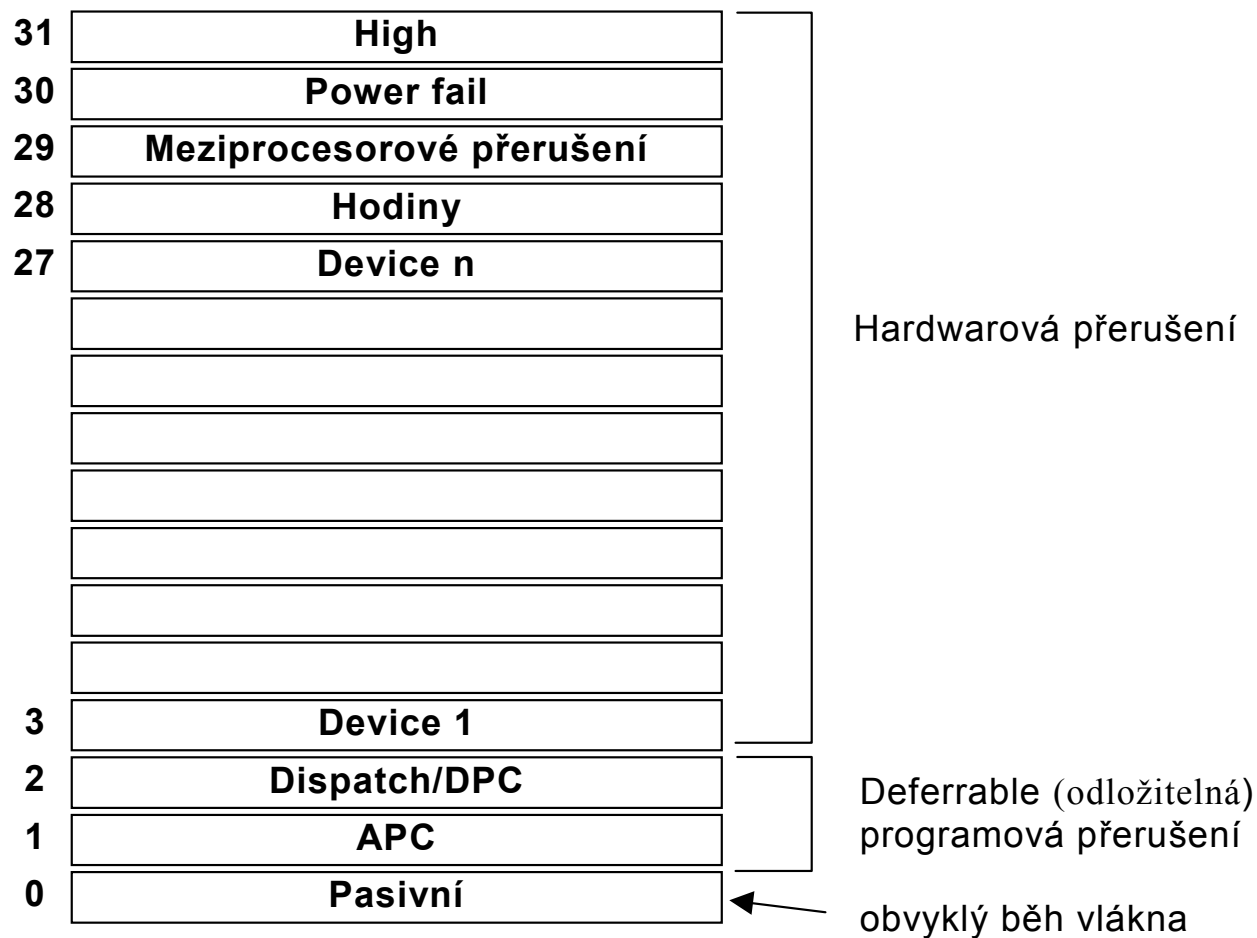
Přerušujeme tento program kvůli rodičovské domluvě...

Struktura operačního systému WinNT



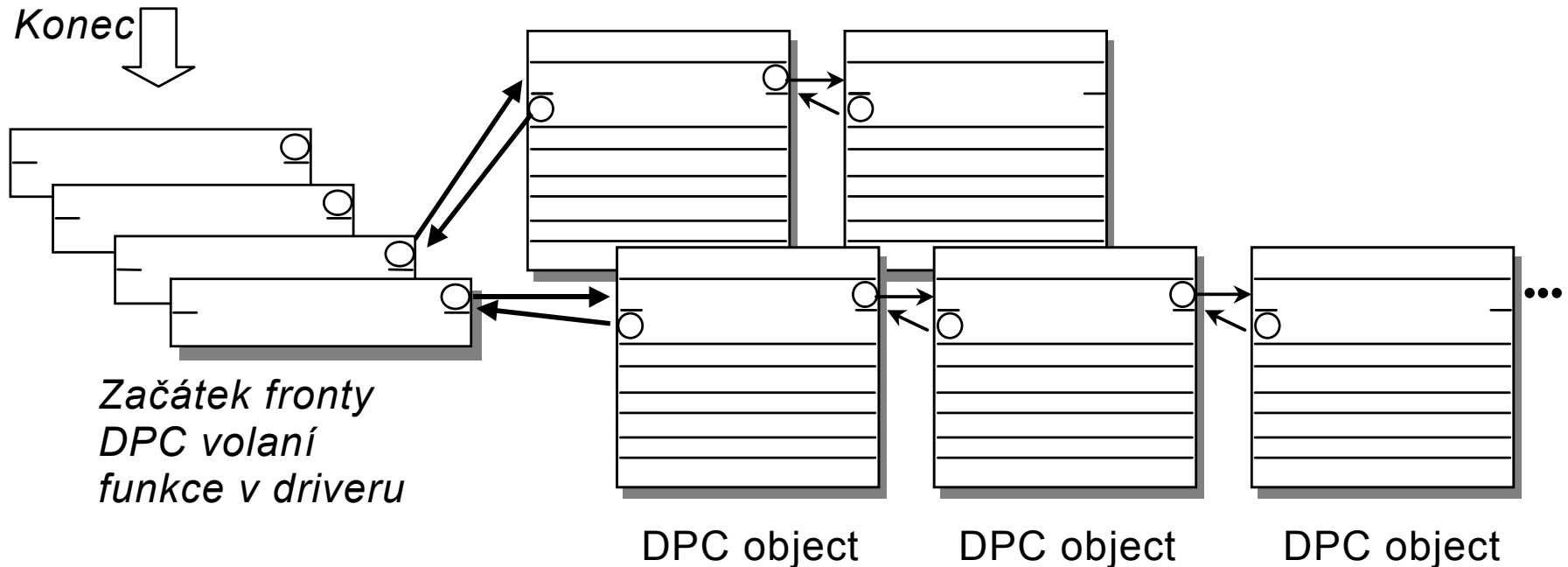


Priorita přerušení skrze IRQL



DPC – Deferred Procedure Calls

- Odložené zpracování z vyšší IRQL na nižší "dispatch" úroveň
 - Driver zařadí požadavek do fronty
 - **Pro každou CPU existuje jedna fronta**
 - **DPC se vykonávají po vyřízení vyšších IRQL**



Zpracování mnohých DPC končí odesláním
zprávy příslušnému vláknu

Co to vlastně počítač provádí?

Administrative tools -> Performance Monitor : Add Counter

- **% DPC time** - čas spotřebovaný na obsluhu všech IRQL 2
- **% interrupt time** - čas spotřebovaný na obsluhu všech IRQL > 2
- **Interrupts/sec.** - počet přijatých přerušení za vteřinu
- **DPC queued/sec.** - počet odložených volání

Pokud neběží žádný proces a systém není idle => přerušení či DPC



[http://www.uibk.ac.at/geotechnik/res/lame_II.html]

Zprávy a události

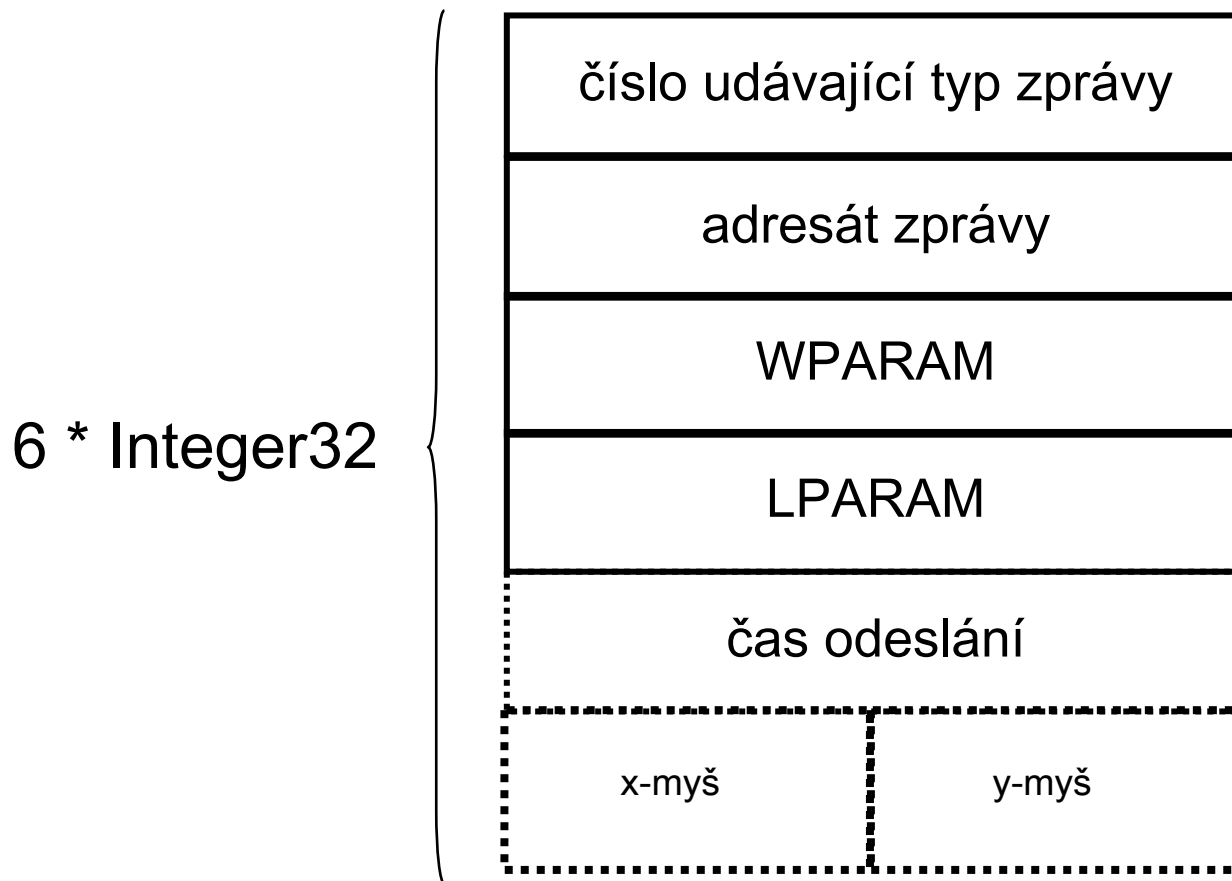
základ Windows a X-Windows



Co jsou to zprávy?

