6 Programming with Images

6.1 Graphics and Pictures Across Platforms

6.2 Displaying Static Vector/Bitmap Graphics

6.3 Structured Graphics: Display Lists, Scene Graphs

6.4 Sprites

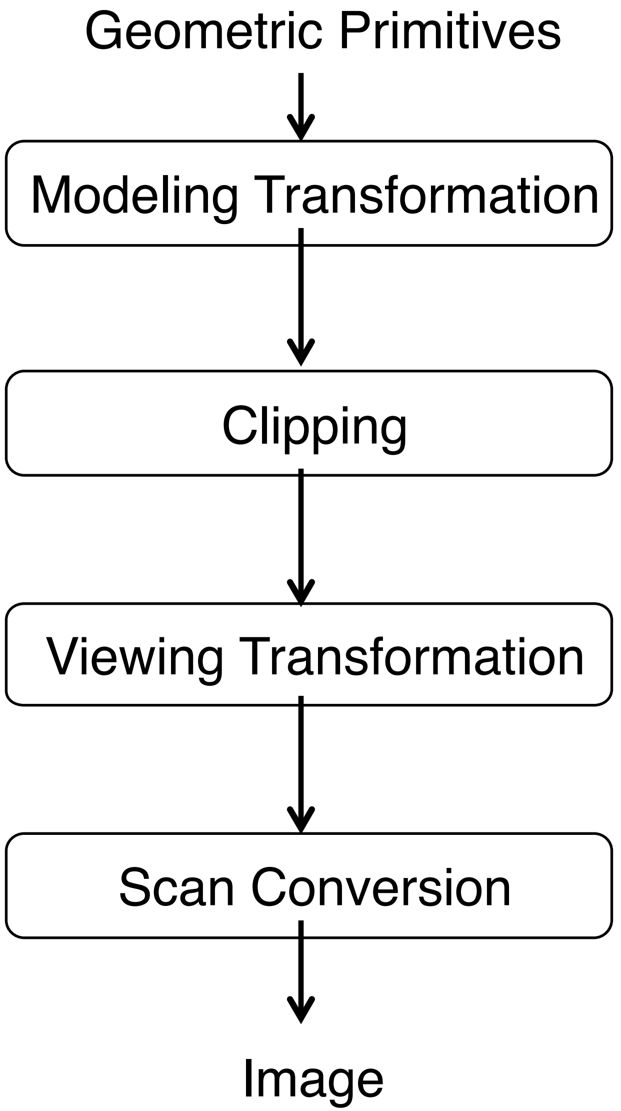
Literature:

http://carlfx.wordpress.com/2012/04/09/javafx-2-gametutorial-part-2/

Vector Graphics vs. Bitmap Graphics

- Vector Graphics
  - Picture synthesized from geometric primitives (points, lines, curves)
  - Easy access through programming interface
  - Dynamic adaptation to current program state is easily possible
  - Basis for dynamic graphics (animation)
- Bitmap Graphics
  - Picture pre-recorded, in many cases sampled from analog original (e.g. through camera, scanner)
  - Programming interface restricted to picture manipulations
  - Difficult to adapt dynamically
  - Essentially static

2D Rendering Pipeline



Geometric Primitives

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

Image

Adapted from: Funkhouser, Princeton U

Low-Level 2D Graphics

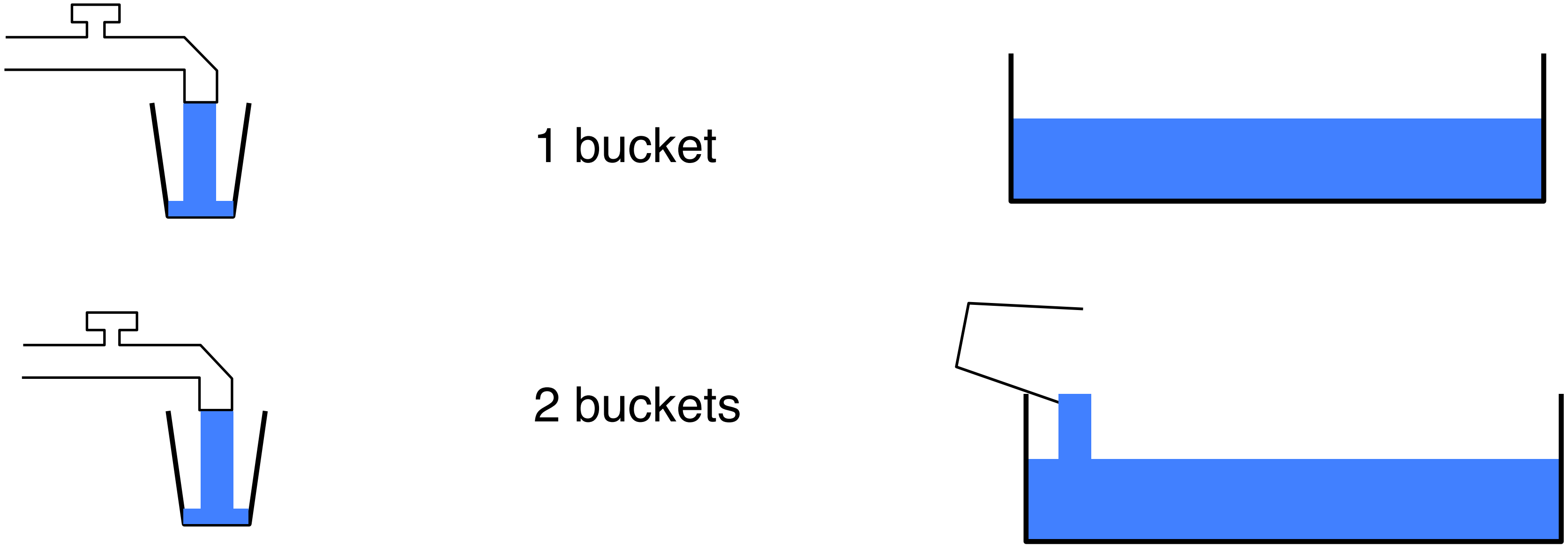
- Immediate mode:
  - Screen content is created pixel by pixel
  - (vs.: Retained Mode)
- Sometimes: Full-screen & hardware-accelerated
- Main problems:
  - *Screen Flickering*:
    due to interference between screen update and drawing commands
  - Simultaneous drawing from different processes/threads on same target
- Traditional solutions:
  - Double-buffering, multi-buffering
  - Screen locking
- Please note:
  - Most of these principles (or better remedies thereof)
    are built into modern graphics frameworks already!

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

Double Buffering

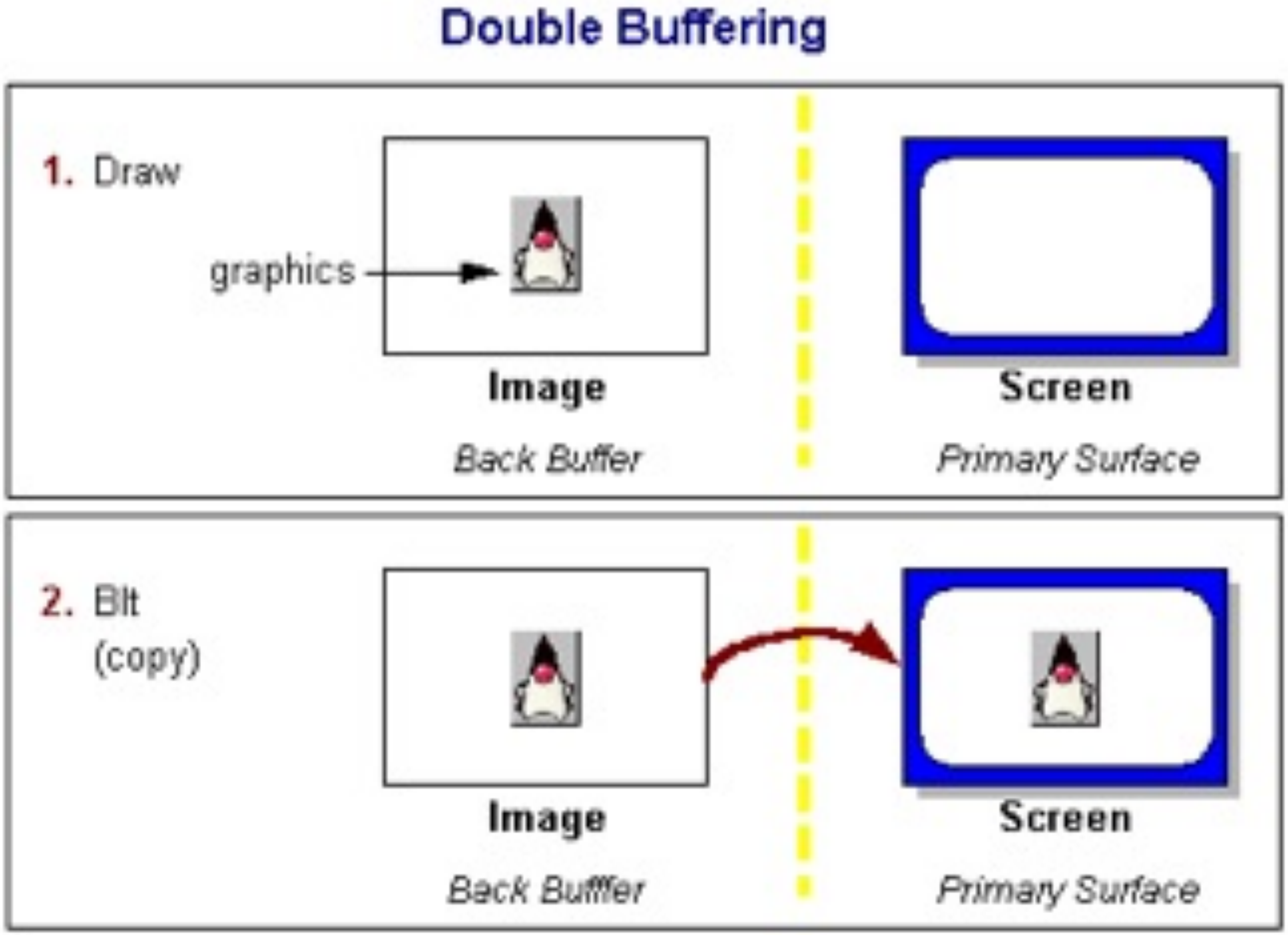
- Idea:
  - Draw to a separate memory area from screen buffer
  - Draw all contents at once
- Implementations may use very fast buffer switching (change pointers)
- Double buffering is implicitly used in most modern graphics frameworks!
- One traditional explanation for speedup effect:
  - Consider filling an outdoor pool with water…



