# 1 Development Platforms for Multimedia Programming

Literature:

http://www.w3.org/TR/SMIL/
http://www.openlaszlo.org/

## SMIL - Idea and History

- Synchronized Multimedia Integration Language (pronounced: "Smile")
- Standard language for co-ordinated combination of time-dependent media elements into a multimedia presentation
  - Temporal dependencies are described explicitly (declarative language)
  - Integrates time-independent media (text, still image)
  - Suitable for "Streaming”
- Standardization by W3C (WWW Consortium)
  - First Draft November 1997
  - SMIL 1.0 Standard June 1998
  - since 1998: Implementations by CWI/Oratrix, HELIO, REAL and others
  - 1999: Plans for an extended and improved version ("Boston SMIL")
  - SMIL 2.0 Standard August 2001
  - SMIL 2.1 Recommendation Dec. 2005 (e.g. profile for mobile devices)
  - SMIL 3.0 Recommendation Dec. 2008

## Slideshow as SMIL Document

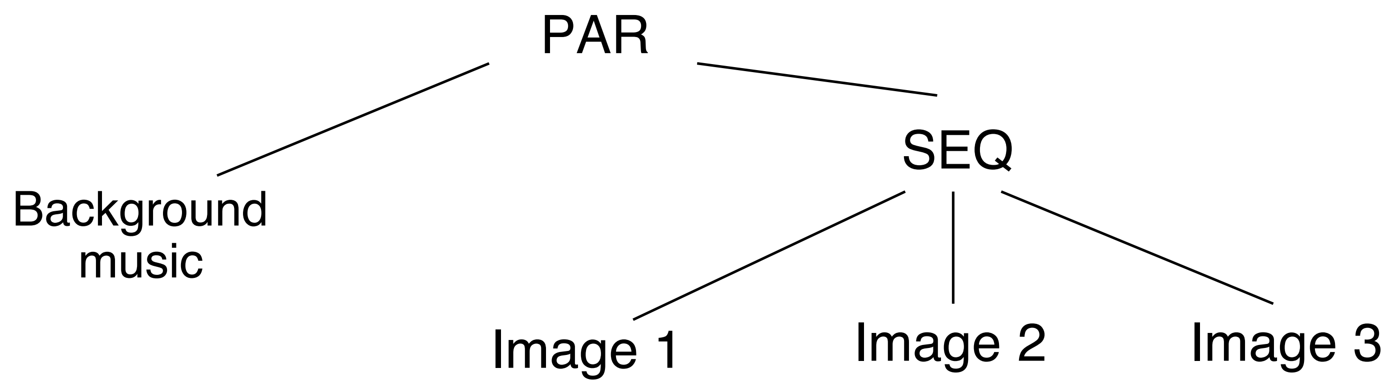
```
<smil xmlns="http://www.w3.org/2001/SMIL20/Language">
  <head>
      <layout>
         <root-layout width="356" height="356‟
                   backgroundColor="black"/>
         <region id="imgReg" width="256" height="256‟
                   left="50" top="50"/>
      </layout>
  </head>
  <body>
      <seq>
         <img region="imgReg" src="tiger.jpg" dur="4s"/>
         <img region="imgReg" src="elephant.jpg" dur="4s"/>
         <img region="imgReg" src="butterfly.jpg" dur="4s"/>
       </seq>
  </body>
</smil>
```

Spatial Structure (Layout)

Temporal Structure (Execution)

## *Concept:* Time Container

- A *time container* groups together media elements ("children") which have a common rule for their synchronization when the group is presented.
  - Analogous to graphical containers (e.g. panels)
  - Synchronization is analogous to spatial layout managers
- Typical time container types (synchronization types)
  - Sequential (one child after the other)
  - Parallel (all children in parallel)
  - Exclusive (one child at a time, but no defined order)
- Time containers can be nested into a hierarchical structure



## Examples for Exclusive Time Containers

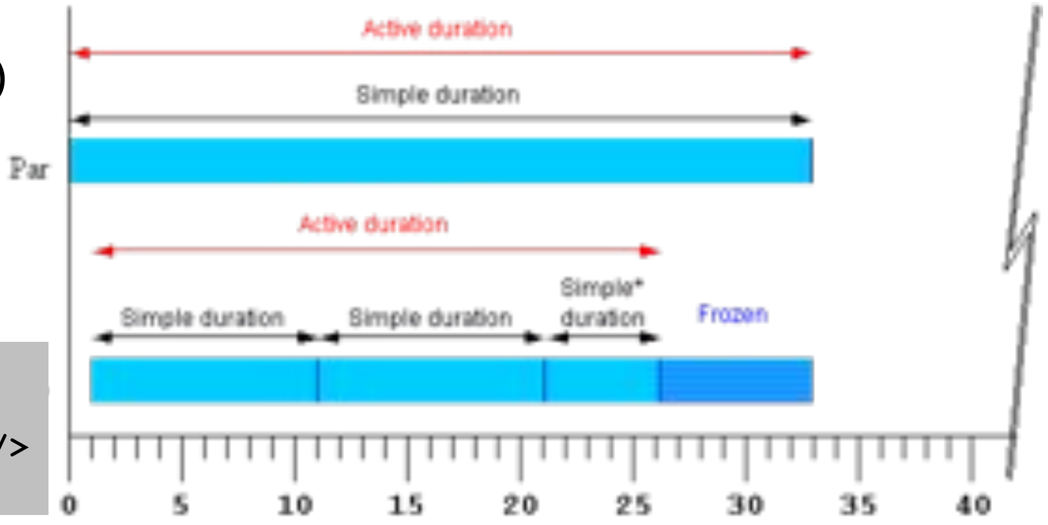
- Interactive playlist:
  - One song out of a collection of songs is played at a time
- Audio descriptions for visually impaired users:
  - Video is suspended and audio description is played
  - After audio description has ended, video resumes
- Interactive video sub-titles:
  - Multiple language versions of sub-titles are available
  - Only one language version is shown at a time

## *Concept:* Timing Behaviour of Media Elements

- A media element has
  - a *begin* (triggered by some event)
  - a *simple duration*
  - an *active duration:* simple duration modified by *repeat* specification
- Simple duration of a time container:
  abstracts from the timing behaviour of its children.
- An element can be
  - *inactive*, when it is not presented
  - *active*, when it is presented
    (has begun its active duration)
- *Fill* at end of active duration:
  - either *removed*
  - or *frozen*

```
<par begin="0s" dur="33s">
   <video begin="1s" dur="10s"
     repeatCount="2.5" fill="freeze" .../>
</par>
```

From SMIL 3.0 specification



## Interactivity in Document-Based Platforms

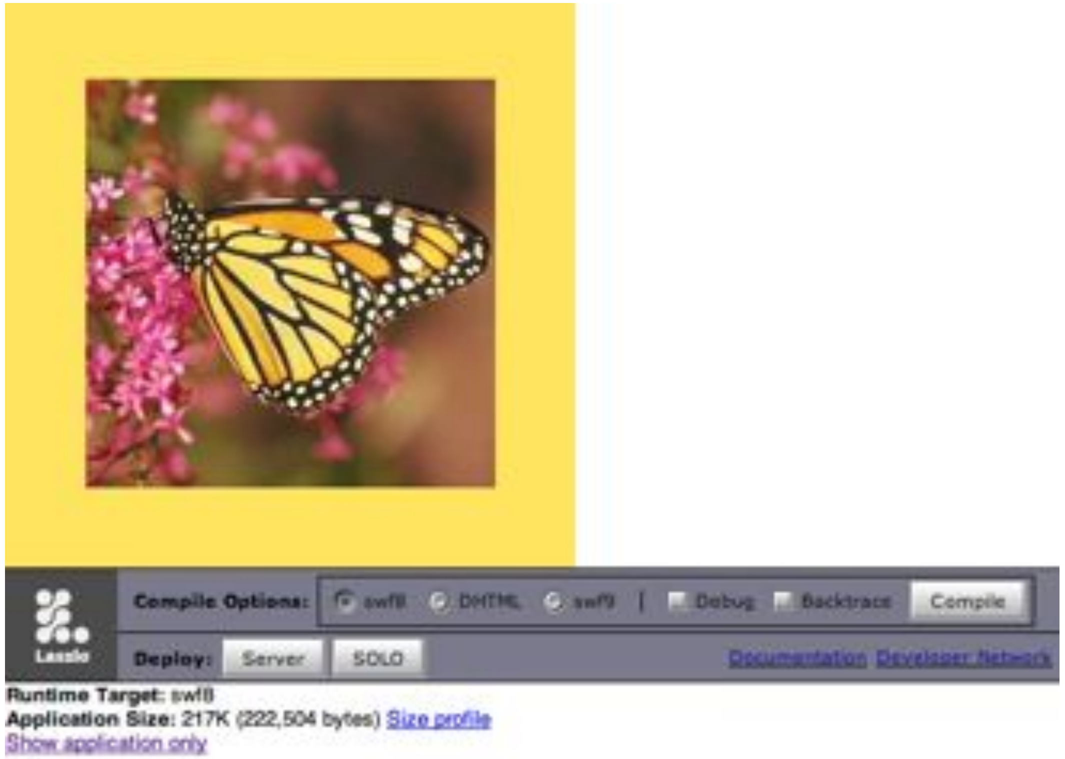
- Simple forms of interactivity are easily supported:
  - Sequential presentation
  - Activation of elements triggered by events (reactive)
- Higher degrees of interactivity (reactive, proactive, directive):
  - Require integration of program/script code

