



Vývoj aplikací v prostředí .NET

© Katedra řídicí techniky,
ČVUT-FEL Praha

4. přednáška



Pro samostudium

Takto označené snímky

- slouží jako další rozšíření přednášky;
- nebudeme se u nich zastavovat
- ale nebudou se ani zkoušet.



Základy kreslení

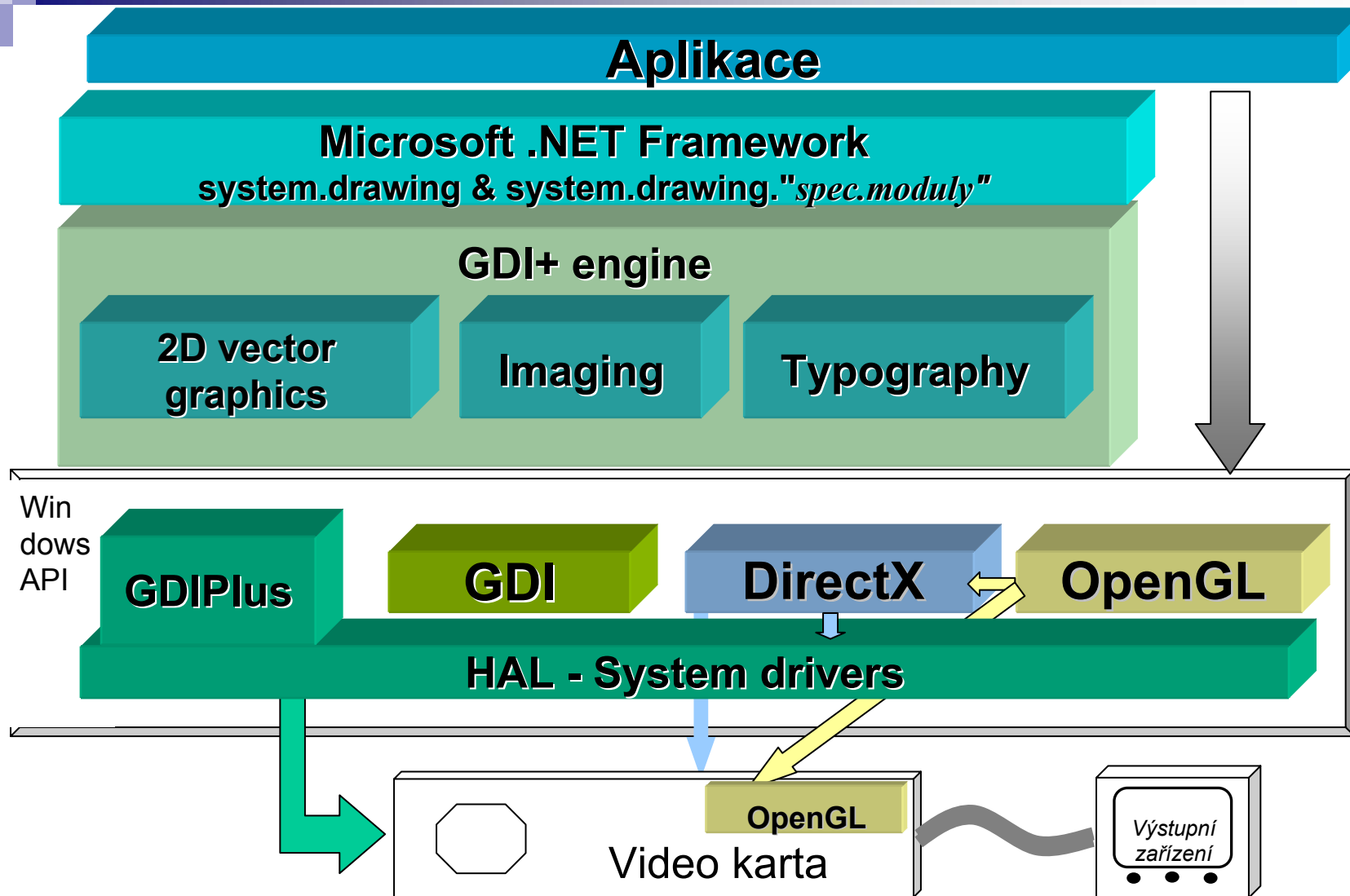
v teorii a praxi



■ Použité zkratky

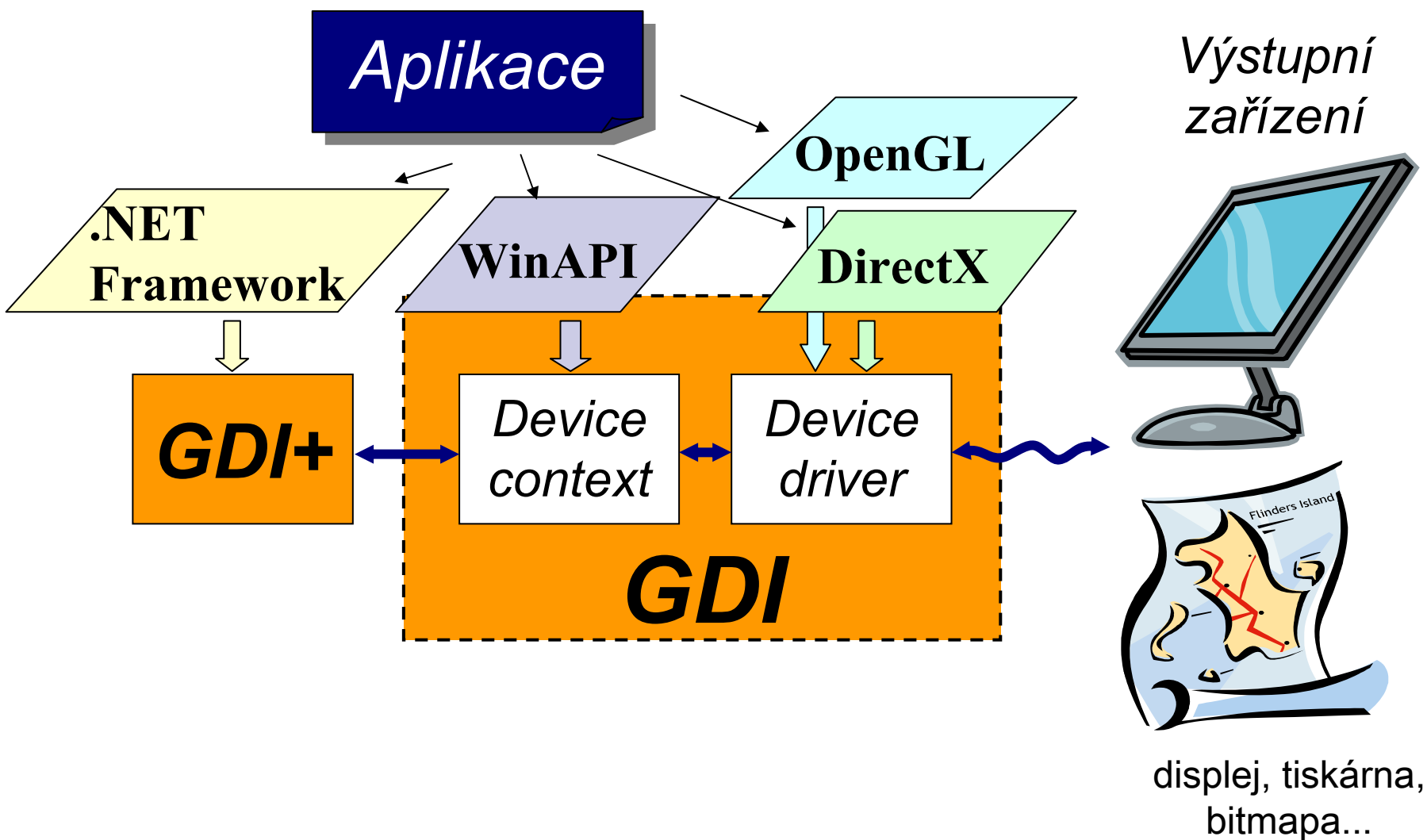
- **GDI** - *Graphics Driver Interface*
- **Open GL** (*Open Graphics Language*) - SGI Silicon Graphics
- **DirectX, Direct 3D**
- *Microsoft Direct X Technology*
- **HAL** – *Hardware Abstract (Adaptation) Layer*

Architektura GDI+



Orientační tabulka vlastností

	GDI+	GDI	DirectX	OpenGL
Použití	<i>snadné</i>	<i>složitější</i>	<i>náročné</i>	<i>obtížná tvorba komplikovanějších obrazů</i>
Rychlost	<i>malá</i>	<i>o málo lepší</i>	<i>100%</i>	<i>cca 70-100%</i>
Přenositelnost na jiné platformy	<i>ano</i>	<i>ne</i>	<i>ne</i>	<i>ano</i>
Zaměření	<i>Snadné použití</i>	<i>Zastaralé</i>	<i>Rychlost</i>	<i>Rychlost a přenositelnost</i>



Pen (*čáry*)

color, width, dash, end caps, joins,



Brush (*výplně a písmo*)

color, solid, texture, pattern, gradient



Font, String Format (*řetězce*)

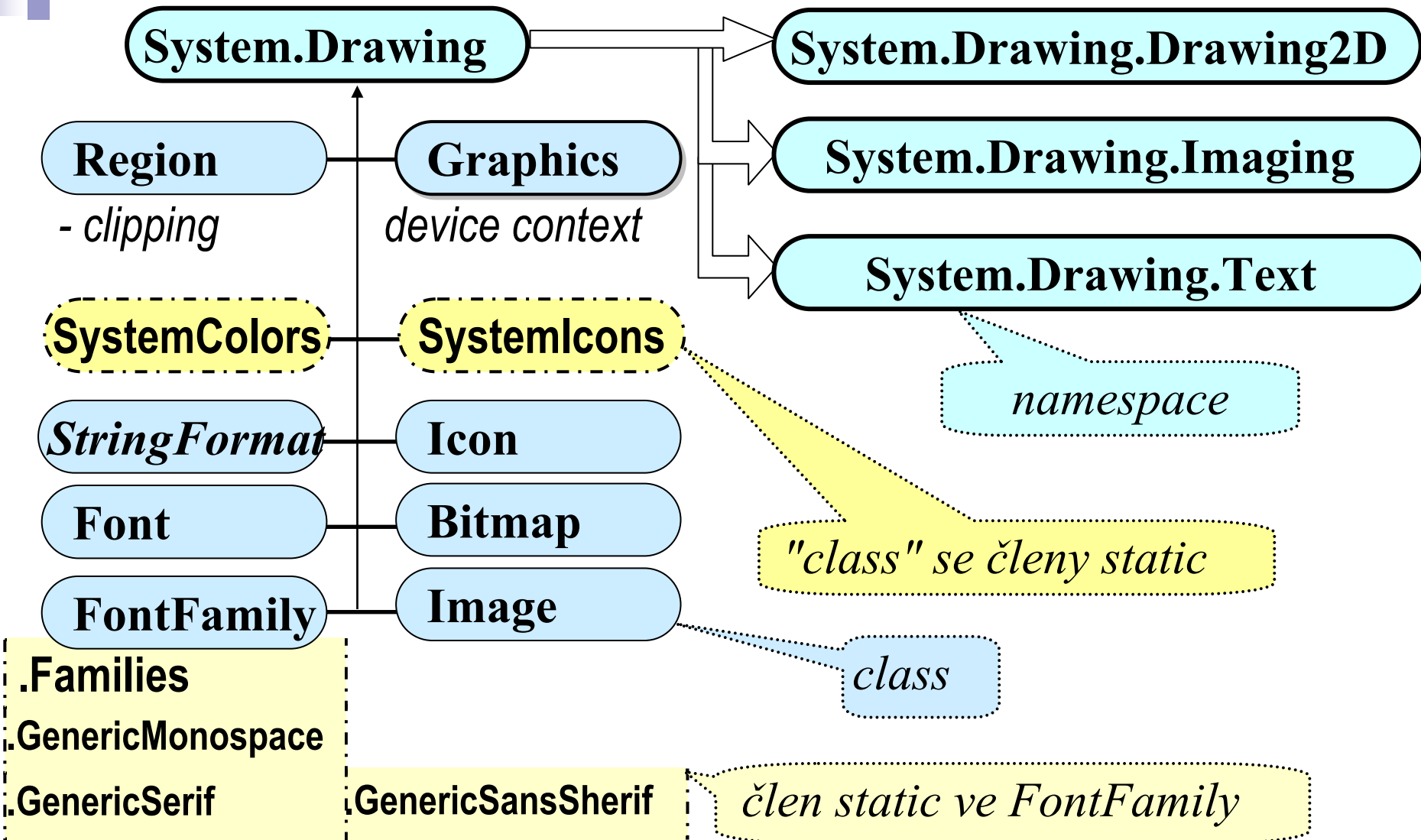
Bitmap/Metafile (*obrázky*)

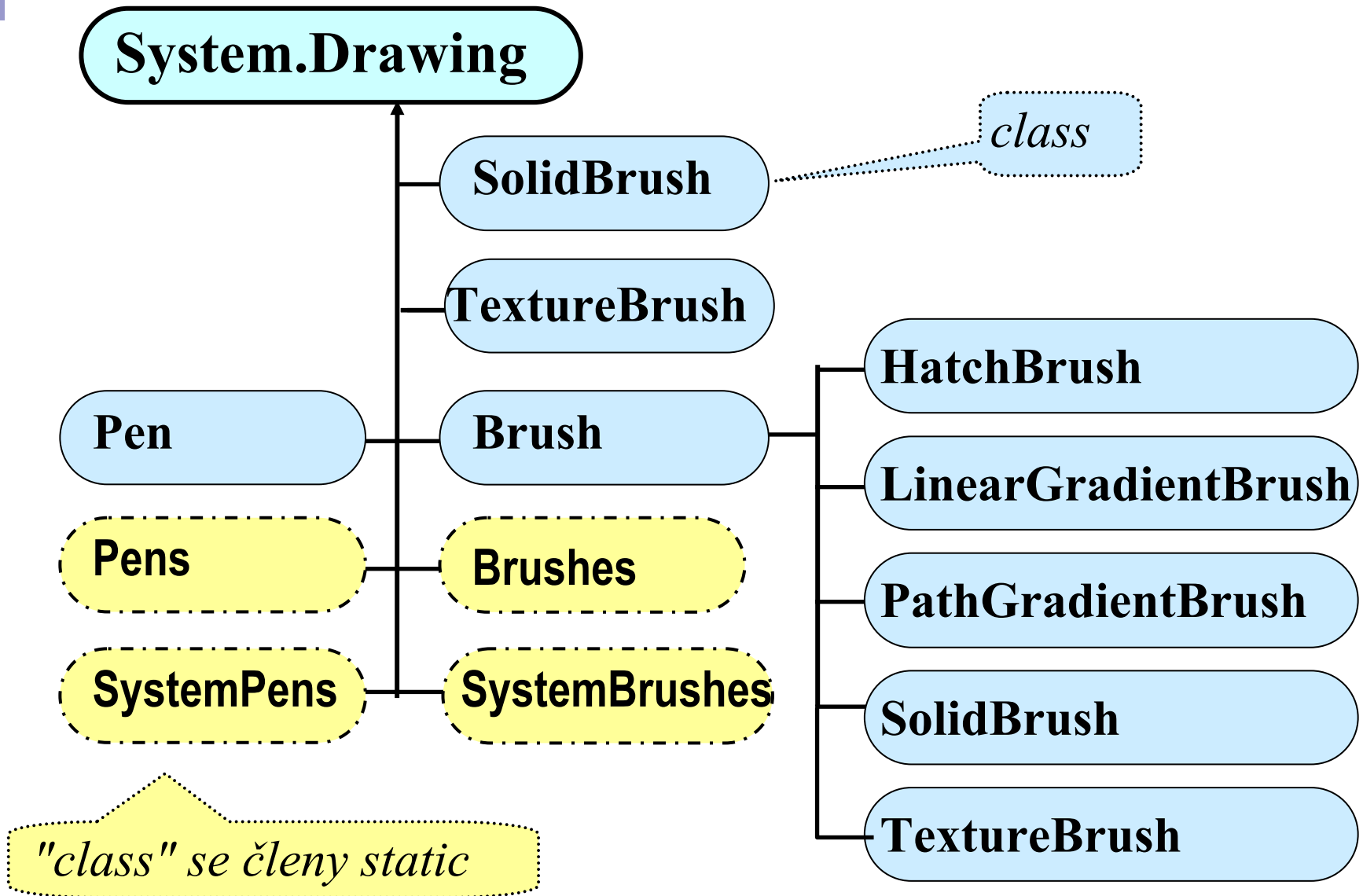
Bmp, gif, jpeg, png, tiff, wmf, ...

