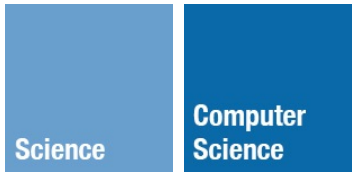
Custom Drawing & Animation

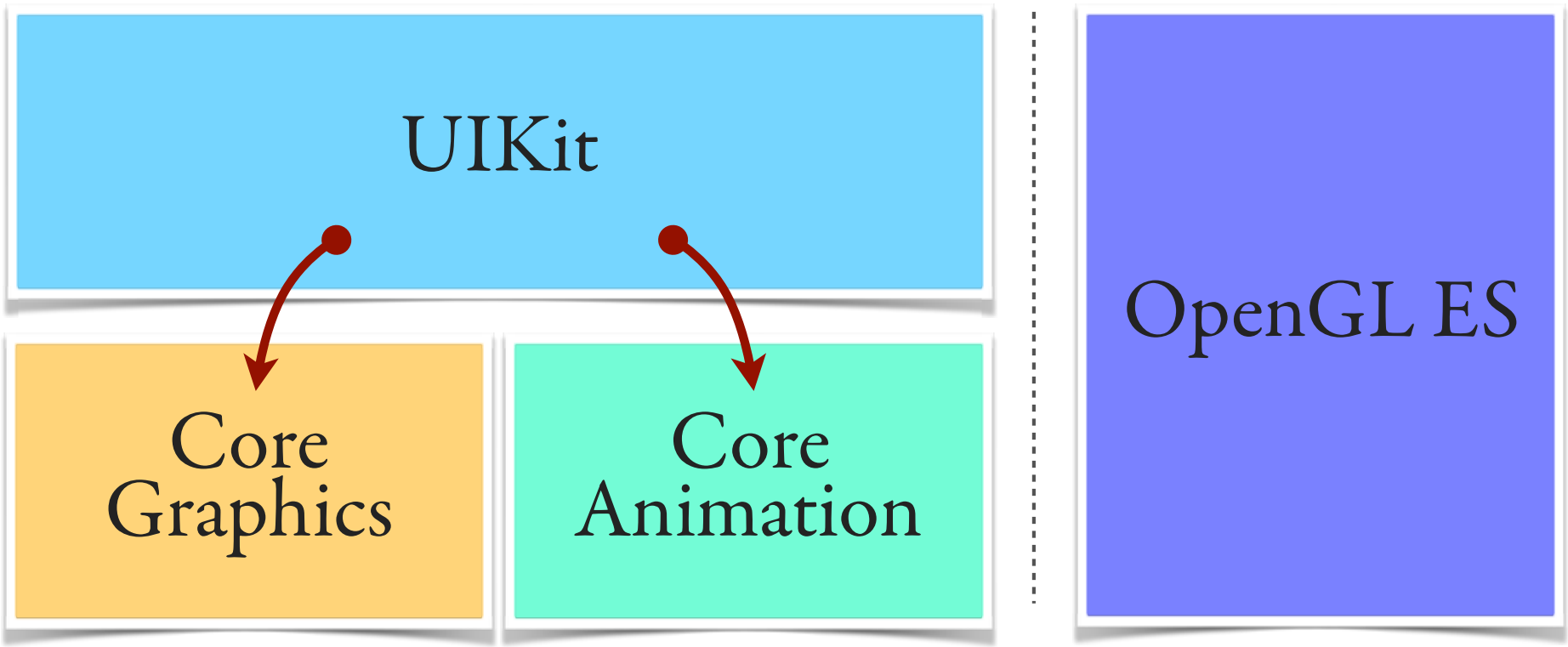


CS 442: Mobile App Development
Michael Saelee <lee@iit.edu>

IIT College of Science
ILLINOIS INSTITUTE OF TECHNOLOGY

Frameworks

- UIKit
- Core Graphics / Quartz
- Core Animation
- OpenGL ES


UIKit
Core Graphics
Core Animation
OpenGL ES

UIKit

(mostly) Swift/ObjC API

`UI...` classes/functions

UIKit

Base view class: UIView

Pre-built controls: UIControls

UIKit

typically, use concrete subclasses as is

(no need to subclass)

e.g., UILabel, UIButton, UITableView, UIImageView

UIKit

can also subclass UIView to draw custom UI elements

UIKit

support for

- 2D drawing
- transformations
- predefined transitions
- basic animation

CG

aka “Quartz 2D”

C API for drawing

CG

support for:

- layer-based graphics
- patterns, gradients, etc.
- color spaces
- working with bitmaps

CG

mostly, UIKit draws using CG

i.e., more than one way of doing anything

UIKit

```swift
// clear with white rectangle
UIColor.whiteColor().set()
UIRectFill(CGRect(x: 0, y: 0, width: 100, height: 100))

// load and draw image at (0,0)
UIImage(named: “image.png")?.drawAtPoint(CGPoint(x: 0, y: 0))
```

CG

```swift
// get current graphics context to draw into
CGContextRef context = UIGraphicsGetCurrentContext();

// clear with white rectangle
CGContextSetRGBFillColor(context, 1.0, 1.0, 1.0, 1.0);
CGContextFillRect(context, self.bounds);

// load image from file
CGDataProviderRef provider = CGDataProviderCreateWithFilename(imageFileName);
CGImageRef image = CGImageCreateWithPNGDataProvider(provider,
                                                    NULL,
                                                    true,
                                                    kCGRenderingIntentDefault);
CGDataProviderRelease(provider);

// draw image at (0,0)
CGContextDrawImage(context,
                   CGRectMake(0, 0, CGImageGetWidth(image), CGImageGetHeight(image)),
                   image);
CGImageRelease(image);
```

CA

API for animation

and *compositing*

verb [trans.] [usu. as n.] (**compositing**)
combine (two or more images) to make a single picture, esp. electronically : *photographic compositing by computer.*