1 bucket

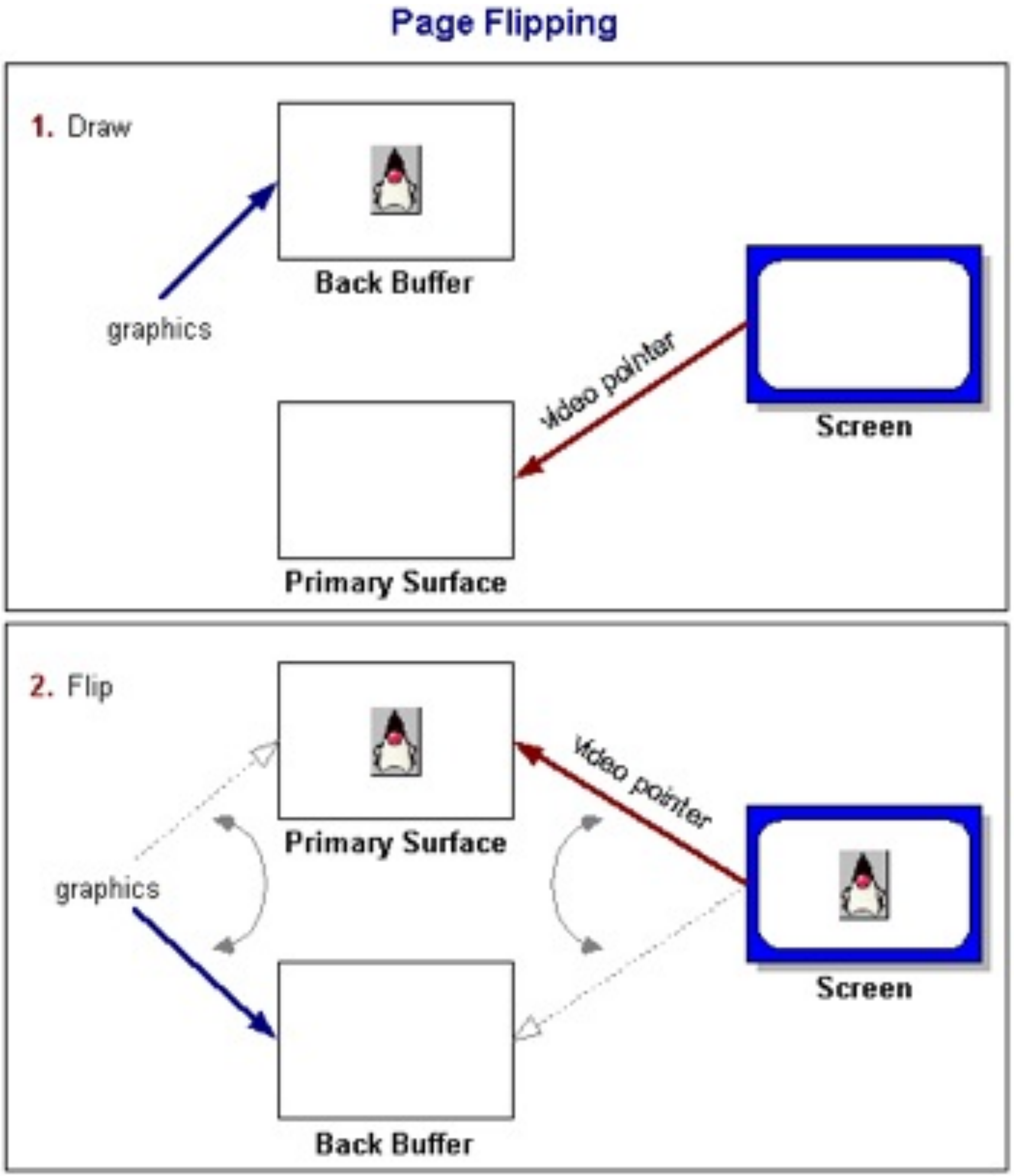
2 buckets

Double Buffering for Images



http://docs.oracle.com/javase/tutorial/extra/fullscreen/doublebuf.html

Page Flipping



http://docs.oracle.com/javase/tutorial/extra/fullscreen/doublebuf.html

Screen/Surface Locking

- *Locking* reserves a part of the screen (*surface* in Pygame)
  - No other process can interfere
    (“one change at a time, please”)
- Locking takes place automatically every time when drawing
- Manual locking/unlocking may improve performance
  - Add a lock/unlock pair of commands around a logically contingent group of graphics commands
  - Reduces number of (implied) locking/unlocking operations

6 Programming with Images

6.1 Graphics and Pictures Across Platforms

6.2 Displaying Static Vector/Bitmap Graphics

6.3 Structured Graphics: Display Lists, Scene Graphs

6.4 Sprites

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

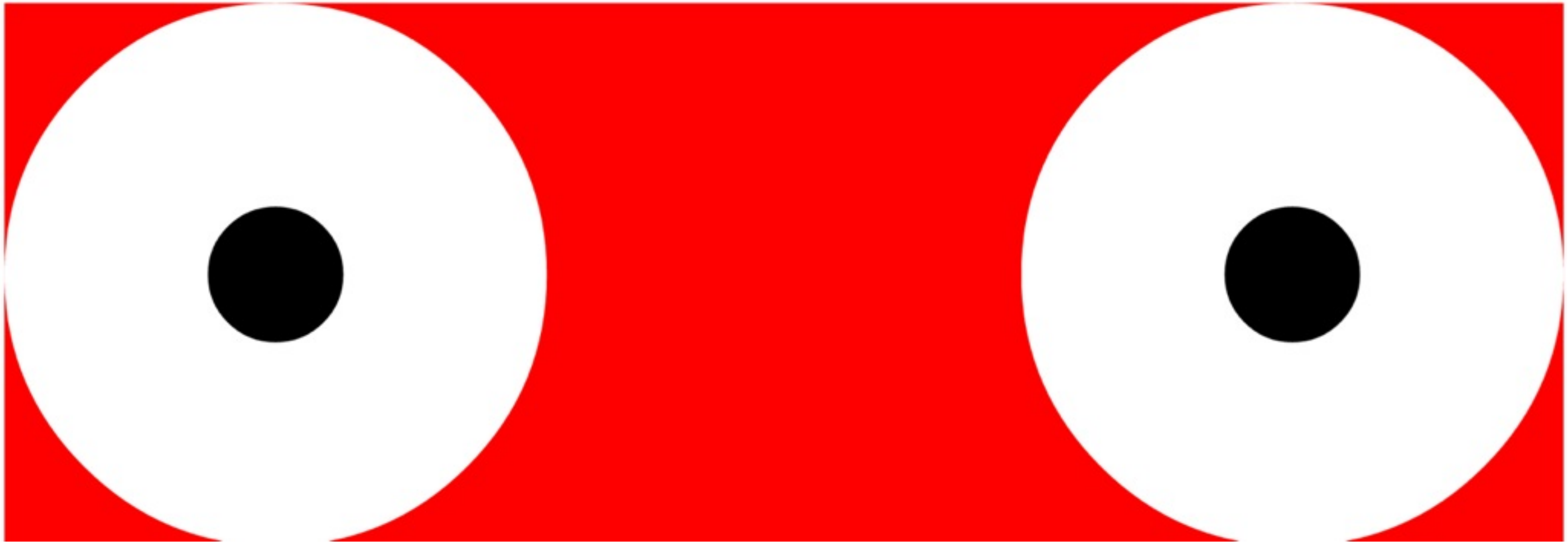
How to Describe Vector Graphics Content

```
<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>
```

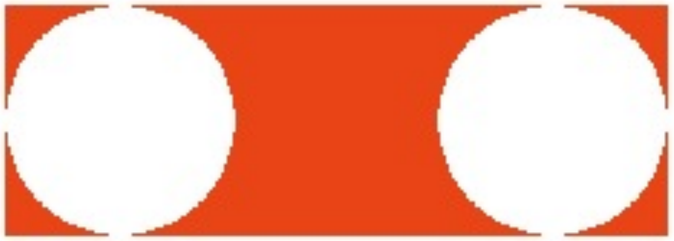
- Declarative document:
  - Vector graphics file format, e.g. SVG
  - Vector graphics is ***not*** part of pure media *integration* languages like SMIL!
- Graphical editor:
  - Either to create a graphics file (e.g. Illustrator, Inkscape)
  - Or as integral part of authoring tool (e.g. Flash)
- Program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

QUIZ

- How can we realize the reuse of structures like the two concentric circles in the drawing (wich appears twice) in SVG?
- Are there alternatives, what are the differences?



