4 Programming with Images

Literature:

P. Ackermann: Developing Object-Oriented Multimedia Software based on the MET++ Application Framework, dpunkt 1996

B. B. Bederson, J, Grosjean, J. Meyer: Toolkit Design for Interactive Structured Graphics, *IEEE TSE* vol. 30 no. 8, pp. 535-546, 2004

Vector Graphics vs. Bitmap Graphics

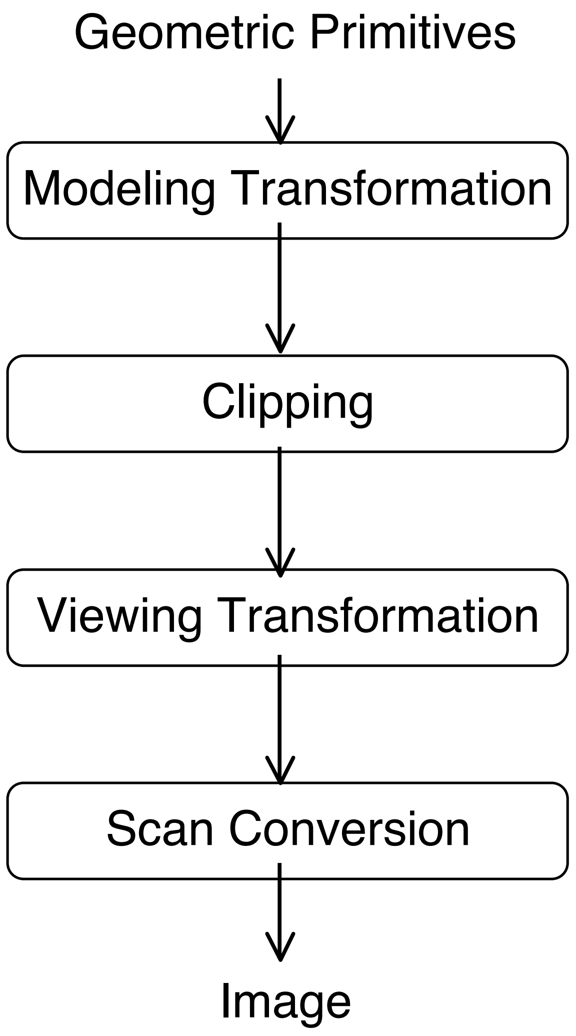
- Vector Graphics
  - Picture synthesized from geometric primitives (points, lines, curves)
  - Easy access through programming interface
  - Can be adapted to current situation (visualization of internal program state)
  - Basis for dynamic graphics (animation)
- Bitmap Graphics
  - Picture pre-recorded, in most cases sampled from analog original
  - Programming interface restricted to manipulations
  - Difficult to adapt
  - Purely static

How to Specify Vector Graphics

```
<svg xmlns="http://www.w3.org/2000/svg" version="1.0"
  viewBox= "0 0 250 100">
    <rect x="10" y="10" width="230" height="80" fill="red"/>
    <circle cx="50" cy="50" fill="white" r="40"/>
    <circle cx="200" cy="50" fill="white" r="40"/>
</svg>
```

- Declarative document:
  - Vector graphics file format, e.g. SVG
  - Vector graphics is ***not*** part of pure media *integration* languages like SMIL!
- Graphical editor:
  - Either to create a graphics file (e.g. Illustrator, Inkscape)
  - Or as integral part of authoring tool (e.g. Flash)
- Program code:
  - Drawing by method/procedure calls (e.g. Java 2D, Python/Pygame)

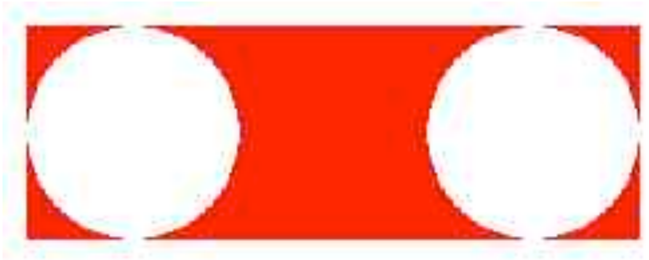
2D Rendering Pipeline



Modeling Transformation	Placing geometric primitives in *world coordinates*	Specified explicitly
Clipping	Restriction to displayed area	Implicit (stage size)
Viewing Transformation	Transformation to screen coordinates	Automatic
Scan Conversion	Transformation to pixel values	Automatic

Adapted from: Funkhouser, Princeton U

Simple Drawing in Python/Pygame



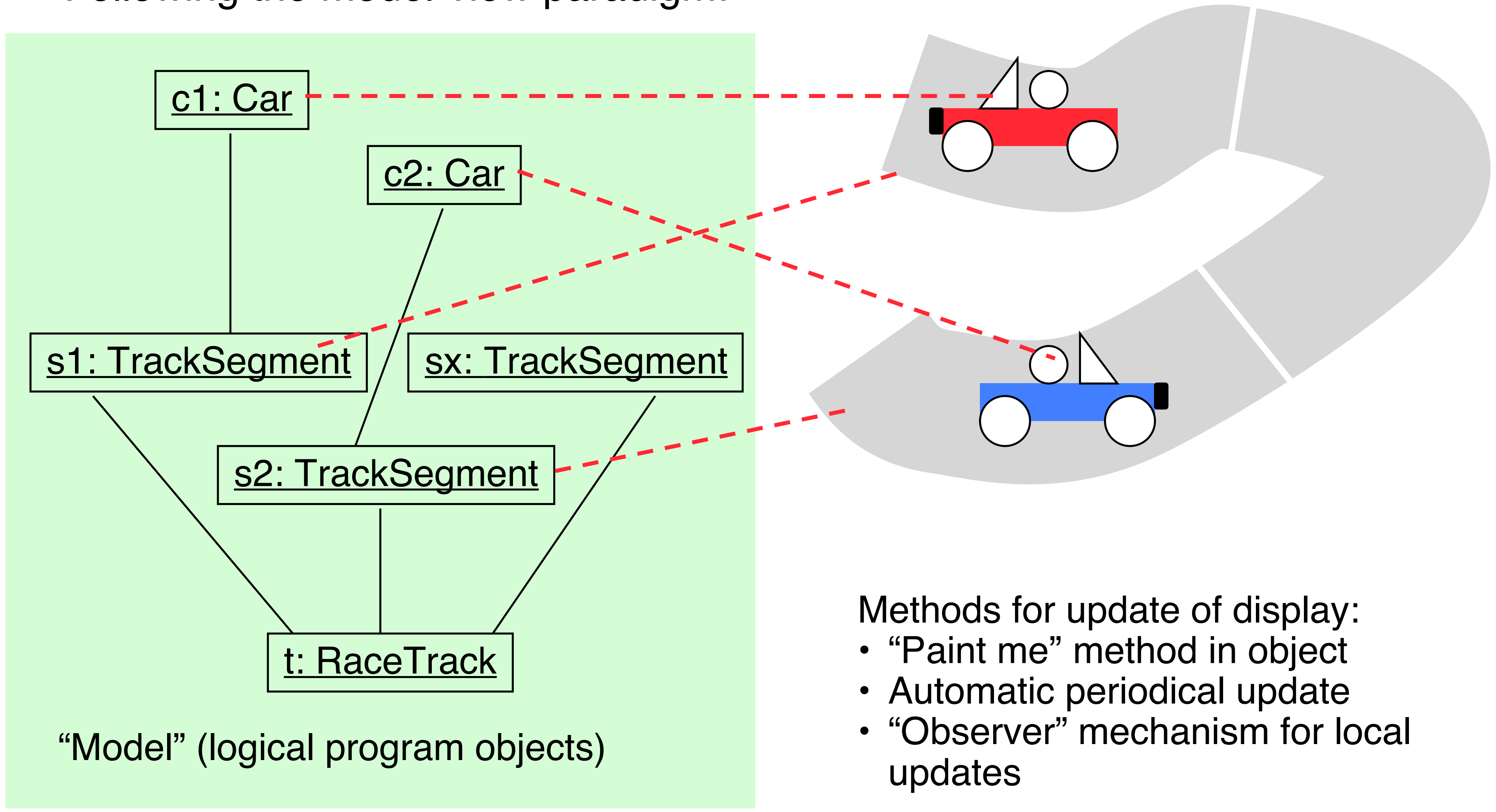
```
screen = pygame.display.set_mode((250,100),0,32)
red = pygame.color.Color("red")
white = pygame.color.Color("white")

pygame.init()

pygame.draw.rect(screen,white,Rect((0,0),(250,100)))
while True:
    for event in pygame.event.get():
        if event.type == QUIT:
            exit()
    screen.lock()
    pygame.draw.rect(screen,red,Rect((10,10),(230,80)))
    pygame.draw.circle(screen,white,(50,50),40)
    pygame.draw.circle(screen,white,(200,50),40)
    screen.unlock()
    pygame.display.update()
```