- SMIL:
  - Web scripting languages (e.g. JavaScript, ECMAScript)
  - Interfaces to SMIL provided via DOM

```
interface ElementTimeControl {
  void               beginElement();
  void               beginElementAt(in float offset));
  void               endElement();
  void               endElementAt(in float offset);
};
```

# OpenLaszlo - Idea and History

- Laszlo Systems, California (www.laszlosystems.com):
  - Laszlo Presentation Server
  - Since 2004 free software (common public license)
- XML-based description of multimedia presentation
  - Language LZX
- Laszlo Server software:
  - Accesses document files
  - Uses Flash or JavaScript for interactivity in browsers
- To present an OpenLaszlo document:
  - Start server software
  - Put LZX file into server directory
  - Access server address with appropriate path


Compile Options: swf8 DHTML swf9 | Debug Backtrace Compile
Deploy: Server SOLO
Documentation Developer Network
Laszlo
Runtime Target: swf8
Application Size: 217K (222,504 bytes) Size profile
Show application only

## Slideshow as OpenLaszlo Doc.: Timed Version

```
<canvas bgcolor="#FFE45F" width="356" height="356">
  <resource src="pics/tiger.jpg" name="img1"/>
  <resource src="pics/elephant.jpg" name="img2"/>
  <resource src="pics/jbeans.jpg" name="img3"/>
  <resource src="pics/peppers.jpg" name="img4"/>
  <resource src="pics/butterfly.jpg" name="img5"/>

  <view name="slide" x="50" y="50" resource="img1‟
      oninit="canvas.changeSlides()"/>

  <method name="changeSlides">
      lz.Timer.addTimer(new LzDelegate(this, "change1"), 4000);
      lz.Timer.addTimer(new LzDelegate(this, "change2"), 8000);
      lz.Timer.addTimer(new LzDelegate(this, "change3"), 12000);
      lz.Timer.addTimer(new LzDelegate(this, "change4"), 16000);
  </method>

  <method name="change1">
      slide.setAttribute("resource","img2");
  </method>
  <method name="change2">
      slide.setAttribute("resource","img3");
  </method> ...

</canvas>
```

## Slideshow as OpenLaszlo Doc.: Interactive Version

```
<canvas bgcolor="#FFE45F" width="356" height="356">
   <resource src="pics/tiger.jpg" name="img1"/>
   ...
   <resource src="pics/butterfly.jpg" name="img5"/>

   <script>
        slideindex = 0;
        slides = new Array("img1", "img2", "img3", "img4", "img5");
   </script>

   <view name="slide" x="50" y="50" resource="img1">
        <handler name="oninit">
                lz.Focus.setFocus(this);
        </handler>
        <handler name="onkeydown" args="akeycode">
                if (akeycode==37) {
                        if (slideindex &gt; 0) {
                                slideindex -=1
                        }
                }
                if (akeycode==39)  {
                        if (slideindex &lt; slides.length-1) {
                                slideindex +=1
                        }
                }
                slide.setAttribute("resource",slides[slideindex]);
        </handler>
   </view>
</canvas>
```

## Observations on Document-Based Multimedia Authoring

- Concepts from multimedia frameworks appear in multimedia document languages as well:
  - Event handlers
  - Object-oriented program structure (OpenLaszlo)
- Combination of document syntax and program syntax is problematic
  - Embedding JavaScript into XML
- There is a trade-off between document syntax and program syntax
  - Document syntax (XML): Structure, static elements, views
  - Script code: Dynamics, handlers, “back end” functions

# 1 Development Platforms for Multimedia Programming

Literature:

http://javafx.com

## JavaFX - Idea and History

javafx.com: JavaFX is an expressive rich client platform for creating and delivering rich Internet experiences across all screens of your life. JavaFX 1.0 was released on December 4, 2008. As of February 1st, 2009, there had already been over 100,000 downloads of the tools and SDK. Today, JavaFX is available on over 50 million desktops.

- Chris Oliver, 2006 (?): “Form follows function” (F3)
  - Working for company “SeeBeyond”, but personal project
- Acquisition of SeeBeyond by Sun, 2005
  - F3 is not in the center of interest, apparently
  - First announcement of JavaFX (ex F3) May 2007 (JavaOne conference)
- Builds on Java runtime environment:
  - Common programming model for multimedia applications across many platforms, including mobile devices
- Builds on JavaScript language (not Java!)
- Is sometimes understood as Sun’s (now Oracle’s?) response to Flash and Silverlight technologies
- Components: SDK (compiler, runtime, libraries), NetBeans IDE tool, “Production suite” (plugins and converters for media software/elements)

## Slideshow in JavaFX - Timed Version (1)

```
package javafxslideshow0;

import javafx.animation.KeyFrame;
import javafx.animation.Timeline;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.scene.paint.Color;
import javafx.scene.Scene;
import javafx.stage.Stage;

var imageArray = [
        Image {
          url: "{__DIR__}pics/tiger.jpg"
          backgroundLoading: true
        },
        Image {
          url: "{__DIR__}pics/elephant.jpg"
          backgroundLoading: true
        },
        …
     ];
```

*(Contd.)*

## Slideshow in JavaFX - Timed Version (2)

*(Contd.)*

```
var slideIndex: Integer;

def timeline : Timeline =
    Timeline {
        repeatCount: 1
        keyFrames : [
            KeyFrame {
                time : 0s
                values : [
                    slideIndex => 0
                ]
            },
            KeyFrame {
                time : 4s
                values : [
                    slideIndex => 1
                ]
            },
          …
        ]
    }
(Contd.)
```

Var: Variable
Def: Constant

## Slideshow in JavaFX - Timed Version (3)

*(Contd.)*

```
def stage: Stage = Stage {
    title: "Slide Show"
    resizable: false
    scene: Scene {
        width: 356
        height: 356
        fill: Color.rgb(255,228,95)
        content: [
            ImageView {
                x: 50
                y: 50
                image: bind imageArray[slideIndex]
                }
        ]
    }
}

function run(args : String[]) {
    timeline.play();
}
```

## *Language Concept:* Object Literals

- Large parts of multimedia applications are hierarchical object structures:
  - Layouts: Nested space containers
  - Timing behaviour: Nested time containers
  - Scene graphs
- In traditional object-oriented languages, objects are constructed dynamically using imperative statements:

```
var stage: Stage = new Stage();
stage.title = "Slide Show‟;
```

  - Constructor syntax is limited in expressivity
- Powerful syntax for *static* definition of *nested* object structures is helpful:

```
def stage: Stage = Stage {
    title: "Slide Show"
    scene: Scene {
        width: 356
        height: 356
        content: [
            ImageView {
                x: 50
                y: 50
                image: bind imageArray[slideIndex]
        ]
     }
}
```

## Crossover Document – Program

JavaFX

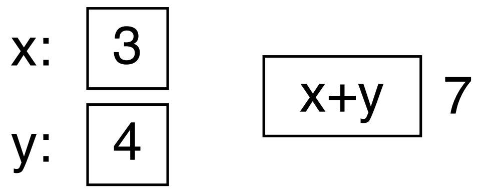
```
def stage: Stage = Stage {
    title: "Slide Show"
    scene: Scene {
        width: 356
        height: 356
        content: [
            ImageView {
                x: 50
                y: 50
                image: bind imageArray[slideIndex]
        ]
     }
}
```

(Fictive!) XML Equivalent

```
<stage>
    <title>Slide Show</title>
    <scene>
        <width>356</width>
        <height>356</height>
        <content>
            <ImageView>
                <x>50</x>
                <y>50</y>
                <image>bind imageArray[slideIndex] </image>
        </content>
    </scene>
</stage>
```

## *Language Concept:* Expression Binding

- Expressions in program are evaluated at different times
  - Expressions containing constants only:
    - » Evaluated statically (before execution)
  - Expressions containing variables:
    - » Evaluated dynamically when expression value is needed
  - Expressions containing variable *bindings*:
    - » Evaluated dynamically whenever value of a contained variable changes



Value of x changes (e.g. to 4)