Drawing Vector Graphics in Python/Pygame



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

pygame.init()
screen.fill(white)

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()
```

Not really necessary

QUIZ

- Why is there a loop in the preceding code example?
  Isn’t everything static there?

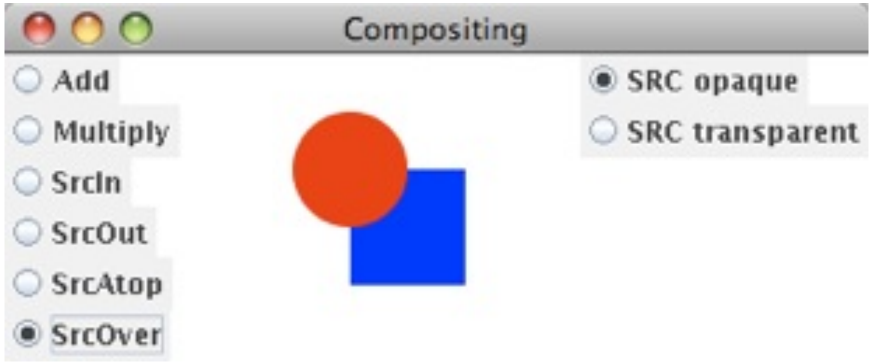
Bitmap Image

- Usually a special data type in multimedia framework
  - Often subclass of (2D vector) graphical elements
- Input/Output functionality:
  - Reading picture files and conversion into internal representation (decoder)
    - » Image constructor in JavaFX
    - » Pygame: pygame.image.load method
    - » CreateJS: Load queue (preloader) and createjs.Bitmap class
  - Conversion of internal representation into external code (coder) and writing external file
- Internal representation:
  - Reference to image information (class `Image` in JavaFX)
  - Or integrated with surface (e.g. in Pygame)

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

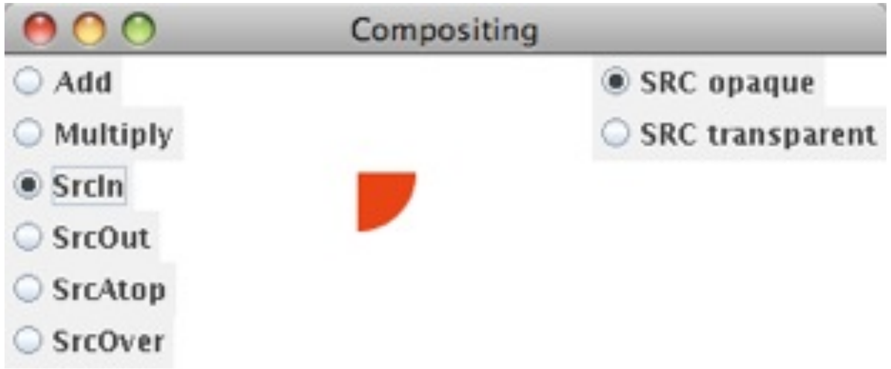
Compositing Example, Opaque



Compositing

○ Add
○ Multiply
○ SrcIn
○ SrcOut
○ SrcAtop
● SrcOver

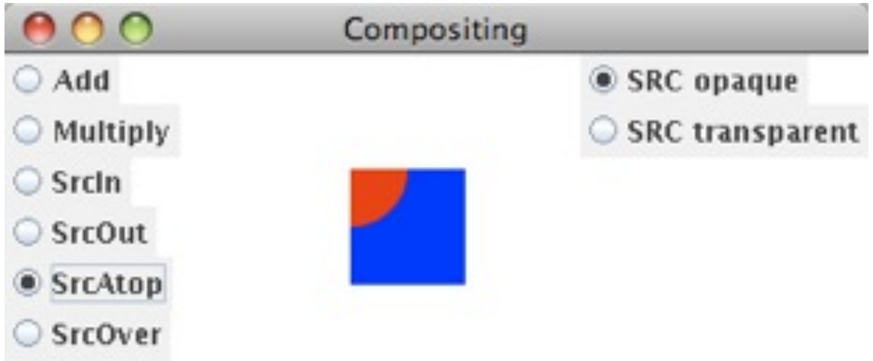
● SRC opaque
○ SRC transparent



Compositing

○ Add
○ Multiply
● SrcIn
○ SrcOut
○ SrcAtop
○ SrcOver

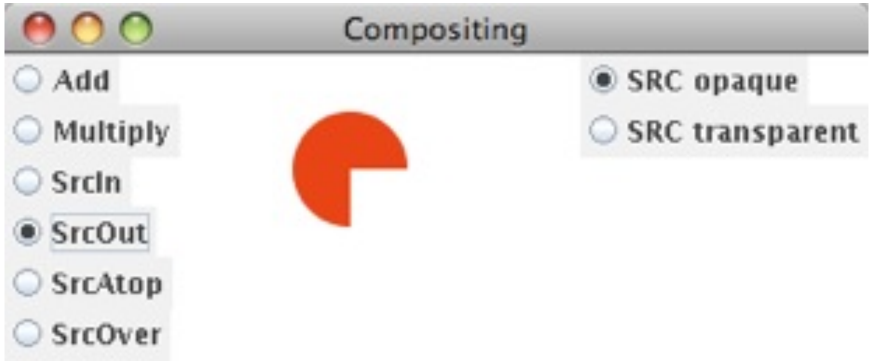
● SRC opaque