Locking

- *Locking* reserves a part of the screen (*surface* in Pygame)
  - No other process can interfere
- Locking takes place automatically everytime when drawing
- Manual locking/unlocking improves performance
  - Reduses number of locking/unlocking operations

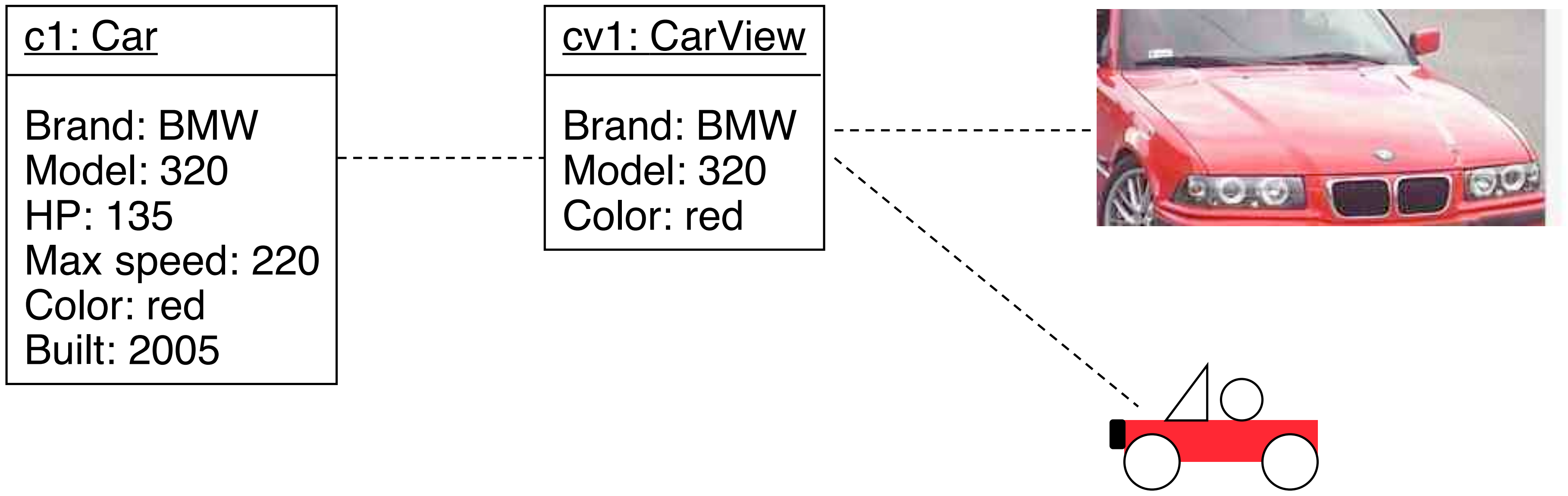
Program Objects and Visual Representations

- Following the model-view paradigm:



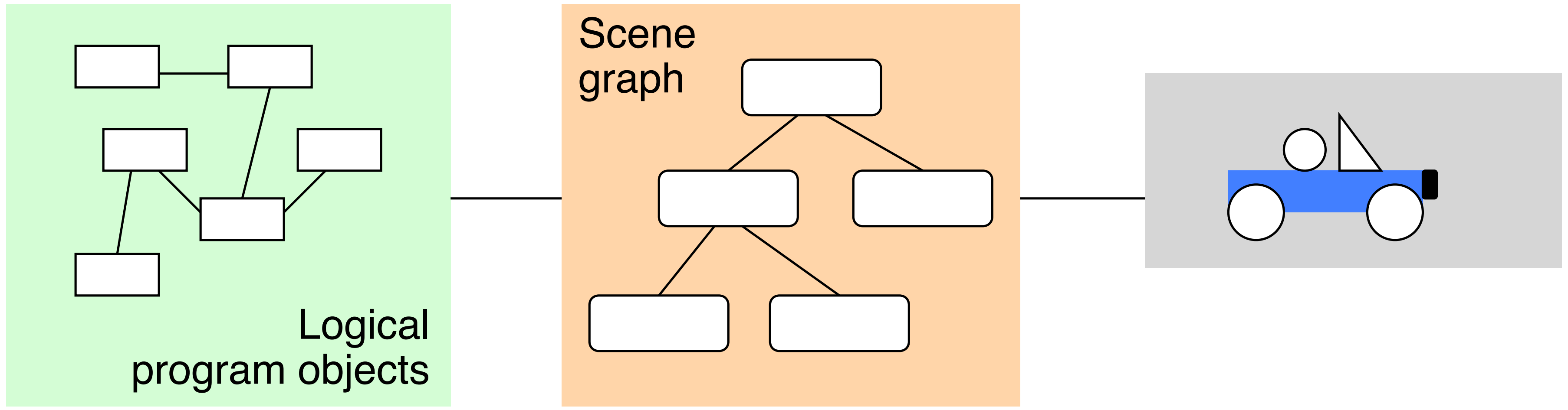
Methods for update of display:
- "Paint me" method in object
- Automatic periodical update
- "Observer" mechanism for local updates

Different Abstraction Levels for Objects



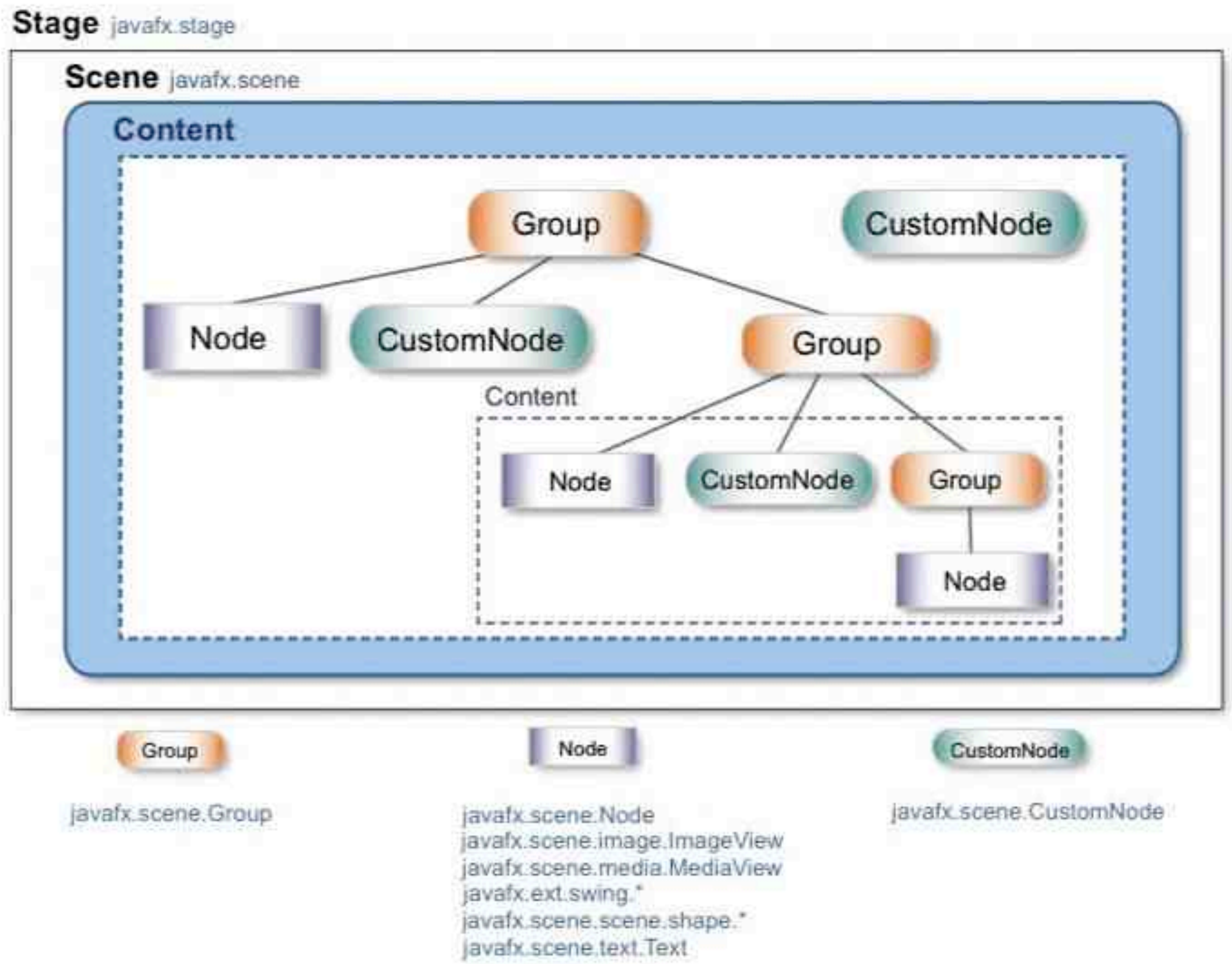
A view-oriented abstraction of program objects is helpful.
Graphical representations exist on different abstraction levels.