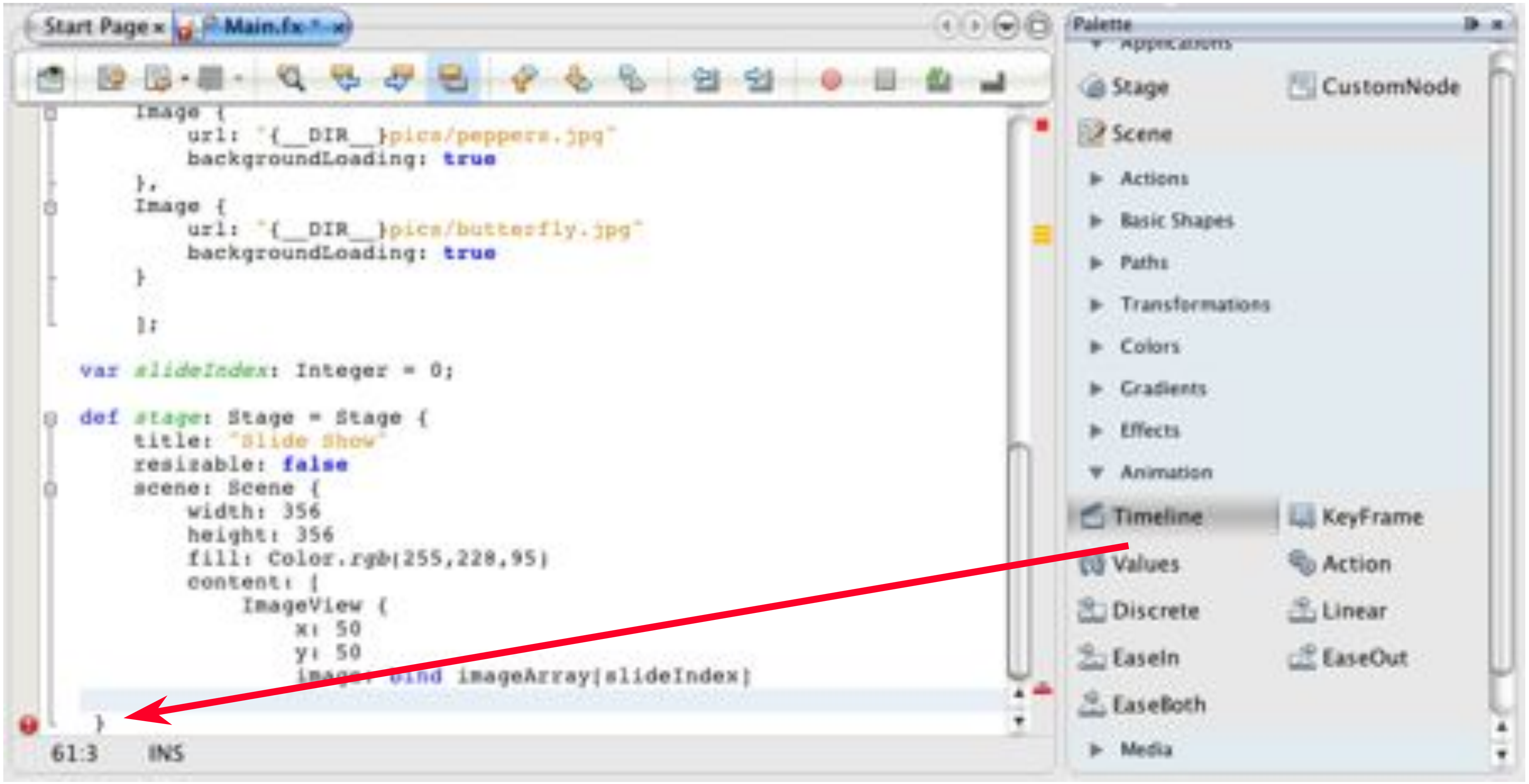
With expression binding:
x+y is *re-evaluated* (to 8)

- Expression binding realizes built-in *Observer* mechanism

## *Concept:* TimeLine and Key Frames

- From JavaFX documentation:
  “`Timeline` provides the capability to update the property values along the progression of time.“
- Some variables take new values at certain points in time
  - Example slide index in slide show
- Timeline:
  - Defines a sequence of key frames
  - Each key frame defines a certain configuration of values
  - Each key frame is associated to a point in time
- Timelines are very suitable to express animations
  - Interpolation of values (see later)
- A timeline is a sequential time container.

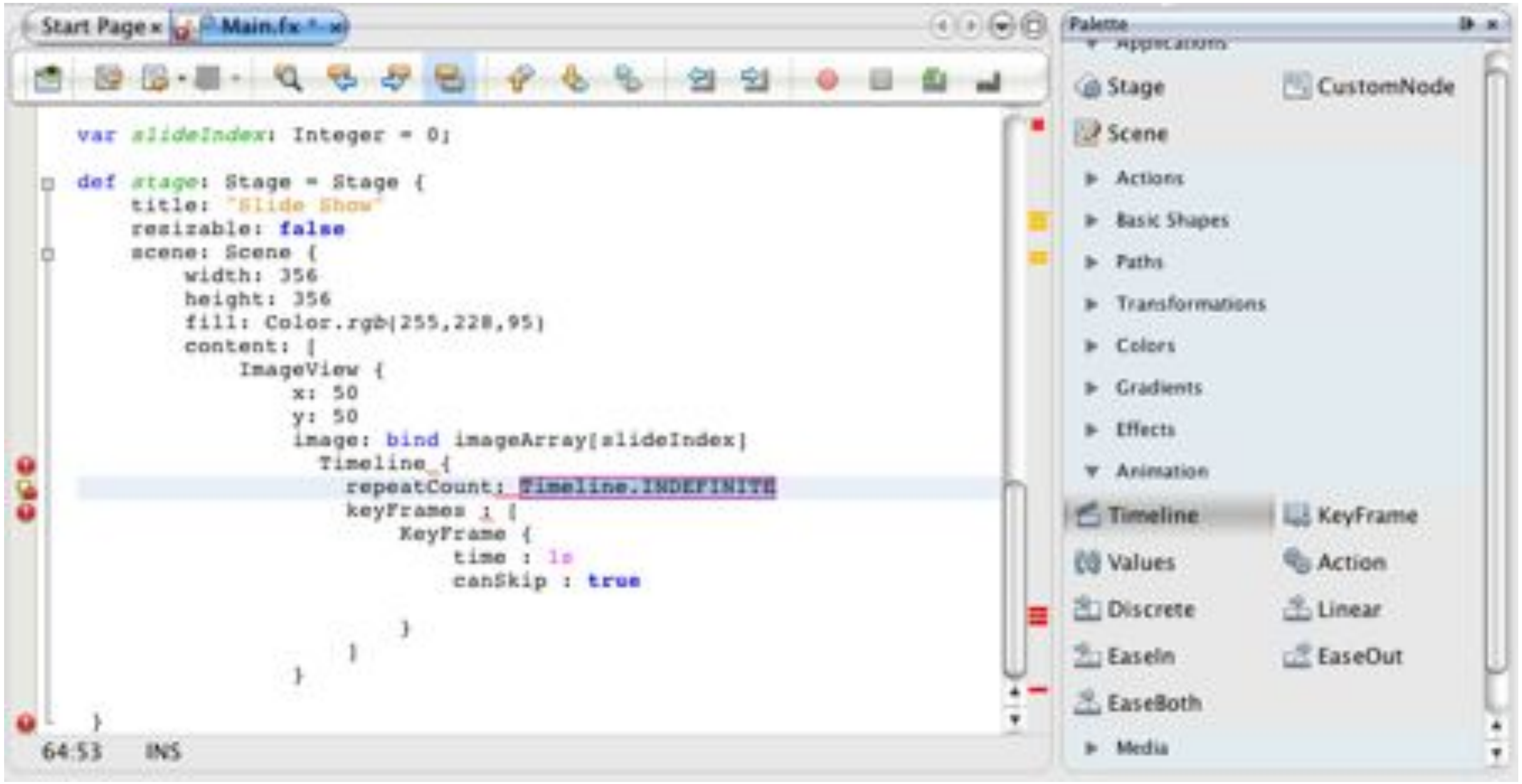
# *Development Tool Concept:* Code Palette (1)



```
    Image {
        url: "{__DIR__}pics/peppers.jpg"
        backgroundLoading: true
    },
    Image {
        url: "{__DIR__}pics/butterfly.jpg"
        backgroundLoading: true
    }

    ];

var slideIndex: Integer = 0;

def stage: Stage = Stage {
    title: "Slide Show"
    resizable: false
    scene: Scene {
        width: 356
        height: 356
        fill: Color.rgb(255,228,95)
        content: [
            ImageView {
                x: 50
                y: 50
                image: bind imageArray[slideIndex]

}
```

JavaFX with NetBeans IDE

## *Development Tool Concept:* Code Palette (2)



```
var slideIndex: Integer = 0;

def stage: Stage = Stage {
    title: "Slide Show"
    resizable: false
    scene: Scene {
        width: 356
        height: 356
        fill: Color.rgb(255,228,95)
        content: [
            ImageView {
                x: 50
                y: 50
                image: bind imageArray[slideIndex]
              Timeline {
                repeatCount: Timeline.INDEFINITE
                keyFrames : [
                    KeyFrame {
                        time : 1s
                        canSkip : true

                    }
                ]
            }

}
```

JavaFX with NetBeans IDE

Code is inserted independently of semantics/type correctness...