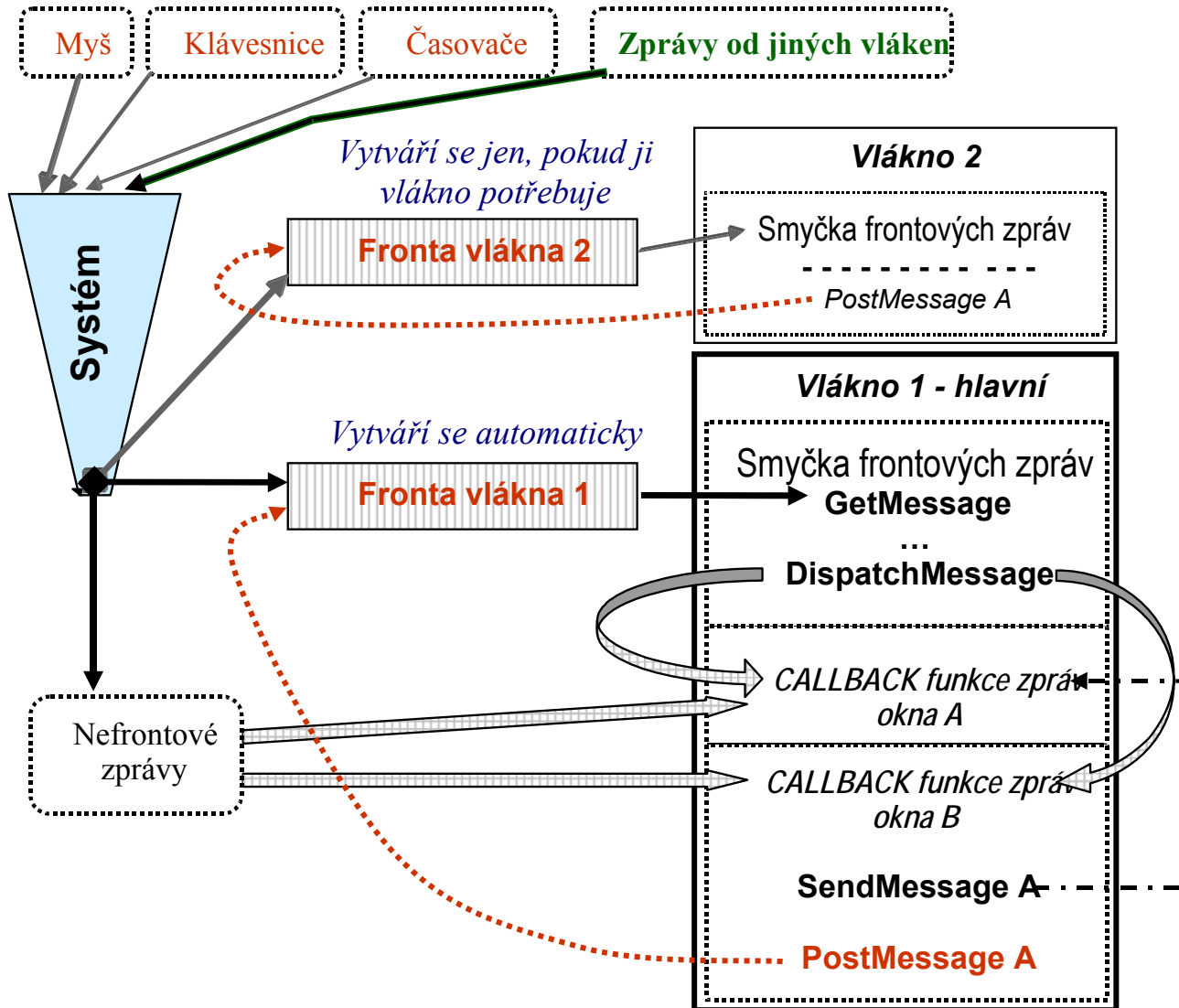
- **zprávy = informace o událostech**

navrženy 1984 v Massachusetts Institute of Technology pro X-Windows jako elegantní a jednoduché řešení pro menší množství sdílených dat.

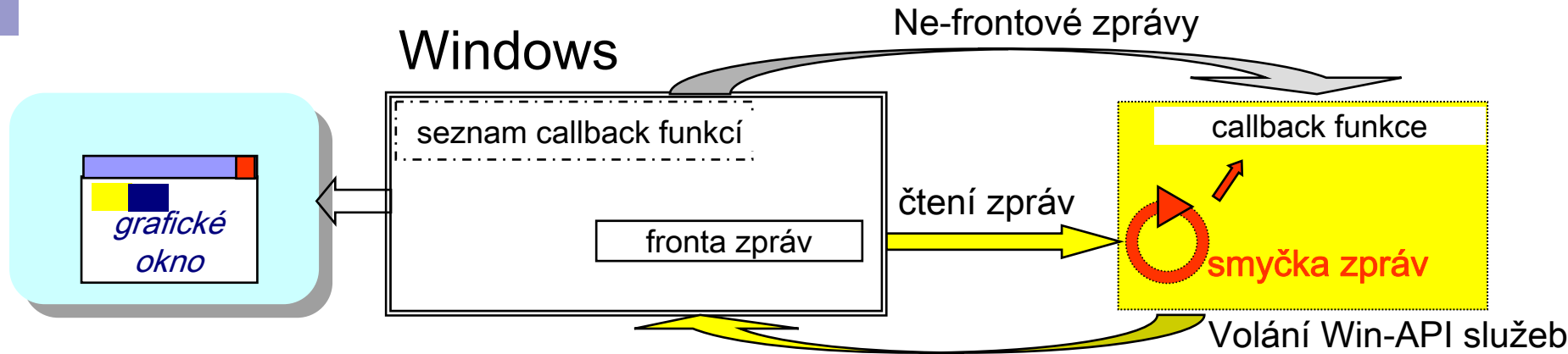


Jak se zpracovávají zprávy?

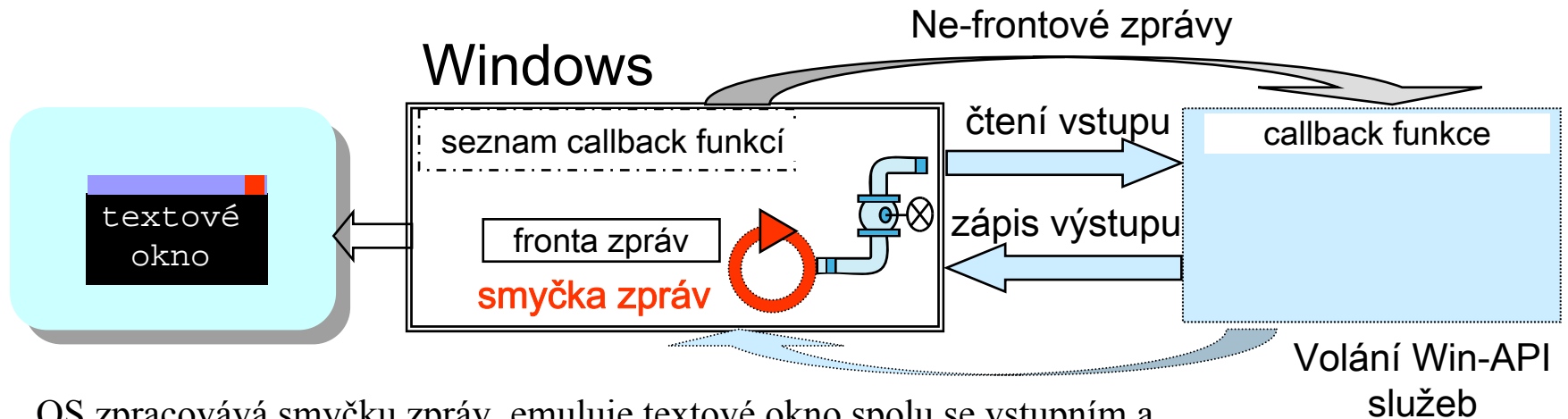
Zjednodušení orientační schéma
– přesnější popis procesu viz další snímky