Scenes, Objects and Groups



- *Scene:* Collection of all relevant (view-oriented) objects
  - Abstract representation of the "world" (in a certain state)
- Often several objects are grouped into one (view-oriented) representation
  - Operations shall be applied to whole group (movement, copy, …)
- Two-level view mechanism:
  - Model
  - Scene graph (abstract view)
  - Concrete view

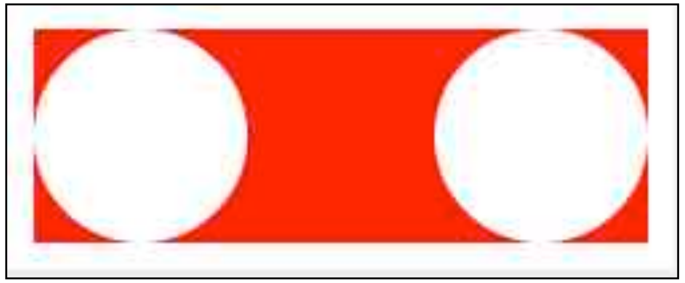
JavaFX Scene Graph



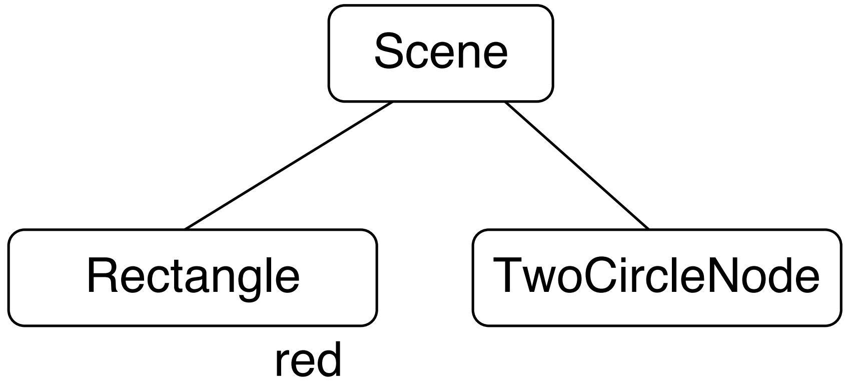
Netbeans.tv

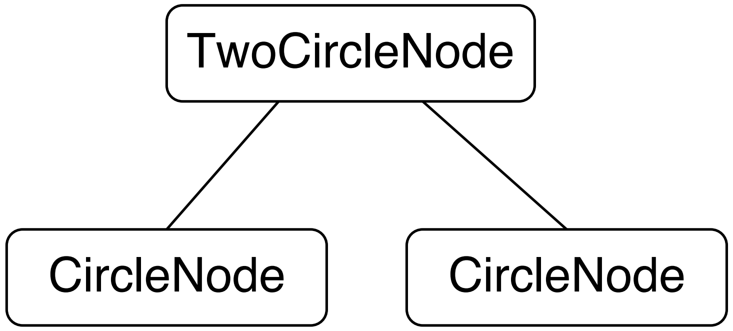
Scene Graph Example with JavaFX

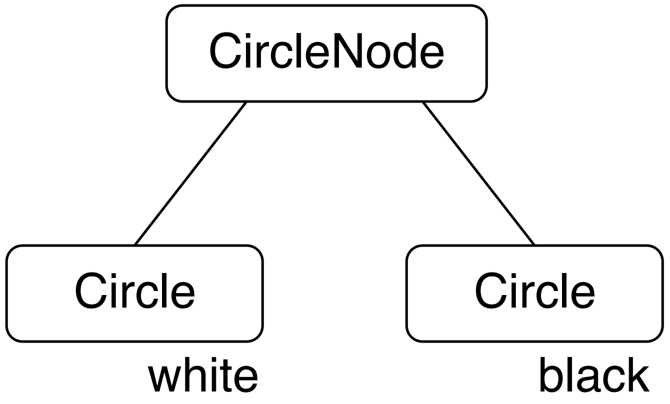
```
Stage {
    title: "2D Graphics"
    scene: Scene {
       width: 250, height: 100
       content: [
          Rectangle {
             x: 10, y: 10
             width: 230, height: 80
             fill: Color.RED
          },
          Group {
            translateX: 10
            translateY: 10
            content: [
                Circle {
                   centerX: 40, centerY: 40, radius: 40
                   fill: Color.WHITE
                },
                Circle {
                   centerX: 190, centerY: 40, radius: 40
                   fill: Color.WHITE
                },
            ]
          }
       ]
    }
}
```

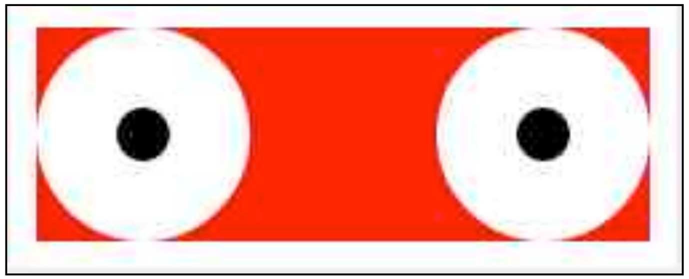


Object-Oriented Scene Graph Example









Object-Oriented Scene Graph in JavaFX (1)

```
class CircleNode extends CustomNode {
    override function create(): Node {
        return Group {
            content: [
                Circle {
                    centerX: 40, centerY: 40, radius: 40
                    fill: Color.WHITE
                },
                Circle {
                    centerX: 40, centerY: 40, radius: 10
                    fill: Color.BLACK
                }
            ]
        }
    }
}
```

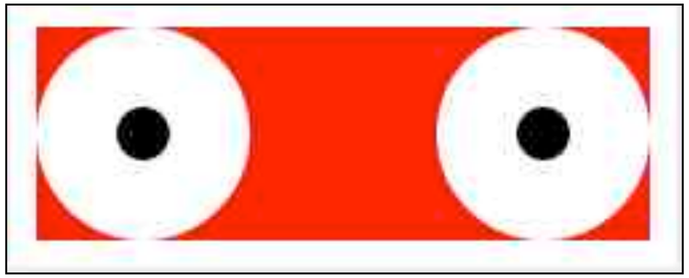
Coordinates relative to *local* coordinate system!