CA

all UIViews are backed by CA **layers**

CA

can create a hierarchy of CALayers
within a single view

(in addition to creating a hierarchy of views)

CA

generally, layers are:

- more lightweight
- more flexible
- more complex

CA

CALayer properties can be automatically animated

CA

support for:

- simple value interpolation
- key frame animation
- transition effects
- animation groups

UIKit

CA

```swift
// load image and add to view at position (100,100)
let imageView = UIImageView(image: UIImage(named: "image.png"))
imageView.center = CGPoint(x: 100, y: 100)
view.addSubview(imageView)

// animate using a block -- bounce between start position and (300,300)
UIView.animateWithDuration(5.0,
  delay: 0.0,
  options: .Repeat | .Autoreverse,
  animations: {
    view.center = CGPoint(x: 300, y: 300)
  },
  completion: nil)
```

```objc
// create new CA layer and populate with image
CALayer *layer = [CALayer layer];
UIImage *image = [UIImage imageNamed:@"image.png"];
layer.contents = image.CGImage;
layer.frame    = CGRectMake(0, 0, image.size.width, image.size.height);

// add layer to view layer
[self.view.layer addSublayer:layer];

// create basic animation to interpolate position between (100,100) and (300,300)
CABasicAnimation *anim = [CABasicAnimation animationWithKeyPath:@"position"];
anim.fromValue = [NSValue valueWithCGPoint:CGPointMake(100, 100)];
anim.toValue = [NSValue valueWithCGPoint:CGPointMake(300, 300)];
anim.duration = 5.0;
anim.autoreverses = YES;
anim.repeatCount = HUGE_VALF;
anim.timingFunction = [CAMediaTimingFunction functionWithName:kCAMediaTimingFunctionEaseInEaseOut];
[layer addAnimation:anim forKey:@"backandforth"];
```

OpenGL

industry standard 2D/3D graphics API

OpenGL

technically, OpenGL ES;
i.e., for **E**mbedded **S**ystems

OpenGL

OpenGL ES 2.0 not backwards compatible (fixed-function vs. shaders)

OpenGL

hardware acceleration

iPad 2: CPUx2, **GPUx9**,

iPad 3: CPUx1, **GPUx2-3**, etc.

OpenGL

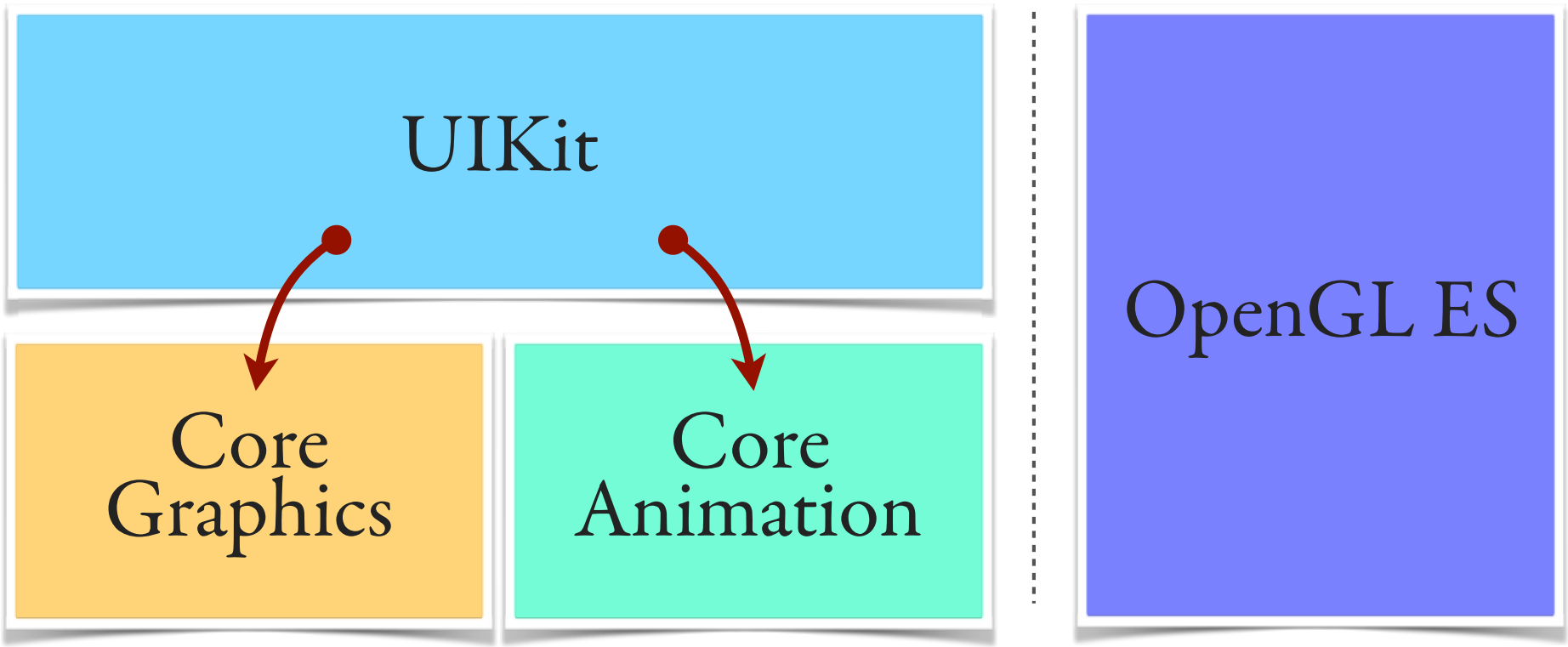
OpenGL render destination:

CAEAGLLayer in UIView

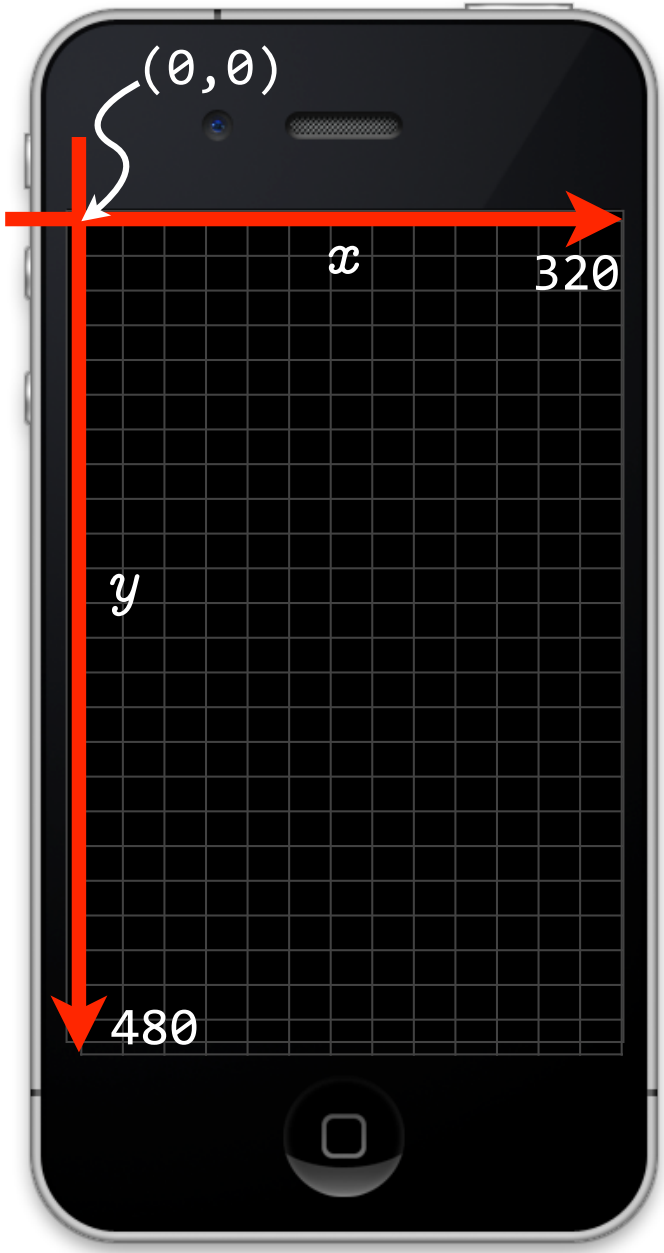
OpenGL

generally, don't mix OpenGL and UIKit/CA/CG functions

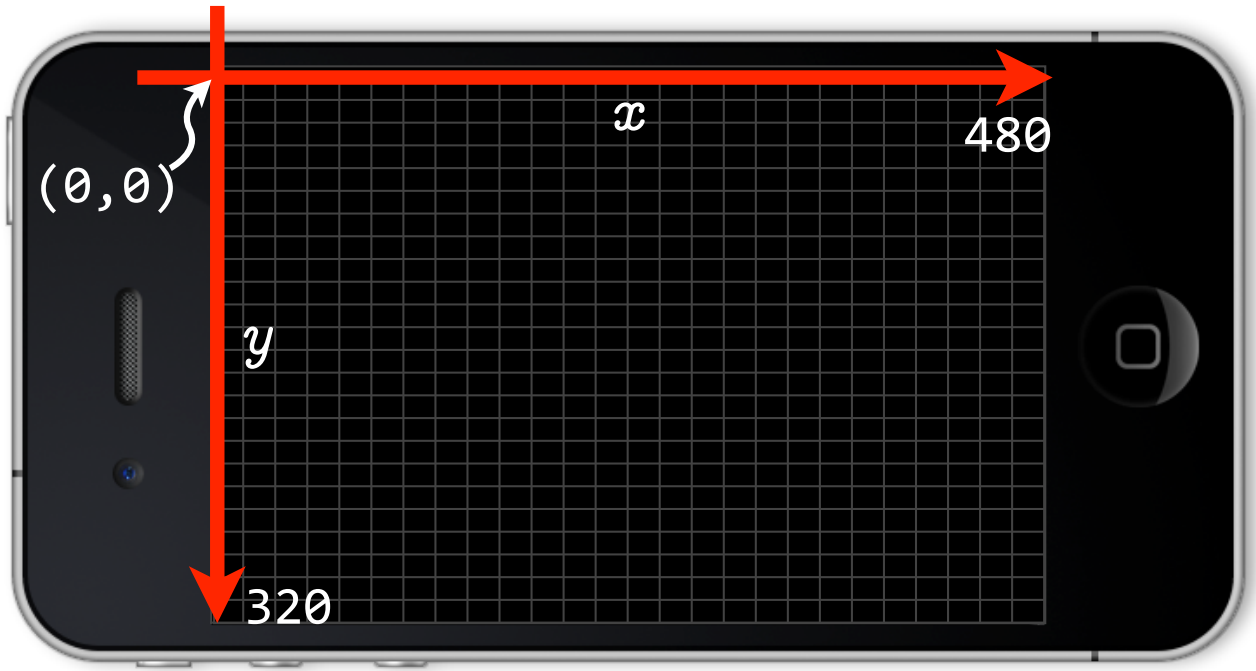
(e.g., no layer transforms)


UIKit
Core Graphics
Core Animation
OpenGL ES

§ Drawing



“ULO”



“ULO”

320 x 480 **points**

(not necessarily = pixels!)