Aplikace typu GUI - Graphical User Interface



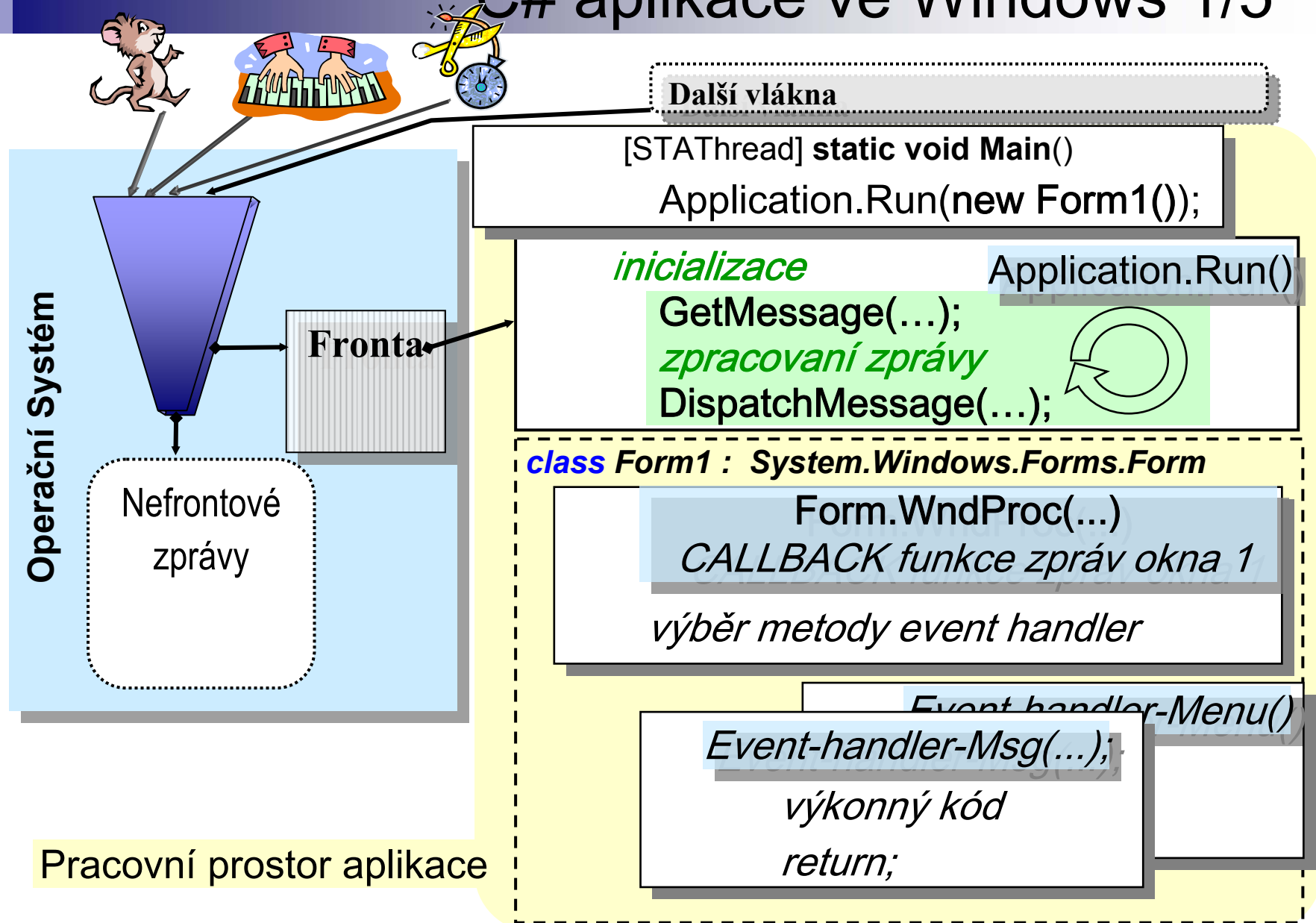
Aplikace typu Console



OS zpracovává smyčku zpráv, emuluje textové okno spolu se vstupním a výstupním proudem (pomocí Win API služby pipe)

- *Překlad pojmu Event Handler je v české literatuře značně nejednotný.*
 - *Použijeme raději nesklonný technický termín event handler, resp. ve zkratce pouze jako handler, pokud nebude hrozit záměna.*
- *Pozor, nezaměňujme*
 - **handler** - ovladač realizovaný příkazy programu, zpravidla kódem metody
 - **handle** - číselný identifikátor přidělený OS vytvořenému prvku

C# aplikace ve Windows 1/5



C# aplikace ve Windows 2/5

Další vlákna

```
[STAThread] static void Main()  
Application.Run(new Form1());
```

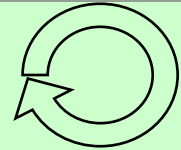
inicializace

Application.Run()

GetMessage(...);

zpracování zprávy

DispatchMessage(...);



class Form1 : System.Windows.Forms.Form

Form.WndProc(...)

CALLBACK funkce zpráv okna 1

výběr metody event handler

žádný není
definovaný

EventHandler-Menu()
EventHandler-Menu(...);

výkonný kód

return;