Object-Oriented Scene Graph in JavaFX (2)

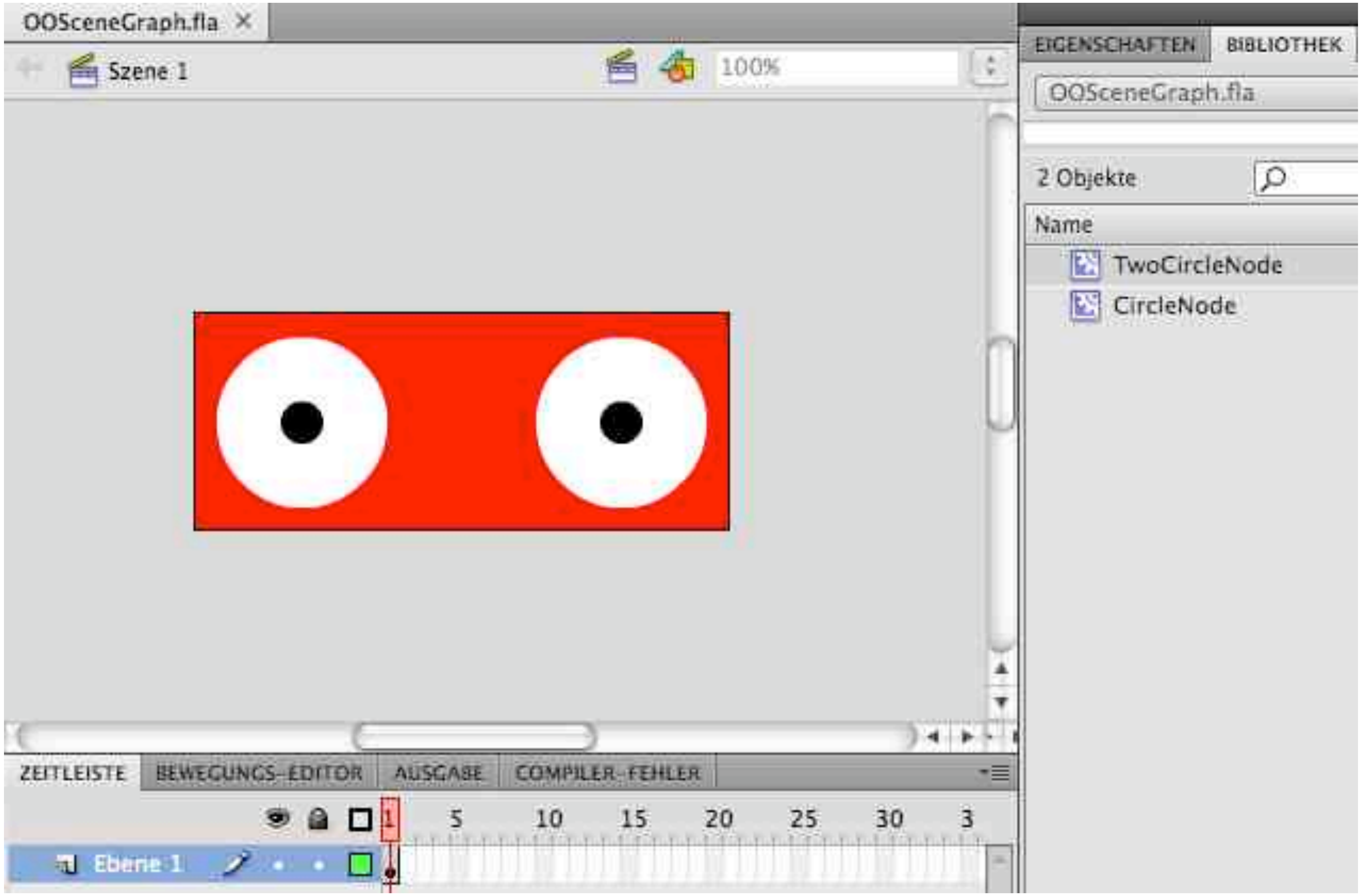
```
class TwoCircleNode extends CustomNode {
    override function create(): Node {
        return Group {
            content: [
                CircleNode{},
                CircleNode{
                    translateX: 150
                }
            ]
        }
    }
}
```

Object-Oriented Scene Graph in JavaFX (3)

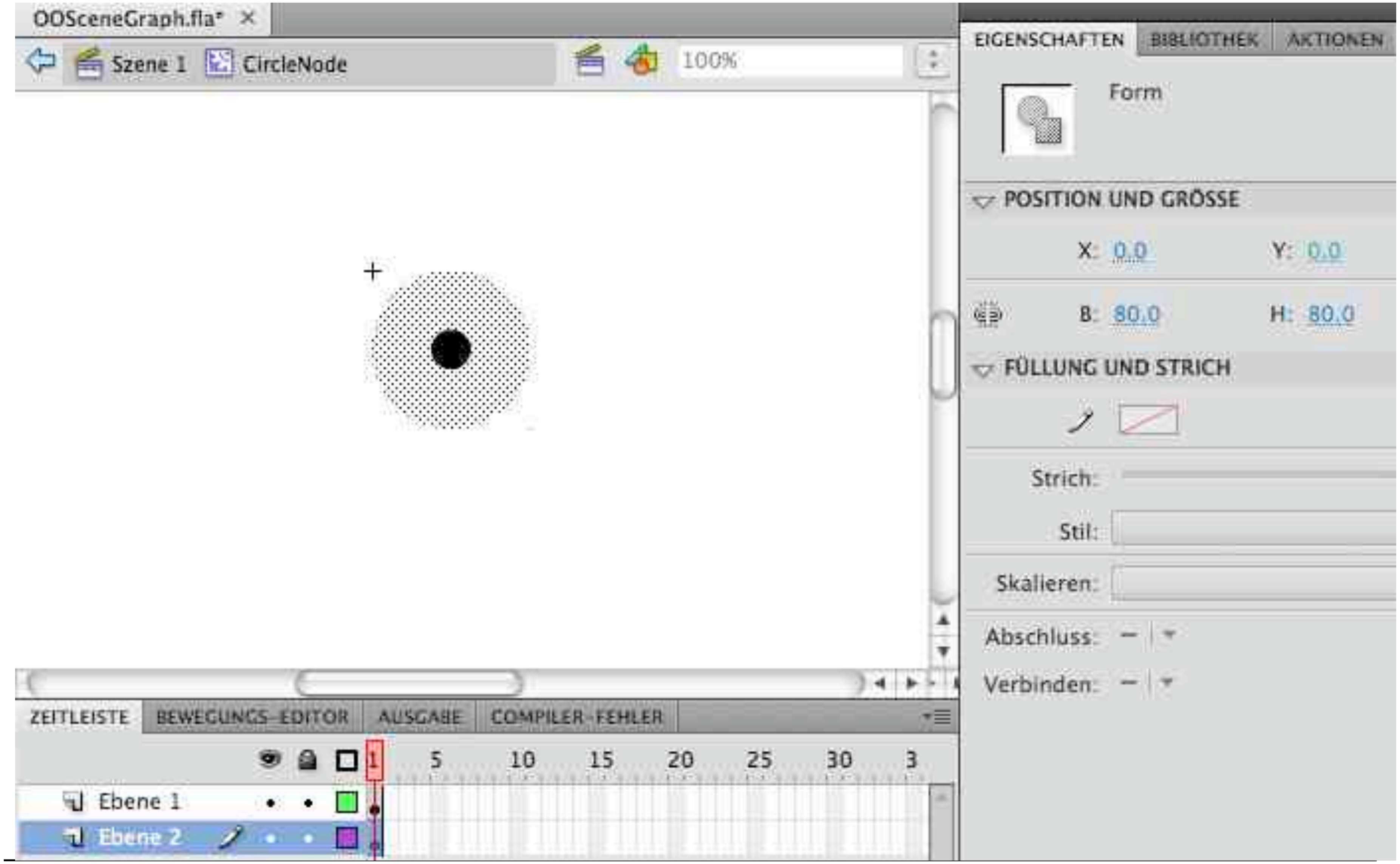
```
Stage {
    title: "OO Scene Graph"
    scene: Scene {
        width: 250, height: 100
        content: [
            Rectangle {
               x: 10, y: 10
               width: 230, height: 80
               fill: Color.RED
            },
            TwoCircleNode {
                translateX: 10, translateY: 10
            }
        ]
    }
}
```



Object-Oriented Scene Graph in Flash (1)



Object-Oriented Scene Graph in Flash (2)



Form

POSITION UND GRÖSSE

X: 0.0 Y: 0.0

B: 80.0 H: 80.0

FÜLLUNG UND STRICH

Strich:

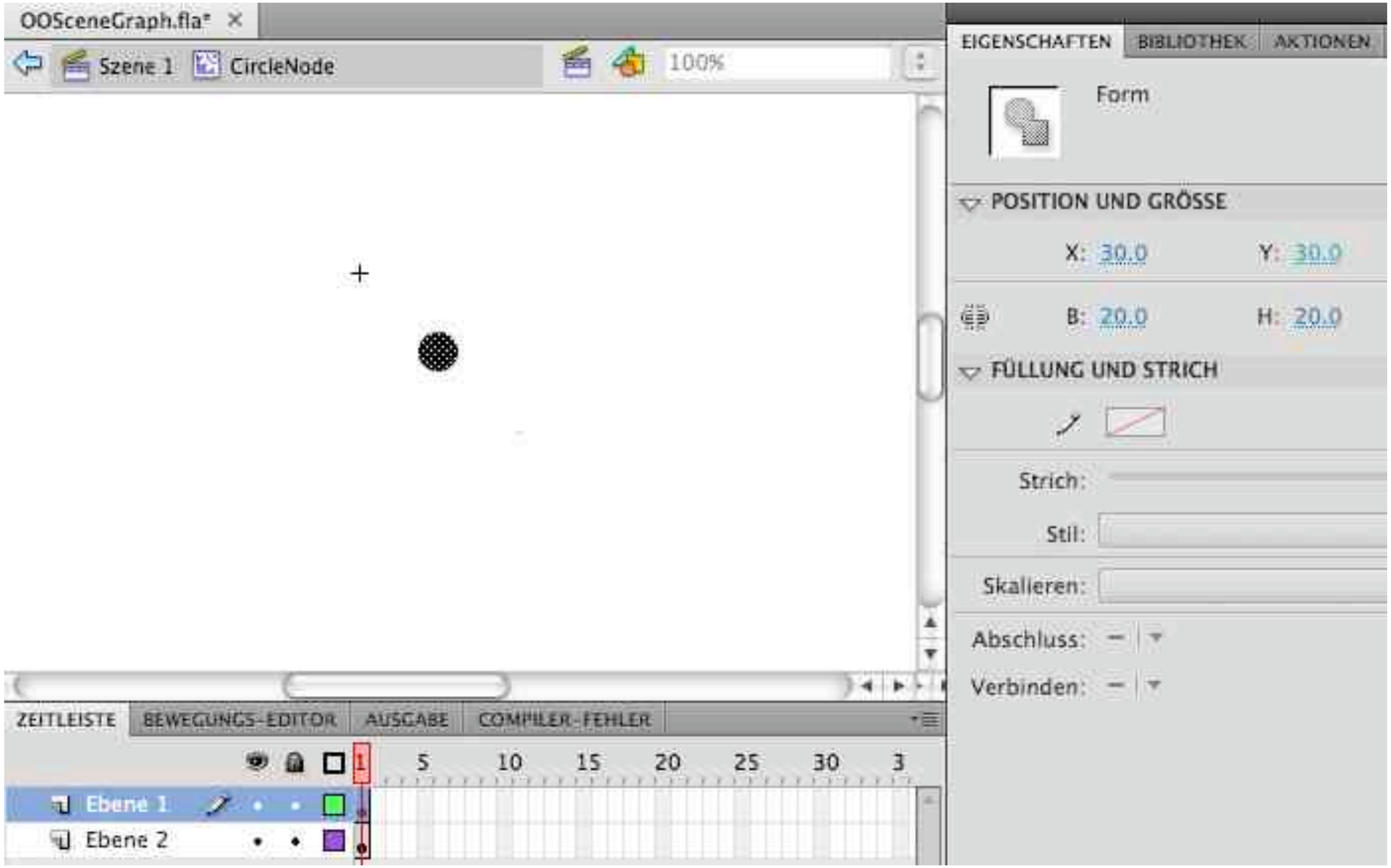
Stil:

Skalieren:

Abschluss: –

Verbinden: –

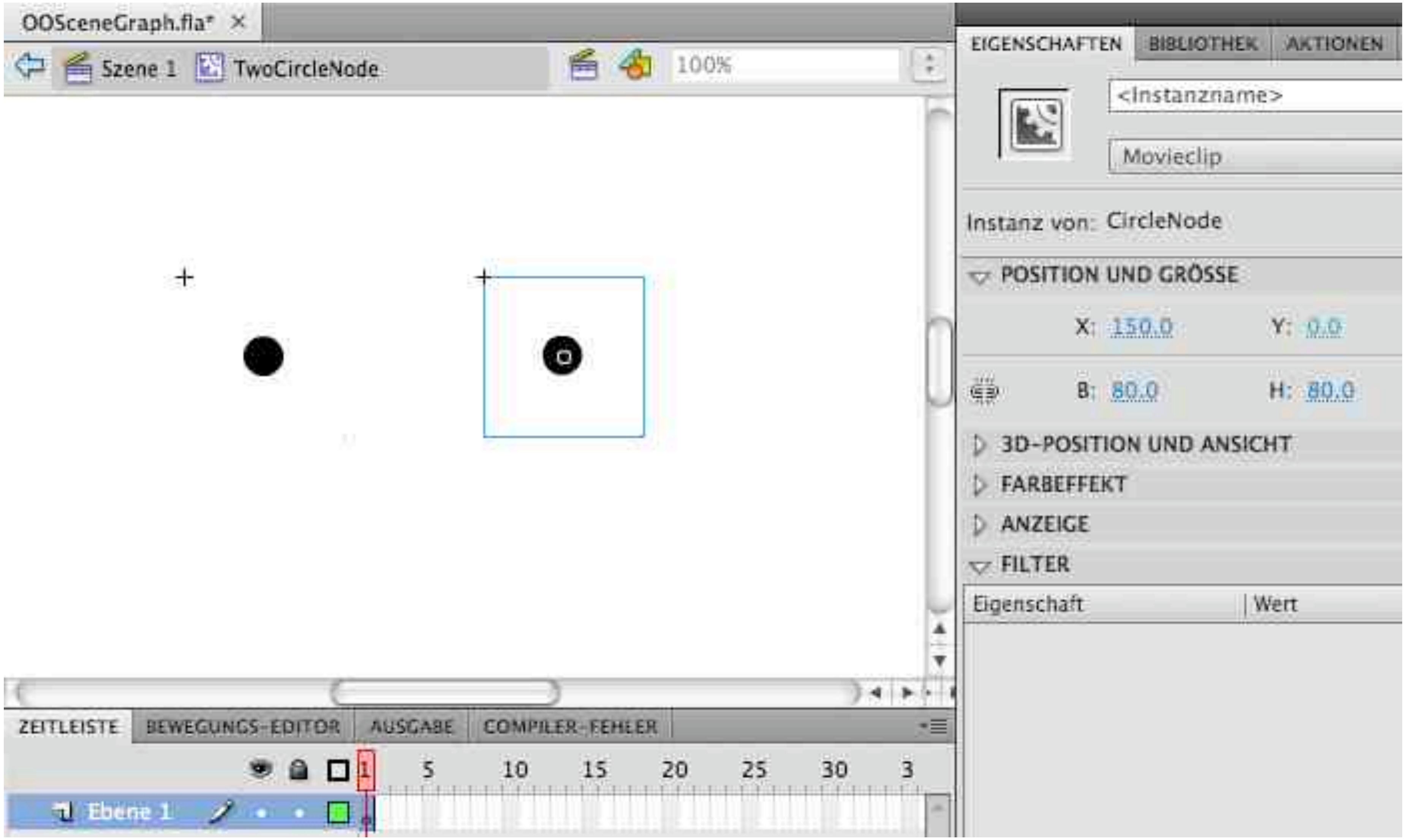
Object-Oriented Scene Graph in Flash (3)



X: 30.0 Y: 30.0

B: 20.0 H: 20.0

Object-Oriented Scene Graph in Flash (4)