≈ resolution independence

scale factor × points = pixels

(retina display: scale = 2.0)

principal data types:
CG**Point**, CG**Size**, CG**Rect**

```
/* Points. */
struct CGPoint {
  var x: CGFloat
  var y: CGFloat
}
```

```
/* Sizes. */
struct CGSize {
  var width: CGFloat
  var height: CGFloat
}
```

```
/* Rectangles. */
struct CGRect {
  var origin: CGPoint
  var size: CGSize
}
```

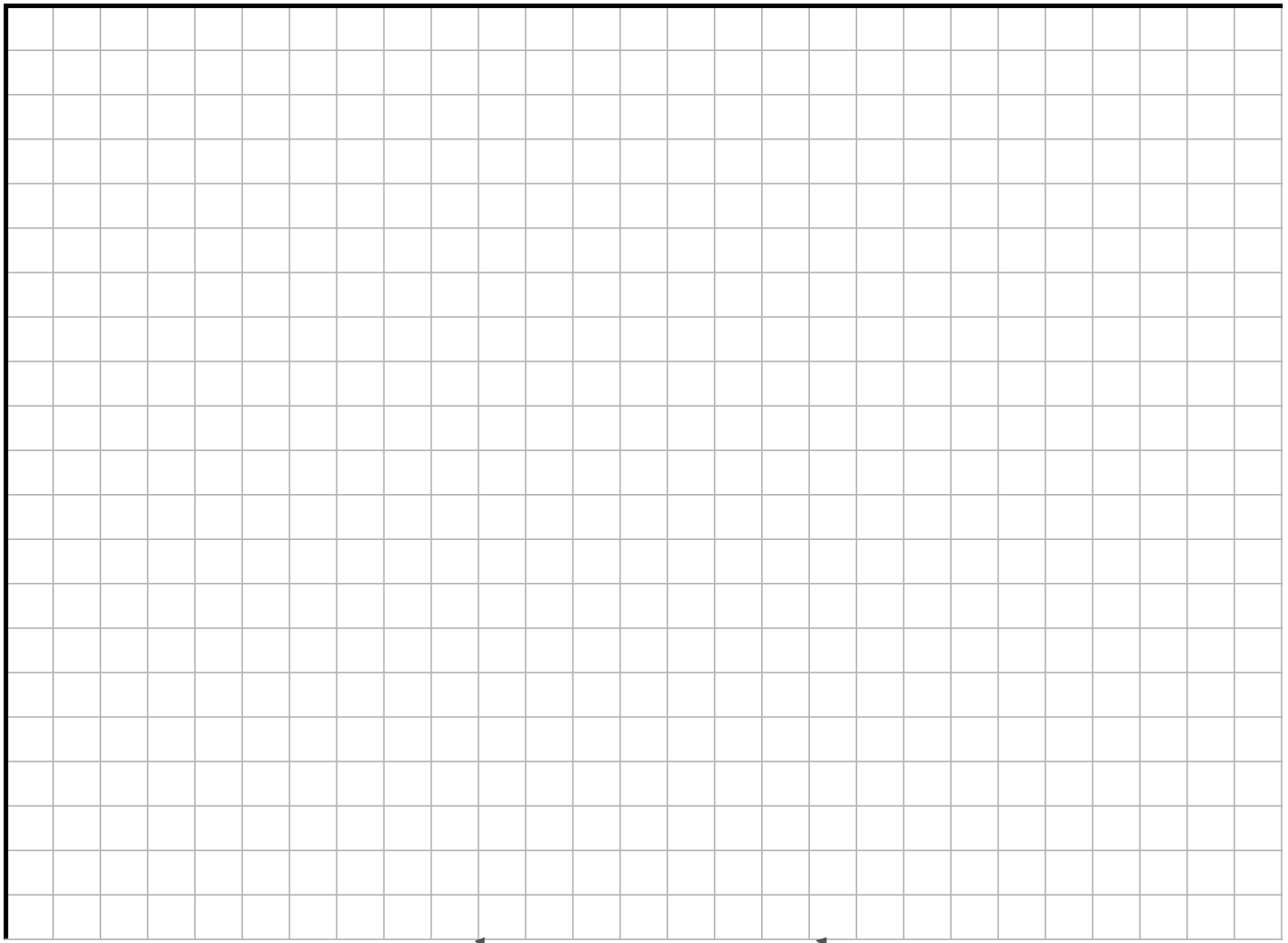
```
/* Return the left/mid/right x-value of ‘rect’. */
CGFloat CGRectGetMinX(CGRect rect);
CGFloat CGRectGetMidX(CGRect rect);
CGFloat CGRectGetMaxX(CGRect rect);

/* Return the top/mid/bottom y-value of ‘rect’. */
CGFloat CGRectGetMinY(CGRect rect);
CGFloat CGRectGetMidY(CGRect rect);
CGFloat CGRectGetMaxY(CGRect rect);

/* Return the width/height of ‘rect’. */
CGFloat CGRectGetWidth(CGRect rect);
CGFloat CGRectGetHeight(CGRect rect);


/* Standardize ‘rect’ -- i.e., convert it to an
   equivalent rect which has positive width and height. */
CGRect CGRectStandardize(CGRect rect);

/* Return true if ‘rect’ is empty (that is, if it has
   zero width or height), false otherwise. A null rect
   is defined to be empty. */
bool CGRectIsEmpty(CGRect rect);
```

locating/placing things:

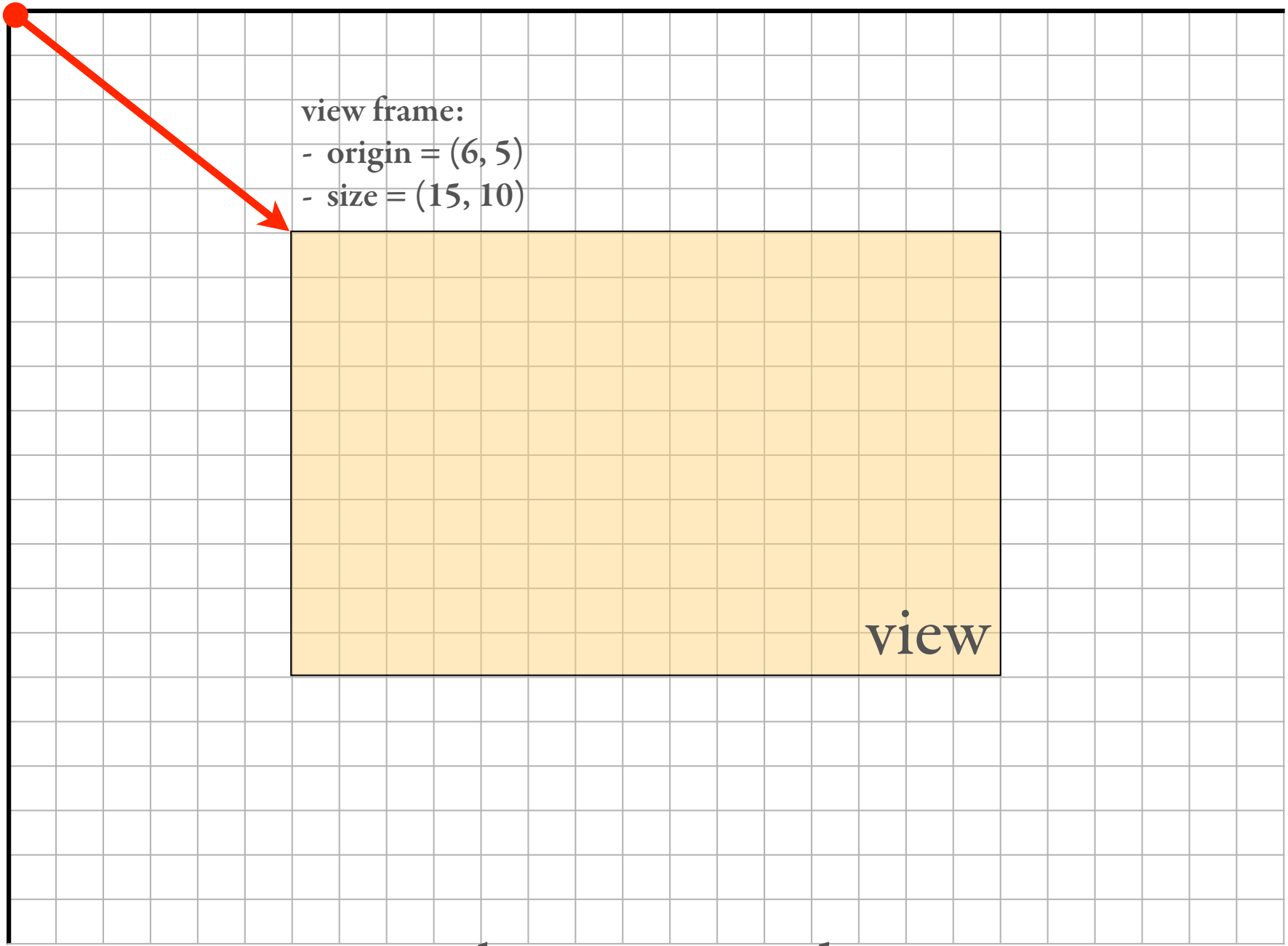
frame & **bounds** rectangles

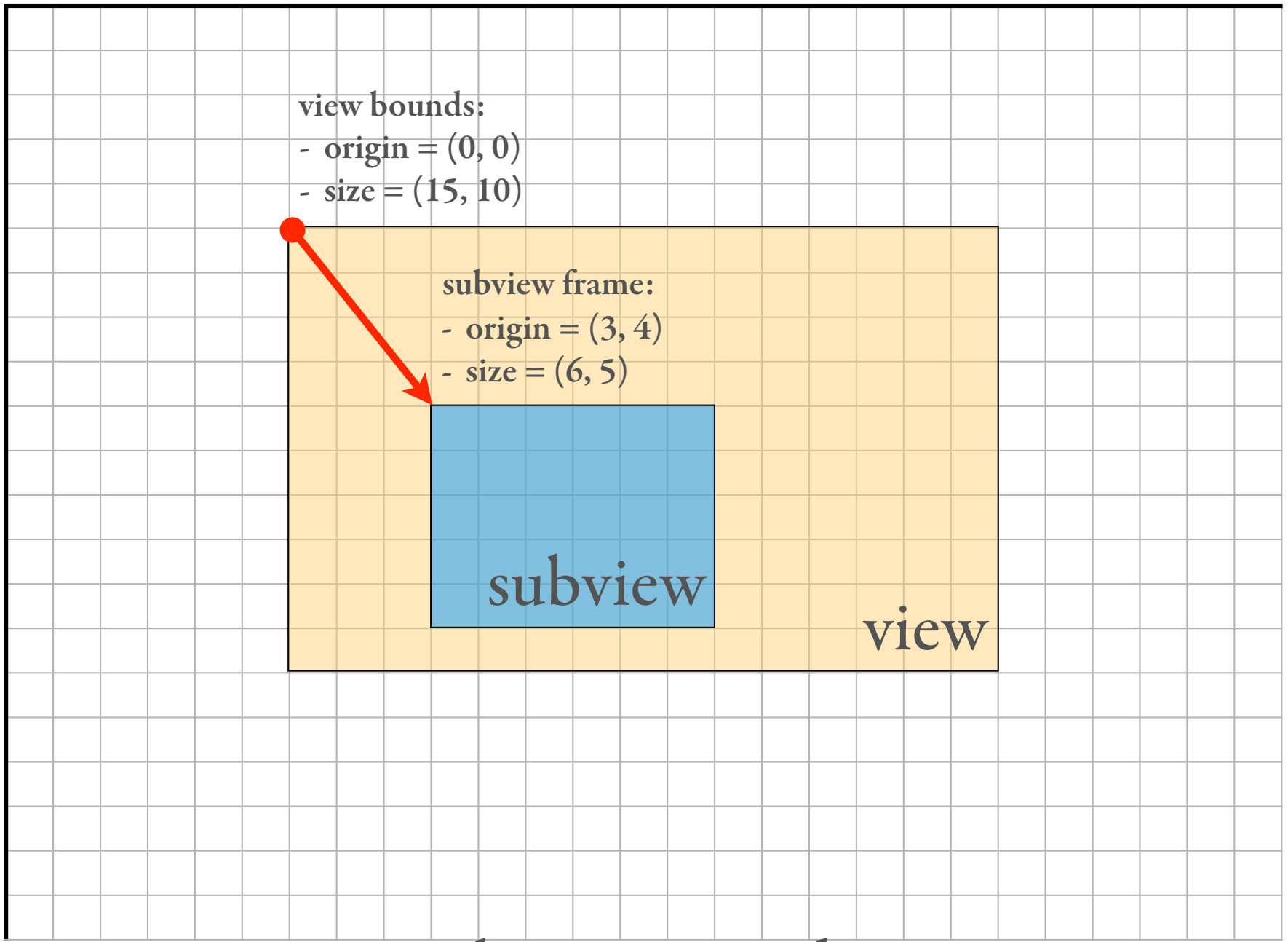
frame = origin & size in
superview's coordinate system