## Slideshow in JavaFX - Interactive Version

```
package javafxslideshow1;
…
var imageArray = […];

var slideIndex: Integer = 0;

def stage: Stage = Stage {
  …
  content: [
    ImageView { …
      image: bind imageArray[slideIndex]
      onKeyPressed: function( e: KeyEvent ):Void {
        if ((e.code == KeyCode.VK_LEFT) and (slideIndex > 0)){
          slideIndex -=1
        };
        if ((e.code == KeyCode.VK_RIGHT) and
            (slideIndex+1 < sizeof imageArray)){
          slideIndex +=1
        }
      }
    }
  ]
}
```

## Processing - History and Idea

"Processing is an open source programming language and environment for people who want to program images, animation, and interactions. It is used by students, artists, designers, researchers, and hobbyists for learning, prototyping, and production."

- Developed by Ben Fry and Casey Reas
  - MIT Media Lab
  - Aesthetics and Computation Group
- Free software
  - Made for simple development processes (simple tool)
  - Tries to avoid many complexities of multimedia programming
- Based on Java
  - Runtime errors are often Java errors

## Slideshow in Processing

```
PImage img1, img2, img3, img4, img5;
int savedTime;

void setup() {
  size(365,365);
  background(255,228,95);
  img1 = loadImage("pics/tiger.jpg");
  …
  img5 = loadImage("pics/butterfly.jpg");
}

void draw() {
  image(img1,50,50);
  int passedTime = millis() - savedTime;
  if (passedTime > 4000) {
    image(img2,50,50);
   }
  …
  passedTime = millis() - savedTime;
  if (passedTime > 16000) {
    image(img5,50,50);
   }
}
```

## Observations on Multimedia Scripting Languages

- Integration of nested semi-structured data into program code works better than the other way round.
- Event handling mechanisms can be built into scripting language.
- It makes sense to extend the core of a language for dealing well with events or structured data.
- Development environments can use specialized graphical metaphors for structuring program code of multimedia applications.

# 1 Development Platforms for Multimedia Programming

Literature:

(Abundant literature on Flash...)

## Flash: History

- Jonathan Gay:
  - Software developer for *Silicon Beach Software* (starting in high school...)
  - Involved in various ground-breaking Macintosh applications:
    Airborne!, DarkCastle (1987), SuperPaint II, IntelliDraw (drawings with behaviour)
- 1993: Foundation of *FutureWave Software*
  - Goal: Develop sketching software (*SmartSketch*) for the new “pen computer” and the PenPoint operating system from the company GO
  - GO (and later EO) computers failed
- 1995-96: *SmartSketch* becomes *FutureSplash Animator*
  - Ported to Macintosh and Windows
  - Extended with 2D animation features
  - From the beginning targeted at delivery over the Web
  - Well accepted by important customers (e.g. Microsoft, Disney)
- 1996: FutureWave bought by Macromedia
  - FutureWave Splash becomes *Macromedia Flash 1.0*
- 2005:
  - Adobe acquires Macromedia and its product portfolio

EO

## Flash vs. Director

- Director:
  - 10 years older than Flash
  - Designed for development of interactive CD-ROMs
  - Integrated programming language *Lingo*
  - Oriented towards bitmap graphics
  - Starting from Version 7: integration of Flash content
- Flash:
  - Designed for content delivery over the Internet *(streaming)*
  - Oriented towards vector graphics
  - Early versions (up to version 3) extremely simple in their interaction possibilities, later versions with increasing support for scripting
  - Early usage of Flash heavily criticized for bad usability
    - » Flash intros, breaking with Web paradigms (Jakob Nielsen 2000)
  - Current usage trends:
    - » rich media content (e.g. video), advanced interactive Web sites

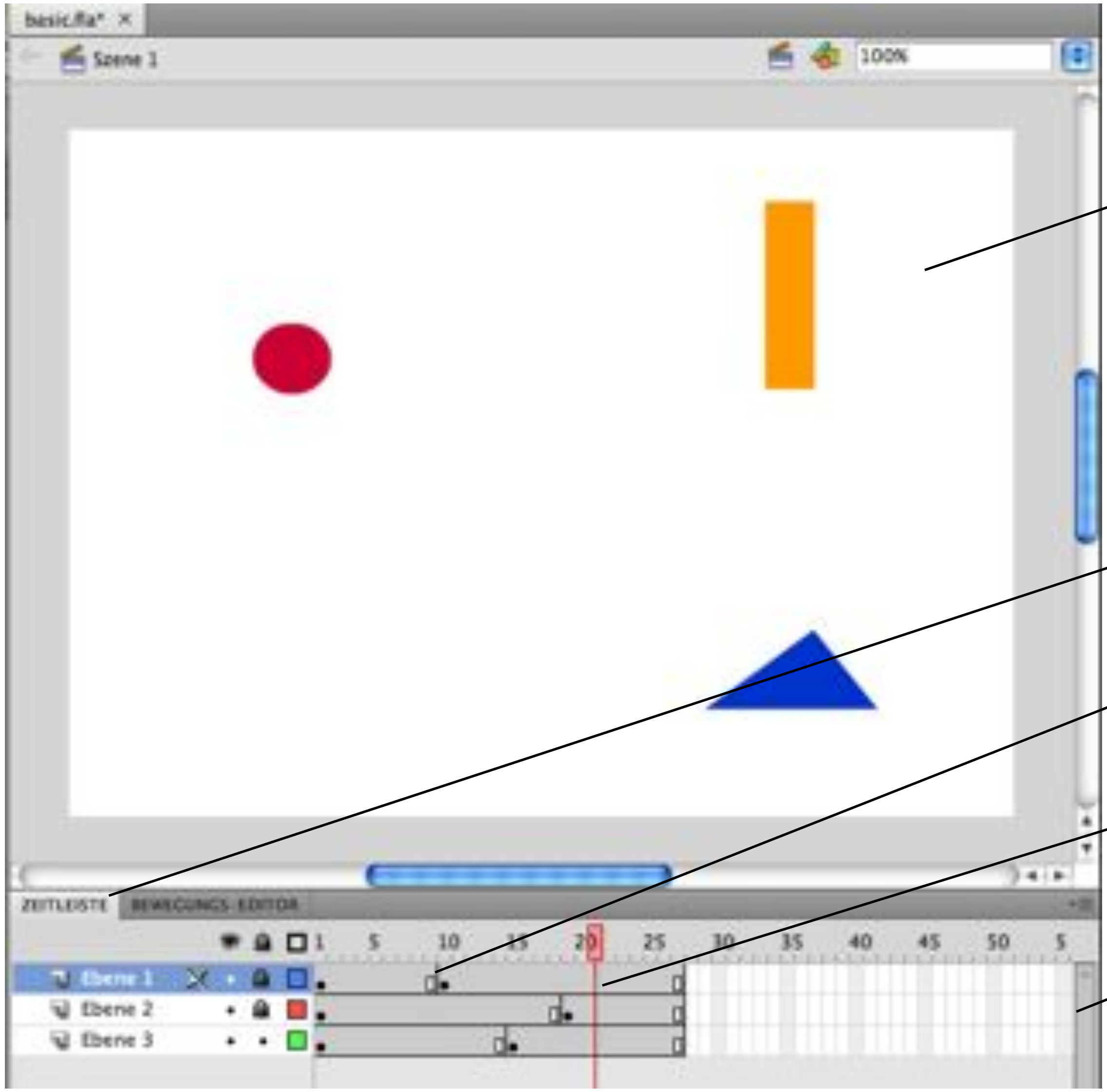
## Shockwave Plugins

- Shockwave:
  - General name for Web plugins playing Macromedia content
- *Shockwave for Director:*
  - Often simply called *Shockwave* plugin!
  - Plays content created with Director (Shockwave Movies)
  - File types: .dcr, .dir, .dxr
  - MIME type:  
    application/x-director
- *Shockwave Flash*
  - Often called *Flash* plugin, different from Shockwave plugin!
  - Plays content in SWF (Shockwave Flash) format
  - File types: .swf, .spl (from FutureSplash)
  - MIME types:  
    application/x-shockwave-flash  
    application/futuresplash

## Shockwave Flash (SWF)

- SWF is often pronounced as “swiff”
- File format for execution-ready presentations
  - Proprietary compiled format of Flash presentations
  - Flash browser penetration over 95%
  - Can be produced by various programs, not only Macromedia Flash
- Specifications of SWF format:
  - Older versions were publicly available, now developer-licensed product
- Players exist for many platforms:
  - PDAs
  - Mobile phones
    - » The pioneer 2003: *i-Mode* system from NTT DoCoMo
    - » 2009: *Flash lite* licensed by many mobile phone manufacturers, 800+ million devices shipped with flash (Adobe statement)
  - Digital music players
  - Set-top boxes, public displays, car infotainment systems, ...
  - Generic Java player applets (older versions only)