- vytvořený *Windows* - ty vracejí jeho **handle**
- i každý GDI objekt patřící DC má svůj **handle**
- *při vytváření se uváží:*
 - provedení ovladače zařízení;
 - fyzická velikost použitelné plochy pro výstup;
 - barevné možnosti výstupu a barevnou paletu zařízení;
 - objekty, které hardware zařízení dovoluje kreslit;
 - vestavěné možnosti limitace kreslicí plochy, tzv. clipping;
 - vestavěné možnosti kreslení křivek, polygonů a podobně;
 - vestavěné možnosti psaní textu.

- DC představuje ve své podstatě transformační program mezi příkazy GDI a vstupem grafické periférie.
- DC dovoluje aplikovat kreslicí příkazy, nezávisle na provedení koncového zařízení
- DC se vytváří nejen pro grafickou kartu, ale třeba i pro tiskárnu nebo bitmapu

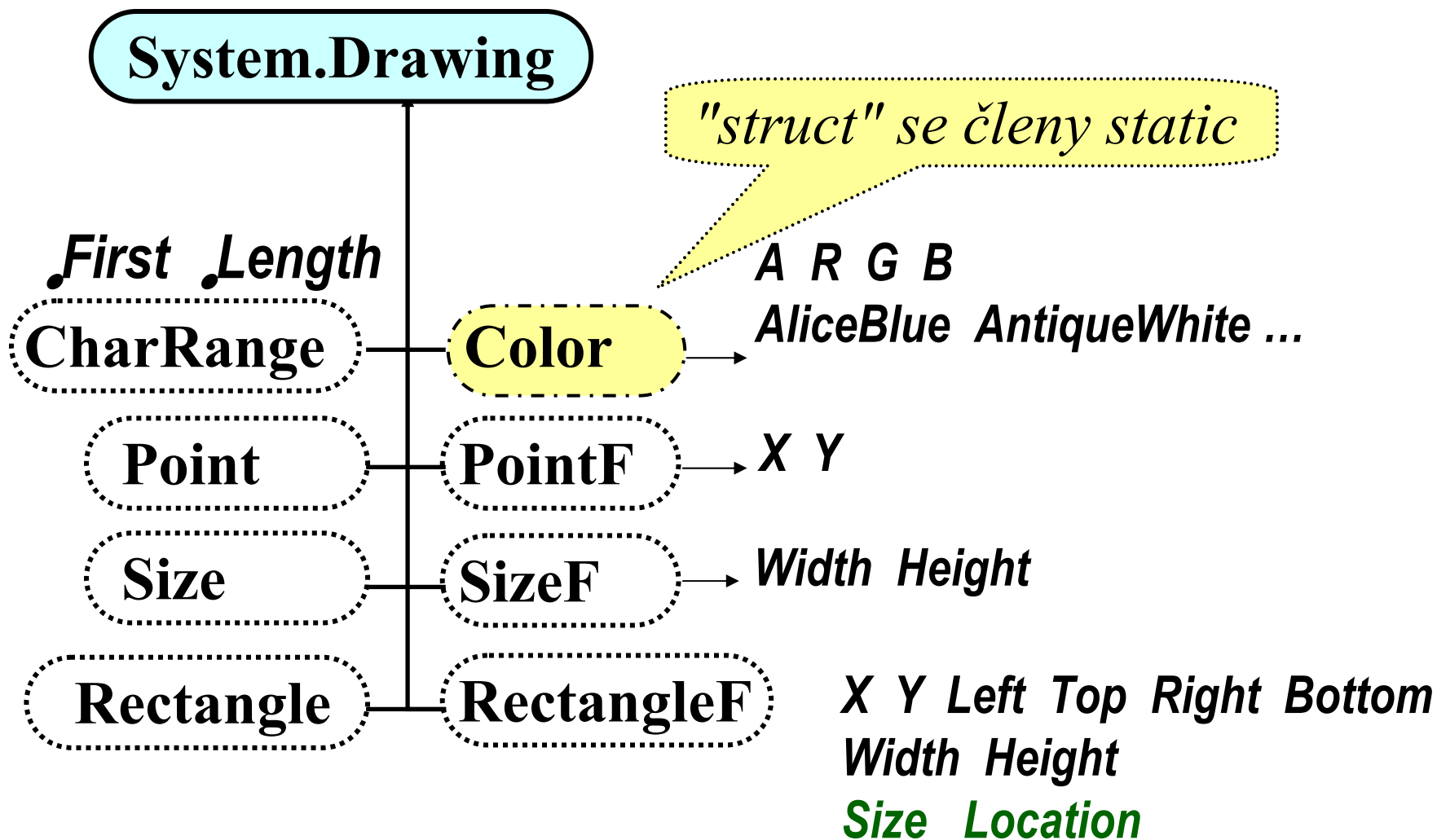
System.Drawing namespace 1/2



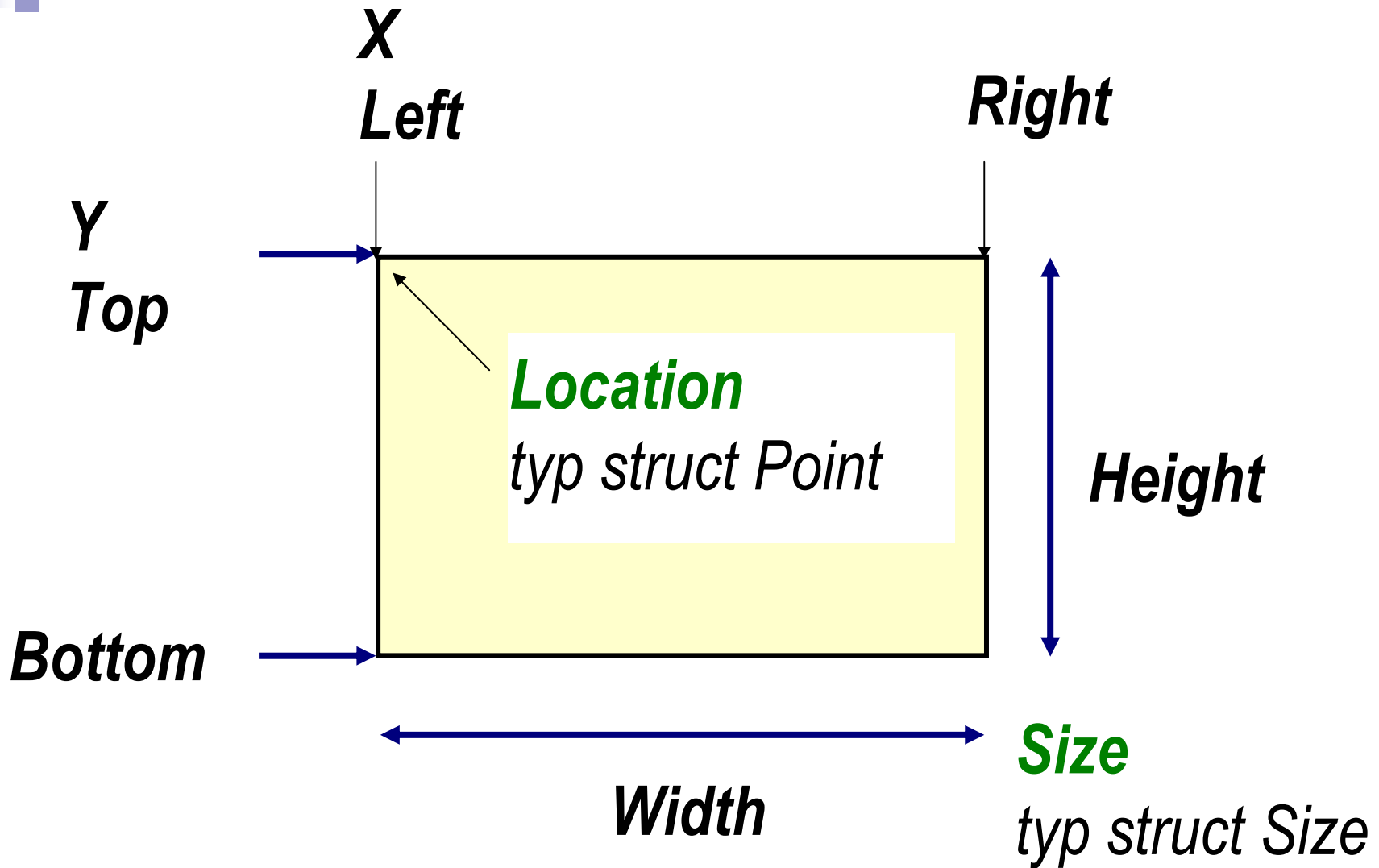


Struktury System.Drawing namespace

Pro samostudium



Pro samostudium Označení souřadnicových property



Začneme od barvy...

*realita
versus
paleta*



Statické property a metody pro barvu

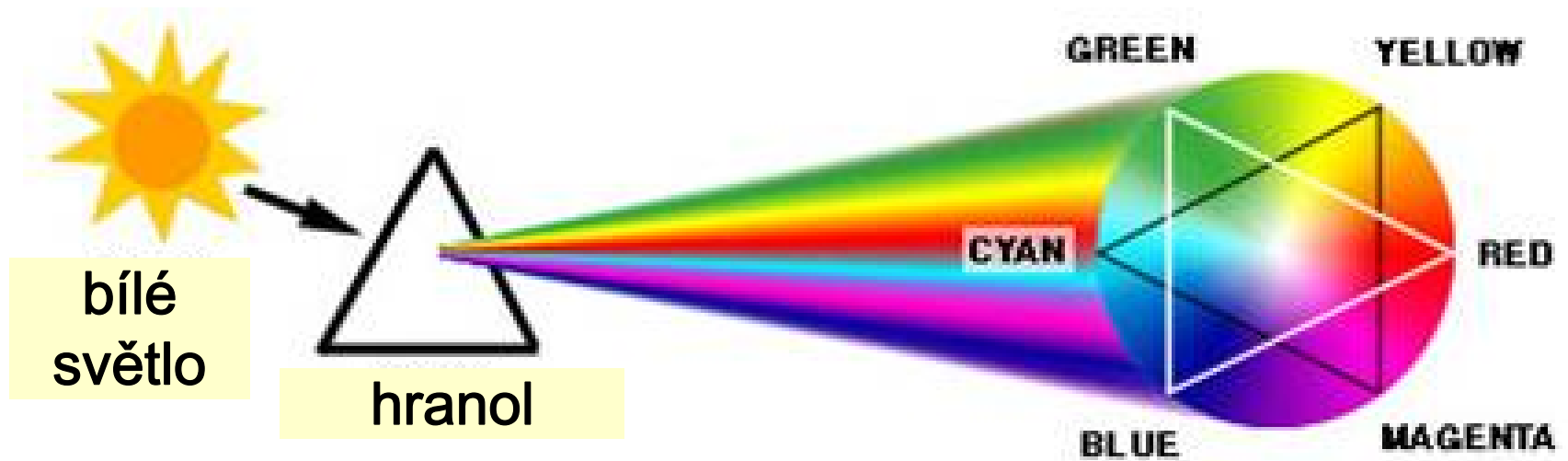
■ struct System.Drawing.Color

- .A - Alpha (neprůhlednost, opacita)
0-plně průhledné, 255-neprůhledné
- .R .G .B složka Red, Green, Blue 0..255
- .AliceBlue .AntiqueWhite ...
.Yellow .YellowGreen
- FromARGB(), FromName()

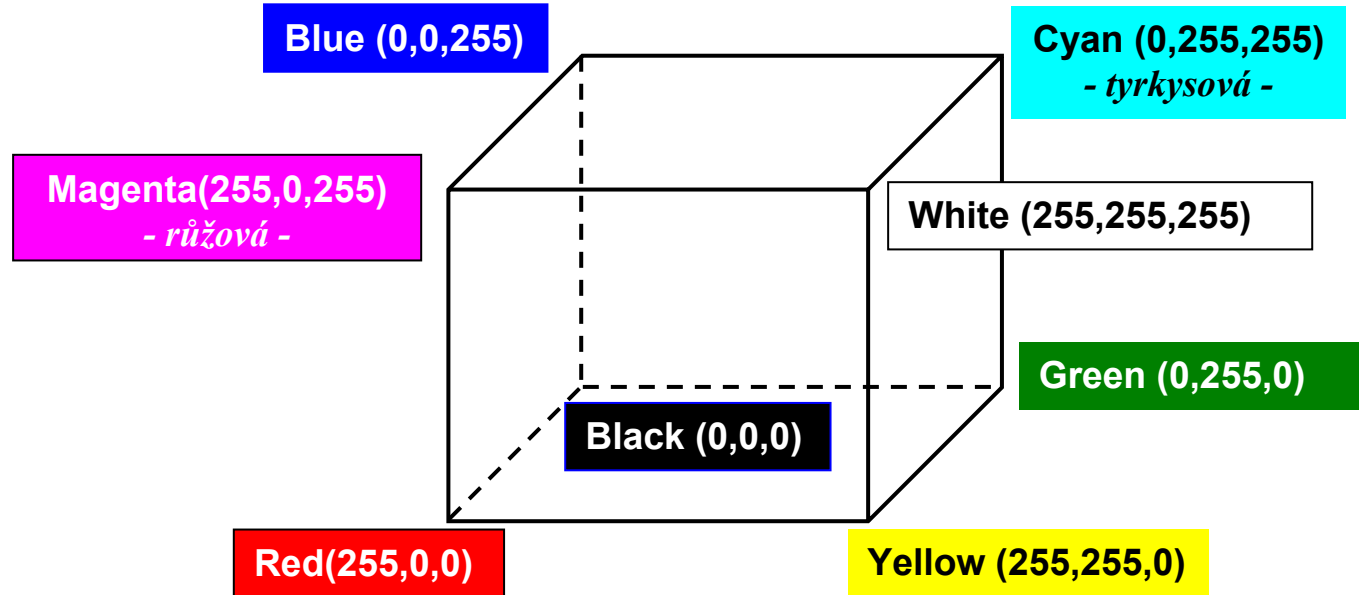
■ class System.Drawing.SystemColors

- .ActiveBorder .ActiveCaption ...
.WindowFrame .WindowText

Aditivní míšení barev



RGB barevná krychle



- *Aditivní míchání barev se používá například na monitorech, vlivy jednotlivých barev se sčítají, neboť každá složka se chová jako zdroj světla.*