EIGENSCHAFTEN | BIBLIOTHEK | AKTIONEN

<Instanzname>

Movieclip

Instanz von: CircleNode

POSITION UND GRÖSSE

X: 150.0 Y: 0.0

B: 80.0 H: 80.0

3D-POSITION UND ANSICHT

FARBEFFEKT

ANZEIGE

FILTER

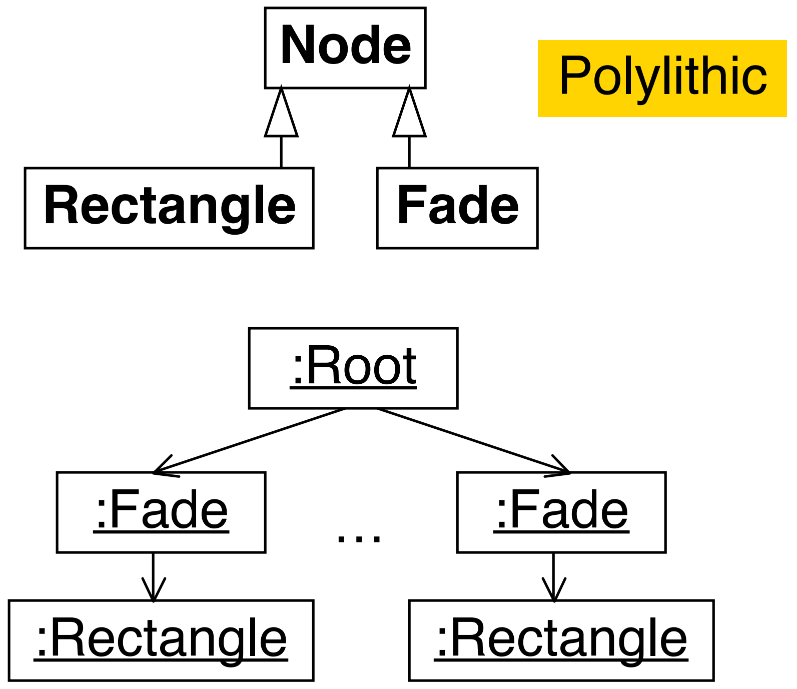
Eigenschaft	Wert

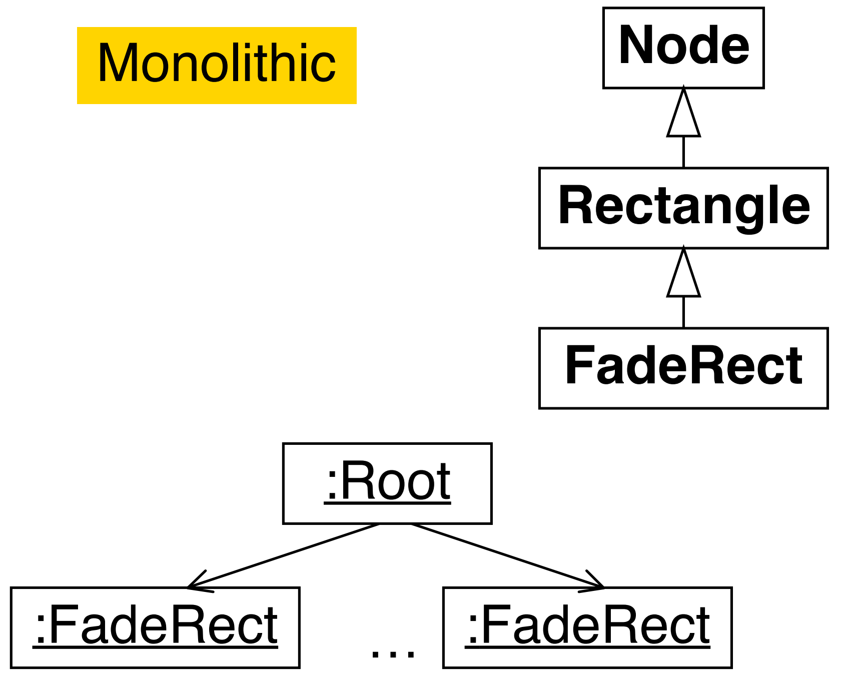
University of Maryland “Piccolo” Framework

- “A revolutionary way to create robust, full-featured *graphical applications* in Java and C#, with striking visual effects such as *zooming, animation* and *multiple representations*.”
  - Piccolo is a layer built on top of a lower level graphics API.
  - Piccolo.Java is written in 100% java, and is based on the Java2D API.
  - Piccolo uses a "scenegraph" model, this means that Piccolo keeps a hierarchical structure of objects and cameras.
- “History”:
  - Ken Perlin, New York University: “Pad” zoomable interface
  - Ben Bederson, Jim Hollan, Bellcore: “Pad++” (1994)
  - Ben Bederson et al, UMD: Jazz (ca. 1999)
    - » Many objects
  - Ben Bederson, Jesse Grosjean, UMD: Piccolo (2004)
- Also NOT an official Java standard but in widespread use

Monolithic and Polylithic Class Hierarchies

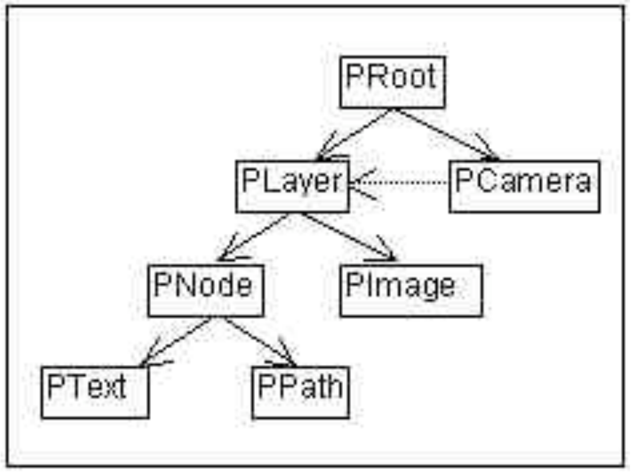
- *Monolithic:* Primarily uses compile-time *inheritance* to structure and extend functionality
- *Polylithic:* Primarily uses run-time *composition* to structure and extend functionality
  - More flexible, but creation of MANY objects





Piccolo Terminology

- ***PNode:*** Any object that wants to paint itself on the screen should inherit from the node class. In addition to painting on the screen all nodes may have other "child" nodes added to them.
- ***PCamera:*** Cameras are nodes that have an additional view transform and a collection of layers.
- ***PLayer:*** Layer nodes are nodes that can be viewed by one or more cameras. They maintain a list of the cameras that are viewing them, and notify these cameras when they are repainted.
- ***PRoot:*** The ***PRoot*** serves as the topmost node in the Piccolo runtime structure.
- ***PCanvas:*** The ***PCanvas*** views the scene graph through a ***PCamera***. It forwards input events to that camera, and uses that camera to draw itself.

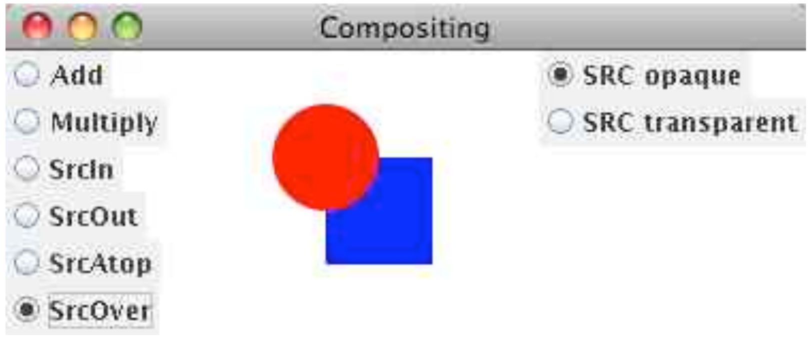


$$\begin{tikzcd}
 & & PRoot \arrow[dl] \arrow[dr] & \\
 & PLayer \arrow[dl] \arrow[dr] & & PCamera \arrow[ll, dotted] \\
PNode \arrow[d] \arrow[dr] & & PImage & \\
PText & PPath & &
\end{tikzcd}$$

Compositing, Alpha Channel

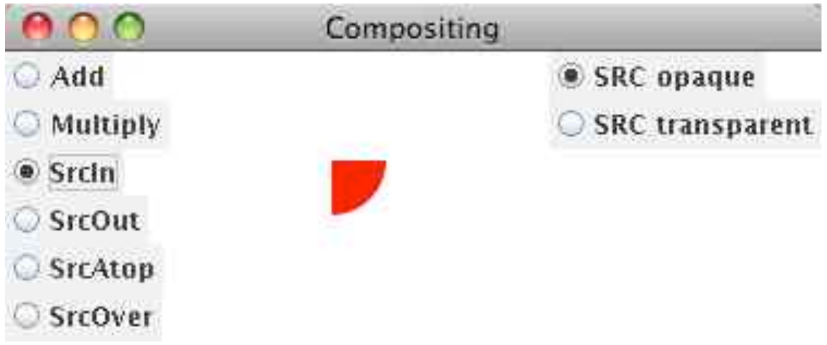
- Graphical elements may overlap
- Rules for deciding what to draw in the area where a second graphical element (source) is drawn onto an existing element (destination)
  - Standard: SRC_OVER
  - Other options:
    - » Only draw where destination exists (SRC_ATOP)
    - » Overwrite destination, draw source where destination exists (SRC_IN)
    - » Overwrite destination, draw source where destination does not exist (SRC_OUT)
    - » Merge source and destination (ADD, MULTIPLY)
- Result of composition depends on transparency (alpha) value of source