## Timeline and Stage



*Stage* shows current frame

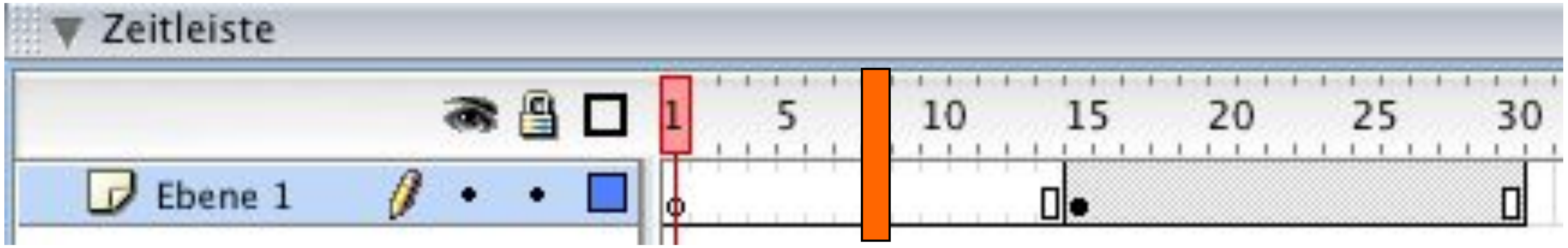
*Timeline*

Frame change

Playback head

current picture consists of three parallel layers

# Timeline Symbols



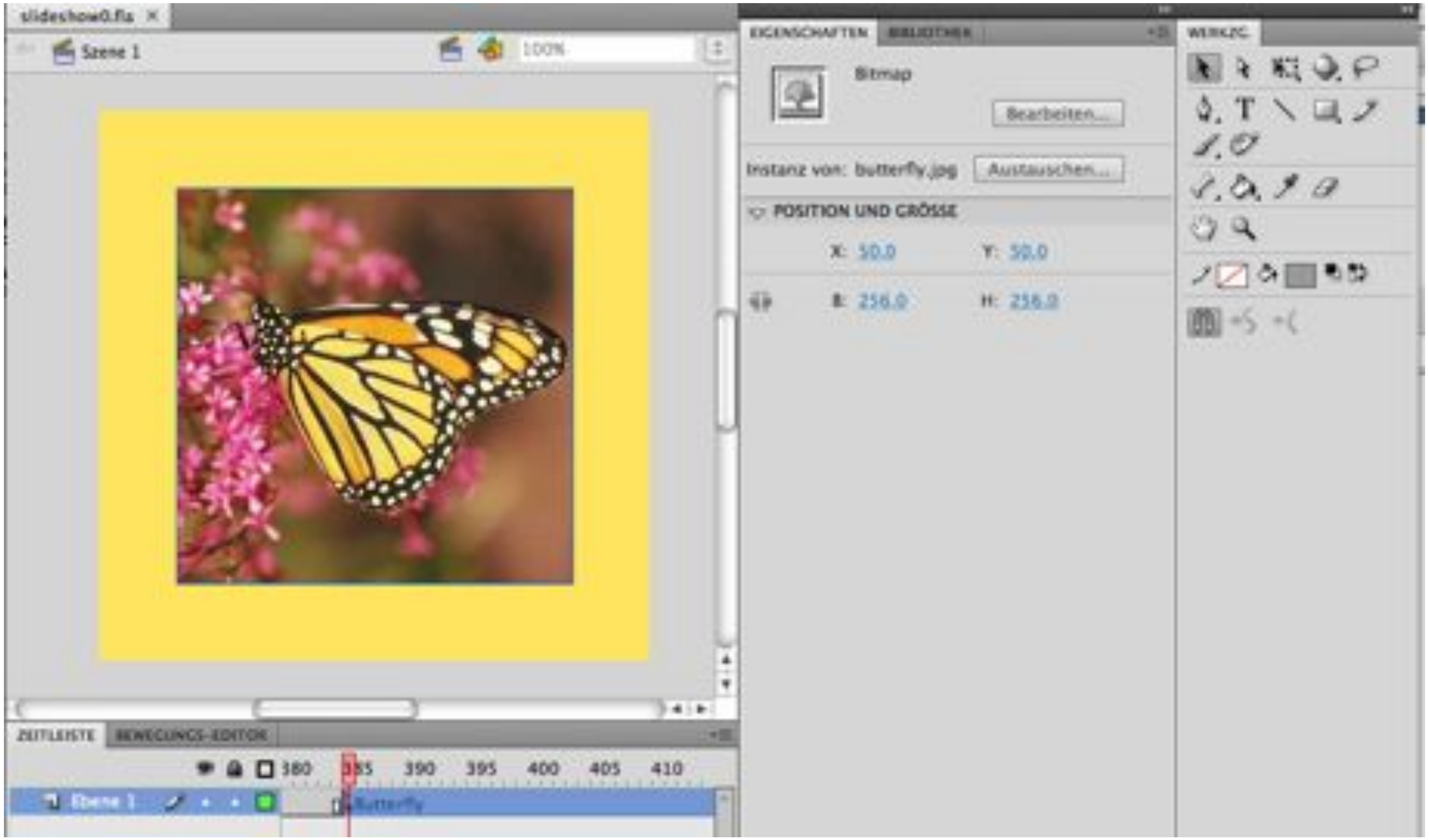
Frame no. 8

- The timeline contains *frames* (Bilder)
- *Key frames (Schlüsselbilder)* are defined explicitly (drawn by hand)
  - Representation in Flash:
    hollow dot = empty key frame
    black dot = key frame with content
- Default treatment of frame sequences: repeat last frame
  - Grey bar: Sequence of identical frames
  - Square: Last frame of a sequence
  - Changes in key frame affect all subsequent frames till next key frame!

## SWF

- The Macromedia Flash file format (SWF) (pronounced “swiff”) delivers vector graphics and animation over the Internet to the Macromedia Flash Player.
  - Pure delivery format
- Design goals:
  - On-screen display
    - Designed for rendering
  - Extensibility
    - Tagged format
  - Network delivery
    - Compact binary format
  - Simplicity
  - Scalability regarding power of hardware
  - Scriptability
    - Stack machine code compatible to “ActionScript” language

## Slide Show in Adobe Flash (CS4)



Bitmap

Bearbeiten...

Instanz von: butterfly.jpg Austauschen...