bounds = origin & size in
local view's coordinate system



application window


view frame:
- origin = (6, 5)
- size = (15, 10)
view
application window

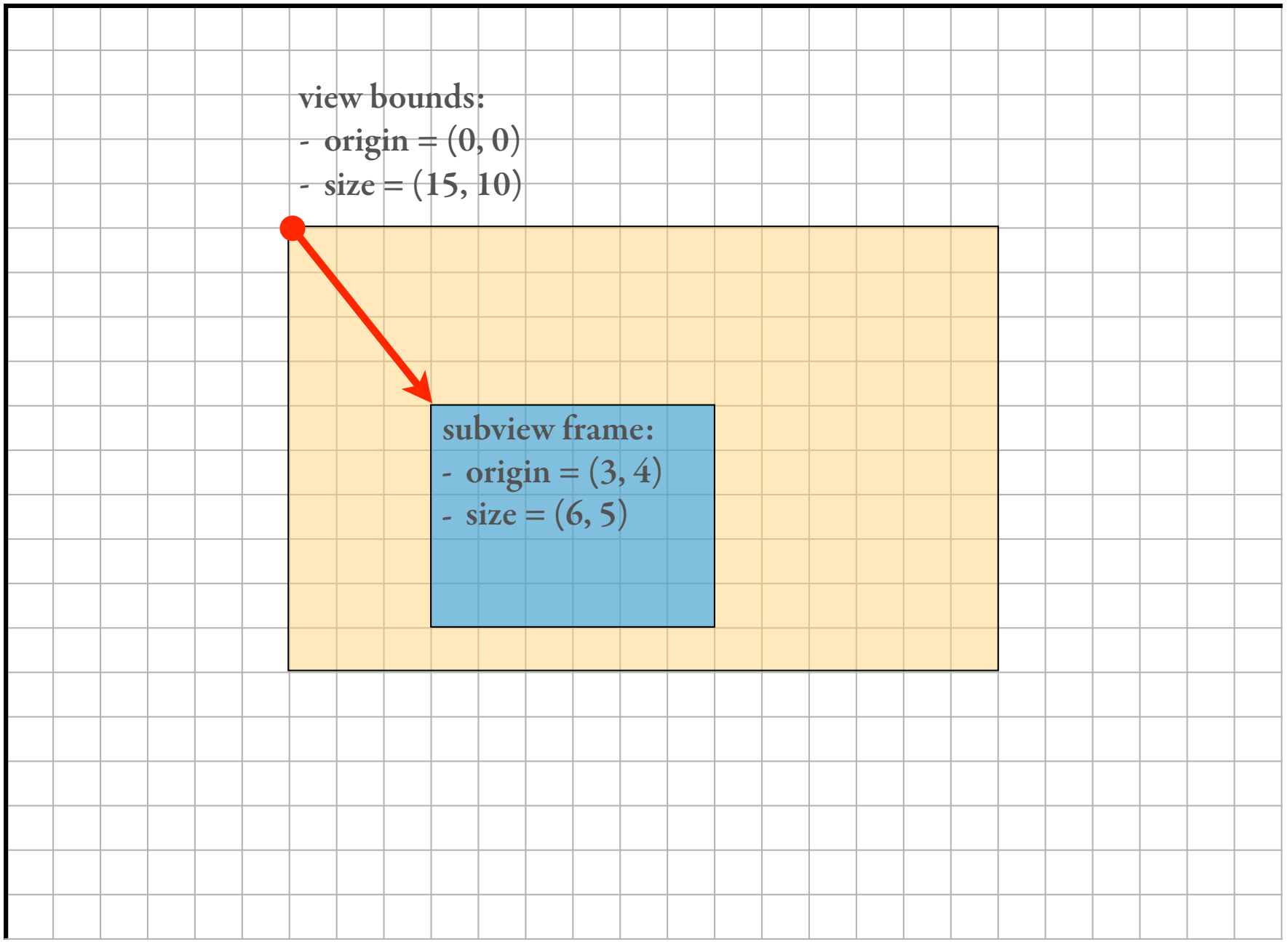

view bounds:
- origin = (0, 0)
- size = (15, 10)
subview frame:
- origin = (3, 4)
- size = (6, 5)
subview
view
application window