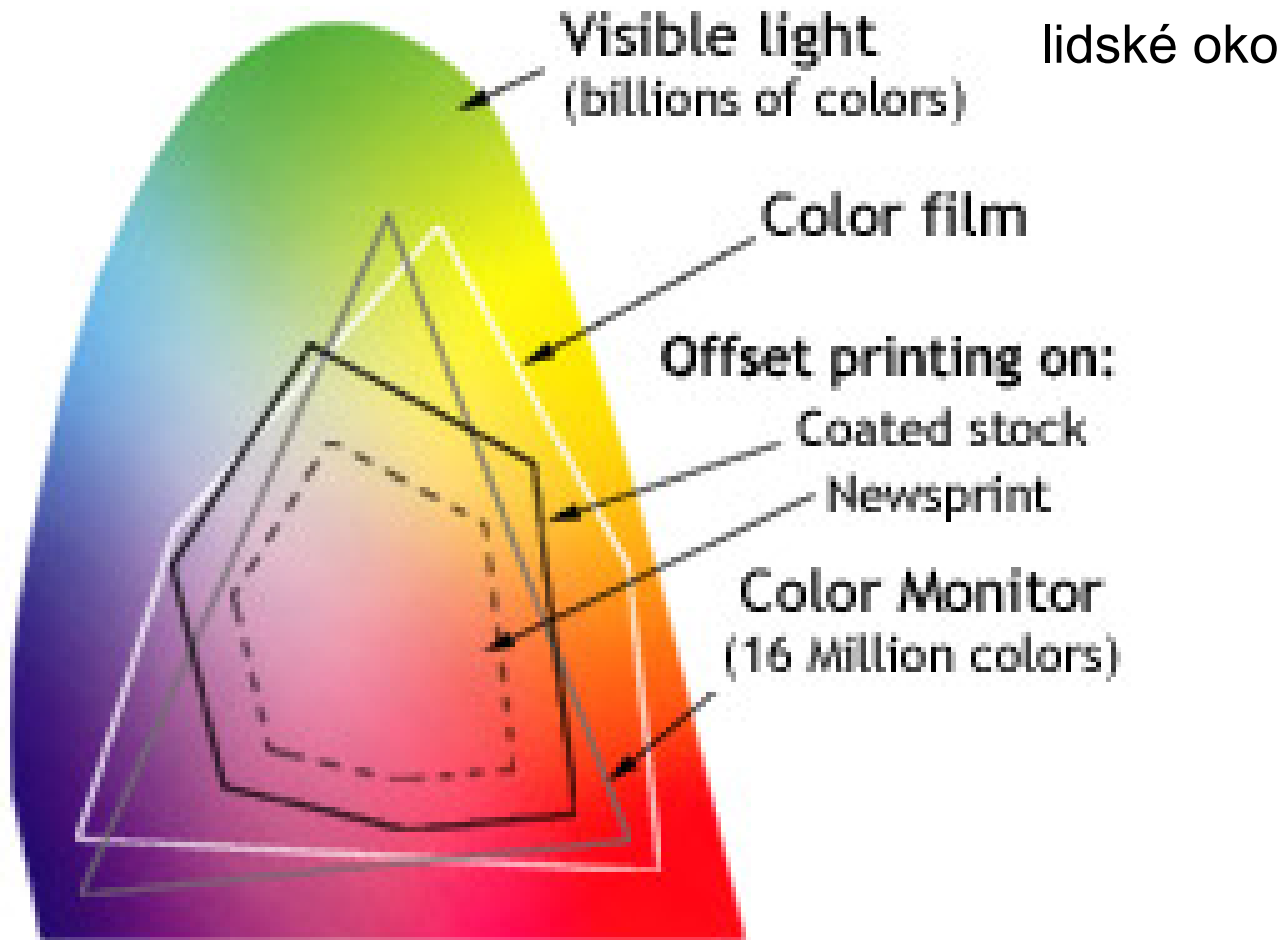
- *Subtraktivní míchání barev se projevuje při tisku - vlivy jednotlivých barvy se odečítají, jelikož **každá složka se chová jako filtr bílého světla**.*
- *CMY barevnou krychli dostaneme prohozením protilehlých vrcholů RGB krychle, tj- černá ↔ bílá, žlutá ↔ modrá, apod.*



$$\begin{array}{l}
 \text{Cyan} \\
 \text{Magenta} \\
 \text{Yellow}
 \end{array}
 \begin{bmatrix} C \\ M \\ Y \end{bmatrix} = \begin{bmatrix} 255 \\ 255 \\ 255 \end{bmatrix} - \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

- *Přidáním black dostaneme CMYK:
 $K = \min(C, M, Y); C = C - K; M = M - K; Y = Y - K;$*

Dostaneme však stejný gamut?



Zdroj: <http://www.kathleenmahoney.com/>

gamut = rozsah barev zařízení



- *CMYK má jiný gamut než RGB, tj. část RGB barev nelze vytisknout na běžných tiskárnách a některé CMYK barvy zas zobrazit na monitorech!*
- *A mnohé barevné odstíny spatříme jen v přírodě.*

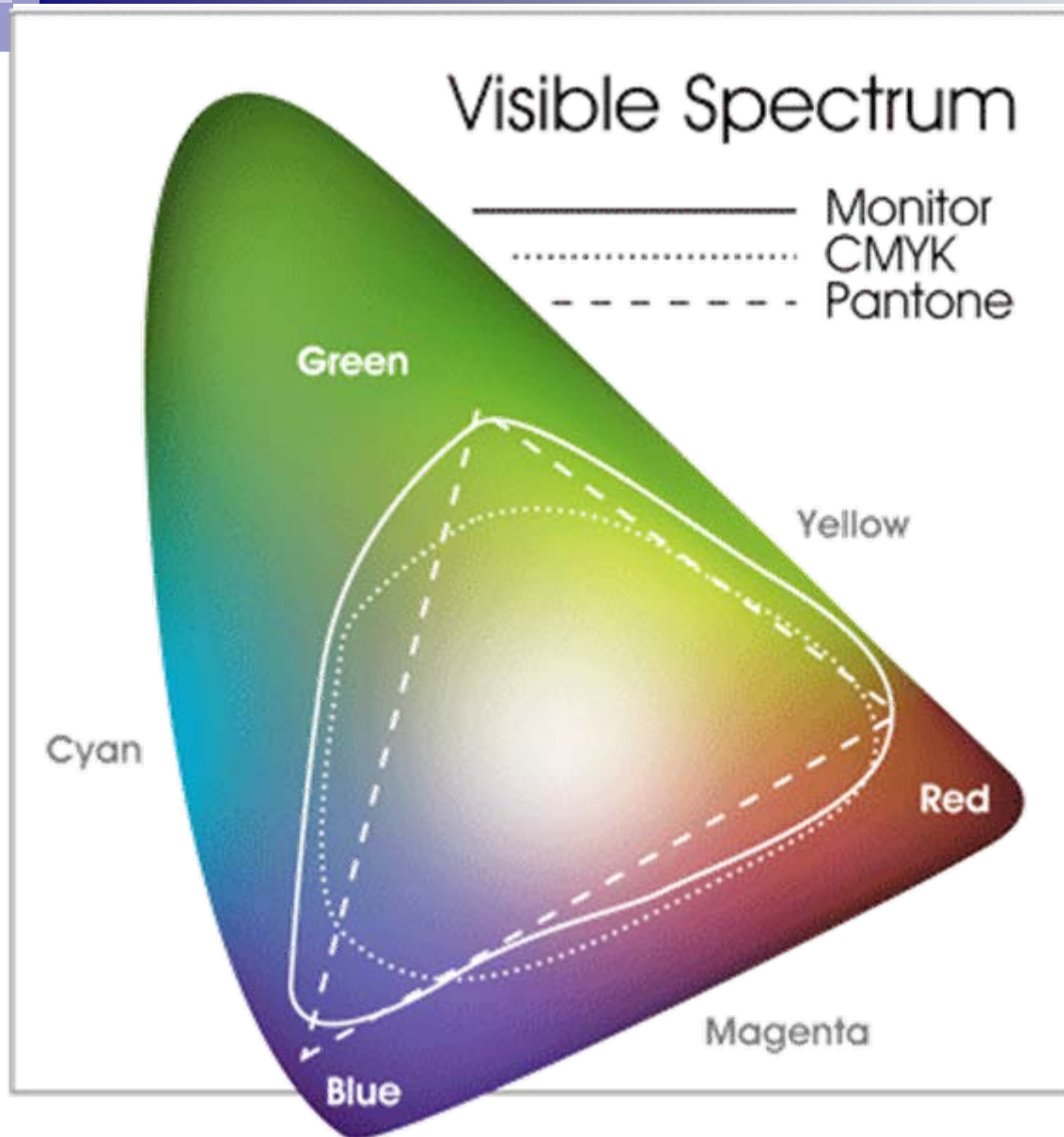
Pantone systém tiskařských barev

Pantone Color Matching System

Pantone® Red / Pantone® Warm Red

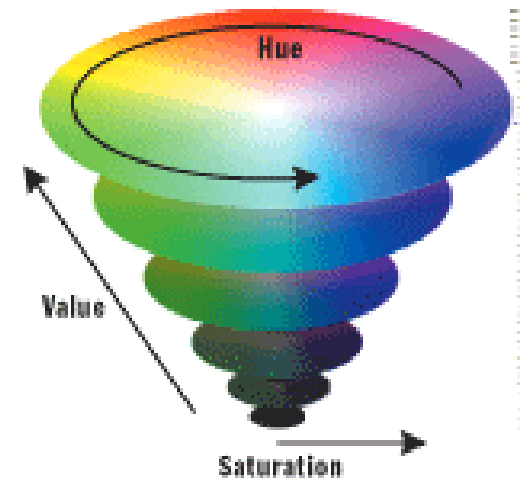
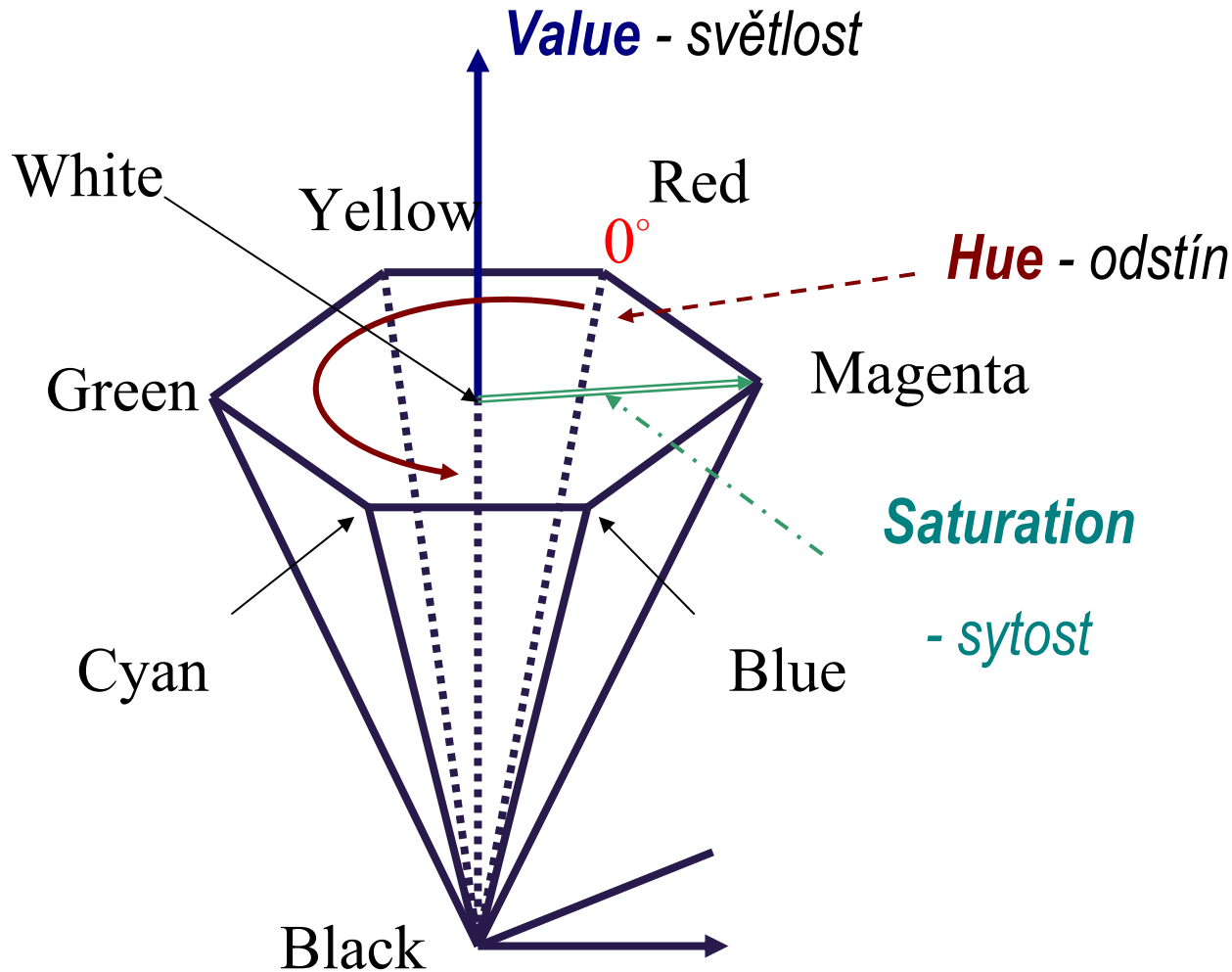
- nejen standard přesných barev pro profesionální tisk
- ale i tvorba barev mimo CMYK z barevných pigmentů