Compositing Example, Opaque



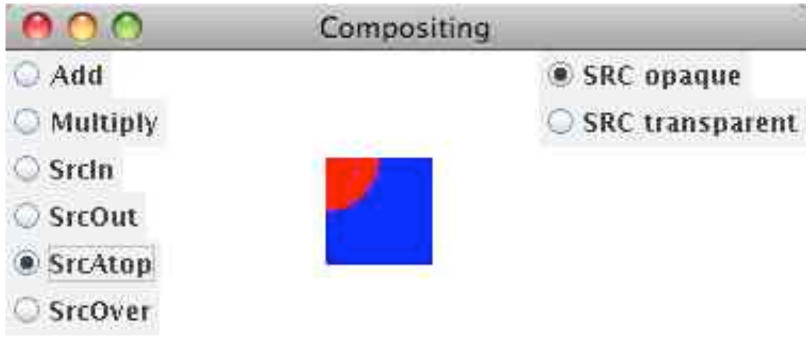
Add Multiply SrcIn SrcOut SrcAtop [✓ SrcOver]

[✓ SRC opaque] SRC transparent



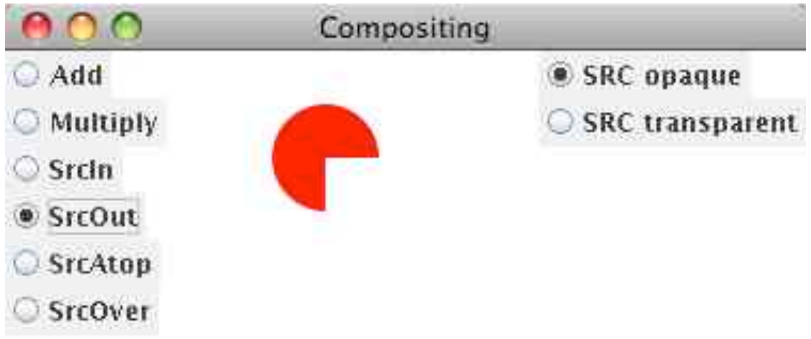
Add Multiply [✓ SrcIn] SrcOut SrcAtop SrcOver

[✓ SRC opaque] SRC transparent



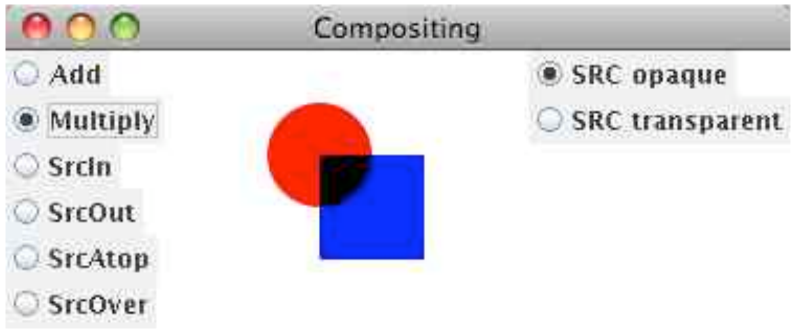
Add Multiply SrcIn SrcOut [✓ SrcAtop] SrcOver

[✓ SRC opaque] SRC transparent



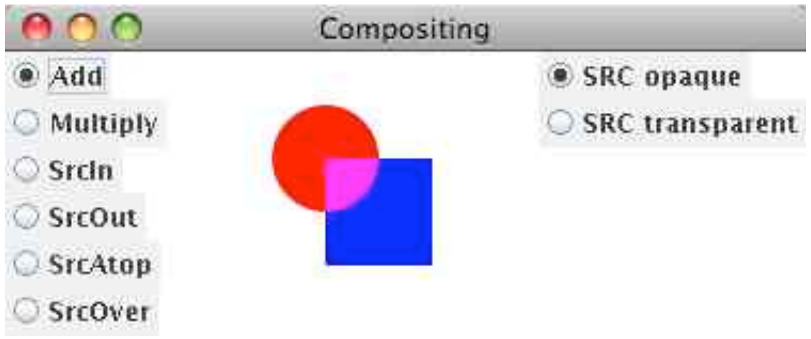
Add Multiply SrcIn [✓ SrcOut] SrcAtop SrcOver

[✓ SRC opaque] SRC transparent



Add [✓ Multiply] SrcIn SrcOut SrcAtop SrcOver

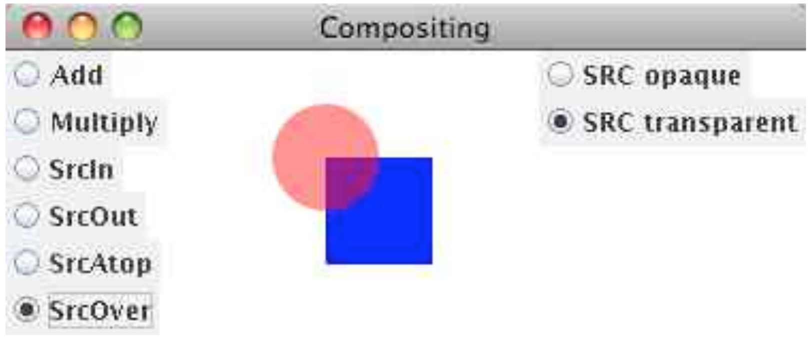
[✓ SRC opaque] SRC transparent



[✓ Add] Multiply SrcIn SrcOut SrcAtop SrcOver

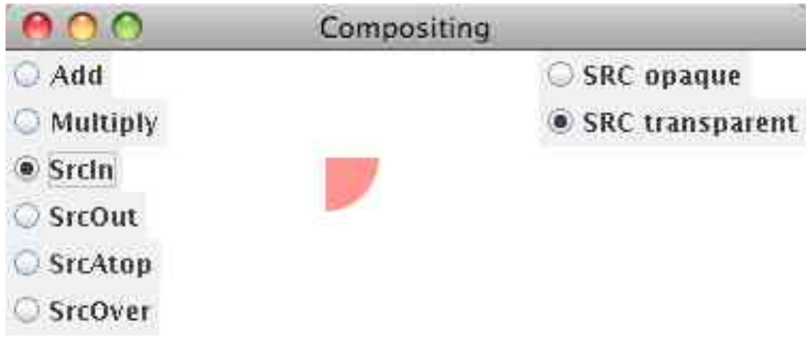
[✓ SRC opaque] SRC transparent

Compositing Example, Transparent


Compositing

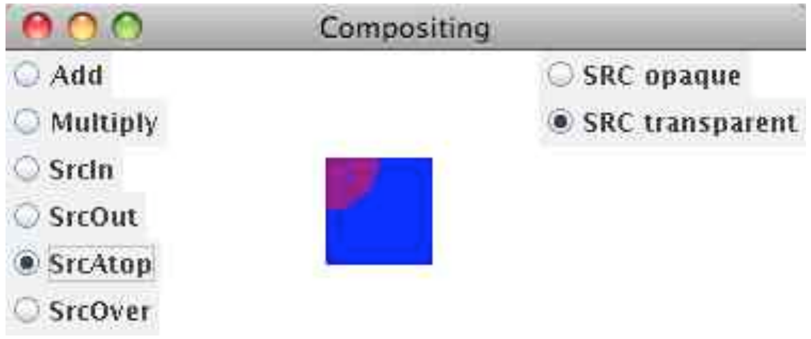
Add Multiply SrcIn SrcOut SrcAtop [✓ SrcOver]

SRC opaque [✓ SRC transparent]


Compositing

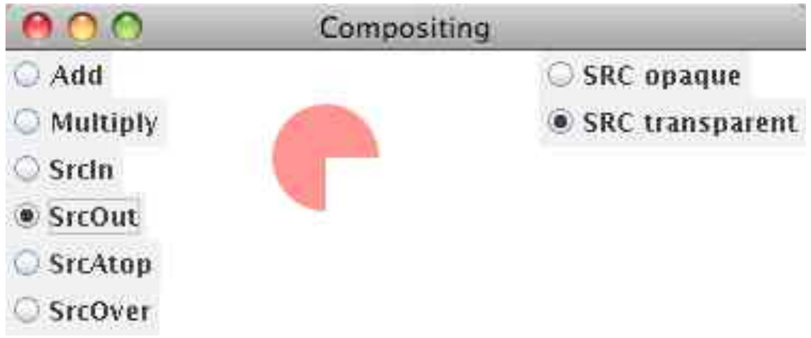
Add Multiply [✓ SrcIn] SrcOut SrcAtop SrcOver

SRC opaque [✓ SRC transparent]