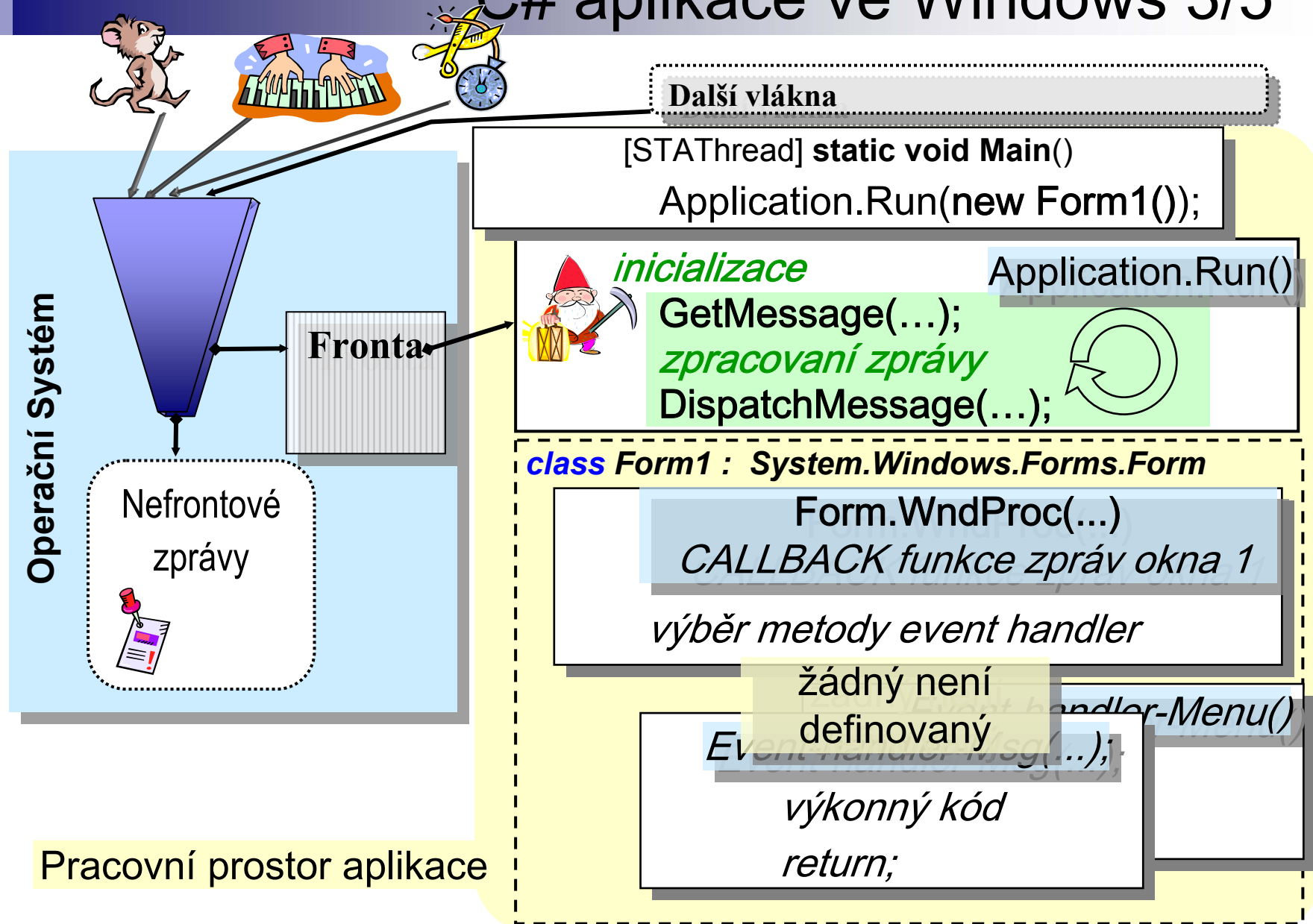
Operační Systém

Fronta

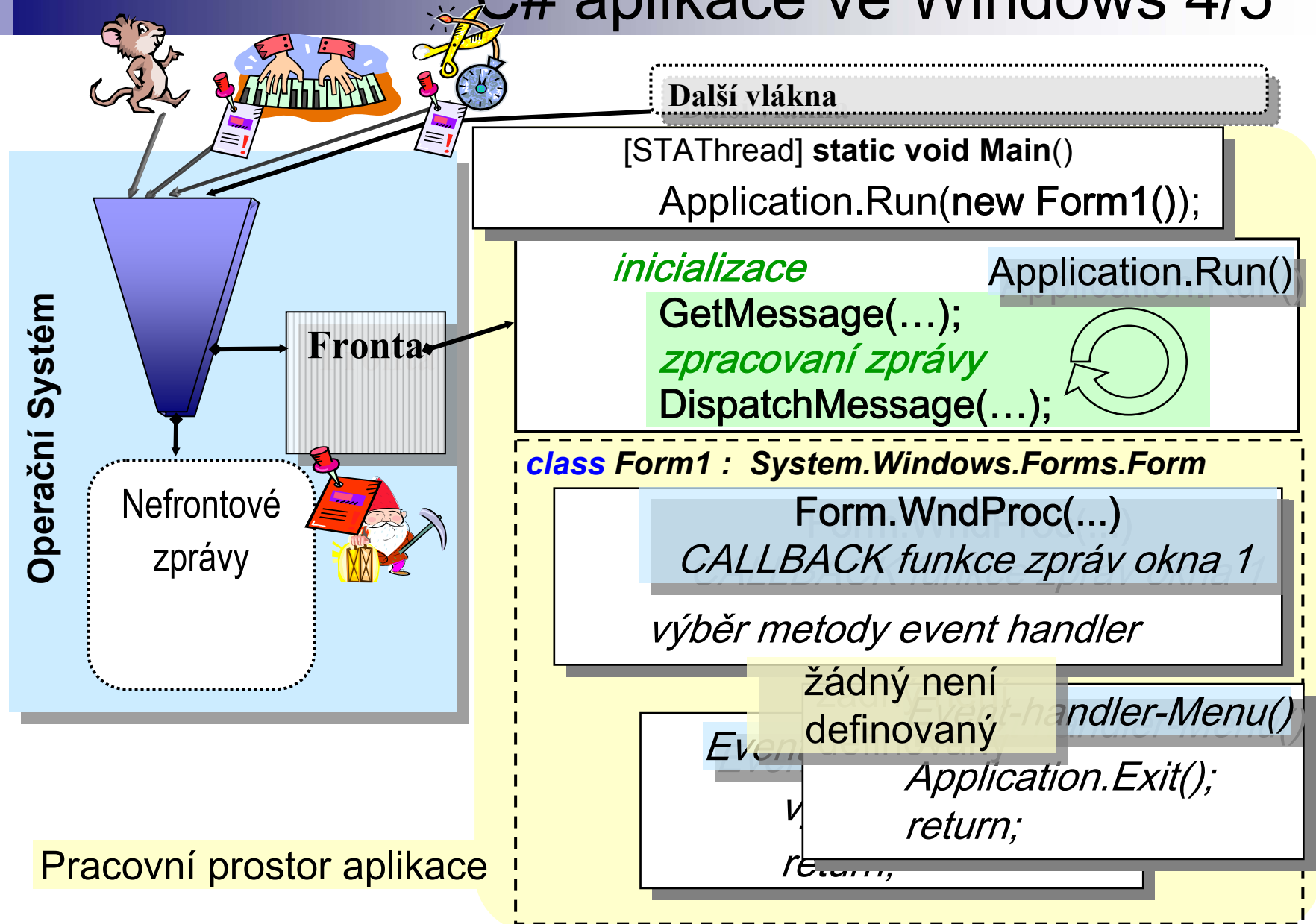
Nefrontové
zprávy

Pracovní prostor aplikace

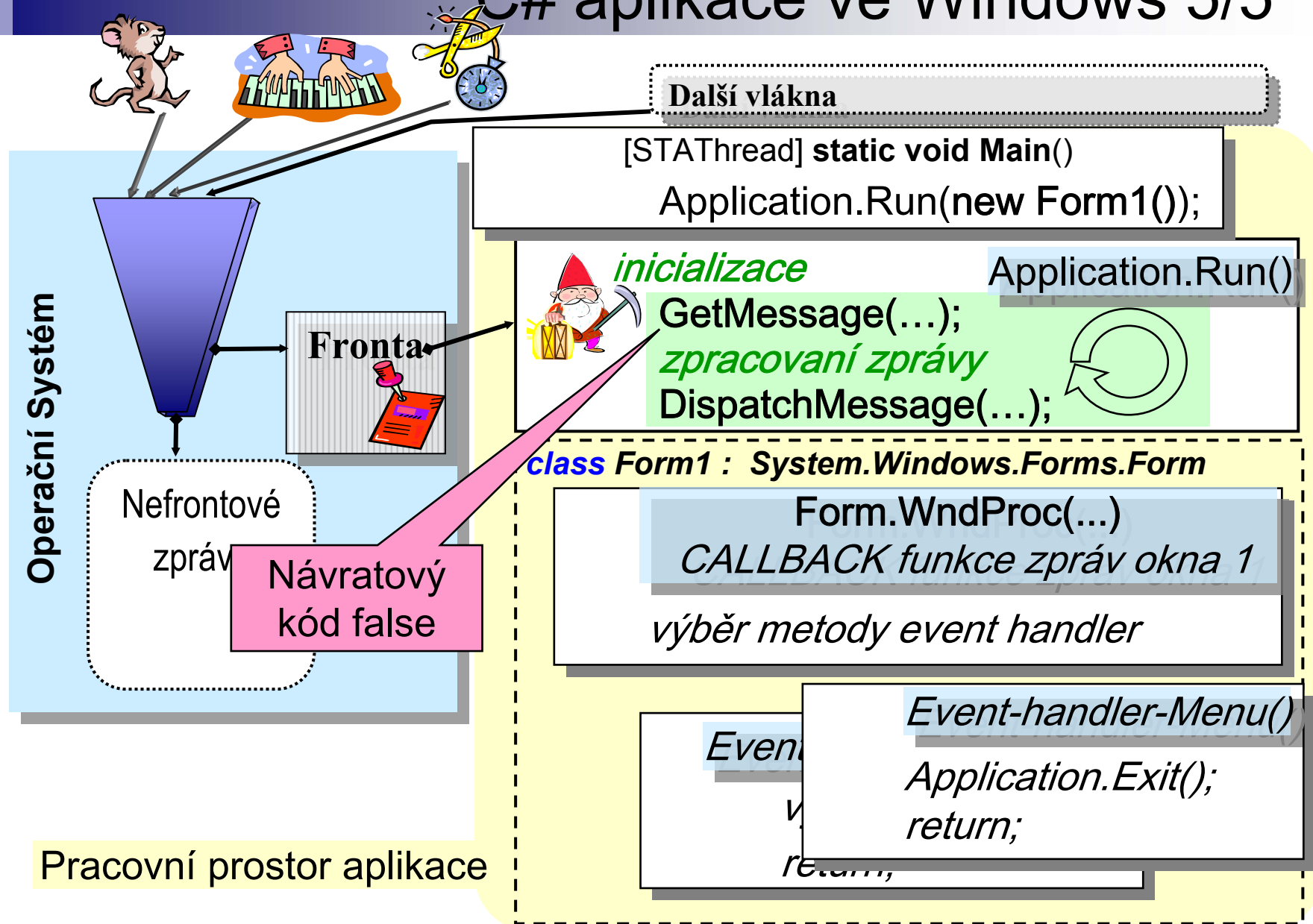
C# aplikace ve Windows 3/5



C# aplikace ve Windows 4/5



C# aplikace ve Windows 5/5



System namespace

- **Application**
.Exit(), .ExecutablePath
- **Clipboard**
.GetDataObject() .SetDataObject()
- **Cursor**
.Current
- **Cursors**
.Arrow .Wait
- **Screen**
.GetWorkingArea()...
- **Environment**
.CommandLine .OperatingSystem .TickCount

Shrnutí pro samostudium

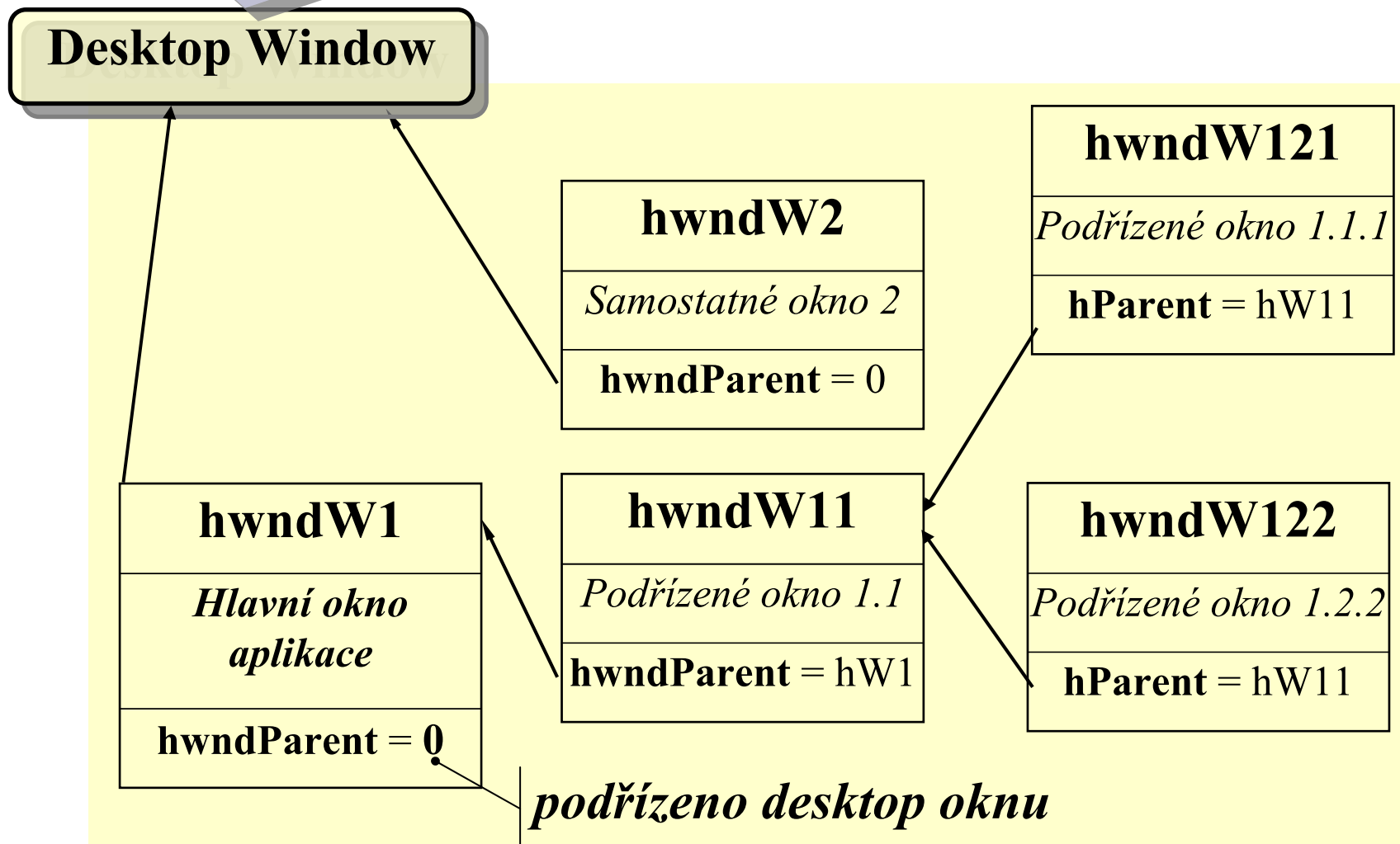
- **Architektura OS Windows se opírá o strukturu zvanou server-klient.**
V roli serveru vystupuje OS, zatímco jeho klienty jsou běžící procesy a jejich vlákna. Komunikace probíhá oběma směry. OS Windows vykonává operace zadané klienty přes služby Win API, dále zprostředkovává distribuci zpráv mezi klienty a některými jimi vytvořenými prvky. Klientům zasílá zprávy hlásící výskyt události.
- **OS předává většinu zpráv voláním zaregistrované callback funkce. Pouze frontové zprávy (*queued messages*) se řadí do fronty (*paměti typu FIFO*), v níž čekají na zpracování podle pořadí jejich příchodu. Proces si musí sám číst frontové zprávy pomocí smyčky zpráv.**
 - ta mu dovoluje i **vytřídit zprávy vyžadující speciální zpracování**, např. tzv. „Hot/Shortcut keys“ = funkční klávesy.
 - K frontovým zprávám se řadí všechny vstupní zprávy (např. od myši, klávesnice), dále zpráva WM_PAINT (požadavek na překreslení okna), WM_QUIT (ukončení aplikace), WM_TIMER (hlášení o uplynutí intervalu časovače) a některé další.*
- *Pozn. TIMER má i nefrontovou verzi, tj. se zprávou přes callback funkci*