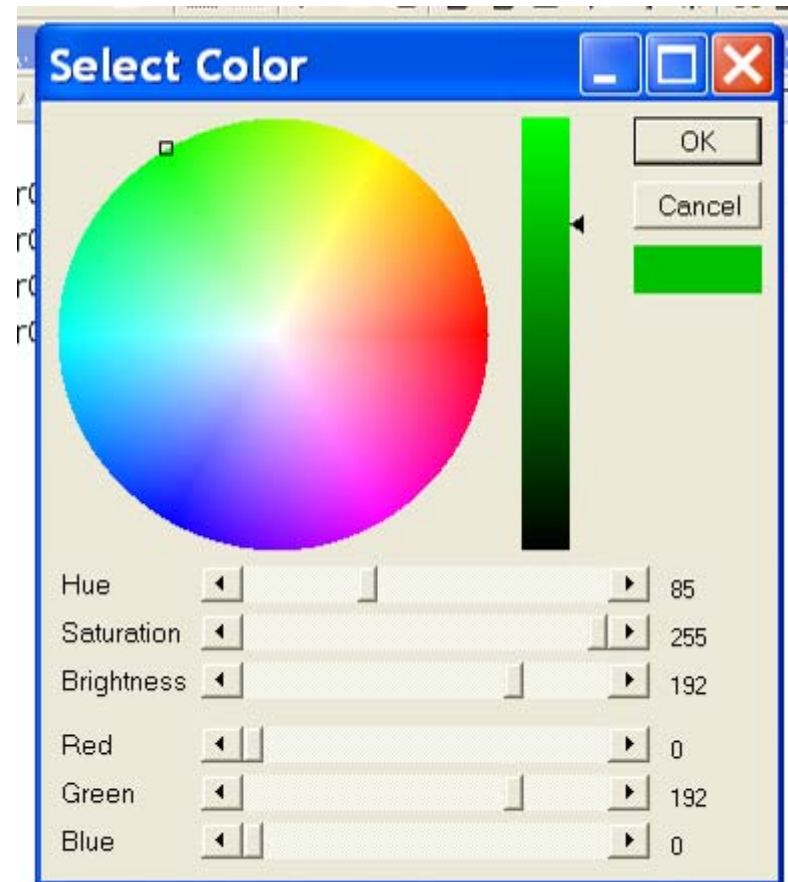


Zdroj: www.graphiccomm.com/

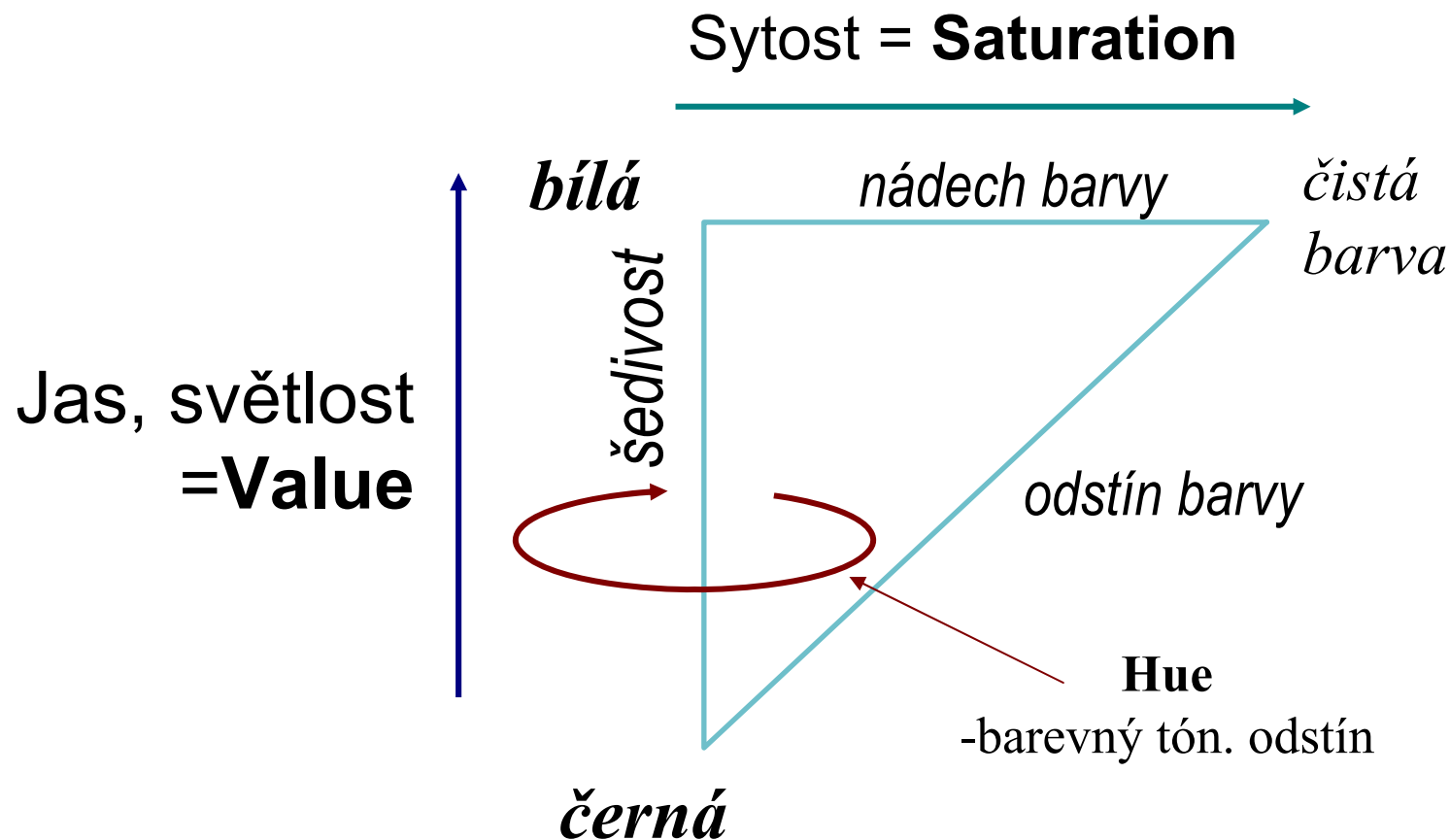
HSV systém pro intuitivní tvorbu barev



Převod mezi RGB a HSV je poměrně složitý, v příloze k přednášce najdete <http://msdn.microsoft.com/msdnmag/issues/03/07/GDIColorPicker/> - nyní nedostupný, v něm třída ColorHandler obsahuje HSVToRGB a RGBToHSV metody.



Význam parametrů HSV

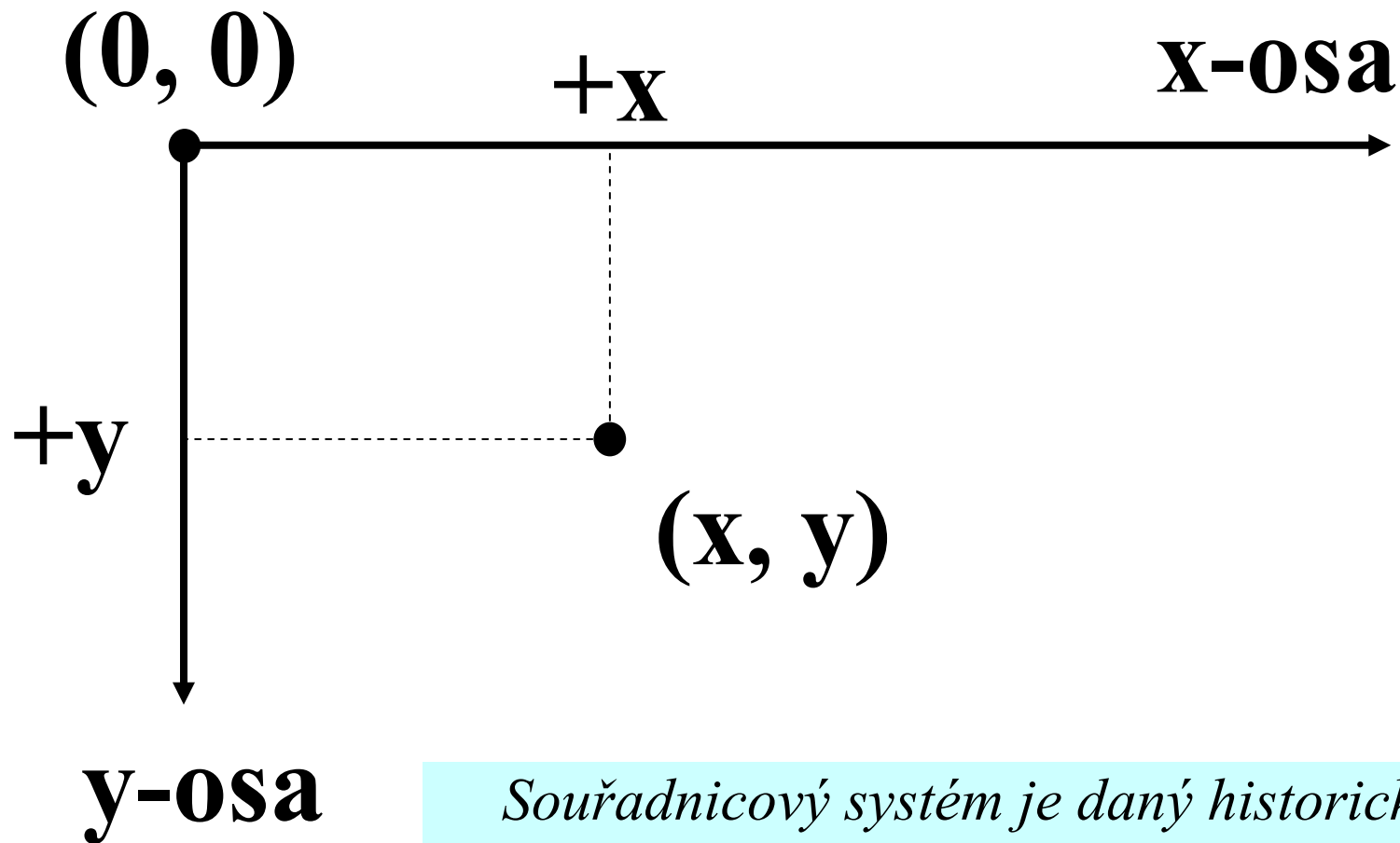


Kreslení v GDI+ není nijak složité...

pokud rozumíme jeho základním prvkům

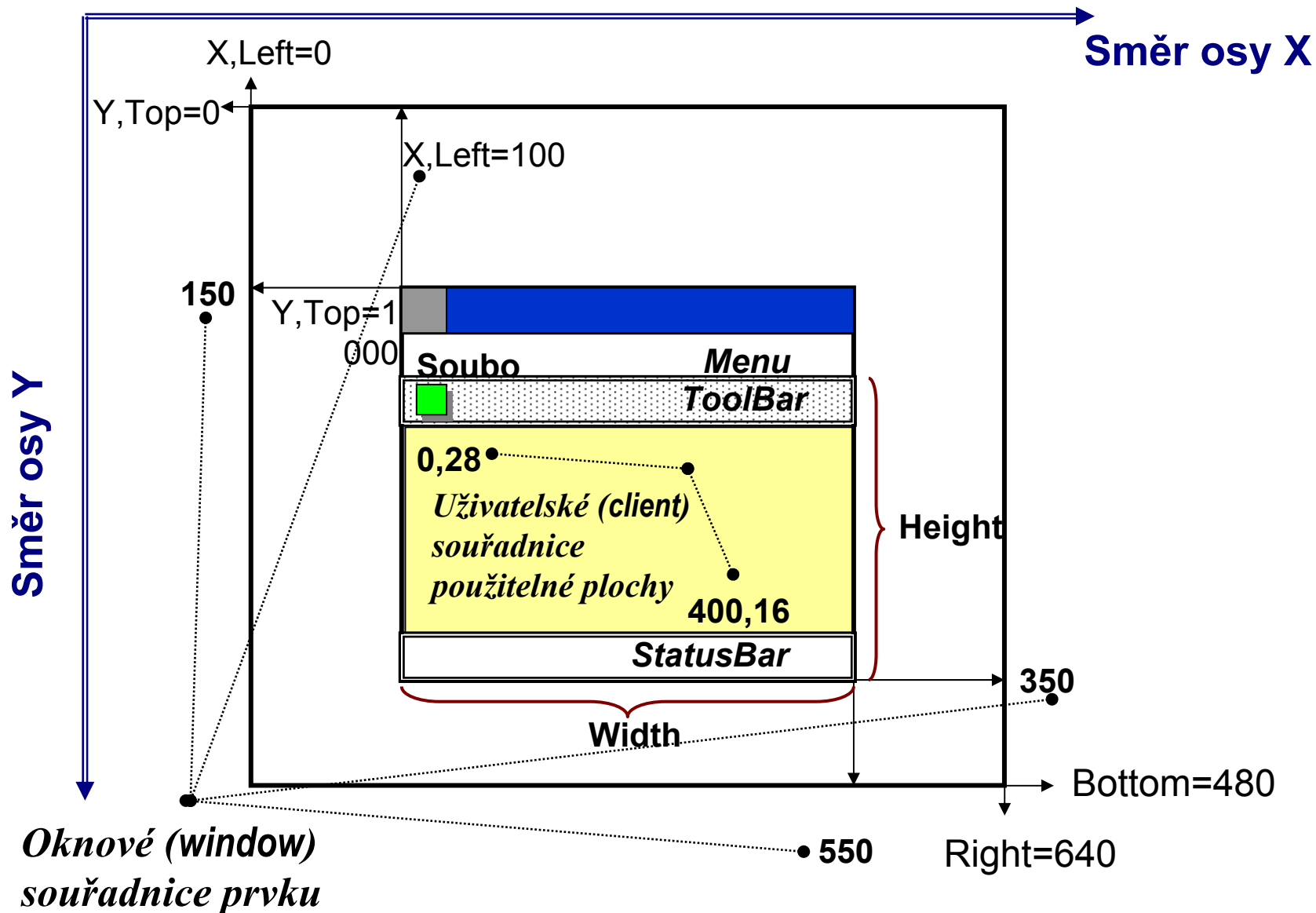


Souřadnicový systém v GDI+



Souřadnicový systém je daný historicky rozkladem obrazu u CRT monitorů

Souřadnice okna versus klientské souřadnice



Draw...

...**Line**(pt1, pt2)



...**Lines**(pt[])



...**Arc**



...**Curves, Bezier**



...**String** (string, x,y) **Text**

pt... – parametr typu Point

Některé grafické příkazy pro obrázky

Draw...

...Icon

...IconUnscaled

...Image

...ImageUnscaled



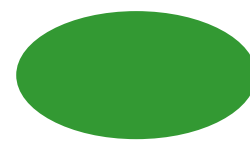
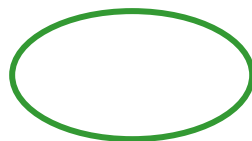
*The Great Wave Off Kanagawa, ©1831 Katsushika Hokusai,
Hakone Museum in Japan*

Některé grafické příkazy s Draw a Fill

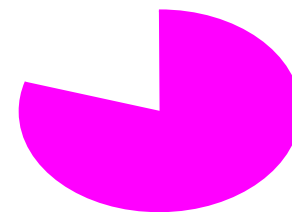
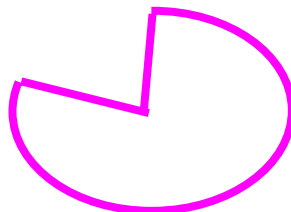
Draw...

Fill...

...**Ellipse** (rect)



...**Pie** (rect, start, end)



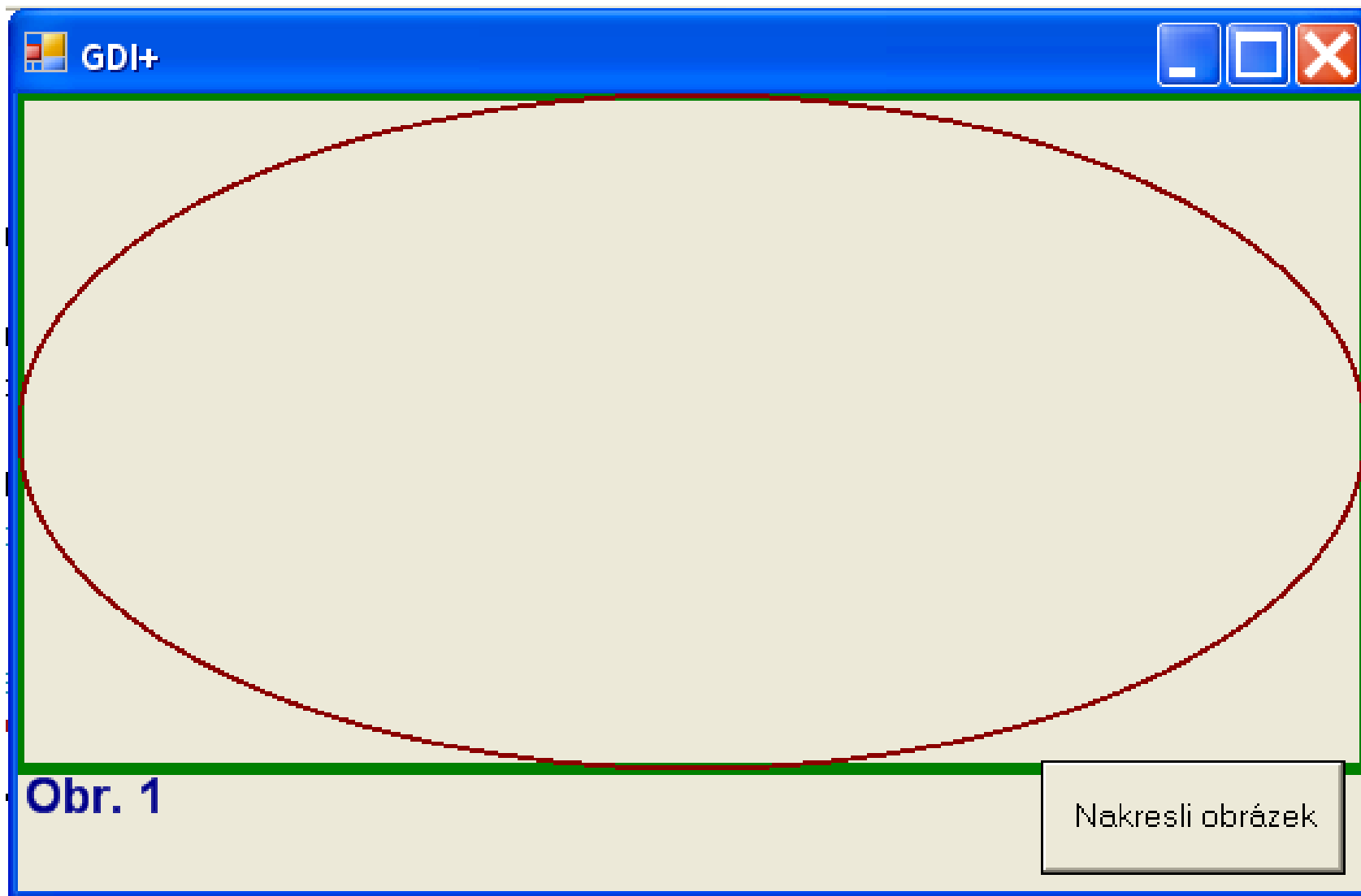
...**Rectangle** (rect)