Compositing

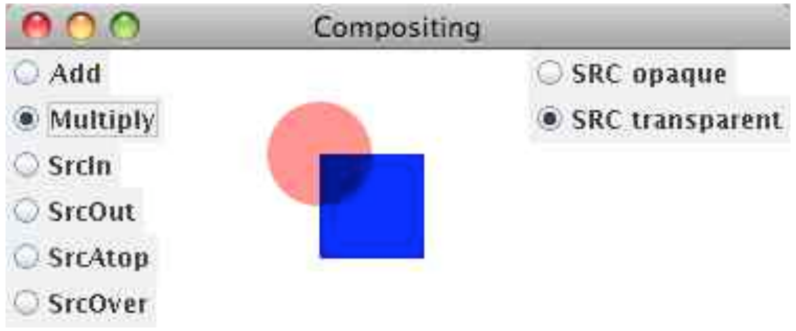
Add Multiply SrcIn SrcOut [✓ SrcAtop] SrcOver

SRC opaque [✓ SRC transparent]


Compositing

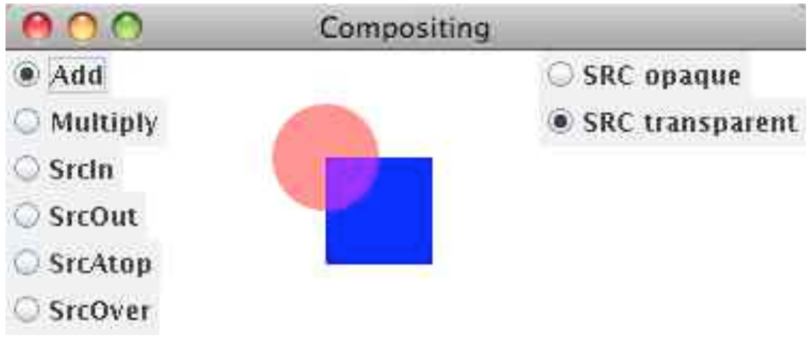
Add Multiply SrcIn [✓ SrcOut] SrcAtop SrcOver

SRC opaque [✓ SRC transparent]


Compositing

Add [✓ Multiply] SrcIn SrcOut SrcAtop SrcOver

SRC opaque [✓ SRC transparent]


Compositing

[✓ Add] Multiply SrcIn SrcOut SrcAtop SrcOver

SRC opaque [✓ SRC transparent]

4 Programming with Images

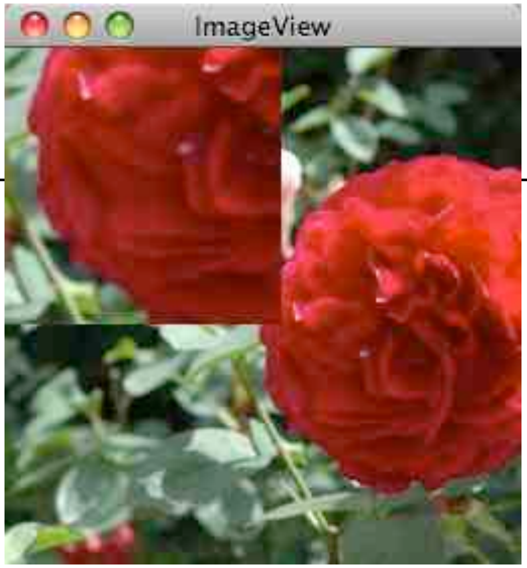
Bitmap Image

- Usually a special data type in multimedia framework
  - Often subclass of (2d vector) graphical elements
- Inout/Output functionality:
  - Reading picture files and conversion into internal representation (decoder)
  - Conversion of internal representation into external code (coder) and writing external file
- Internal representation:
  - Either: Reference to image information (class **`Image`** in JavaFX)
  - Or: Actual image buffer, accessible as data array
    - » JavaFX: **`bufferedImage`** property of **`Image`** delivers **`BufferedImage`**

Viewports

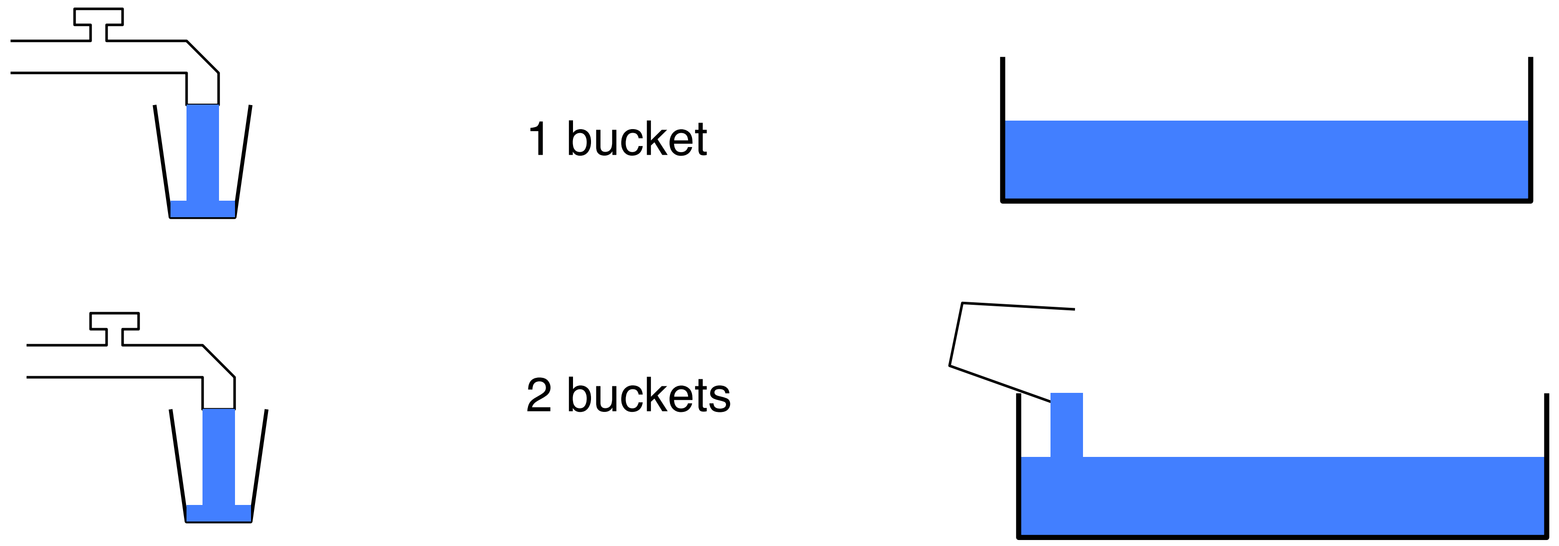
- A *viewport* is a separate representation of the same image which may display a subimage in transformed form

JavaFX



```
Stage {
    title: "ImageView"
    scene: Scene {
        content: [
            ImageView {
                image: image
            },
            ImageView {
                image: image
                viewport: Rectangle2D {
                    minX: 100, minY: 100, width: 110, height: 110
                }
                scaleX: 1.5, scaleY: 1.5
                smooth: true
            }
        ]
        fill: Color.BLACK
    }
}
```

Double Buffering



- Double buffering means to keep a copy of the image to be displayed in memory – different from the frame buffer
  - Updating: Copy from internal buffer to frame buffer
- Double buffering is implicitly used in most modern graphics frameworks

Blitting

- BLIT = Block Image Transfer
  - Bit blit = Name for a hardware technology for speeding up image transfer into frame buffer
  - Also used for optimization technique:
    - » Combine small local changes into a larger buffer
    - » Display large image at once – faster than individual updates
- Possible only by using a second buffer besides frame buffer
  - Double buffering

4 Programming with Images

Literature:

Will McGugan: Beginning Game Development with Python and Pygame, Apress 2007

Sprite

- A *sprite (Kobold, Geist)* is a movable graphics object which is presented on top of the background image.
  – Mouse pointer images are examples of sprites
- Hardware sprite:
  – Outdated technique for hardware-supported fast display of moving image
- Software sprite:
  – Any moving picture displayed over background
- Pygame sprite:
  – Special class designed to display movable game objects

Simple Sprite in Pygame

```
class MagSprite(pygame.sprite.Sprite):

    def __init__(self):
        pygame.sprite.Sprite.__init__(self)
        self.image = pygame.image.load(sprite_imgfile)
        self.rect = self.image.get_rect()

    def update(self):
        self.rect.center = pygame.mouse.get_pos()

sprite = MagSprite()
allsprites = pygame.sprite.Group()
allsprites.add(sprite)

while True:
    for event in pygame.event.get():
        ...
        screen.blit(background,(0,0))
        allsprites.update()
        allsprites.draw(screen)
        pygame.display.update()
```