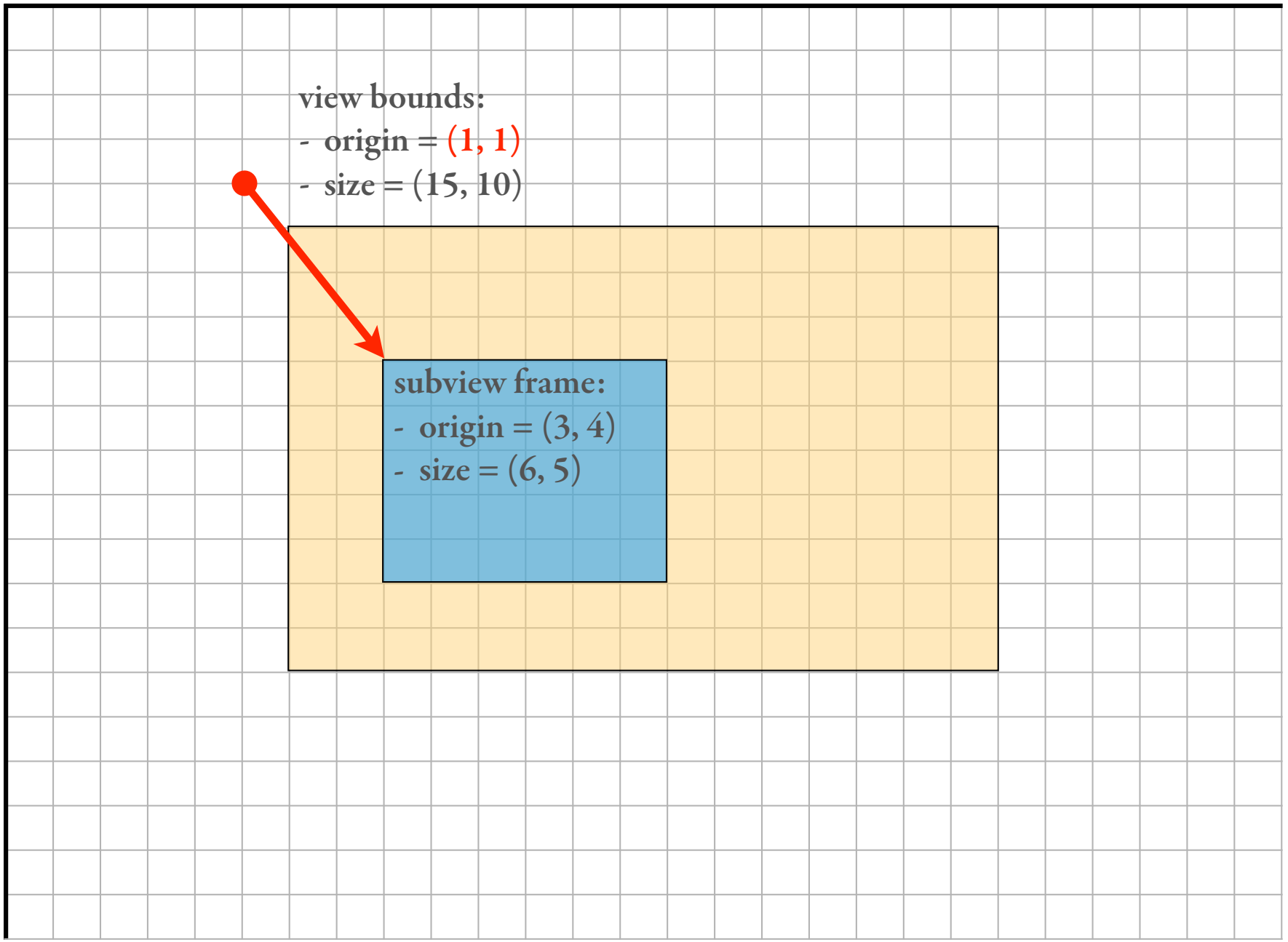
size of frame, bounds are automatically linked

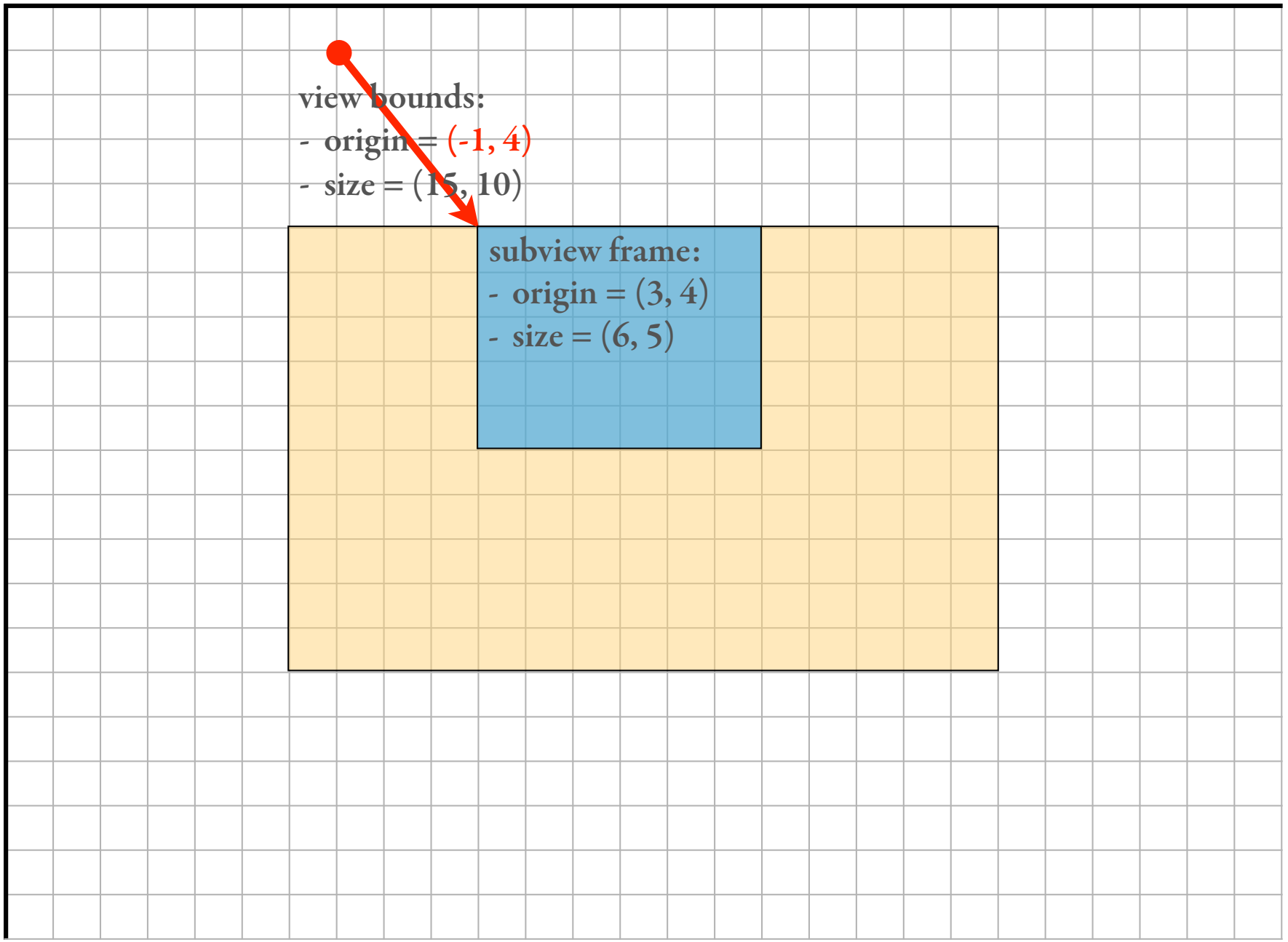
reposition view by changing
frame **origin** or **center**