...**Polygon** (pt[])



pt, rect – parametr typu Point, Rectangle

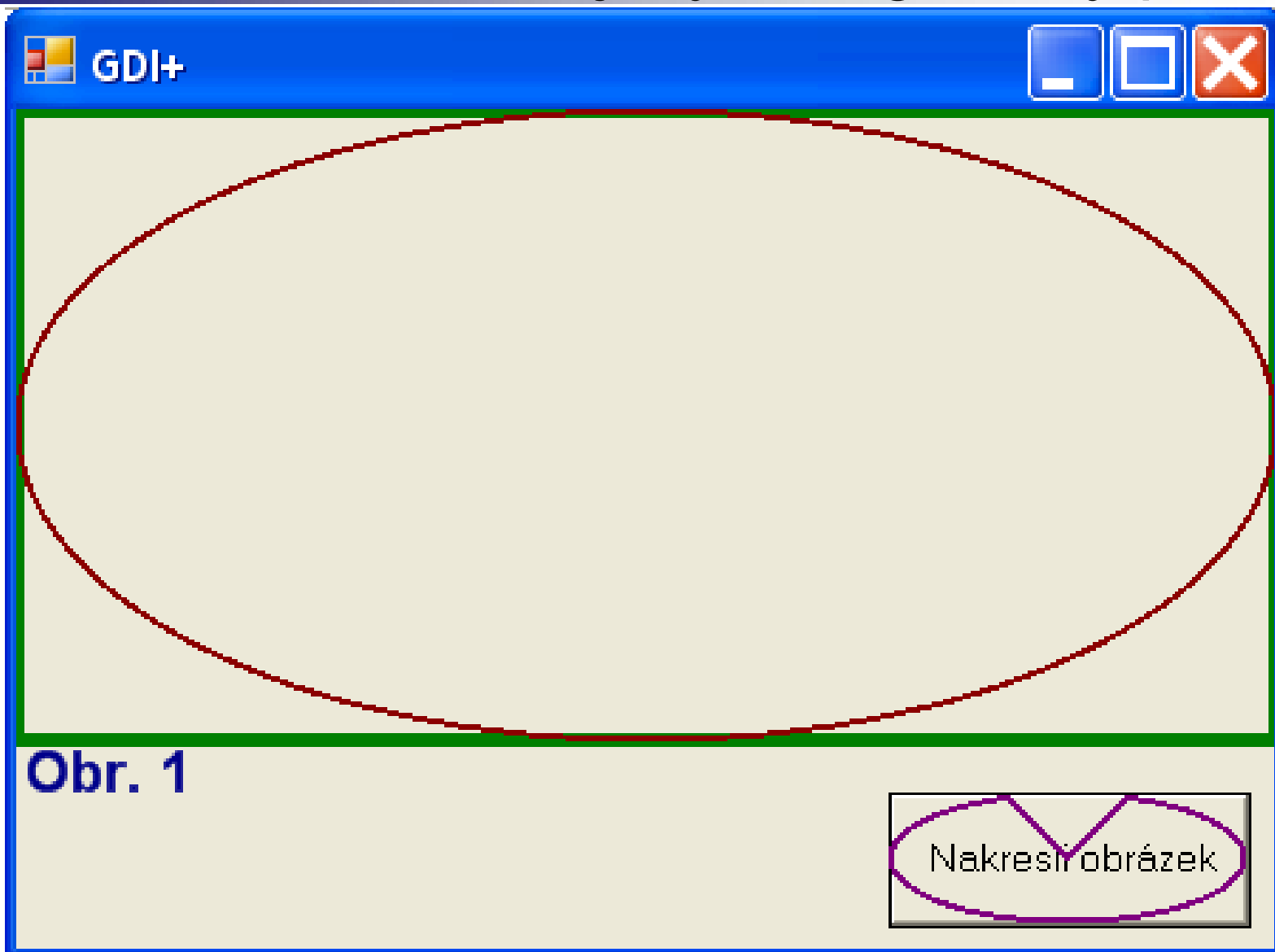


```
private void btnNakresliObrazek_Click(  
    object sender, System.EventArgs e)  
{  
    // vytvoříme DC - Device Context  
    Graphics dc = this.CreateGraphics();  
    /** příkazy pro kreslení **/  
    dc.Dispose(); // uvolnění systémového zdroje  
}
```

```
// Ale mnohem lepší konstrukce je  
using(Graphics dc = this.CreateGraphics())  
{  
    /** příkazy pro kreslení **/  
    // Dispose() se nyní provede i při výjimce  
}
```

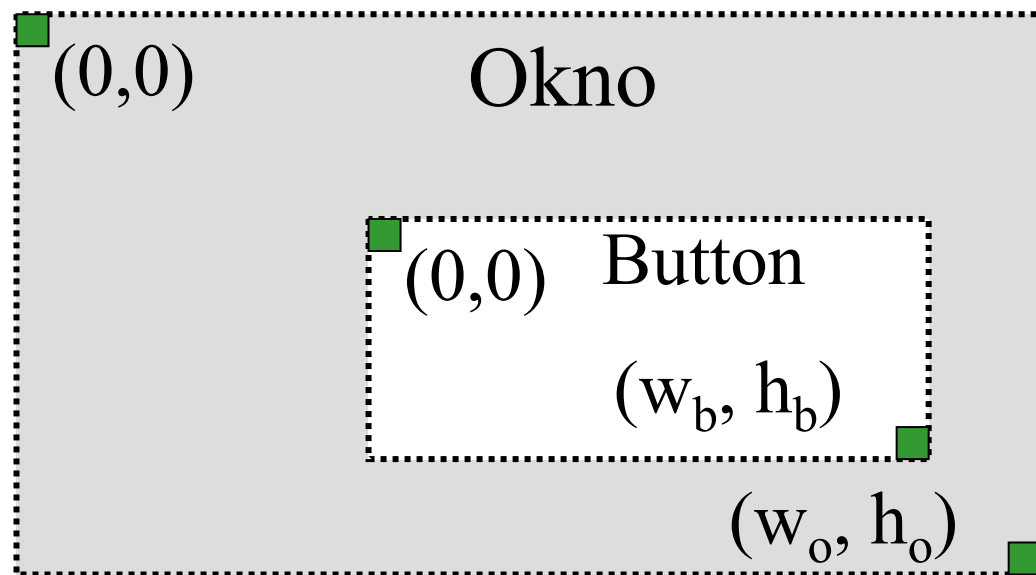
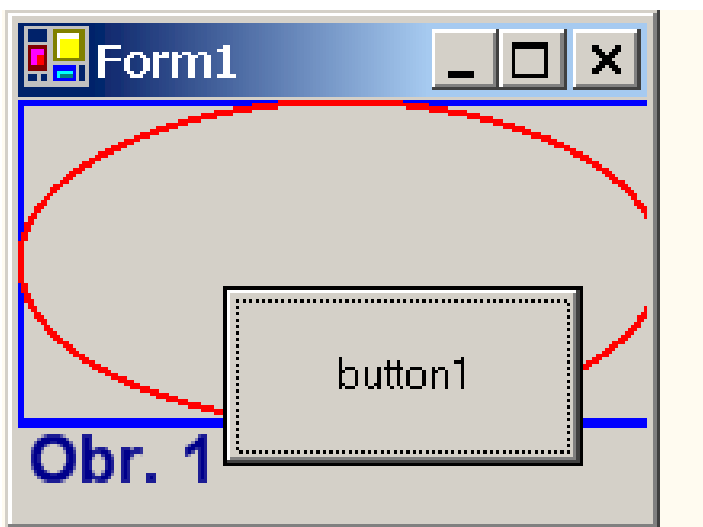
```
using(Graphics dc = this.CreateGraphics()) // Device Context
{
    int yDelka = this.ClientSize.Width; // použitelná část okna
    dc.Clear(this.BackColor);
    Pen greenPen = new Pen(Color.Green, 5); // zelené pero síly 5
    dc.DrawRectangle(greenPen, 0,0,yDelka,yDelka/2); // obdélník
    Pen redPen = new Pen(Color.DarkRed, 2); // červené pero síly 2
    dc.DrawEllipse(redPen, 0, 0, yDelka, yDelka/2); // elipsa
    SolidBrush brush = new SolidBrush( Color.DarkBlue );
    FontStyle style = FontStyle.Bold; // tučné písmo
    Font arial = new Font( "Arial", 12, style ); // 12 bodů velké písmo
    dc.DrawString( "Obr. 1", arial, brush, 0, yDelka/2 ); // napiš Text
}
```

Kreslit lze na jakýkoliv grafický prvek

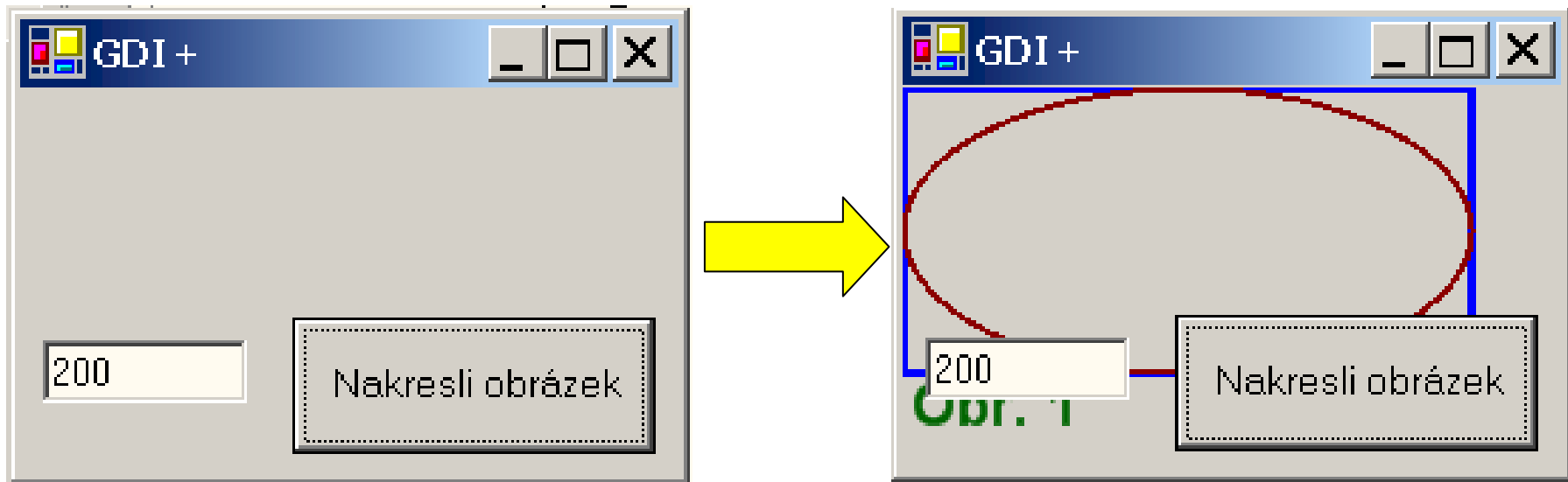


```
using(Graphics dc
    =btnNakresliObrazek.CreateGraphics())
{ dc.DrawPie(
    new Pen(Color.Purple, 2),           // pero
    0, 0,                               // x,y levého horného rohu
    btnNakresliObrazek.ClientSize.Width-3,
    btnNakresliObrazek.ClientSize.Height-3,
    -45, // start angle
    270 // délka oblouku ve směru hodin. ručiček
    );
}
```

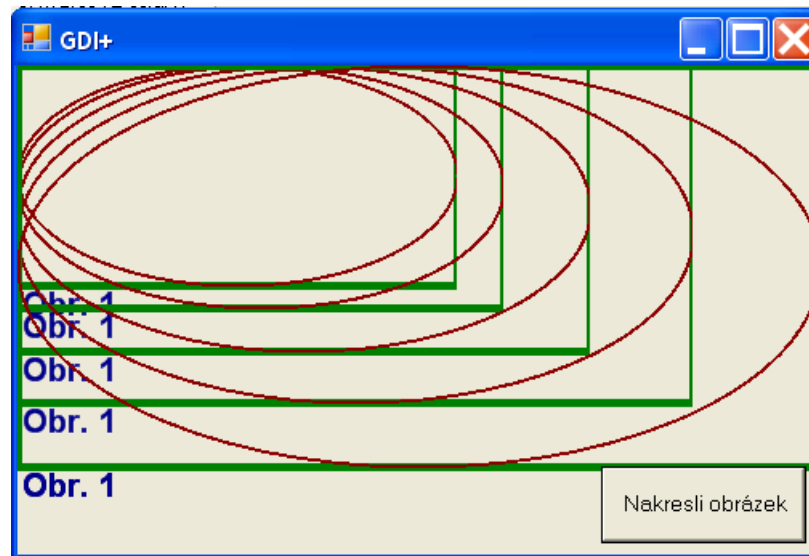
- Každý prvek má své vlastní souřadnice a nelze (*s jeho DC*) kreslit mimo jeho plochu nebo do jiných oken!



- OS zajišťuje tzv. "clipping" - ořezávání kresby na plochu prvku, jemuž patří device context (tj. class Graphics)



- Mazání okna si tady zajišťujeme sami, pokud vynecháme `dc.Clear(this.BackColor);` budou se obrázky kreslit přes sebe:



// `dc.Clear(this.BackColor);`

- Obrázek se nezruší přemístěním okna na jinou pozici
 - OS kopíruje obsah.
- Kresba zůstane zachovaná i při zobrazení menu
 - OS schovává okno.
- Obrázek se ale poruší při zakrytí části okna jiným oknem.
- OS hlásí znehodnocení části okna zprávou WM_PAINT, tj. událostí Paint

```
public Form1()
```

```
{
```

```
/* Po zavolání této metody již existuje viditelné  
okno, ta ho totiž vytvoří */
```

```
InitializeComponent();
```

```
// Poslat WM_PAINT při změně velikosti okna
```

```
this.ResizeRedraw=true;
```

```
/* Pozn. Nepovinné užití this zvyšuje přehlednost  
kódu u rozlehlejších objektů */
```

```
}
```

```
private void Form1_Paint(object sender,  
    System.Windows.Forms.PaintEventArgs e)  
{  
    /* Nesmime si vytvorit vlastni DC, protože by se nic  
    nekreslilo! Řekněte proč! */  
    //using(Graphics dc = this.CreateGraphics())  
    Graphics dc = e.Graphics; // kopie reference  
    // Nemažeme znovu okno – už je smazáno!  
    //dc.Clear(this.BackColor);  
  
    int yDelka = this.ClientSize.Width;  
    /*...kód kreslení je totožný s předchozím příkladem */  
}
```

- OS pošle WM_PAINT zprávu do fronty zpráv v okamžiku, kdy je opět viditelná znehodnocená plocha okna, nebo některá její část.
- Pokud již existuje jiná WM_PAINT zpráva ve frontě zpráv, OS ji vyřadí a sloučí její "clipping" plochu s "clipping" plochou nové zprávy – tím se zamezí zbytečnému překreslování.
- OS vytvoří pro WM_PAINT zprávu Device Context s clipping plochou nastavenou na viditelnou část znehodnoceného úseku okna.
- Při vytváření DC se smaže "Clipping" plocha na barvu BackColor nastavenou v grafickém prvku, jemuž patří WM_PAINT zpráva.