○ SRC transparent



Compositing

○ Add
○ Multiply
○ SrcIn
○ SrcOut
● SrcAtop
○ SrcOver

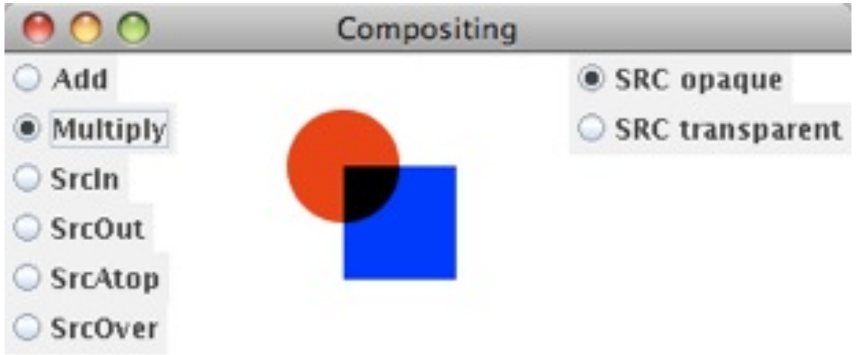
● SRC opaque
○ SRC transparent



Compositing

○ Add
○ Multiply
○ SrcIn
● SrcOut
○ SrcAtop
○ SrcOver

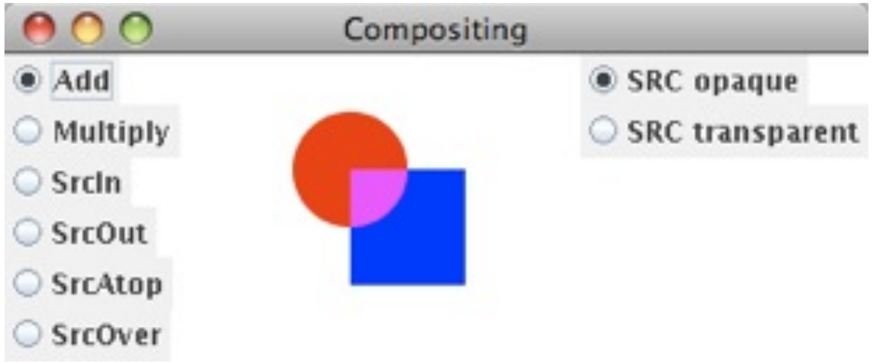
● SRC opaque
○ SRC transparent



Compositing

○ Add
● Multiply
○ SrcIn
○ SrcOut
○ SrcAtop
○ SrcOver

● SRC opaque
○ SRC transparent

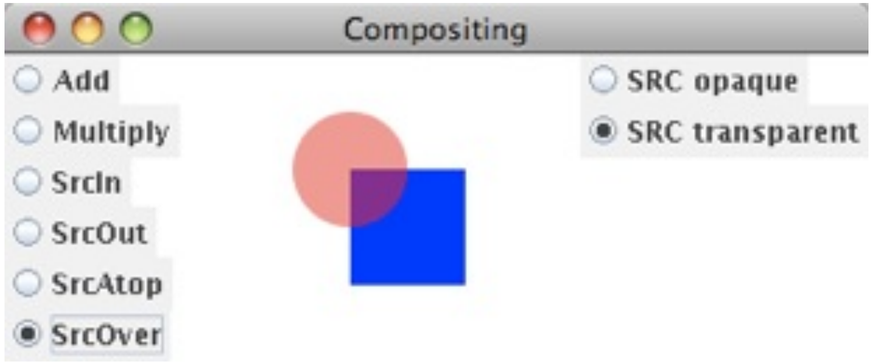


Compositing

● Add
○ Multiply
○ SrcIn
○ SrcOut
○ SrcAtop
○ SrcOver

● SRC opaque
○ SRC transparent

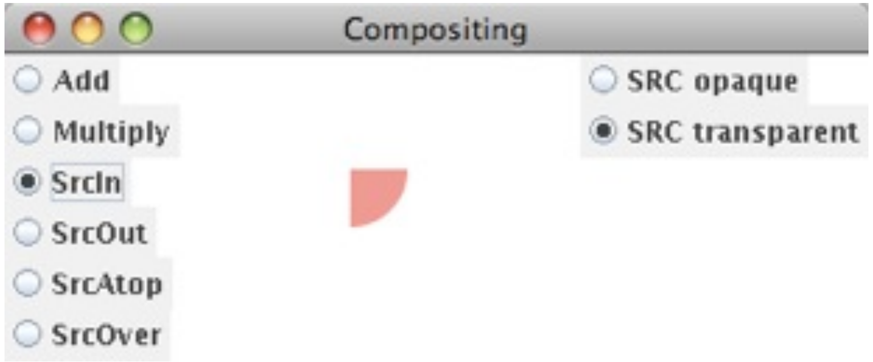
Compositing Example, Transparent



Compositing

○ Add
○ Multiply
○ SrcIn
○ SrcOut
○ SrcAtop
● SrcOver

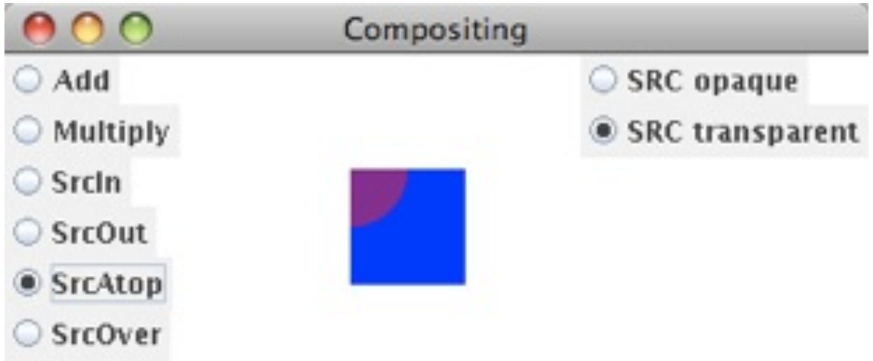
○ SRC opaque
● SRC transparent



Compositing

○ Add
○ Multiply
● SrcIn
○ SrcOut
○ SrcAtop
○ SrcOver

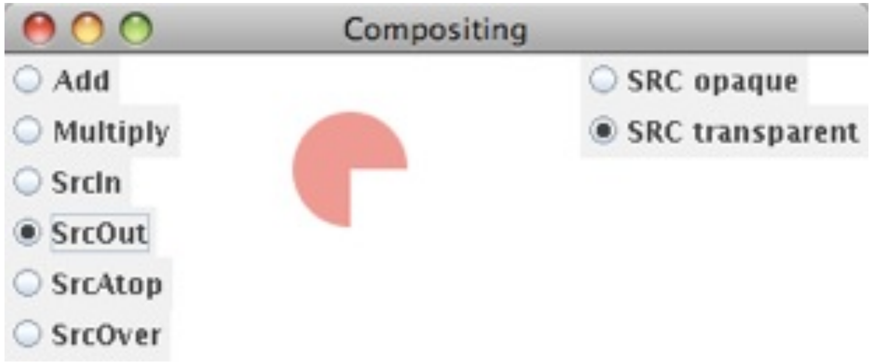
○ SRC opaque
● SRC transparent



Compositing

○ Add
○ Multiply
○ SrcIn
○ SrcOut
● SrcAtop
○ SrcOver

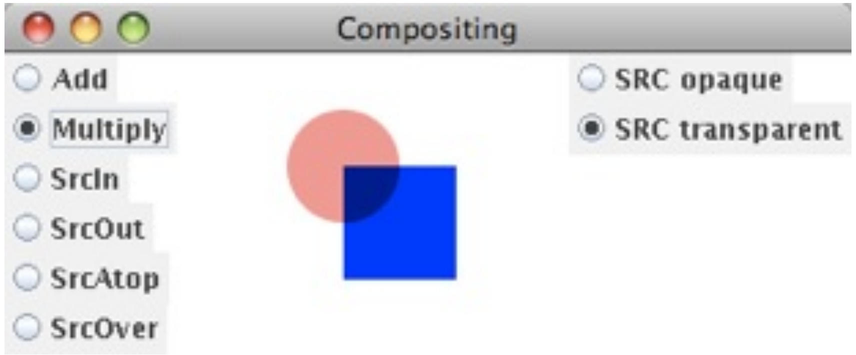
○ SRC opaque
● SRC transparent



Compositing

○ Add
○ Multiply
○ SrcIn
● SrcOut
○ SrcAtop
○ SrcOver

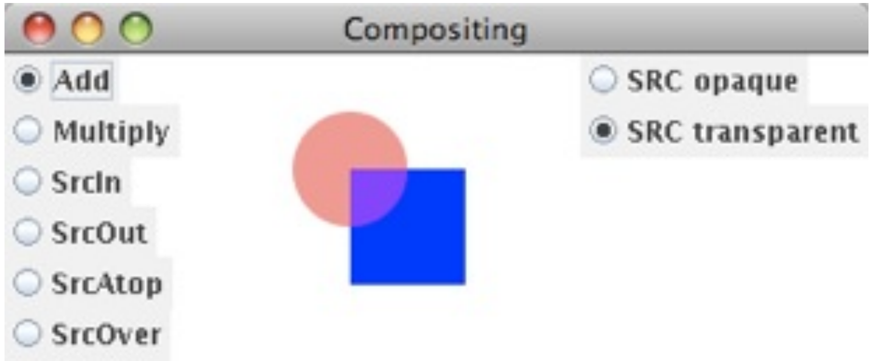
○ SRC opaque
● SRC transparent



Compositing

○ Add
● Multiply
○ SrcIn
○ SrcOut
○ SrcAtop
○ SrcOver

○ SRC opaque
● SRC transparent



Compositing

● Add
○ Multiply
○ SrcIn
○ SrcOut
○ SrcAtop