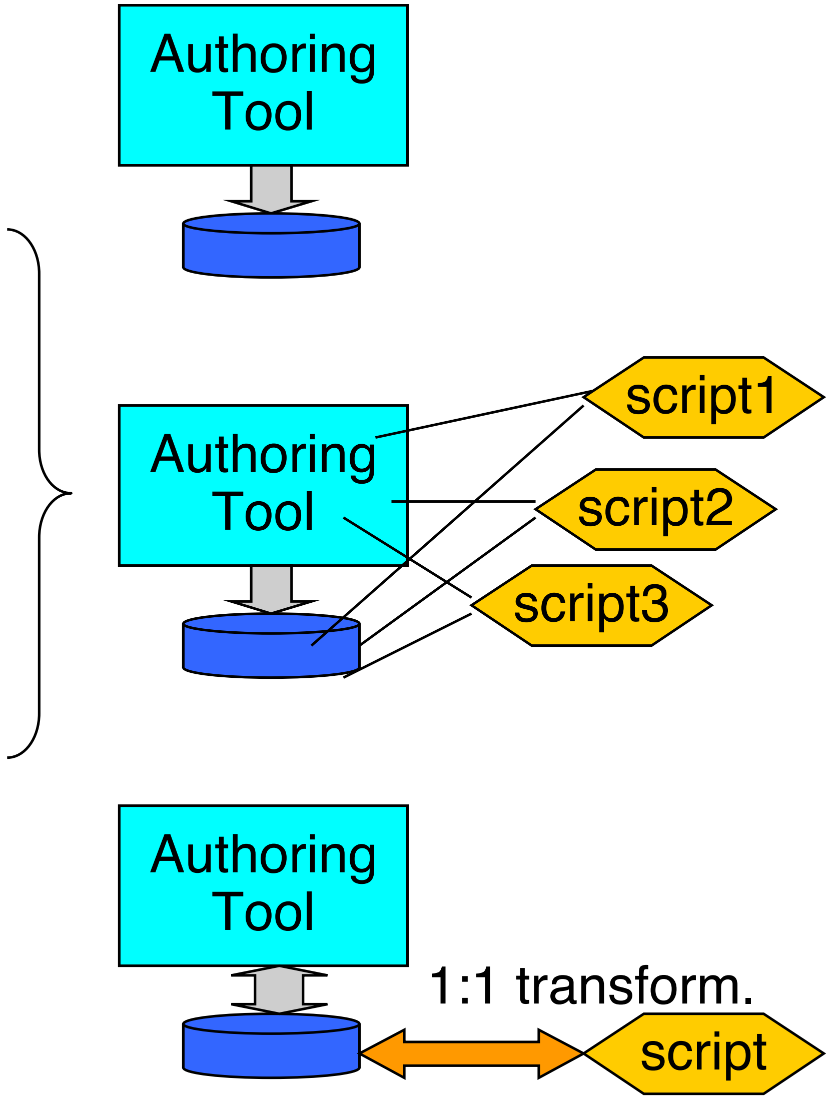
POSITION UND GRÖSSE

X: 50.0 Y: 50.0

B: 256.0 H: 256.0

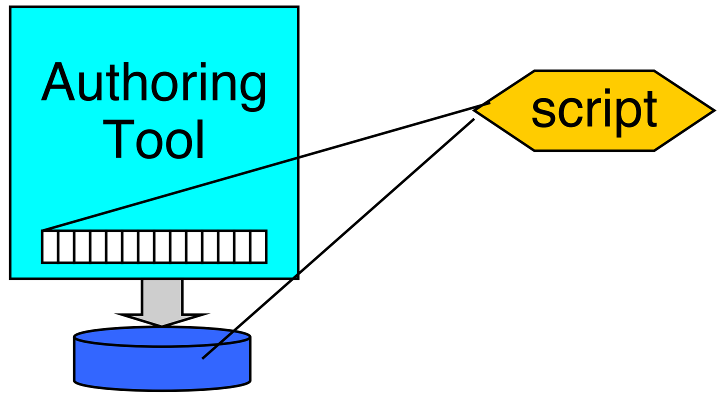
# Scripting Languages for Authoring Tools (1)

- *Script-less authoring:*
  Purely graphical authoring tool
  Scripts/programming avoided
- *Integrated scripting:*
  Scripts added at various places
  in the authoring environment
  to enhance expressiveness;
  scripts are *context-dependent*
- *Separated scripting:*
  Separate script files in addition to the
  file produced with the authoring tool;
  scripts are *self-contained*
- *Script-based development:*
  Authoring tool as a comfortable view
  onto a program (script);
  Whole application can be written as
  a script in a formal language

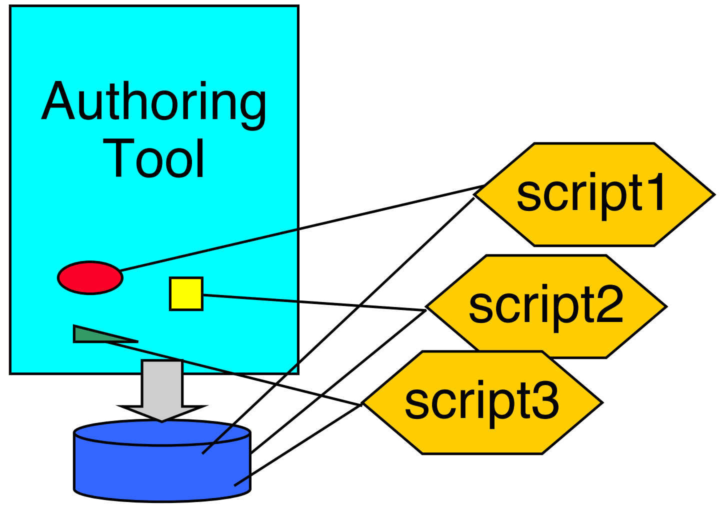


## Scripting Languages for Authoring Tools (2)

- *Control-flow based scripting:* Scripts called at certain places in (global) timeline of animation



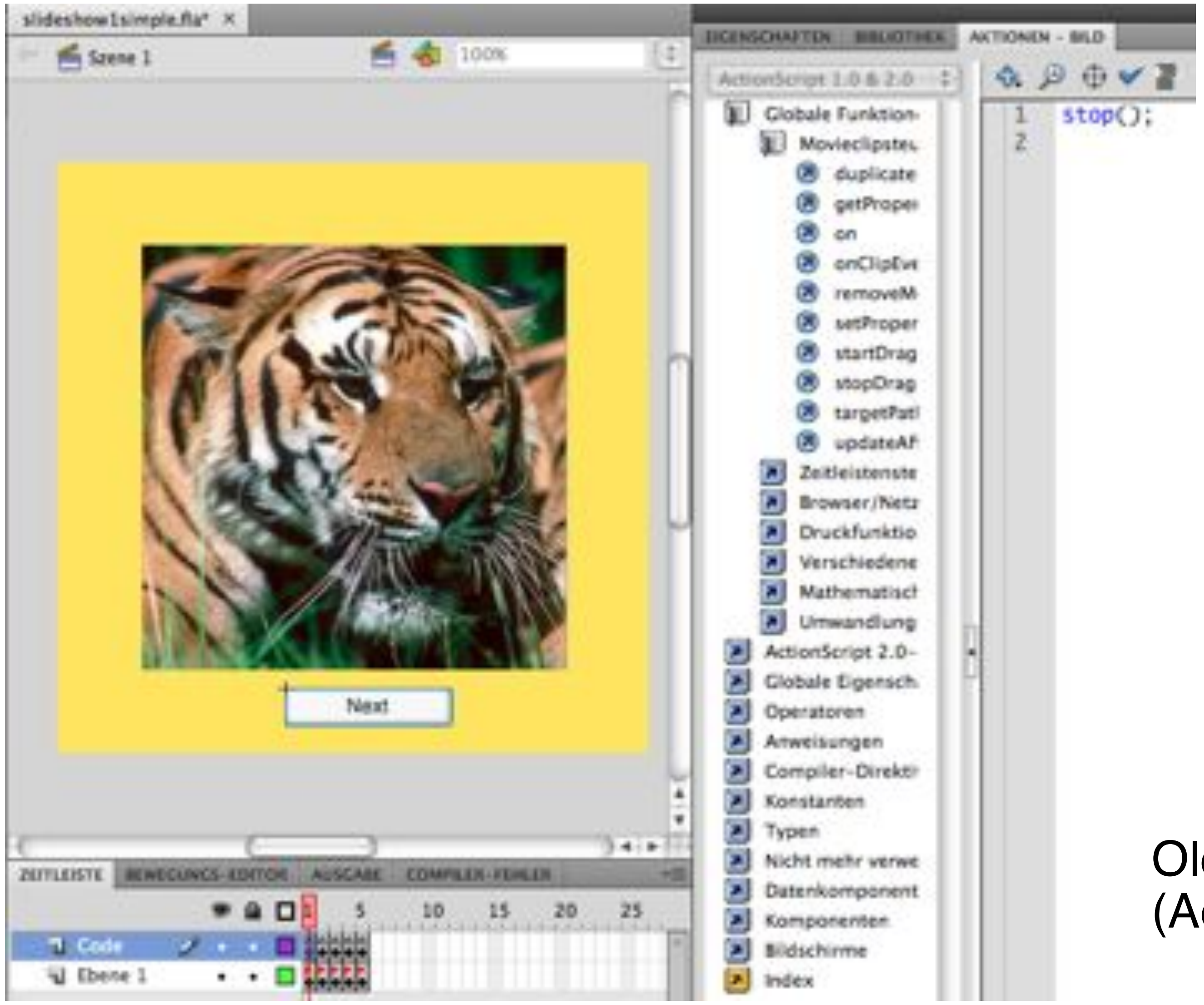
- *Object-based scripting:* Scripts are allocated to individual animation objects and called as *event handlers*



## Example: Scripting in Flash

- Adobe Flash:
  - Graphical authoring system
  - Follows timeline & stage metaphors
  - Extensive graphical authoring of animations
- Scripting in Flash:
  - To be used when graphical authoring is not sufficient
  - E.g. in designing reaction to user input (keyboard, mouse)
- Older versions of Flash:
  - ActionScript 1 & 2 scripting languages
  - Not strictly object-oriented
- Recent versions of Flash (ActionScript 3.0 aka AS3):
  - Enforce object-orientation