Tiskneme opět pomocí DC

```
private void tisknemeToolStripMenuItem_Click(object sender, EventArgs e)
{
    PrintDialog prDiag = new PrintDialog();
    if (prDiag.ShowDialog() != DialogResult.OK) return;
    PrintDocument prndoc = new PrintDocument();
    prndoc.PrinterSettings = prDiag.PrinterSettings;
    prndoc.DocumentName = "GDIprint";
    prndoc.PrintPage +=
        new PrintPageEventHandler(PrintDocumentOnPrintPage);
    prndoc.Print();
}

void PrintDocumentOnPrintPage(object obj, PrintPageEventArgs ppea)
{
    Graphics grfx = ppea.Graphics;
   .SizeF sizef = grfx.VisibleClipBounds.Size;
    Kresba(grfx);
}
```

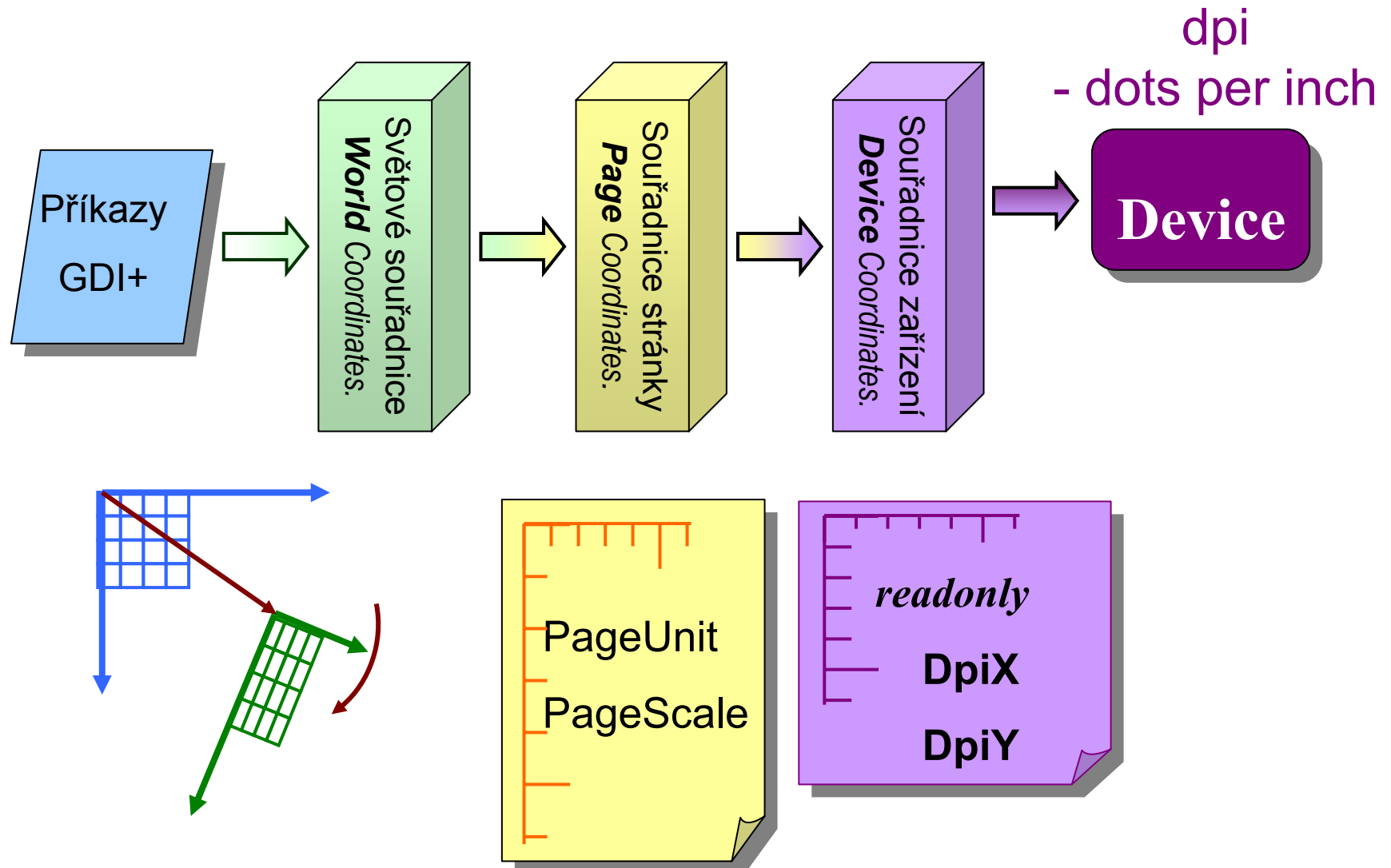

Transformace souřadnic

Světové a zařízení



- Nevýhodou kreslení do prvku s proměnnou velikostí je nutnost neustále přepočítávat velikost kresby podle jeho aktuálních rozměrů.
- Zde pomohou transformace dovolující kreslit na kreslící plochu s pevnou velikostí, která se mapuje do skutečného okna

Transformace souřadnic



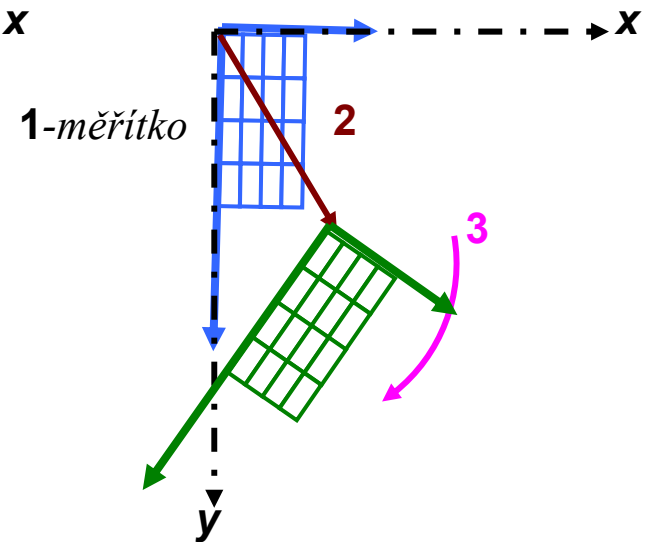
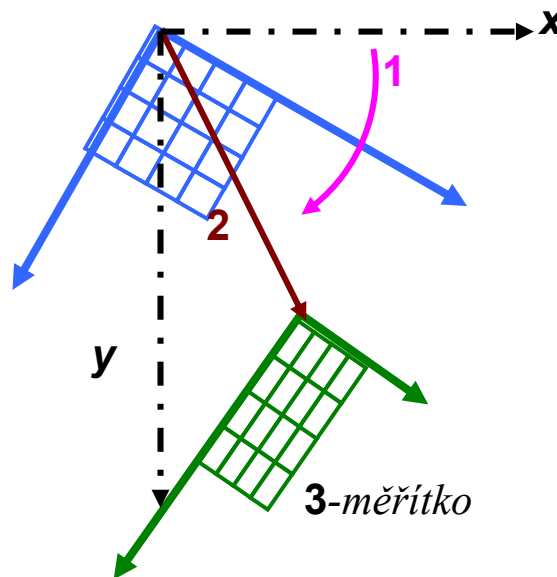
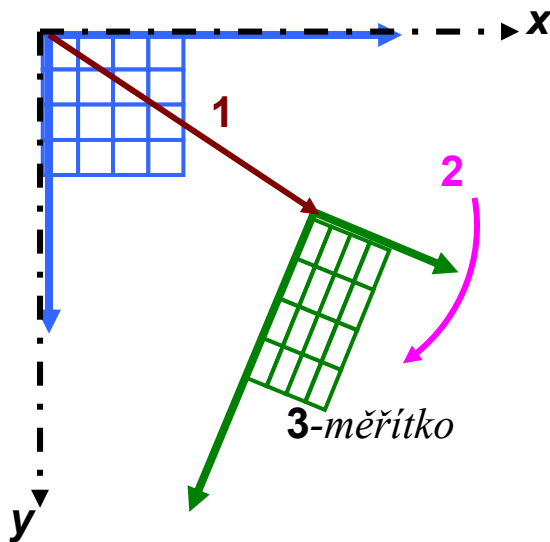
Skládáním transformací

TranslateTransform
(dx, dy);

RotateTransform
(alpha)

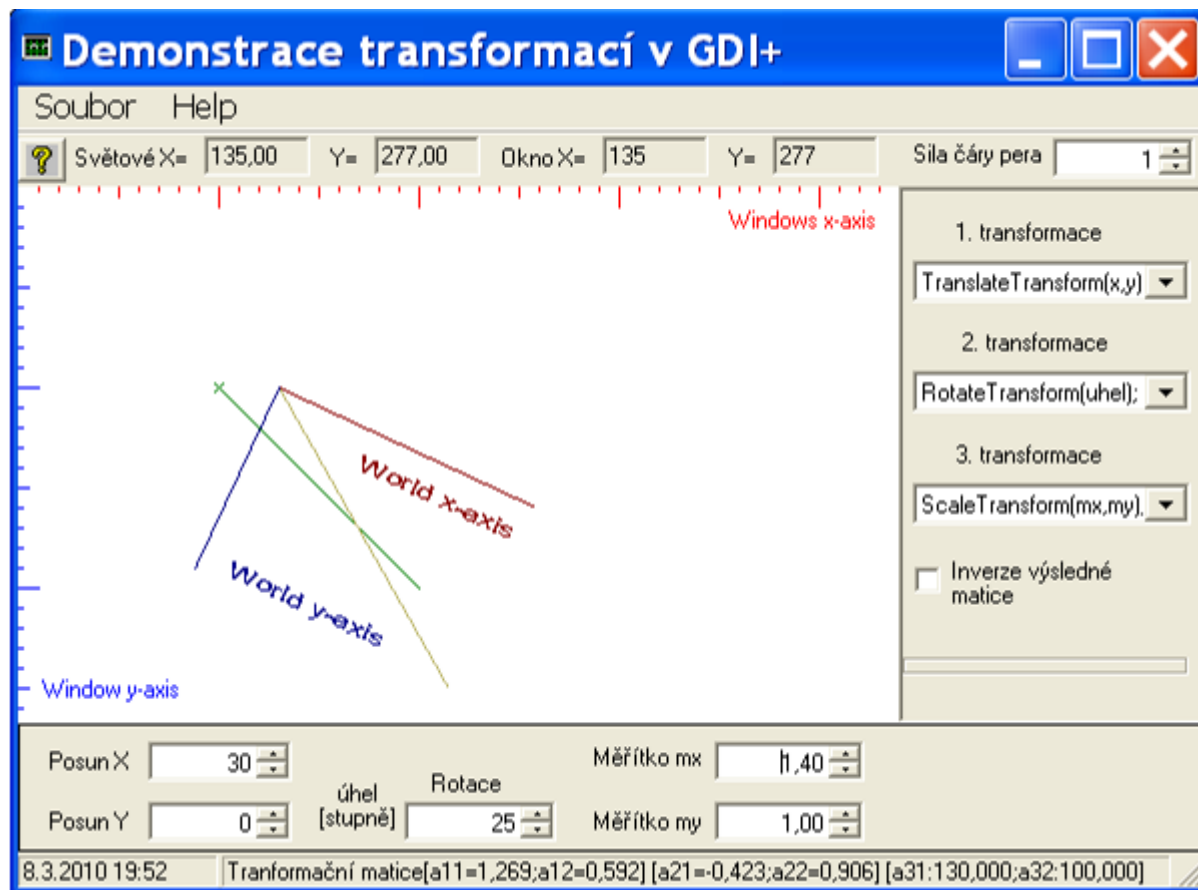
ScaleTransform
(mx, my)

$$\begin{bmatrix} x & y & 1 \end{bmatrix} * \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ \Delta x & \Delta y & 1 \end{bmatrix} * \begin{bmatrix} \cos \alpha & \sin \alpha & 0 \\ -\sin \alpha & \cos \alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} mx & 0 & 0 \\ 0 & my & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

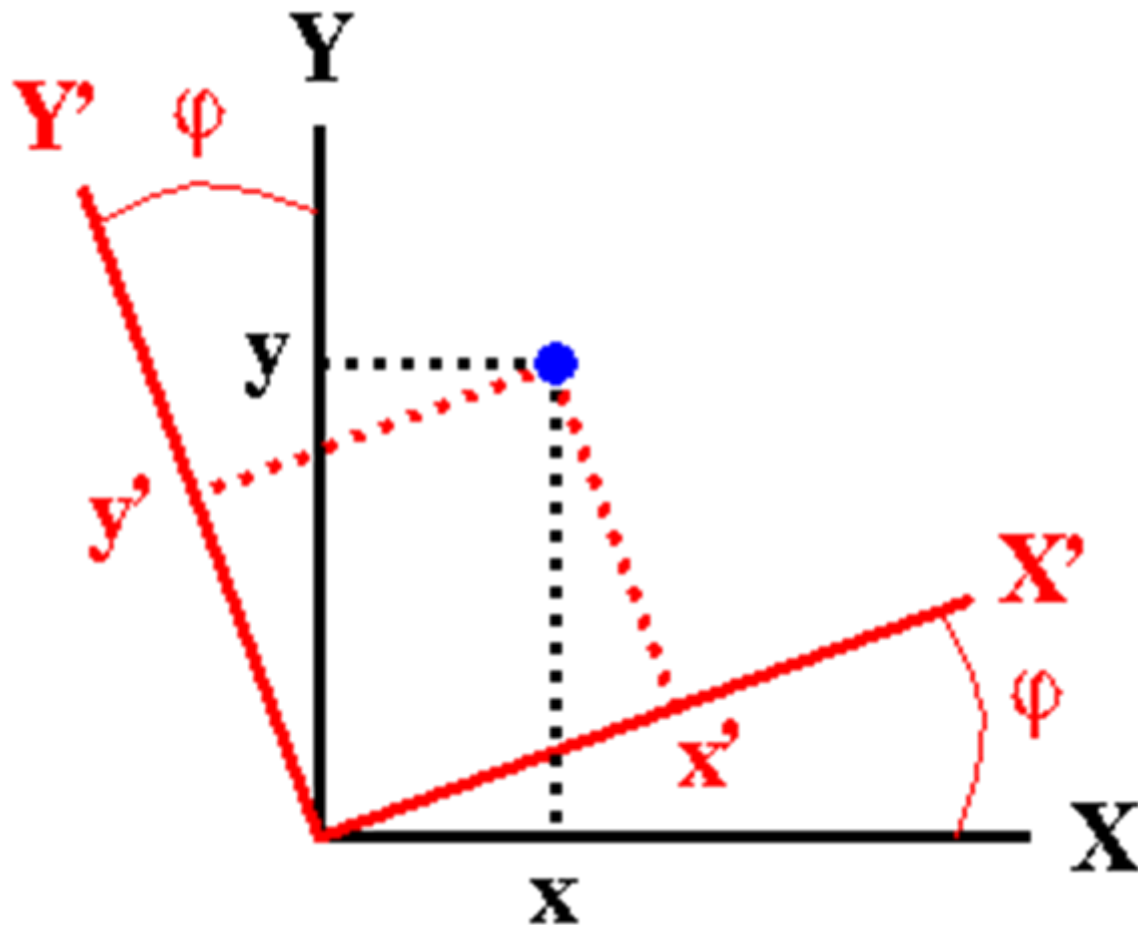


Výsledek závisí na pořadí transformací!