○ SrcOver

○ SRC opaque
● SRC transparent

6 Programming with Images

6.1 Graphics and Pictures Across Platforms

6.2 Displaying Static Vector/Bitmap Graphics

6.3 Structured Graphics: Display Lists, Scene Graphs

6.4 Sprites

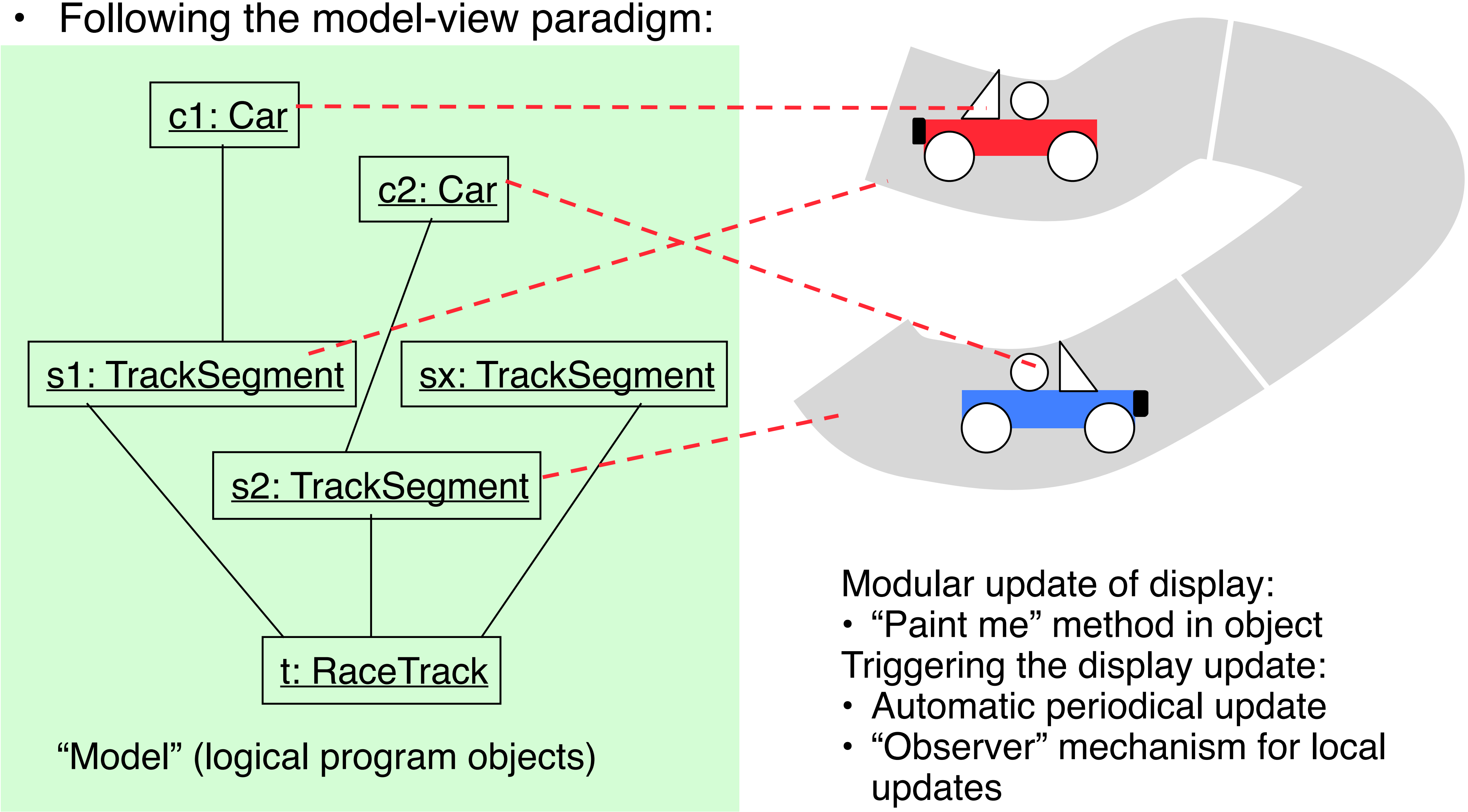
Literature:

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

Program Objects and Visual Representations

- Following the model-view paradigm:



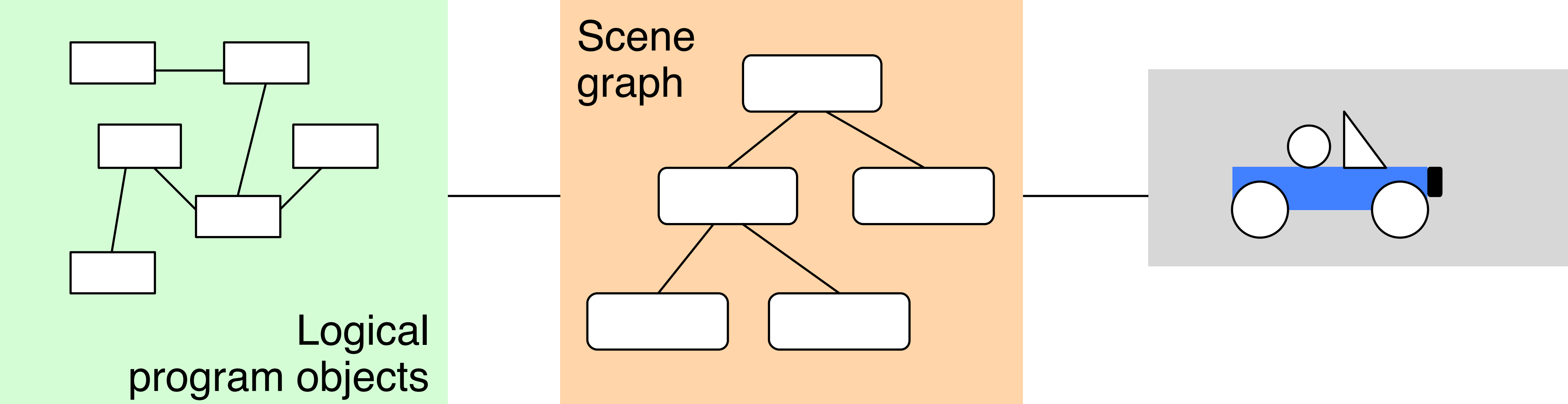
Modular update of display:

- “Paint me” method in object

Triggering the display update:

- Automatic periodical update
- “Observer” mechanism for local updates

Scenes, Objects and Groups

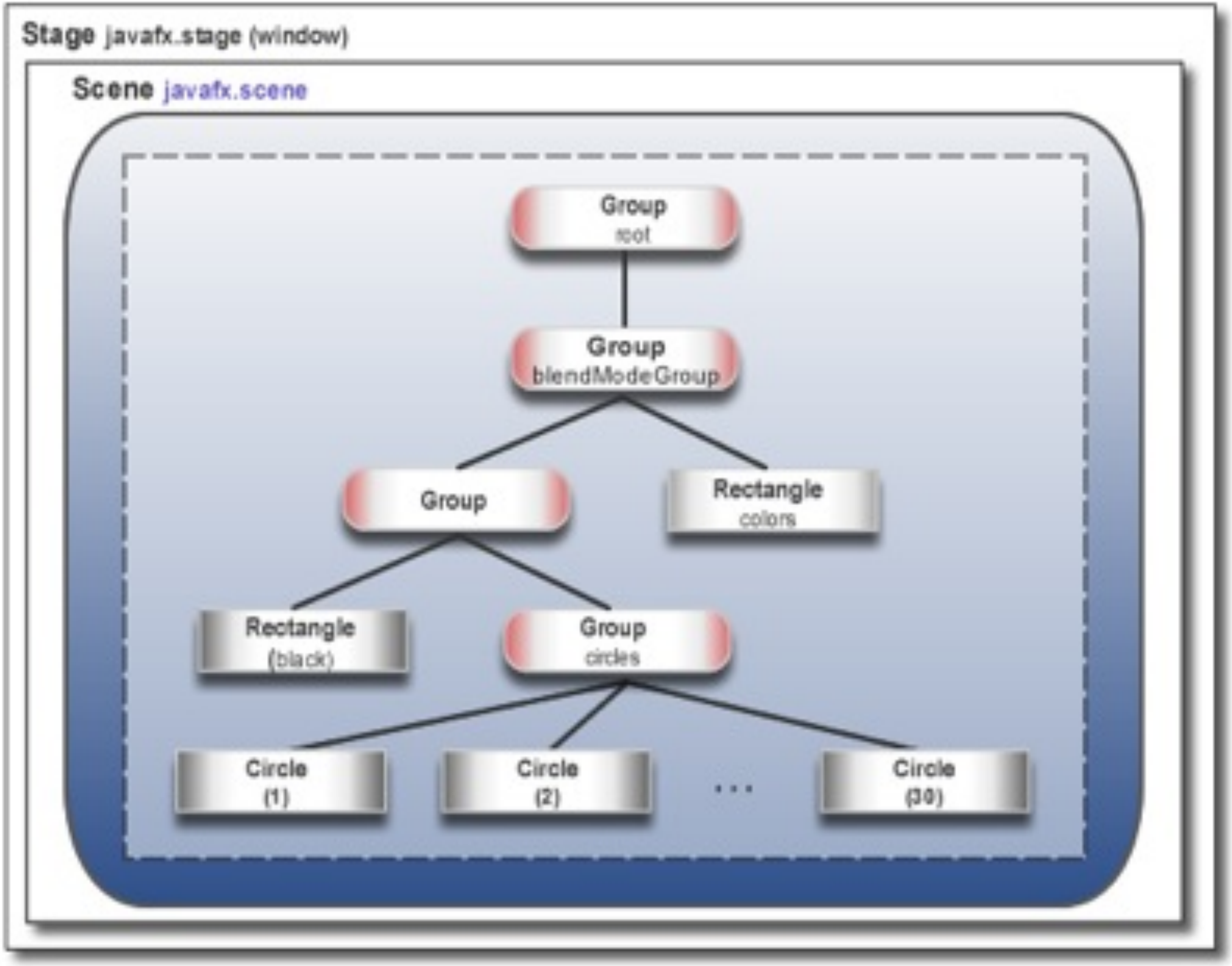


Logical program objects

Scene graph

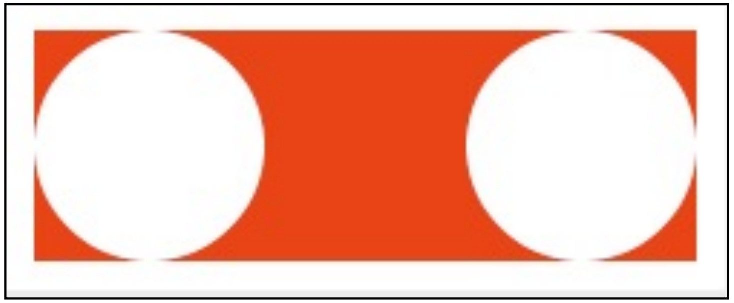
- *Scene:* Collection of all relevant (view-oriented) objects
  - Abstract representation of the “world” (in a certain state)
- Often several objects are grouped into one (view-oriented) representation
  - Operations shall be applied to whole group (movement, copy, …)
- Two-level view mechanism:
  -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

JavaFX Scene Graph



docs.oracle.com

Scene Graph Example with JavaFX

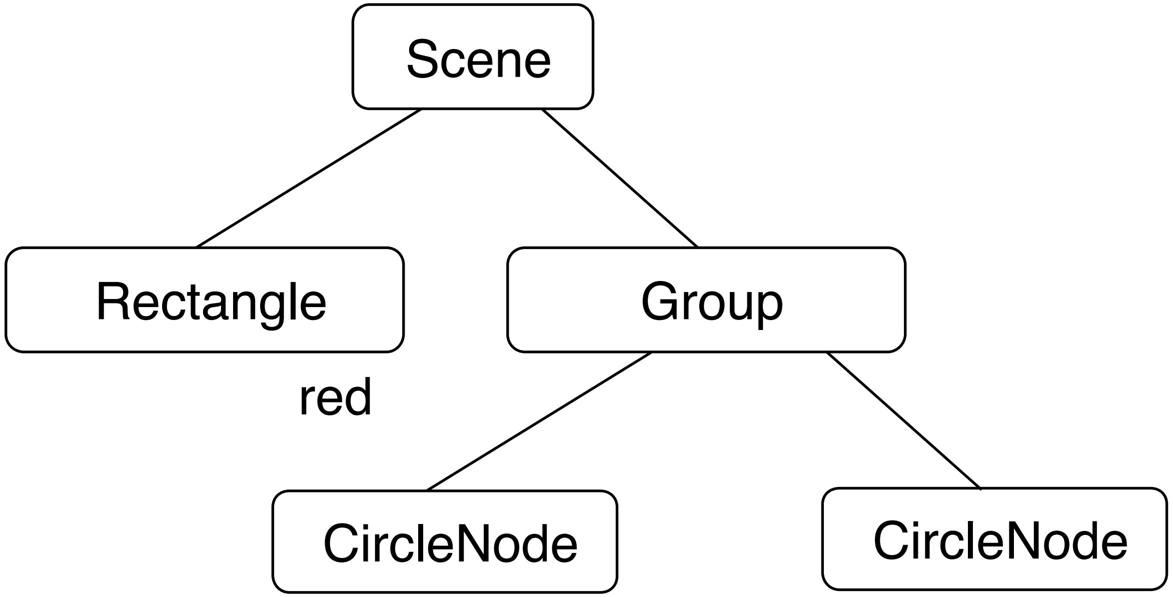


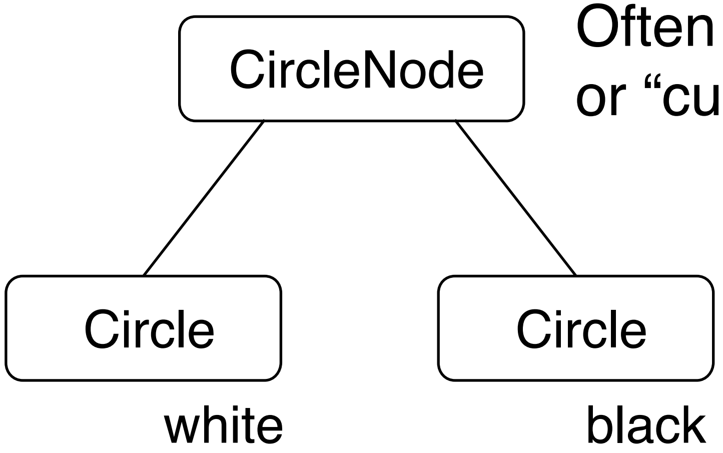
```
Group root = new Group();
Scene scene = new Scene(root, 250, 100);