a jejich objektů

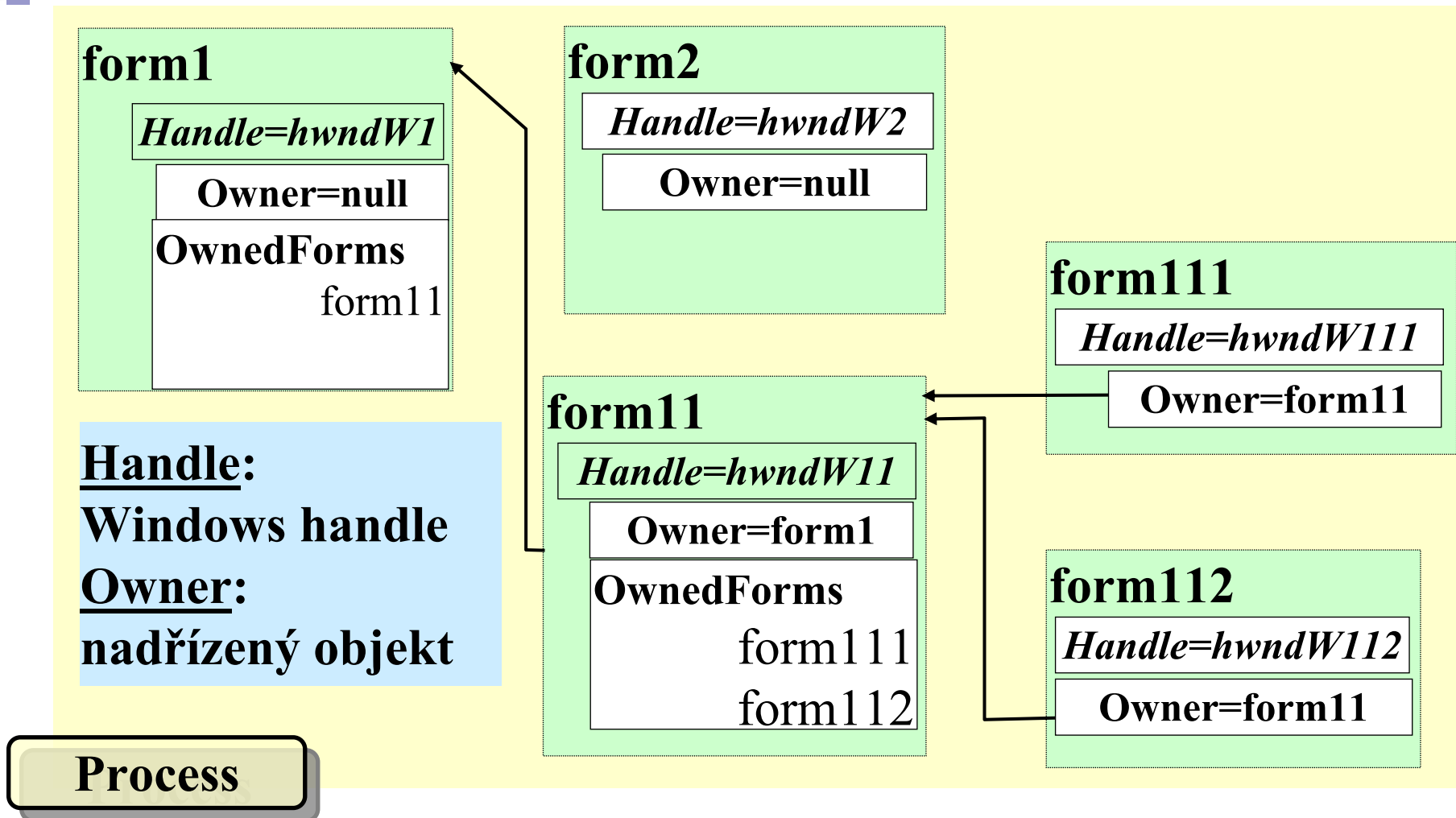


Všechna okna jsou automaticky
podřízena desktop oknu

Hierarchie oken

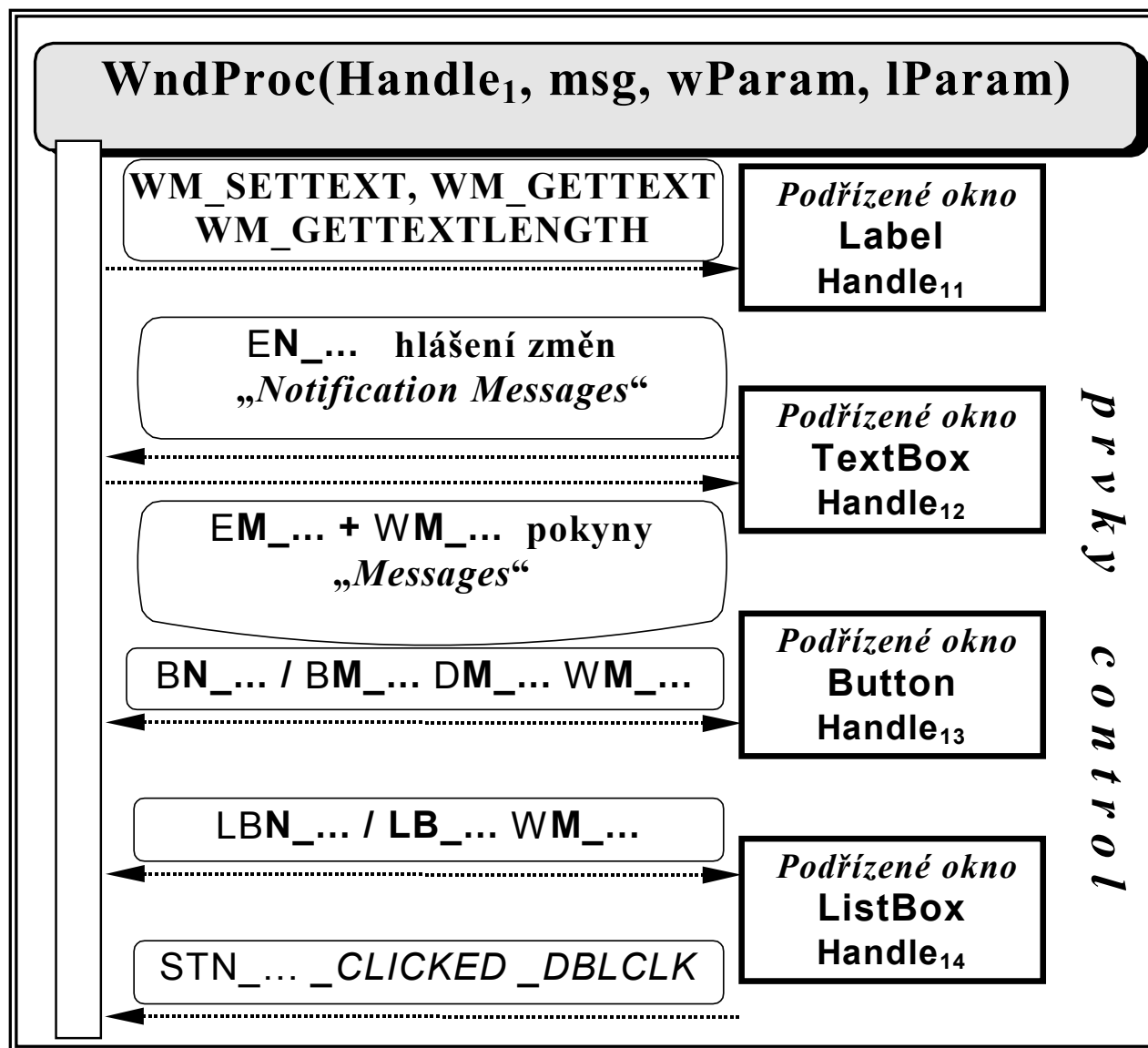


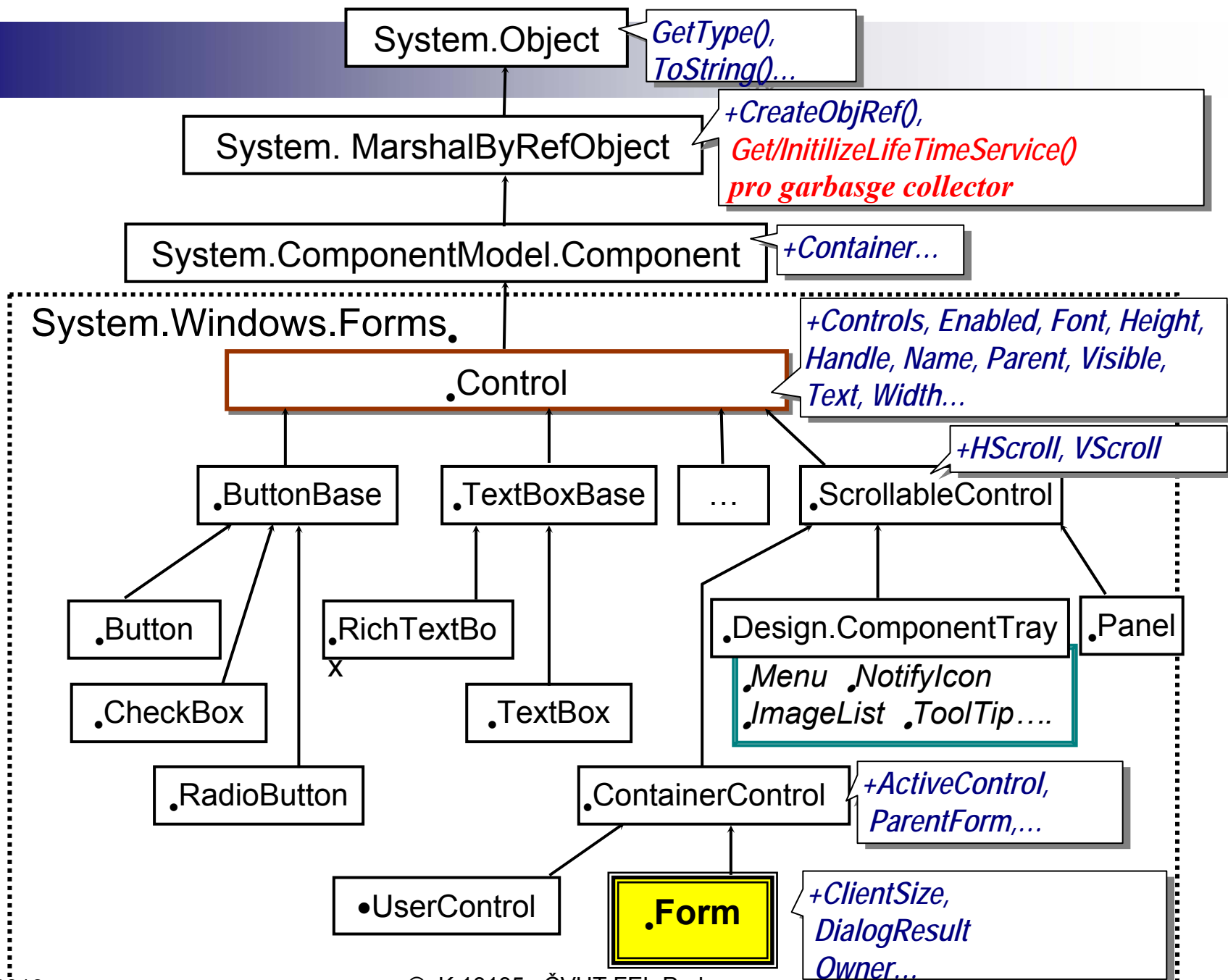
Odpovídající objekty



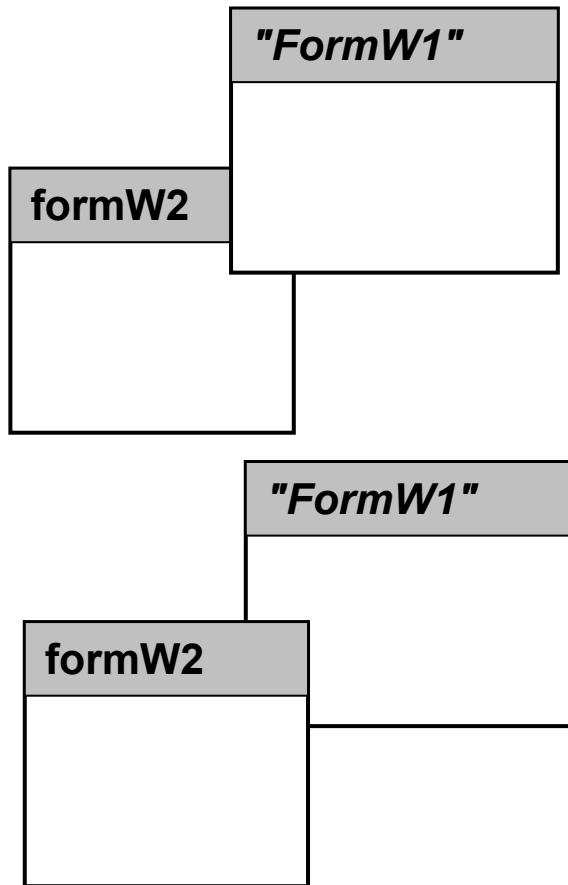
Okno má dvě datové část – v programu a v OS