Demo - transformace souřadnice



- transformuje původní souřadnice x, y bodu do nové otočené soustavy souřadnic



Uložení transformační matice $Matrix\ m = gfx.Transform;$

$$\begin{bmatrix} m.Elements[0] & m.Elements[1] & 0 \\ m.Elements[2] & m.Elements[3] & 0 \\ m.OffsetX & m.OffsetY & 1 \end{bmatrix}$$

respektive

$$\begin{bmatrix} m.Elements[0] & m.Elements[1] & 0 \\ m.Elements[2] & m.Elements[3] & 0 \\ m.Elements[4] & m.Elements[5] & 1 \end{bmatrix}$$

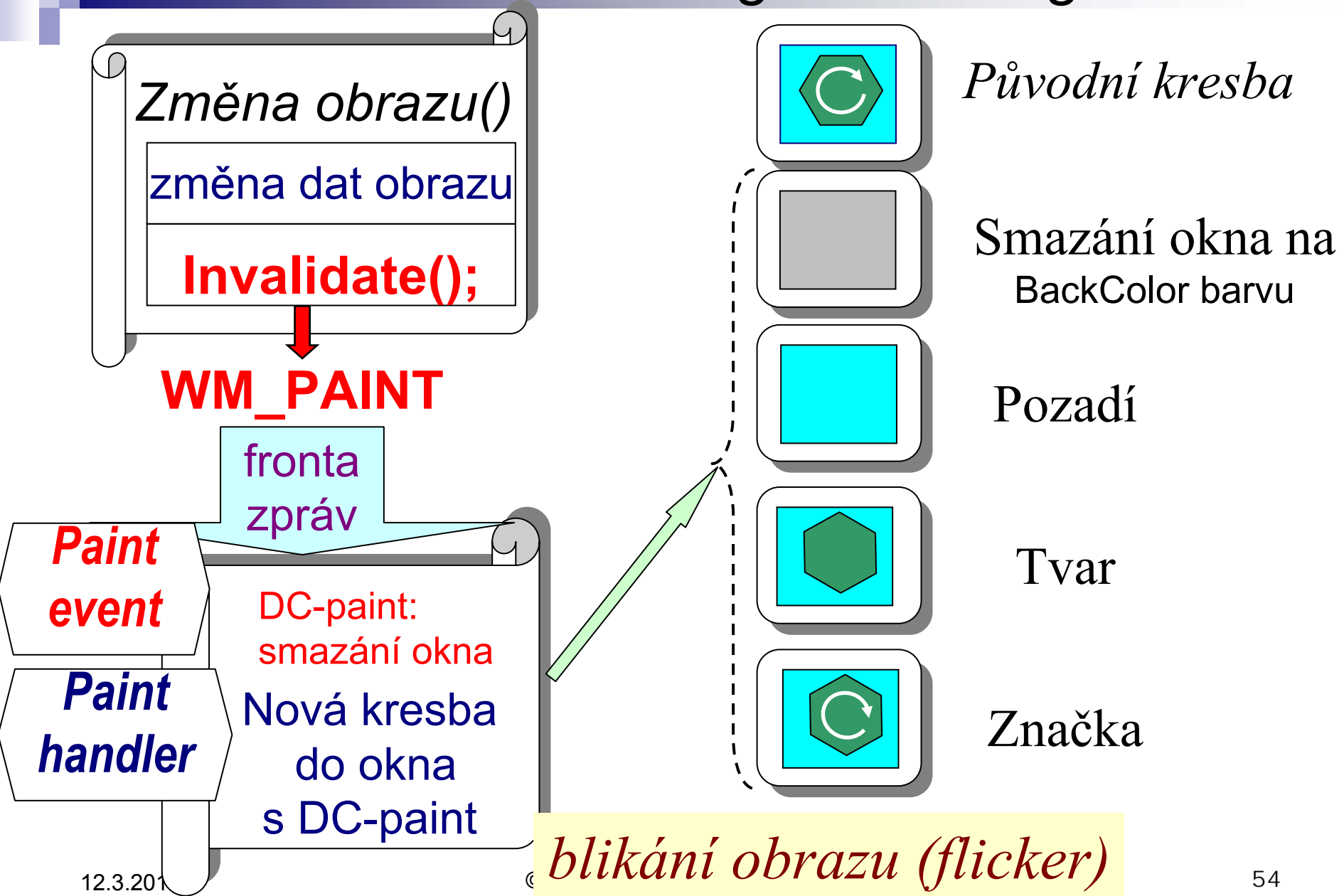
Způsoby kreslení v GDI+

Single buffering

Double buffering



Kreslení Single-Buffering v Paint



Double Buffering

Požadavek na
překreslení

Kreslíme v paměti

Smazání

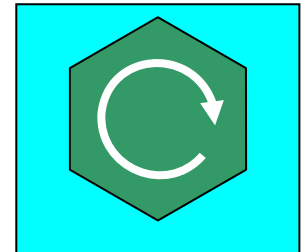
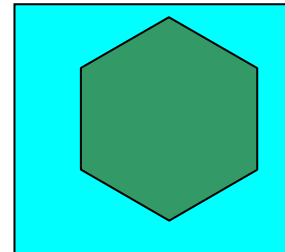
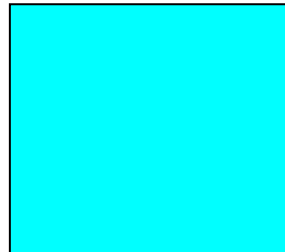
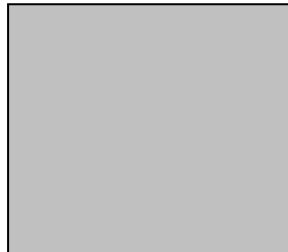
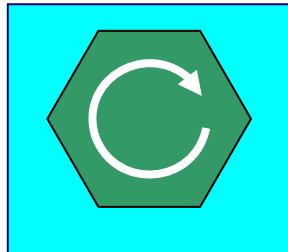
okna

pozadí

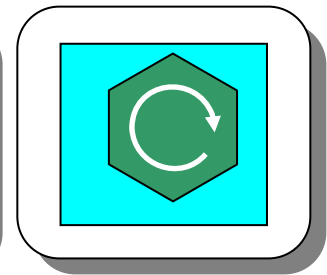
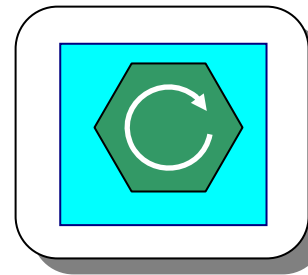
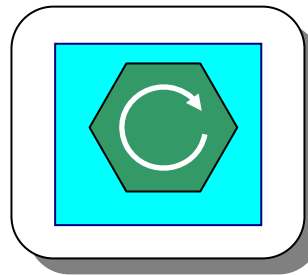
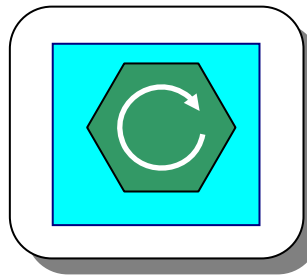
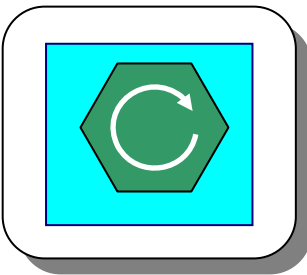
tvar

značka

p
a
m
ě
ť



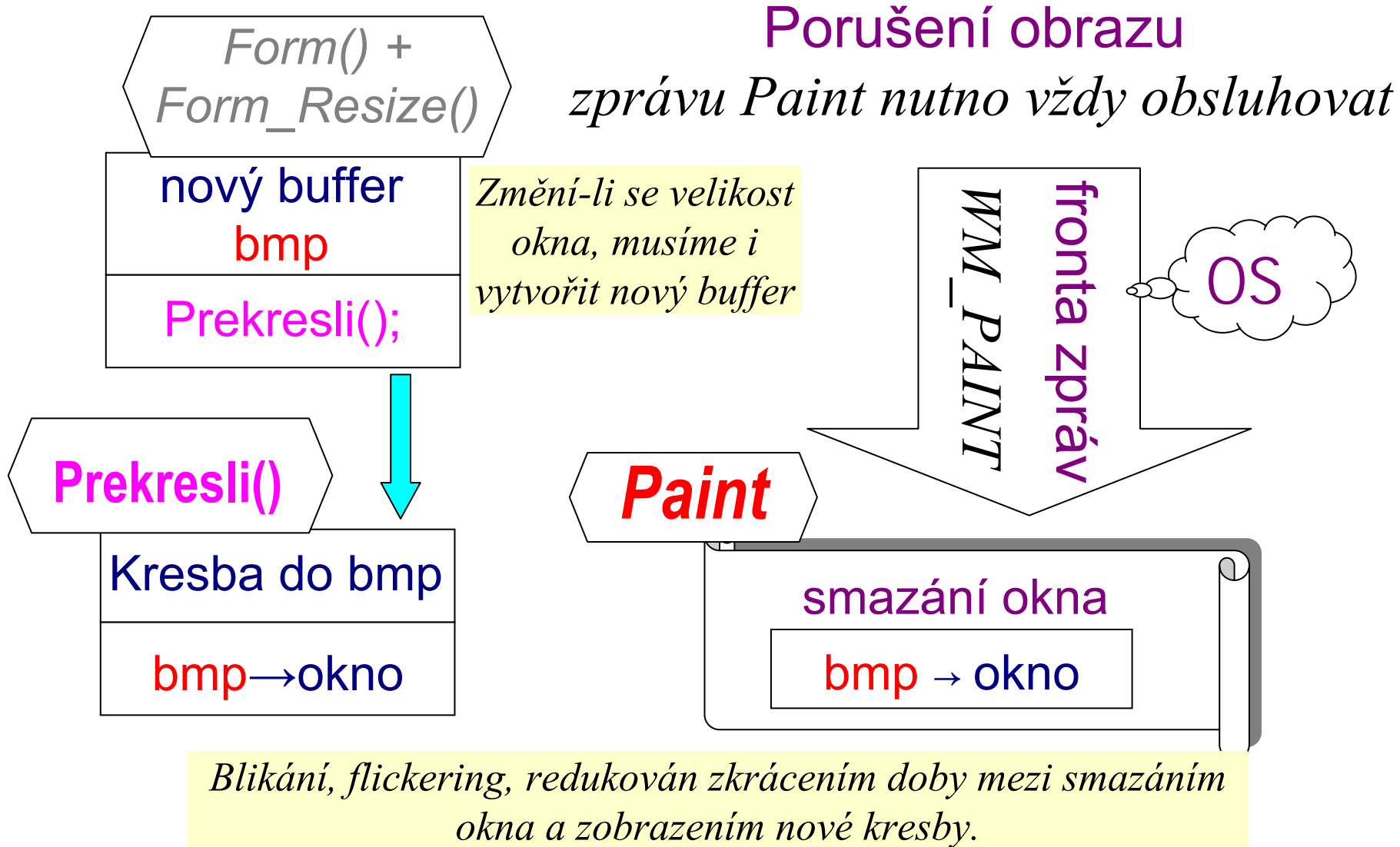
d
i
s
p
l
e
j



kopírování dat



Princip Double-Buffering



- Každá aplikace má svého výchozího **správce grafických bufferů**. Pokud nechceme vytvářet vlastního správce (v některých případech zvyšuje výkon aplikace), použijeme výchozího správce.

// nastavení správce grafických bufferů

```
BufferedGraphicsContext Context = BufferedGraphicsManager.Current;
```

- Je třeba v paměti **vytvořit** pomocí správce grafických bufferů vhodný **buffer** (pomocí metody *Allocate()* správce).

// tvorba bufferu pro kreslení

```
BufferedGraphics BufferPlatna =
```

```
Context.Allocate(vPanel.CreateGraphics(), vPanel.DisplayRectangle);
```

- ```
// přístup k DC bufferu
BufferPlatna.Graphics;
```

- ```
// vykreslení bufferu
BufferPlatna.Render();
```

1. Implementujte metodu **KresliDoBufferu(Graphics dc)**, která vykreslí do předaného DC jeden (další) snímek animace. Pro vykreslení obrázku do DC můžete použít jeho metody *Clear()* a *DrawImage()*.

```
g.Clear(this.BackColor);           // smazání DC  
g.DrawImage(Obraz, Clip);         // vykreslení obrázku
```

2. V obsluze události **Paint skutečného výstupu** vykreslete buffer voláním jeho metody *Render()*.

```
BufferPlatna.Render(); // vykreslení bufferu
```



3. Pro řízení animace použijte *timer*. V **obsluze** jeho **události Tick** vykreslete do bufferu další snímek animace pomocí metody `KresliDoBufferu(Graphics dc)`. Následně buffer vykreslete.

```
KresliDoBufferu(BufferPlatna.Graphics);  
BufferPlatna.Render();
```