Rectangle r = new Rectangle(10, 10, 230, 80);
r.setFill(Color.RED);
root.getChildren().add(r);

Group circles = new Group();
circles.setTranslateX(10);
circles.setTranslateY(10);
root.getChildren().add(circles);

Circle circle1 = new Circle(40, 40, 40);
circle1.setFill(Color.WHITE);
circles.getChildren().add(circle1);

Circle circle2 = new Circle(190, 40, 40);
circle2.setFill(Color.WHITE);
circles.getChildren().add(circle2);

primaryStage.setTitle("JavaFX Scene Graph");
primaryStage.setScene(scene);
```

QUIZ

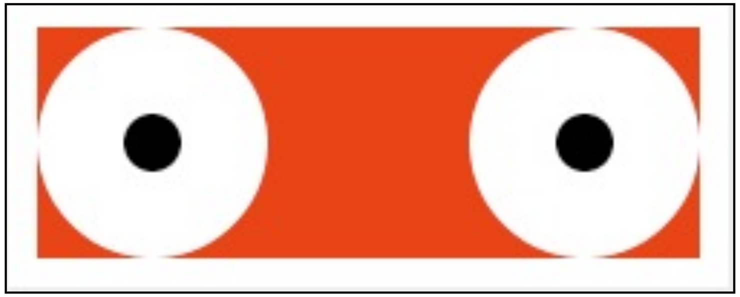
- How does the scene graph of the preceding example look like?

Object-Oriented Scene Graph Example





Often called “custom component” or “custom node"



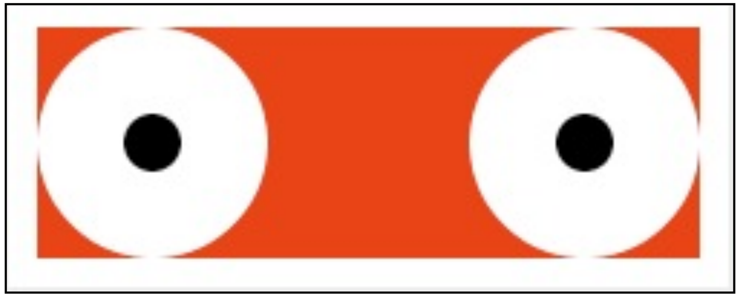
Object-Oriented Scene Graph in JavaFX (1)

```
class CircleNode extends Parent {
      public CircleNode() {
          Circle circle1 = new Circle(40, 40, 40);
          circle1.setFill(Color.WHITE);
          this.getChildren().add(circle1);
          Circle circle2 = new Circle(40, 40, 10);
          circle2.setFill(Color.BLACK);
          this.getChildren().add(circle2);
      }
 }
```

Coordinates relative to *local* coordinate system!

Object-Oriented Scene Graph in JavaFX (2)

```
Group root = new Group();
Scene scene = new Scene(root, 250, 100);
Rectangle r = new Rectangle(10, 10, 230, 80);
r.setFill(Color.RED);
root.getChildren().add(r);
CircleNode c1 = new CircleNode();
CircleNode c2 = new CircleNode();
c2.setTranslateX(150);
Group twoCircles = new Group();
twoCircles.getChildren().add(c1);
twoCircles.getChildren().add(c2);
twoCircles.setTranslateX(10);
twoCircles.setTranslateY(10);
root.getChildren().add(twoCircles);
```

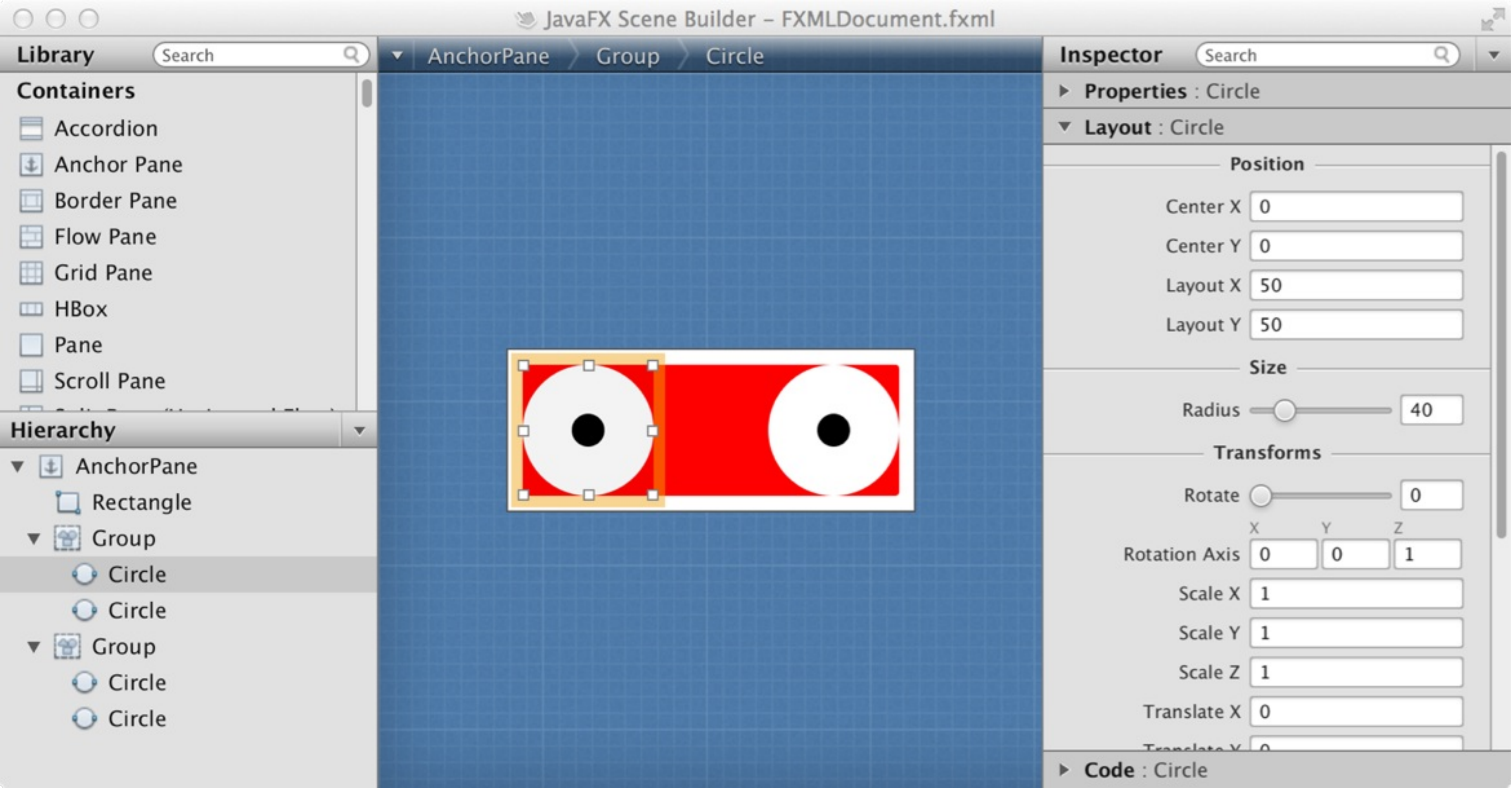


FXML: Markup Language for Scene Graphs