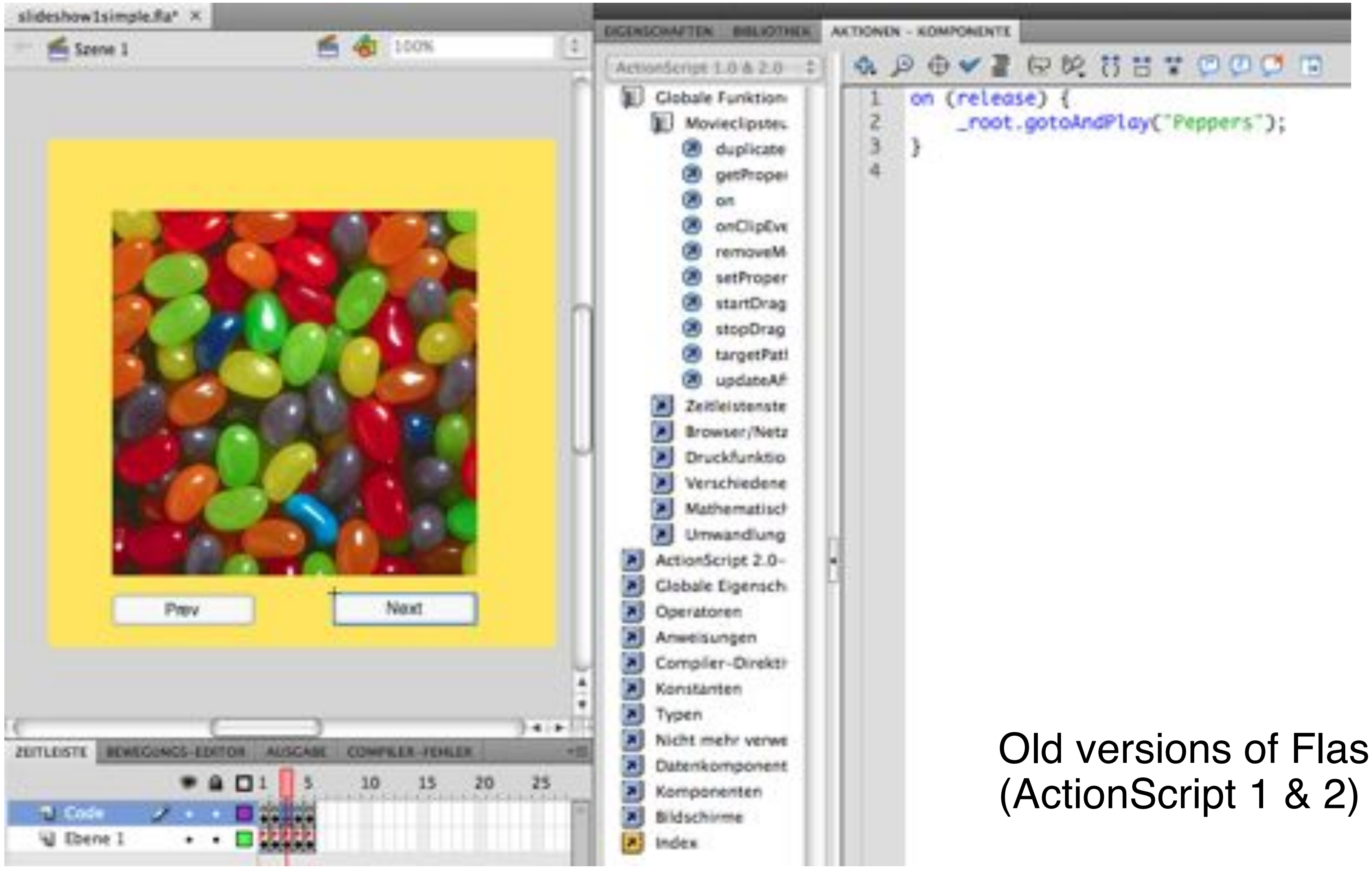
# Integrated Control-Flow Based Scripting



```
stop();
```

Old versions of Flash
(ActionScript 1 & 2)

## Integrated Object-Based Scripting

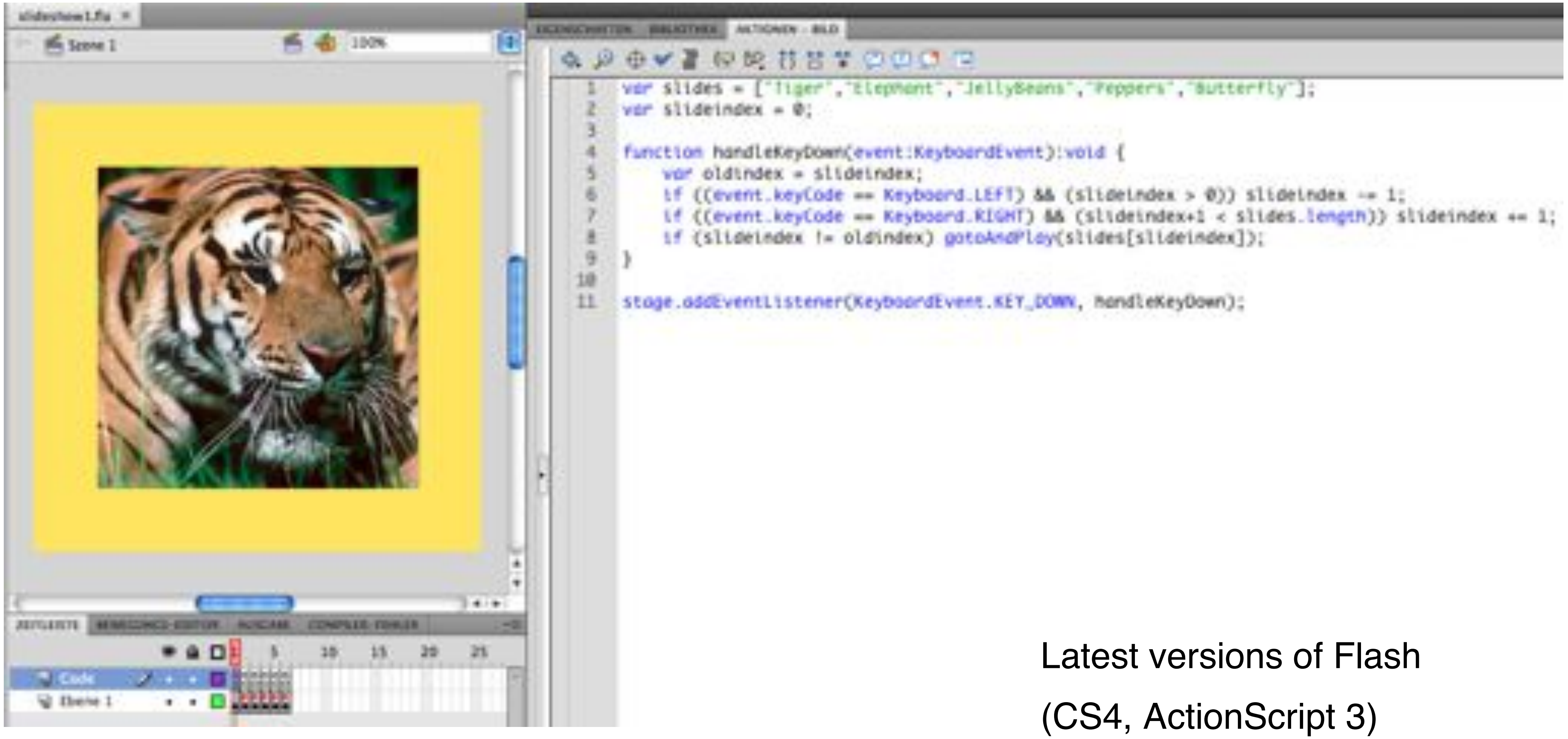

```
on (release) {
    _root.gotoAndPlay("Peppers");
}
```

Old versions of Flash (ActionScript 1 & 2)

## Generic Design

- Simplistic integrated scripting often leads to solutions which are difficult to maintain
  - Core logic not visible clearly
  - Example: Code refers to identifiers of key frames (“Peppers”)
- Generic designs:
  - Try to avoid code duplications
  - Lead to more abstract programs (“programmer style”)
  - There are degrees of genericity!
- Modern versions of Flash enforce more generic programming styles
  - But make it even more difficult for non-programmers…

## More Generic Design (Integrated Object-Based)



```
var slides = ["Tiger","Elephant","JellyBeans","Peppers","Butterfly"];
var slideindex = 0;

function handleKeyDown(event:KeyboardEvent):void {
    var oldindex = slideindex;
    if ((event.keyCode == Keyboard.LEFT) && (slideindex > 0)) slideindex -= 1;
    if ((event.keyCode == Keyboard.RIGHT) && (slideindex+1 < slides.length)) slideindex += 1;
    if (slideindex != oldindex) gotoAndPlay(slides[slideindex]);
}

stage.addEventListener(KeyboardEvent.KEY_DOWN, handleKeyDown);
```

Latest versions of Flash

(CS4, ActionScript 3)

# Object-Orientation: Reusable Highlighting Color Block in AS3

```
class ColorBlock extends MovieClip {

  private var myColor:Color;
  public var myOnRgb:Number;

  public function onLoad() {
      myColor = new Color(this);
  }

  public function onRollOver() {

      myColor.setRGB(myOnRgb);
  }

  public function onRollOut() {

      myColor.setRGB(0xffffff);
  }
}
```

Used built-in technology:

`Color` object controls the color of the movie clip.

Constructor assigns a new color object to the movie clip.

`setRGB` function actually changes the color.