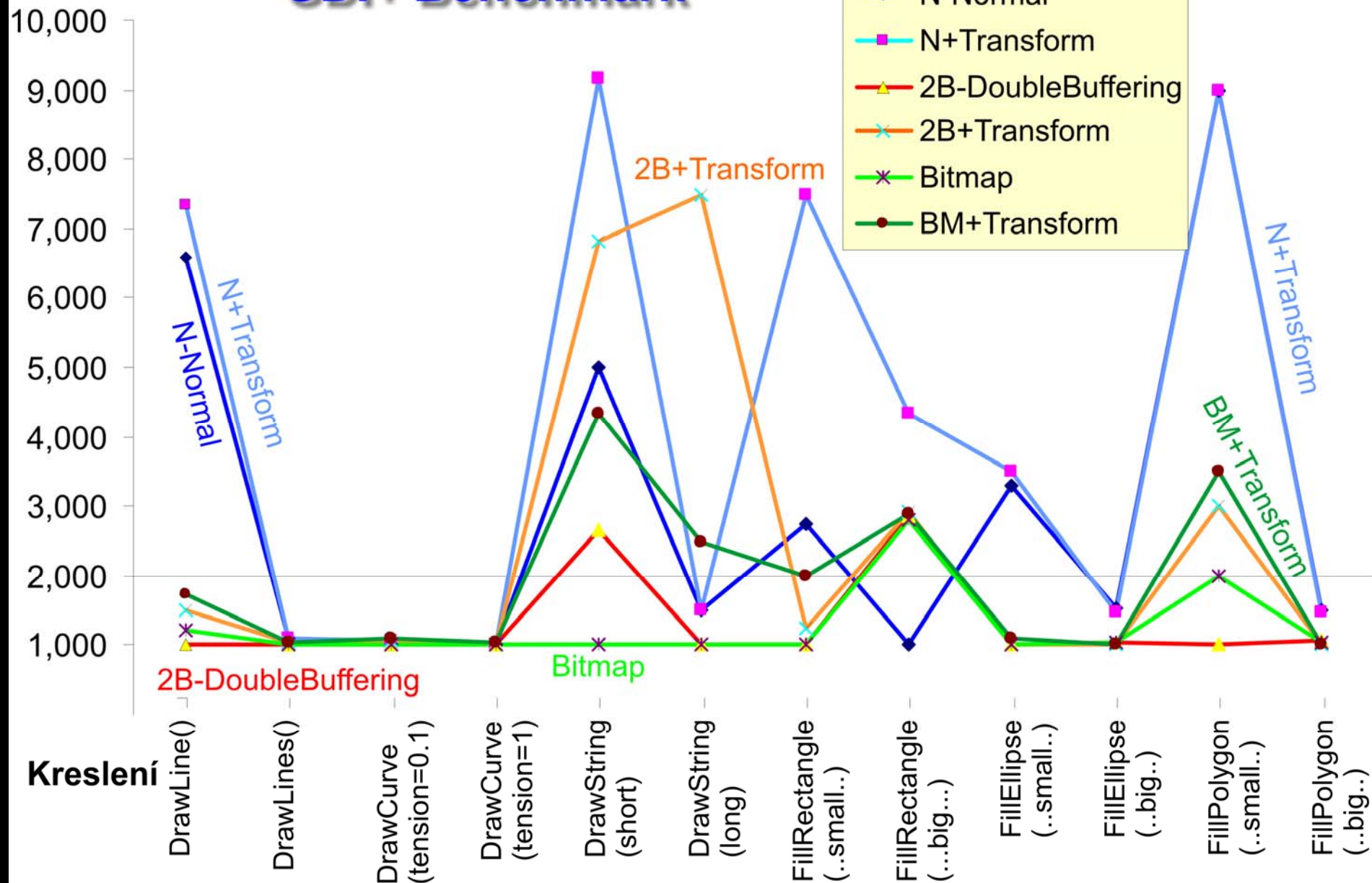
Demo - DoubleBuffering Benchmarks

- Srovnáme GDI+
- DoubleBuffering systémový
- DoubleBuffering pomocí bitmap

12.3.2010

relativní doba

GDI+ Benchmark



Single versus double buffering

- + "Double buffering" zamezí blikání obrazu, ale neurychlí kreslení
 - Má rozdílnou rychlost oproti GDI+ u neefektivních kreslicích algoritmů bývá rychlejší, neposílá příkazy na grafickou kartu, u jiných metod zase výrazně horší, jelikož nevyužívá akcelerace grafické karty.
- Urychlení nabízí jen DirectX, resp. OpenGL



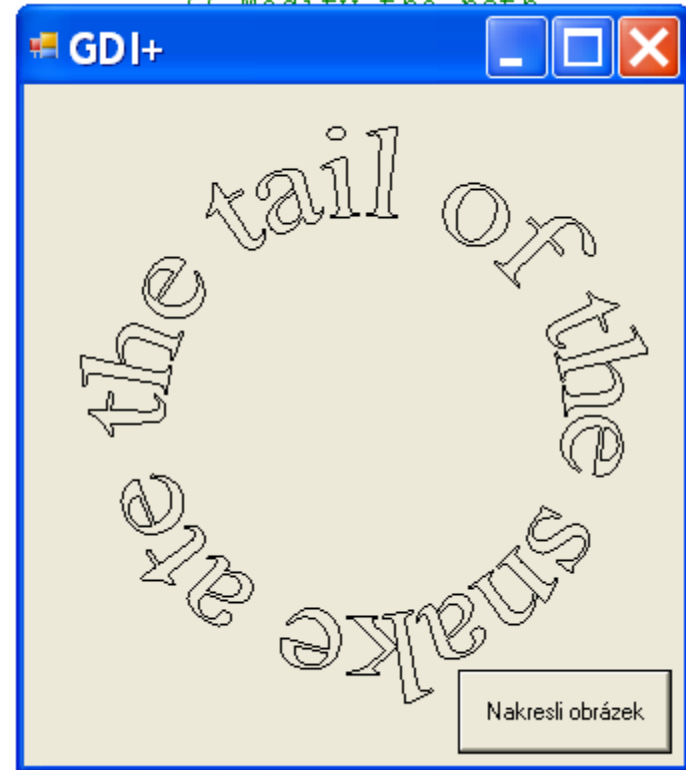
Problémy s textem...

je a není to jednoduché



Nemusíme kreslit jen pouhé čáry

- GDI+ umí hodně z typografie
 - tohle třeba zvládne GraphicsPath



Překlad vytvořený podle knihy "Petzold, Programming Microsoft Windows with C#"

Knihu ale nedoporučuji ke studiu, přišla mi dost nepřehledná a s malým množstvím zajímavých informací.

Uvedený příklad se mi zdál jako jeden z velmi málo dobrých. Jeho kód najdete v programu GDIkresba, přiloženém k přednášce, a to v metodě jako KresbaKruhovehoTextu()

■ PageUnit = GraphicsUnit

.Display 1/75 inch

.Document 1/300 inch

.Inch = palec, coul

.Millimeter

.Pixel

.Point 1 point = 1/72 palce = 0.351 mm

12 pt.=1 pc (pica) = 4,22 mm

.World měřítko světové transformace

Jednotky webových kaskádových stylů

■ *ve WebControls existují v UnitType*

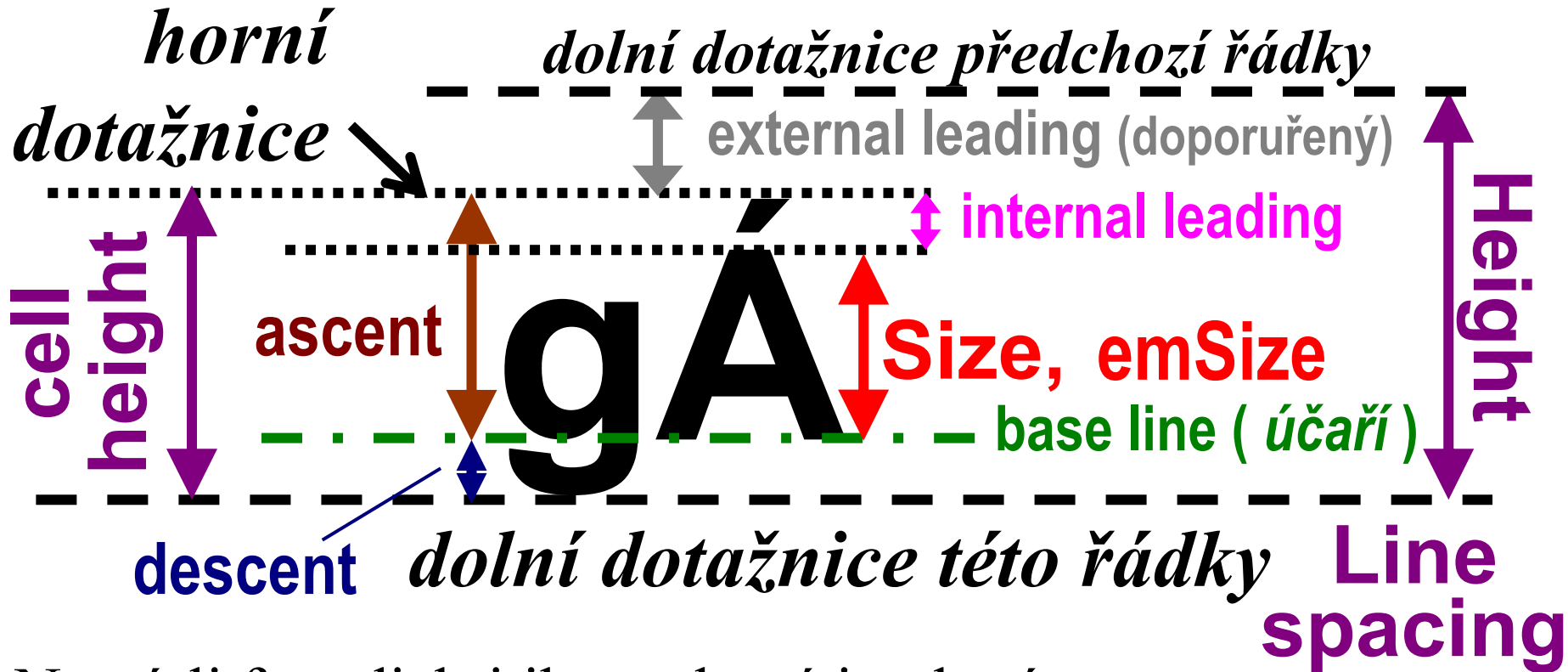
.Em *relativně vůči výšce rodičovského fontu.
Název vznikl podle M, které mívá stejnou
výšku i šířku ve většině fontů.*

.Ex *relativně vůči výšce malého x
v rodičovském fontu.*

*Psaní textu budou věnována
cvičení v dalším týdnu*

Velikost fontu

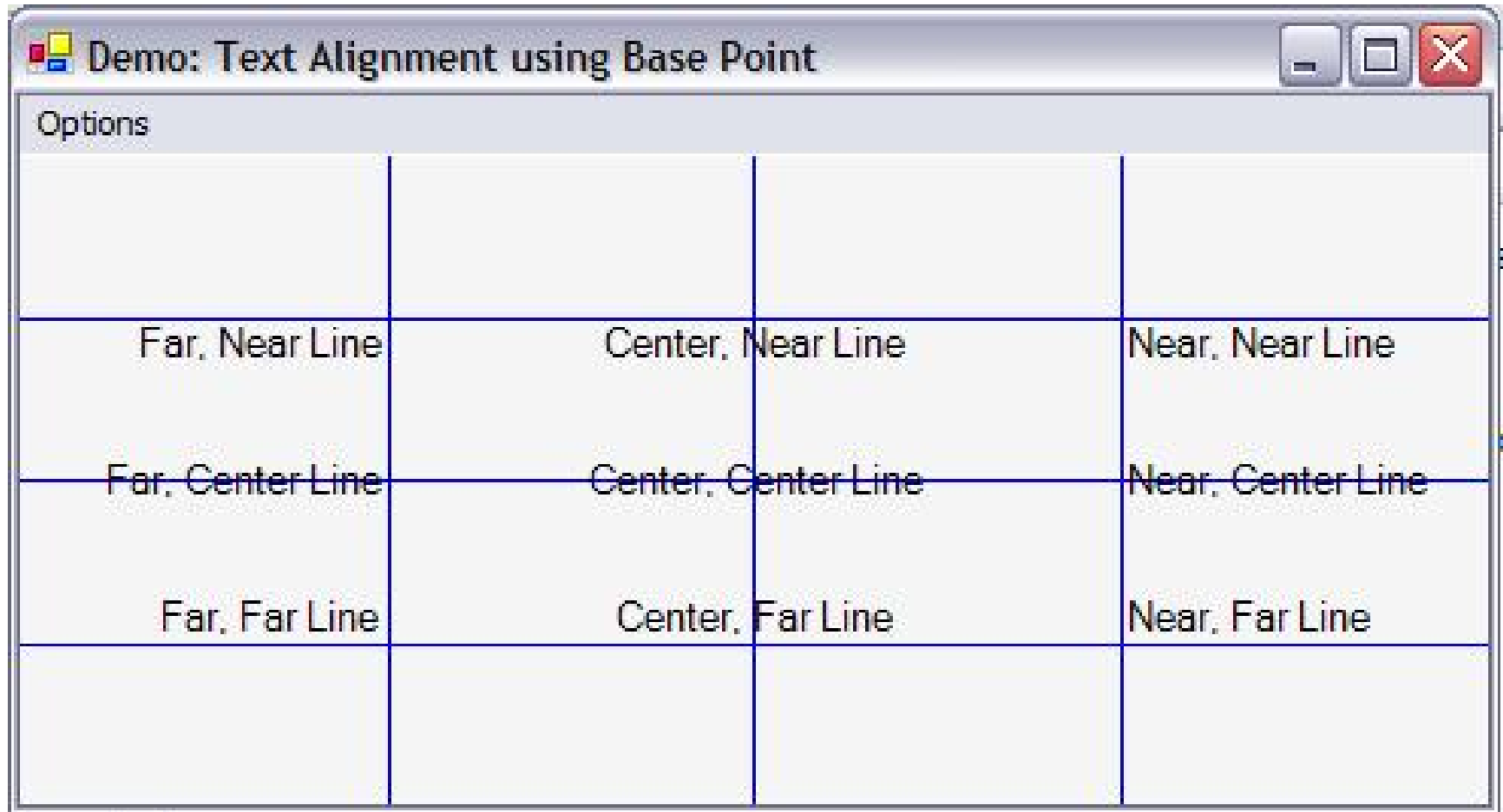
Tento obrázek bude také jednou otázkou u zkoušky.



- Nemá-li font diakritiku, pak má i nulový internal leading a jeho ascent odpovídá emSize
- V .NET se zadává velikost fontu v **emSize**!
- V jiných prostředích tomu bývá často jinak!

```
StringFormat f = new StringFormat();  
f.Alignment = StringAlignment.Near;  
f.LineAlignment = StringAlignment.Near;  
  
dc.DrawString("Near, Near Line",  
               this.Font, Brushes.Black, x, y,  
               f );
```

Výsledek pozicování



Options			
Far, Near Line	Center, Near Line	Near, Near Line	
Far, Center Line	Center, Center Line	Near, Center Line	
Far, Far Line	Center, Far Line	Near, Far Line	

[Zdroj: <http://www.srtsolutions.com/>, [Bill Wagner: C# Development Blog](#)]

Přesnější pozicování – nutno změřit text

Pro samostudium

- *Nejjednodušší, ale nepřesné změření textu...*

```
Font font = new Font("Arial", 100,  
    FontStyle.Regular, GraphicsUnit.Pixel );  
SizeF ex = dc.MeasureString("x", font);  
// {Width = 84.847 Height = 124.21875} [pixel]
```

```
SizeF velikost = dc.MeasureString("Říjen", font);  
// {Width = 273.763 Height = 124.21875} [pixel]
```

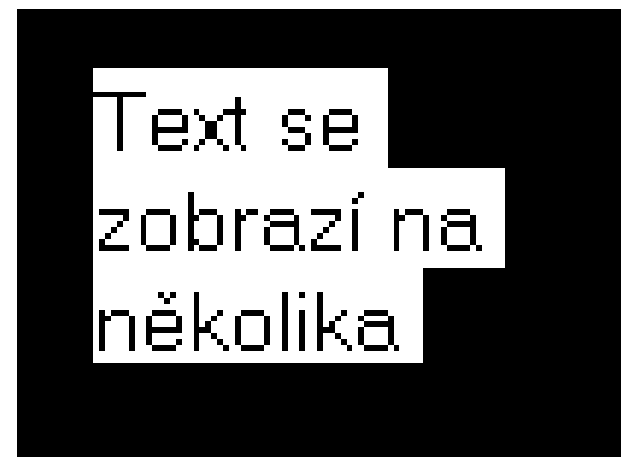
- *Šířka je skoro přesná, ale výška je výškou fontu, ne textu.*
- *Přesnější, ale poměrně složitější metoda je*
`Graphics.MeasureCharacterRanges(..)`
- *V praxi se obvykle volí MeasureString*
s pokusným doladěním pozice

■ `TextRenderer.DrawText`

- metoda dovoluje i psaní do vymezeného čtverce
- **nepodporuje ale výstup na tiskárnu**

```
TextFormatFlags flags = TextFormatFlags.Bottom |  
                        TextFormatFlags.WordBreak;
```

```
TextRenderer.DrawText(e.Graphics, "Text se zobrazí na  
několika řádkách.", this.Font,  
new Rectangle(10, 10, 100, 50),  
SystemColors.ControlText,  
SystemColors.ControlDark, flags);  
}
```



Příště přednáška o přetížených a virtuálních metodách