```
<AnchorPane id="AnchorPane" prefHeight="100.0" prefWidth="250.0"
  xmlns:fx="http://javafx.com/fxml/1" xmlns="http://javafx.com/javafx/2.2">
  <children>
    <Rectangle fill="RED" height="80.0" width="230.0"
       layoutX="10.0" layoutY="10.0"
       />
    <Group>
        <children>
            <Circle fill="WHITE" layoutX="50.0" layoutY="50.0" radius="40.0"/>
            <Circle fill="BLACK" layoutX="50.0" layoutY="50.0" radius="10.0"/>
        </children>
    </Group>
    <Group>
        <children>
            <Circle fill="WHITE" layoutX="200.0" layoutY="50.0" radius="40.0"/>
            <Circle fill="BLACK" layoutX="200.0" layoutY="50.0" radius="10.0"/>
        </children>
    </Group>
  </children>
</AnchorPane>
```

Formatting information can alternatively be kept in CSS-like style sheets

JavaFX Scene Builder Tool



Position

Center X: 0
Center Y: 0
Layout X: 50
Layout Y: 50

Size

Radius: 40

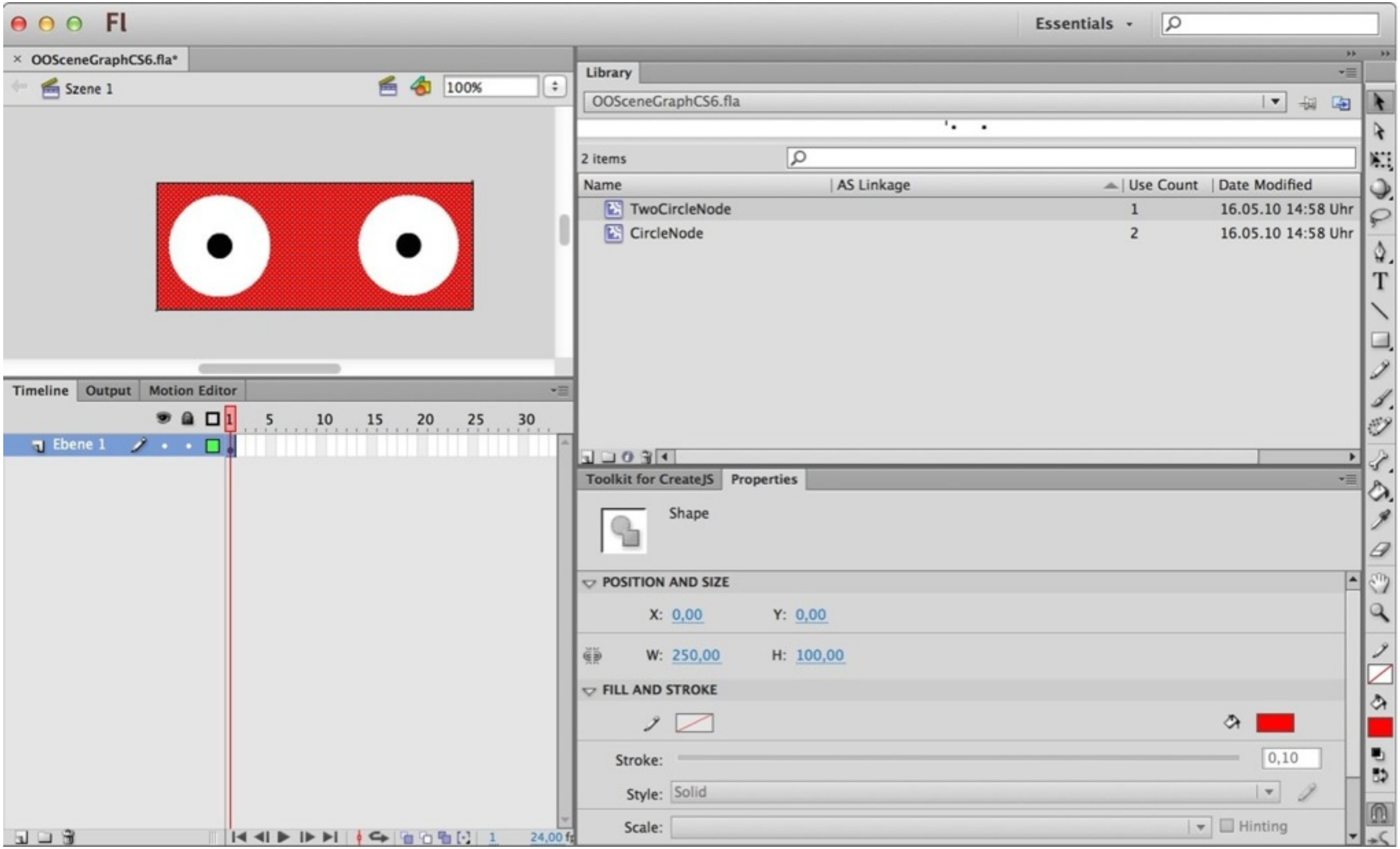
Transforms

Rotate: 0

Rotation Axis: X 0, Y 0, Z 1

Scale X: 1
Scale Y: 1
Scale Z: 1
Translate X: 0

Object-Oriented Scene Graph in Flash (1)



2 items

Name	AS Linkage	Use Count	Date Modified
TwoCircleNode		1	16.05.10 14:58 Uhr
CircleNode		2	16.05.10 14:58 Uhr

Shape

POSITION AND SIZE

X: 0,00 Y: 0,00

W: 250,00 H: 100,00

FILL AND STROKE

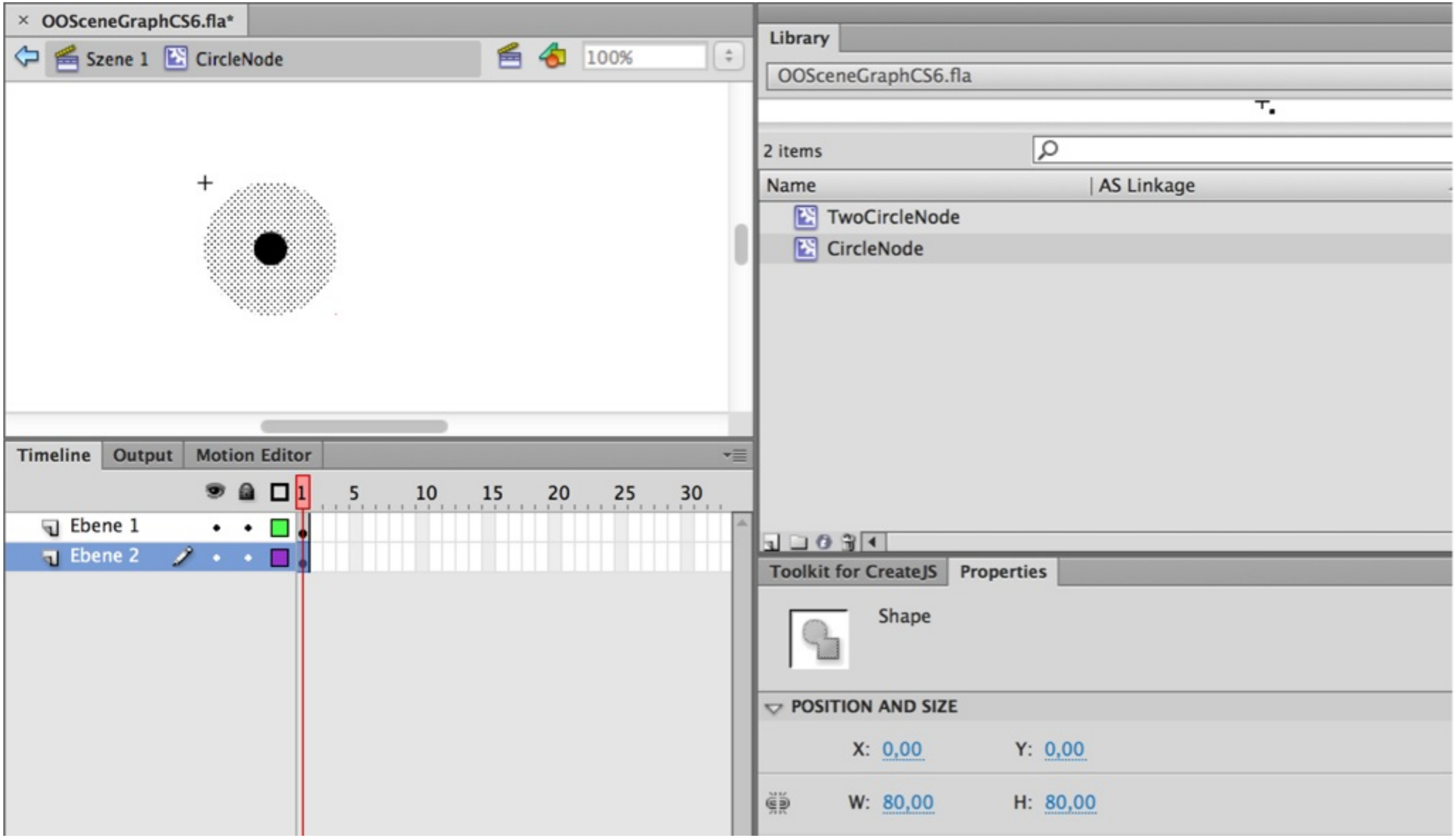
Stroke: 0,10

Style: Solid

Scale:

□ Hinting

Object-Oriented Scene Graph in Flash (2)

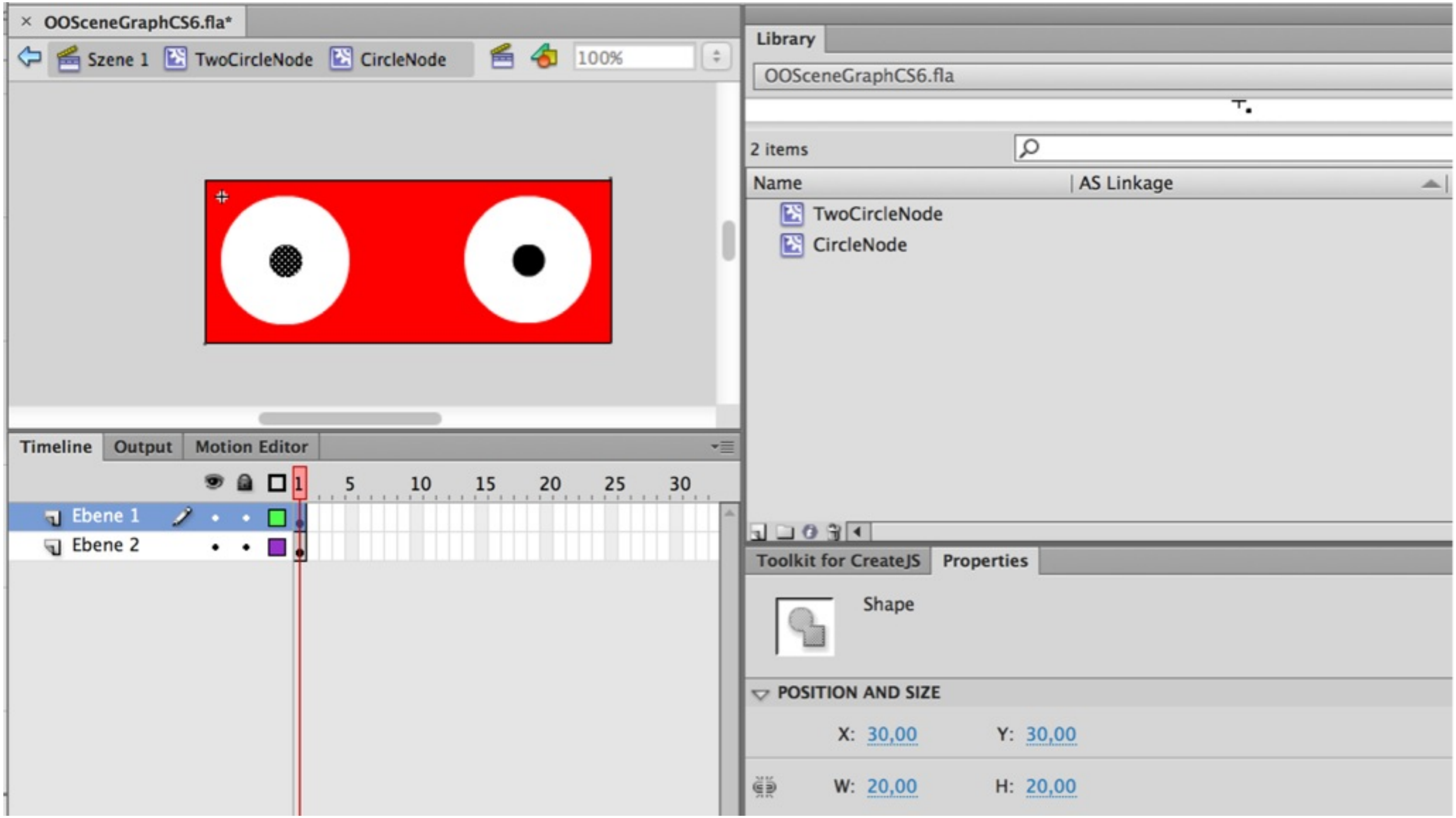


TwoCircleNode
CircleNode

X: 0,00 Y: 0,00

W: 80,00 H: 80,00

Object-Oriented Scene Graph in Flash (3)



OOSceneGraphCS6.fla

2 items

Name	AS Linkage
TwoCircleNode	
CircleNode	

Shape

POSITION AND SIZE

X: 30,00 Y: 30,00

W: 20,00 H: 20,00

QUIZ

- Where is the scene graph in the Flash example?