Okna mají mnoho oken

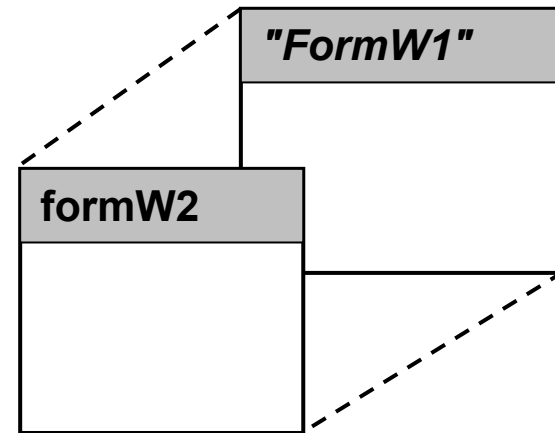




SDI – Single Doc. Interface



MDI – Multi Document Interface



formW2.Owner = this;

/ viditelnost FormW2 odvozena od
vlastníka, avšak před uzavřením
FormW1, se musí zavřít FormW2
/

Hlavní vlastnosti okna

Zprávy okna

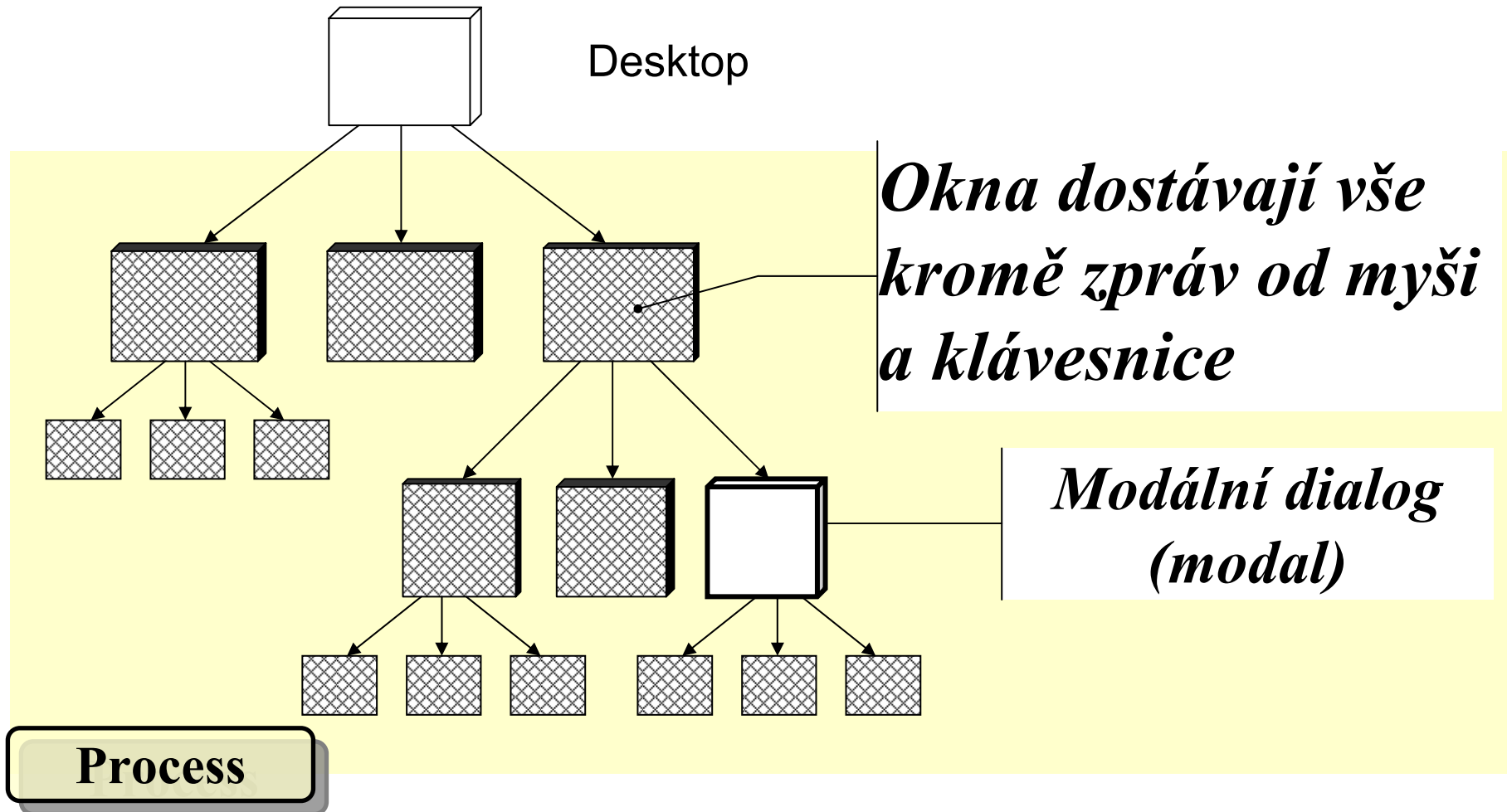
class Form operace

ENABLE	enable	<i>přijímá všechny</i>	Enabled=true;
	disable	<i>NE od myši a klávesnice</i>	Enabled=false;
FOCUS	true	<i>ANO klávesnice</i>	Focus();
	false	<i>NE klávesnice</i>	if(Focused())..
SHOW	hidden	<i>neviditelné</i>	Hide(); Visible=false;
	show	<i>viditelné</i>	Show(); Visible=true;
	minimize	<i>zobrazené na liště</i>	WindowState=Minimized;
	maximize	<i>celá plocha</i>	WindowState=Maximized;
			WindowState=wsNormal;

- **TopMost** - *vždy navrchu*
= true/false;
- **Opacity** - *nastavení neprůhlednosti okna*
= 0 ... 100% *průhledné...neprůhledné*
- **MinimizeBox**
= true/false;
- **MaximizeBox**
= true/false;
- **ShowInTaskbar** - *zobrazovat ikonu na liště*
= true/false

- Modalitou zajistíme, že uživatel může pracovat jenom s jedním oknem, což zjednoduší program.
- Modalita se provádí filtrováním zpráv, zpravidla ve smyčce zpráv.
 - ☐ Modální okno dostává všechny zprávy.
 - ☐ Dokud je modální okno otevřeno, ostatním oknům se neposílají zprávy od myši a klávesnice, avšak **jiné zprávy dál dostávají**.
- Všimněte si, že modalita nemění vlastnost Enable oken - provádí se jen filtrováním zpráv

Modal Dialog Box - ShowDialog()



Zobrazení modálního dialogu

- *vyzkoušíte si na cvičení*



```
private void oAplikaciToolStripMenuItem_Click(  
    object sender, EventArgs e)
```

```
{ // vytvoříme
```

```
    FormAbout formAbout = new FormAbout();
```

```
    formAbout.ShowDialog(); // zobrazíme
```

```
    formAbout.Dispose();
```

```
    /* Zde je Dispose() doporučeno pro okamžité  
    uvolnění systémových zdrojů, tj. dat v OS. */
```

```
}
```

```
if(formAbout.ShowDialog()  
    == DialogResult.OK)  
    { /* zavřen tlačítkem OK */  
    }  
else  
    { /* zavřený jinak */  
    }  
formAbout.Dispose();  
    // v obou případech zrušíme
```

Ukázka modálních oken



1. Otevření
2. Vliv Dispose()
3. Dead-lock

// předání parametru do okna

```
public partial class FormMsg : Form
{
    Form1 form1;

    public FormMsg(Form1 form1)
    {
        this.form1 = form1;
        InitializeComponent();
    }

    /* */ form1.Info = "Modální okno otevřeno"; /*...*/
}

public partial class Form1 : Form
{
    /*...*/

    public string Info { set { statusLabel.Text = value; } }
}
```

Prvky z Form1, které potřebuje okno ovládat, zpřístupníme pomocí definic public prvků

1. Předání odkazu na objekt s public členy, viz předchozí snímek
2. Některý objekt obsahuje statický odkaz sám na sebe či na jiný objekt s public členy, viz Singleton z minulé přednášky
3. Lze vytvořit delegate a ten předat

Otázka k zamyšlení:

Řekněte, jaké má každá metoda výhody a nevýhody...

- `FormMsg formMsg = new FormMsg(...);`
`formMsg.ShowDialog();`
`formMsg.ShowDialog();`
- `FormMsg formMsg = new FormMsg(...);`
`formMsg.ShowDialog();`
`formMsg.Dispose();`
`formMsg.ShowDialog(); // výjimka`

■ **Dead-lock**

```
FormX form = new FormX(...);
```

```
this.TopMost = true;
```

```
form.ShowDialog();
```

```
this.TopMost = false;
```

■ **Dead-lock ++** - pokud bude hlavní okno maximalizované, pak konec aplikace jedině přes Ctrl-Alt-Del

```
FormX form = new FormX(...);
```