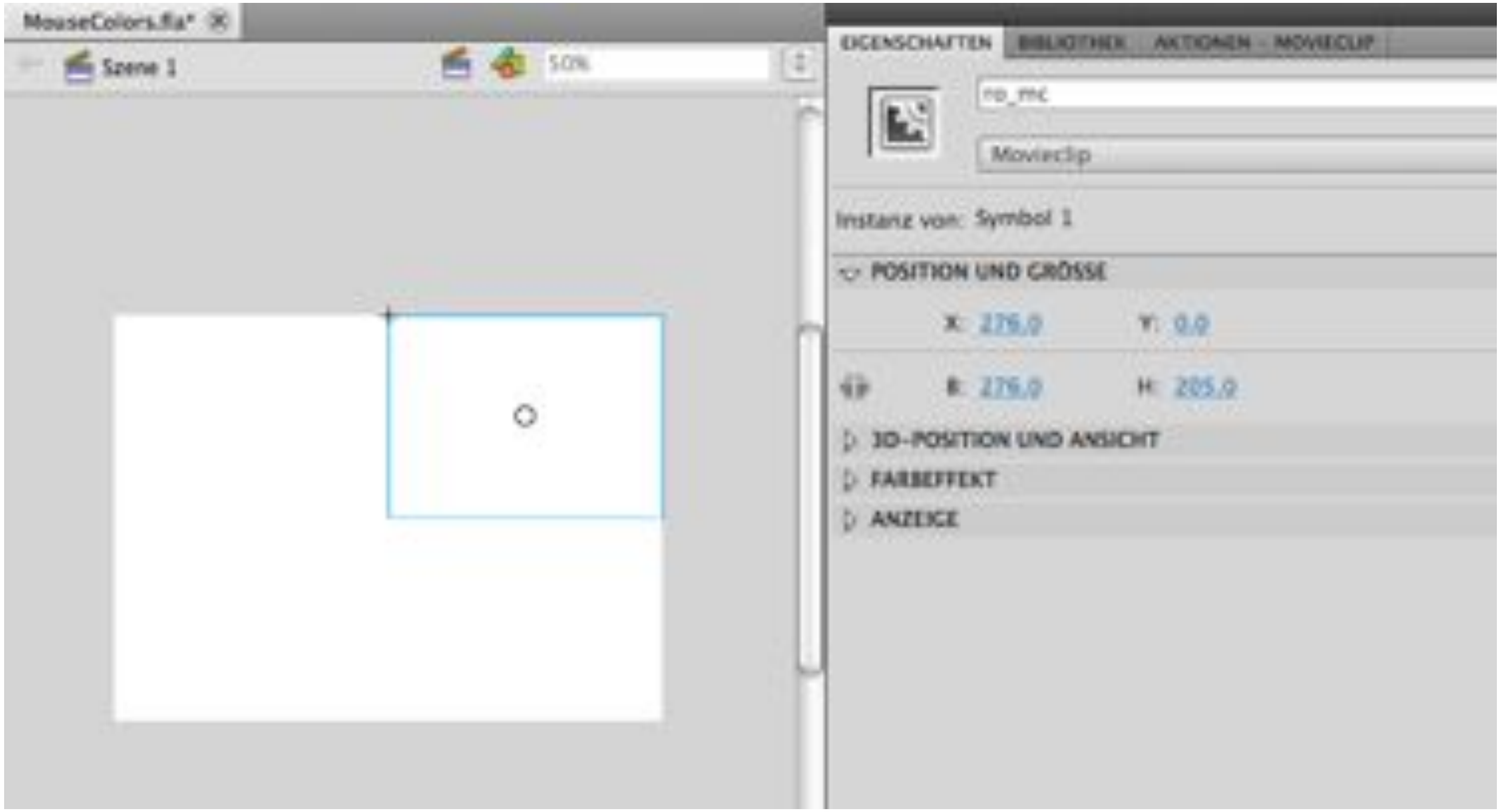
## Creating Instances of the Reusable Symbol

- There is *one* symbol with several instances (example: lo_mc, ro_mc, lu_mc, ru_mc)
- The symbol defines the graphical shape with irrelevant color.
- The instances define their respective “on” color.
- In the graphical authoring tool, the symbol instances are located on the stage.

- Initialisation code:

```
lo_mc.myOnRgb = 0xff0000; //red
ro_mc.myOnRgb = 0x0000ff; //blue
lu_mc.myOnRgb = 0x00ff00; //green
ru_mc.myOnRgb = 0xffff00; //yellow
```

# Graphical Authoring with Object-Oriented Symbols



ro_mc

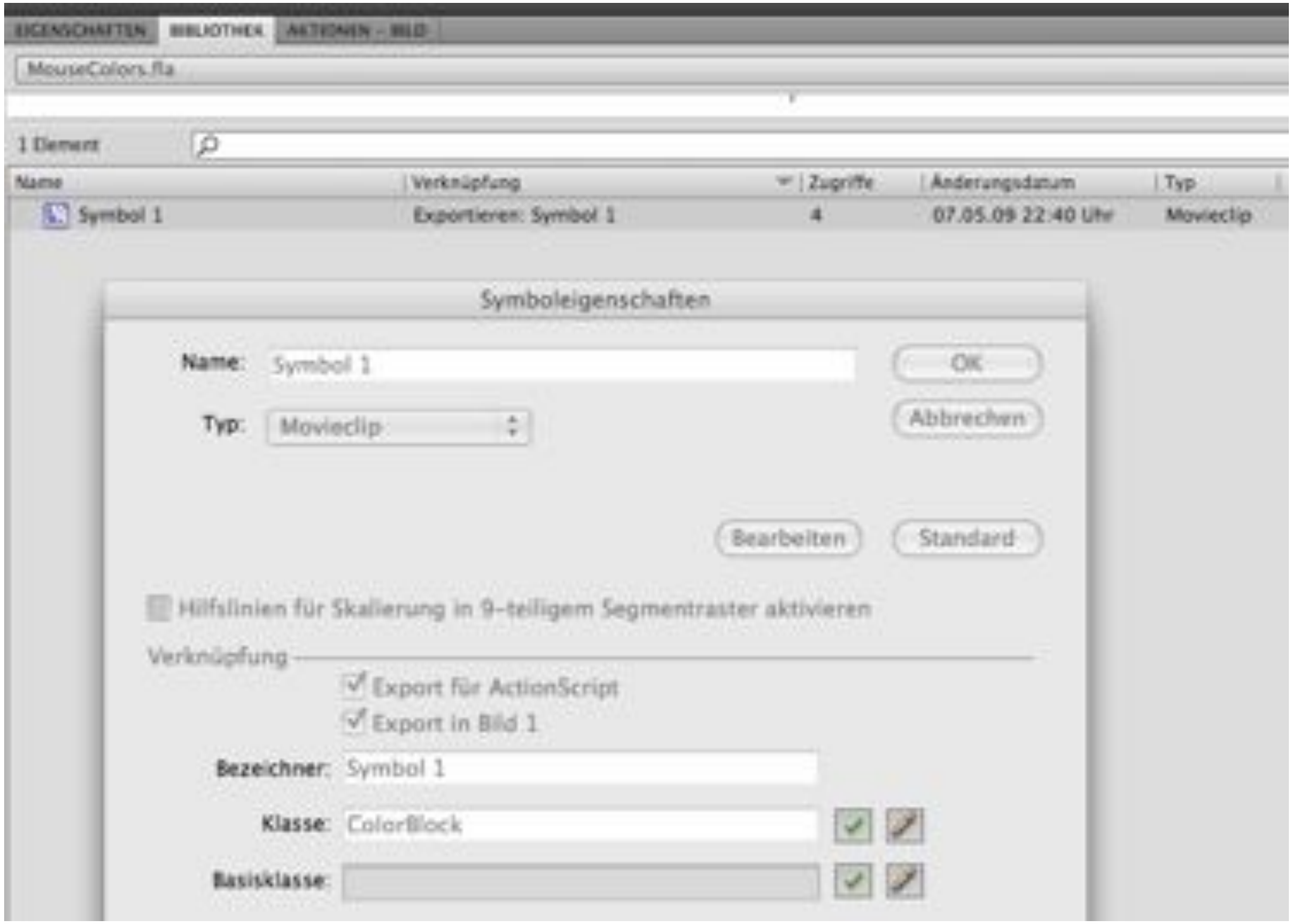
Movieclip

Instanz von: Symbol 1

X: 276,0 Y: 0,0

B: 276,0 H: 205,0

## Connecting Symbol (Movieclip) and Class


EIGENSCHAFTEN | BIBLIOTHEK | AKTIONEN - BILD

MouseColors.fla

1 Element

| Name | Verknüpfung | Zugriffe | Änderungsdatum | Typ |
|---|---|---|---|---|
| Symbol 1 | Exportieren: Symbol 1 | 4 | 07.05.09 22:40 Uhr | Movieclip |

Symboleigenschaften

Name: Symbol 1

Typ: Movieclip

Hilfslinien für Skalierung in 9-teiligem Segmentraster aktivieren

Verknüpfung

[✓] Export für ActionScript

[✓] Export in Bild 1

Bezeichner: Symbol 1

Klasse: ColorBlock

Basisklasse:

## Observations on Multimedia Authoring Tools

- Similar approach to document-based authoring
  - Realizing higher degrees of interactivity is difficult
- Programming (ActionScript in case of Flash) is unavoidable for more complex applications.
- Easy to use for graphical designers
  - Step towards programming is even more difficult then...
- Integration of programming language into graphical design tool is relatively difficult to use.