Scene Graph: Outlook

- Scene graphs are used in many drawing programs
  - Illustrator, CorelDraw
- Scene graphs are a main concept in 3D modeling and programming
  - VRML, X3D, OpenSceneGraph
  - www.openscenegraph.org
  - Python language binding for OpenSceneGraph exists

6 Programming with Images

6.1 Graphics and Pictures Across Platforms

6.2 Displaying Static Vector/Bitmap Graphics

6.3 Structured Graphics: Display Lists, Scene Graphs

6.4 Sprites

Literature:

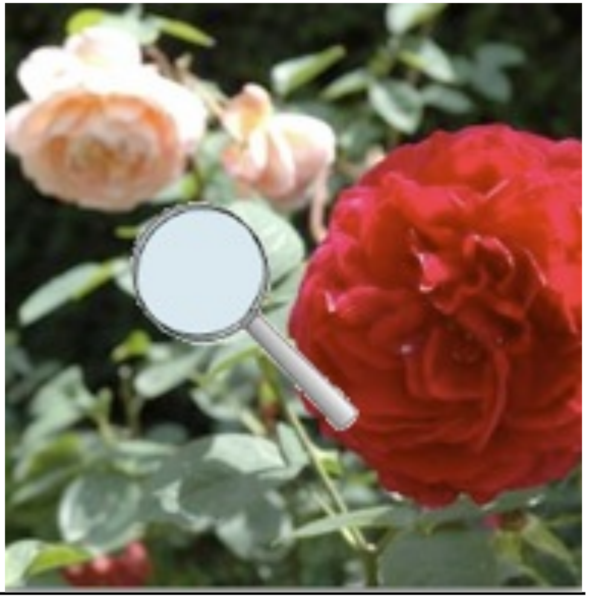
Will McGugan: Beginning Game Development with Python and Pygame, Apress 2007

Sprite

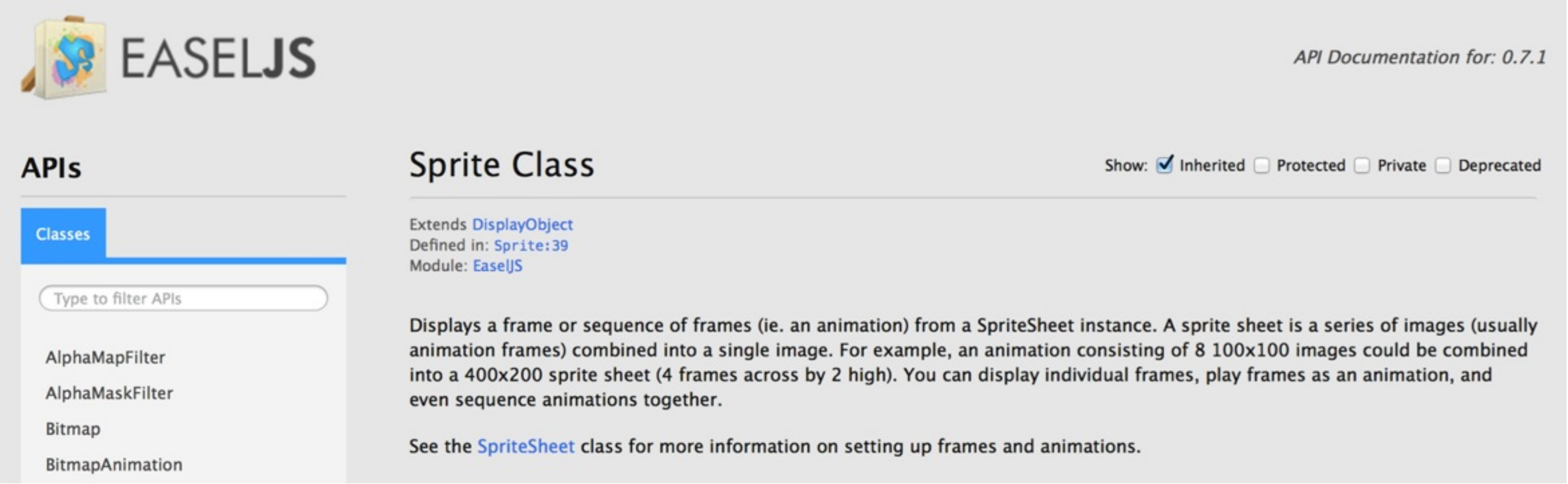
- A *sprite (Kobold, Geist)* is a movable graphics object which is presented on top of the background image.
  - Mouse pointer images are examples of sprites
- Hardware sprite:
  - Outdated technique for hardware-supported fast display of moving image
- Software sprite:
  - Any moving picture displayed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



Sprites in CreateJS/EaselJS



Sprite Class

Show: ☑ Inherited □ Protected □ Private □ Deprecated

Extends DisplayObject
Defined in: `Sprite:39`
Module: EaselJS

Displays a frame or sequence of frames (ie. an animation) from a SpriteSheet instance. A sprite sheet is a series of images (usually animation frames) combined into a single image. For example, an animation consisting of 8 100x100 images could be combined into a 400x200 sprite sheet (4 frames across by 2 high). You can display individual frames, play frames as an animation, and even sequence animations together.

See the SpriteSheet class for more information on setting up frames and animations.

- “Sprite” term used in EaselJS
- Tool for animated image sequence
  - Using a “sprite sheet”
- See Animation chapter for details