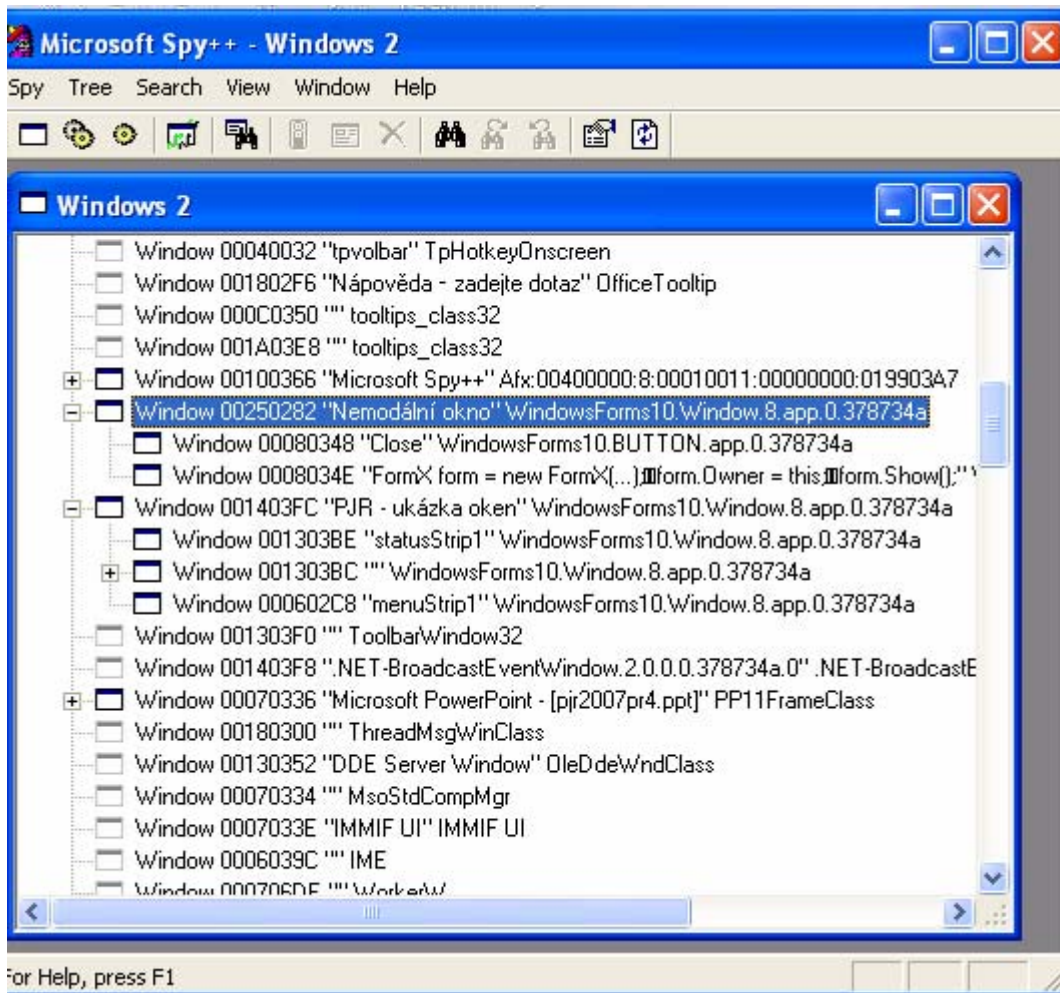
```
form.ShowInTaskbar = false;
```

```
this.ShowInTaskbar = false;
```

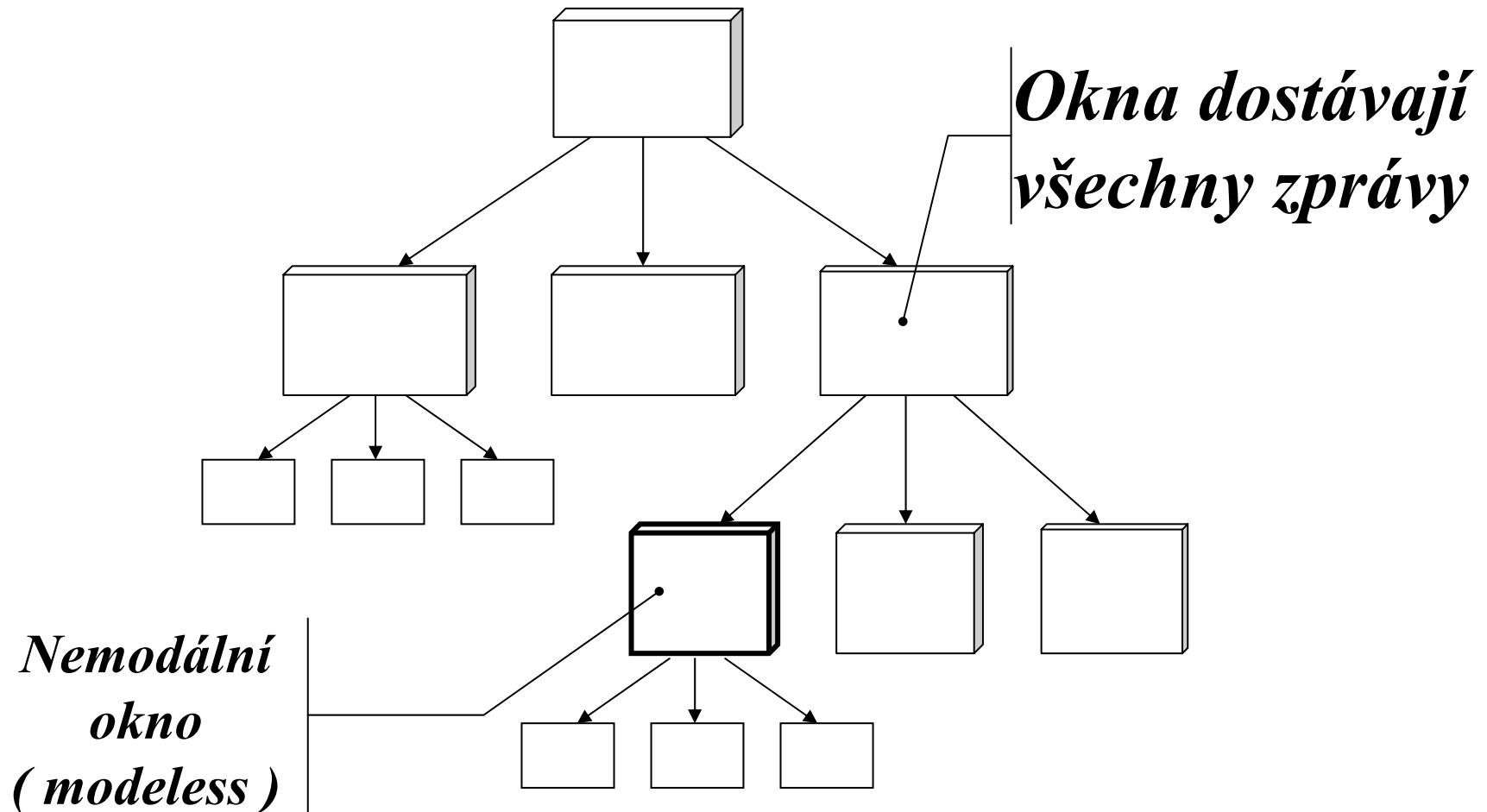
```
this.TopMost = true;
```

```
form.ShowDialog();
```

Window Spy



Modeless Dialog Box - Show()



Zobrazení nemodálního okna

```
public partial class Form1 : Form
{ /*...*/
    private FormNemodalni frm;
    public Metoda()
    {
        frm = new FormNemodalni();
        frm.Owner = this; frm.TopMost = true;
        frm.Show();
    }
}
```

// dostup na vlastněné formy

Form [] formlist = this.OwnedForms; *// pole forem*

Form form = null; *// objekt typu Form*

if(formlist.Length > 0) form = formlist[0]; *// člen [0]*

```
/*.... */ }
```

Ukázka nemodálních oken




- Vlastník okna
- MDI okna
- Automatické dispose

- *Rušení nemodálních oken není tak jednoduché - programování modálních oken je mnohem snazší.*
- *Nemodální okno lze sice zavřít pomocí `Close()`, ale jak zabránit jeho duplicitnímu vytvoření?*

Nemodální okno 1/2

// vytvoříme proměnnou v deklaračním prostoru třídy

FormNemodalni1 formN1 = null;

private void buttonOkno1_Click(object sender,
 EventArgs e)

{ if (formN1 == null)

formN1 = new FormNemodalni1();

/ anonymní metoda pro vynulování proměnné po
zavření okna */*

formN1.FormClosed += delegate { formN1 = null; };

// ----- Pokud je schované, zviditelníme

```
if (!formN1.Visible) formN1.Show();
```

// Ukážeme minimalizované okno v normálním stavu

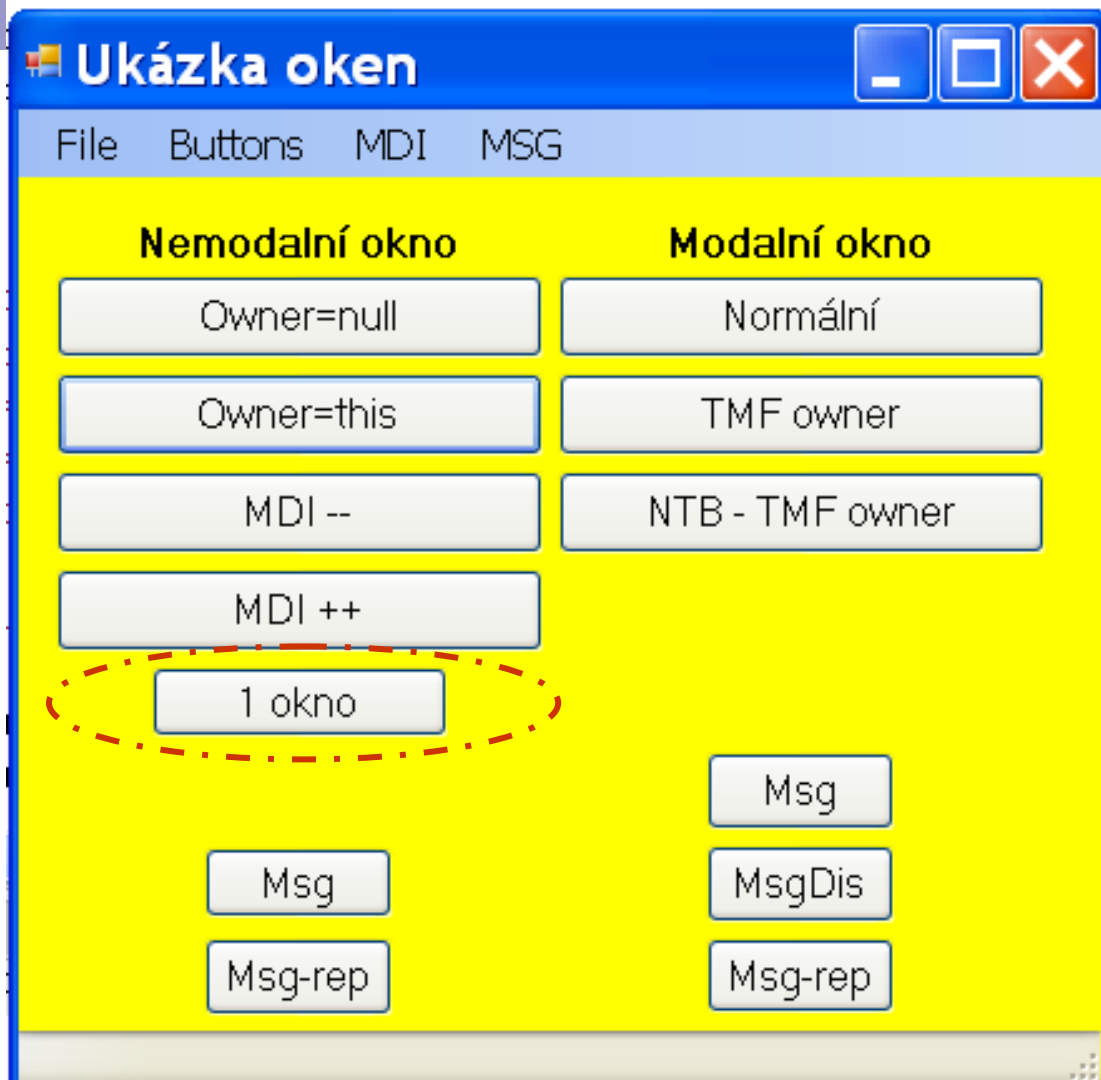
```
if (formN1.WindowState == FormWindowState.Minimized)  
    formN1.WindowState = FormWindowState.Normal;
```

// přeneseme dopředu v Z-indexu

```
formN1.BringToFront();
```

```
}
```

Ukázka nemodálních oken



- Anonymní metoda

Trvalá existence okna

- Každé kliknutí na Help->Nápověda znovu vytváří okno a pokud vytvoření trvá dlouho, zdržuje to...



Přidáme do FormHelp událost FormClosing

```
private void FormHelp_FormClosing(object sender,  
    FormClosingEventArgs e)
```

```
{
```

```
    e.Cancel = true; Hide();
```

```
}
```

// tohle teď nesmíme, jinak nezavřeme aplikaci
~~*formHelp.Owner = this;*~~

Před uzavřením okna, se totiž musí napřed uzavřít všechna jemu podřízená okna.



Posloupnosti událostí

<http://www.cartoonstock.com/>

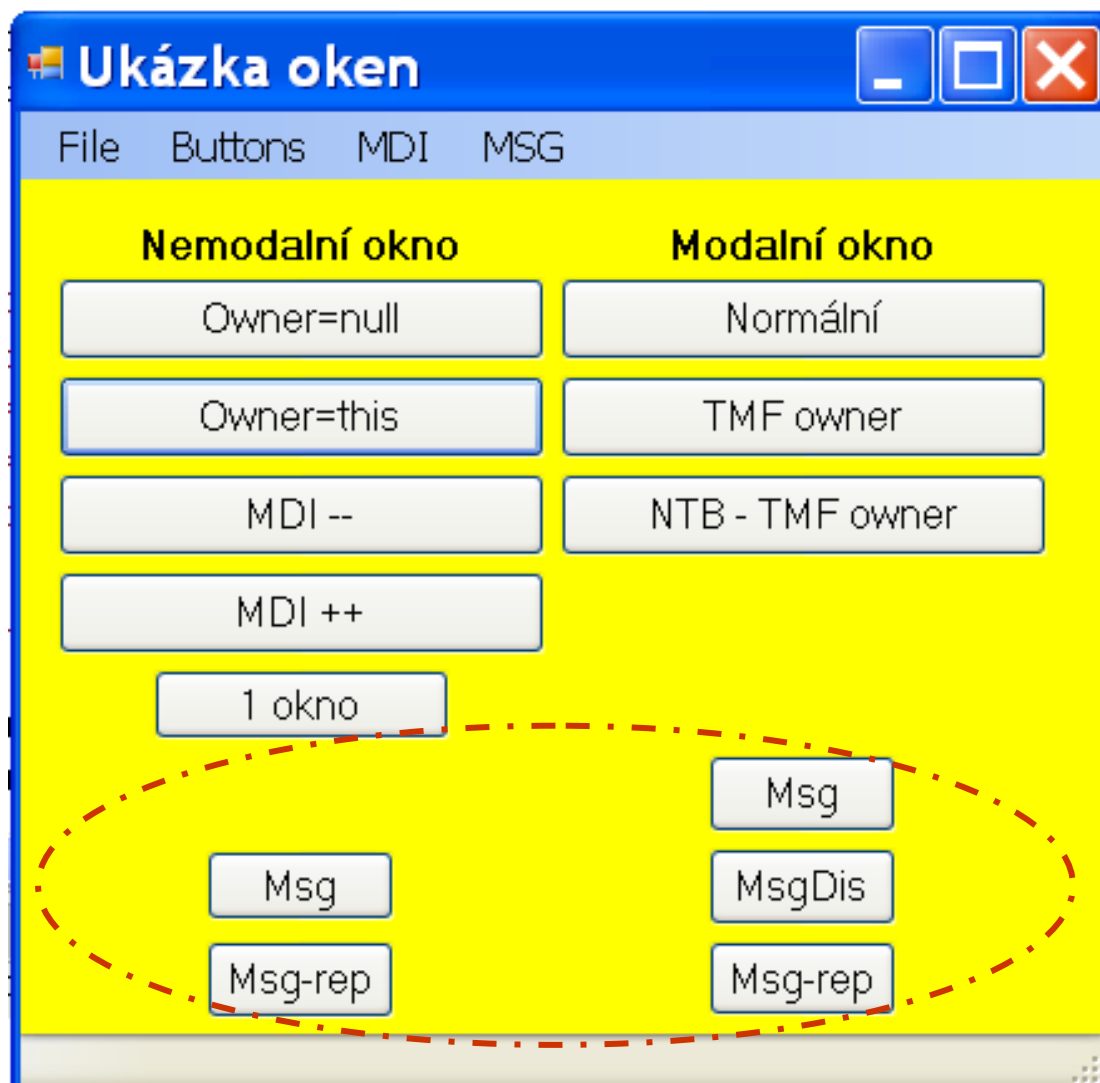
```
namespace WindowsApplication
{
    static class Program
    {
        [STAThread] static void Main()
        {
            // kreslíme se styly (tématy) vzhledu, pokud OS podporuje
            Application.EnableVisualStyles();
            // podpora pro písma se složitým zápisem, např. arabština
            Application.SetCompatibleTextRenderingDefault(false);
            // vytvoření hlavního okna a smyčka zpráv
            Application.Run(new Form1());
        }
    }
}
```

Hlavní zprávy při vytváření okna

- **Form1() { InitializeComponent(); }**
 - *metoda volaná v konstruktoru, která vytvoří okno. Během ní se provedou následující zprávy:*
 - *[Move] - změna polohy – může, ale nemusí se nutně objevit*
 - *Layout - změna ovlivňující uspořádání prvků v okně*
 - *Resize - změna velikosti okna na obrazovce*
 - *SizeChanged - změna vlastnosti Size*
 - *Layout - Resize - SizeChanged - může se opakovat*
 - *[StyleChanged] - změna vlastnosti ControlStyles*
 - *Load - objeví se pouze před prvním zobrazením okna, teprve nyní je známa velikost a poloha okna*
 - *VisibleChanged - změna hodnoty Visible*
 - *Activated - získán Focus a přenesení dopředu*
 - *Paint - vykreslení obsahu*

Další události přicházejí v závislosti na práci s oknem

- V **konstruktoru** objektu neexistuje ještě grafická podoba okna ve Windows
 - v konstruktoru lze pracovat jen s datovými členy třídy
- Událost **Load** se volá **pouze jedenkrát** - po vzniku okna ve Windows, když okna není ještě zobrazené, odpovídá zprávě Windows s kódem WM_CREATE
 - v Load můžeme pracovat se všemi prvky.
- Událost **Activated** se volá, když se okno stane aktivním. Pozor, tato událost se může opakovat.
- Událost **Paint** je pokyn k vykreslení grafických prvků – tomu věnujeme příští přednášku



Otevření Úvodní obrazovky (Splash Screen)

Konstruktor

```
public partial class Form1 : Form
```

```
{ private FormSplash formSplash;
```

```
public Form1()
```

```
{formSplash = new FormSplash();
```

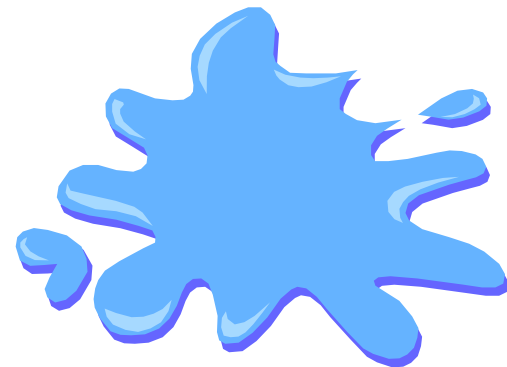
```
formSplash.TopMost = true;
```

```
formSplash.Show();
```

```
InitializeComponent();
```

```
}
```

```
/* ... */ }
```



Zavření Úvodní obrazovky přes Activated

Událost Activated

```
private void Form1_Activated(object sender, EventArgs e)
{
    if (formSplash != null)
    {
        formSplash.Close();
        formSplash.Dispose();
        formSplash = null;
    }
}
```

- **Přepnutí Focus na jiné okno bez zakrytí původního okna**
Deactivate - *ztracen Focus*
Activated - *obnoven Focus*
- **Okno zakryto jiným oknem a zas odkryto**
Deactivate - Activated - Paint
- **Minimalizované a maximalizované okno**
Move - Layout - Resize - SizeChanged - Deactivate
Activated - Move - Layout - Resize - SizeChanged - Paint

■ Přesun okna na ploše

Move - Move - Move - ... - Move - Deactivate - Activated

Při přesunu se nevolá Paint() - operační systém si sám kopíruje grafickou informaci.

■ Změna velikosti

- Layout - Resize - SizeChanged - Layout - Resize
- SizeChanged - Layout - Resize - SizeChanged
- Deactivate - Activated -

Nevolá se Paint() – nutno volat z programu, je-li třeba překreslit okno

■ Psaní do editačního pole

- *okno nedostává žádné zprávy, posílají se prvku TextBox*

■ Zavření okna

- **FormClosing** event
 - dotaz na zavření
- **FormClosed** event
 - objeví se **až po uzavření** okna
- **void Dispose()** metoda
 - volá ji "garbage collector" přes *Finalize()* jako metodu pro uvolnění systémových zdrojů - *více bude později na přenáškách.*

Dotaz před uzavřením okna

```
private void Form1_Closing(object sender,  
    System.ComponentModel.CancelEventArgs e)  
{ if( MessageBox.Show("Ukončit aplikaci?",  
    "Konec programu",  
    MessageBoxButtons.YesNo,  
    MessageBoxIcon.Question  
        ) == DialogResult.No  
    )  
    e.Cancel = true; // zabráníme zavření okna  
}
```


Událost FormClose

- *Při jejím přijetí oknu ve Windows už neexistuje a nelze ho už zobrazit Show() nebo ShowModal().*

Zavírání podřízených oken

- *Všem vlastněným formám se pošle událost FormClosing dříve než nadřazenému oknu.*
- *Pokud některá z podřízených forem odmítne uzavření, zavření se neprovede a událost FormClosing se nadřazenému oknu nepošle.*

Příště bude přednáška o kreslení ve Windows



[Rob Gonsalves]