(changing one automatically adjusts the other)

can change bounds origin to adjust
local coordinate system


view bounds:
- origin = (0, 0)
- size = (15, 10)
subview frame:
- origin = (3, 4)
- size = (6, 5)


view bounds:
- origin = (1, 1)
- size = (15, 10)
subview frame:
- origin = (3, 4)
- size = (6, 5)


view bounds:
- origin = (-1, 4)
- size = (15, 10)
subview frame:
- origin = (3, 4)
- size = (6, 5)

```
/* Fill `rect' with solid color */
void UIRectFill(CGRect rect);
void UIRectFillUsingBlendMode(CGRect rect, CGBlendMode blendMode);

/* Draw 1px frame inside `rect'. */
void UIRectFrame(CGRect rect);
void UIRectFrameUsingBlendMode(CGRect rect, CGBlendMode blendMode);
```

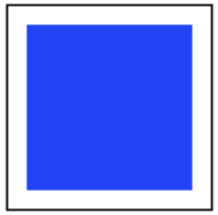
Simple Drawing

```
@interface UIColor
// Convenience methods for creating autoreleased colors
+ (UIColor *)colorWithRed:(CGFloat)red
                    green:(CGFloat)green
                     blue:(CGFloat)blue
                    alpha:(CGFloat)alpha;

// Some convenience methods to create colors. These colors are cached.
+ (UIColor *)blackColor;      // 0.0 white
+ (UIColor *)redColor;        // 1.0, 0.0, 0.0 RGB
+ (UIColor *)greenColor;      // 0.0, 1.0, 0.0 RGB
+ (UIColor *)blueColor;       // 0.0, 0.0, 1.0 RGB
+ (UIColor *)clearColor;      // 0.0 white, 0.0 alpha

// Set the color: Sets the fill and stroke colors in the current drawing context.
- (void)set;

// Set the fill or stroke colors individually.
- (void)setFill;
- (void)setStroke;

// Access the underlying CGColor
@property(nonatomic,readonly) CGColorRef CGColor;
@end
```

Color?

UIKit framework always draws to implicit, current **Graphics Context**

```swift
// establish current drawing context (image buffer)
UIGraphicsBeginImageContext(CGSize(width: 100, height: 100))

// clear background with white box
UIColor.whiteColor().set()
UIRectFill(CGRect(x: 0, y: 0, width: 100, height: 100))

// draw black frame
UIColor.blackColor().set()
UIRectFrame(CGRect(x: 0, y: 0, width: 100, height: 100))

// draw (filled) blue rectangle
UIColor.blueColor().set()
UIRectFill(CGRect(x: 10, y: 10, width: 80, height: 80))

// extract image from context
let image = UIGraphicsGetImageFromCurrentImageContext()
UIGraphicsEndImageContext()
```



CG maintains a *stack* of graphics contexts (empty, by default)

UIView objects automatically push graphics contexts before calling `drawRect:`

```objc
@interface UIView(UIViewRendering)
/* All custom drawing must happen from this method.
   Should limit drawing to 'rect' -- on first call
   'rect' is usually equal to our bounds. */
- (void)drawRect:(CGRect)rect;

/* drawRect: is called lazily. If view must be redrawn,
   we must notify the system by calling one of these
   methods below. */
- (void)setNeedsDisplay;
- (void)setNeedsDisplayInRect:(CGRect)rect;
@end
```

Q: draw in current view vs. adding subview?

- it depends …
- subviews allow us to add/remove objects
  - but adds memory + processing overhead

HelloWorldView
discuss root view frame origin/size

Rectangle1
using setNeedsDisplay to force refresh

Rectangle2
use multiple views to track drawn rectangles
ignore contentStretch for now

GameBoard
- handling

demo

subview size > superview?

default: container views *don't clip* subviews

… but no "scrolling", either

- to implement, parent may adjust *its own bounds* to move children into view
  - or, alternatively, change all child frames (much messier!)

Clipping Demo
- effect of clipsToBounds

ScrollView Demo
automatic pan support based on contentSize
also: pinch to zoom

demo

view *transformations*:

translating, scaling, rotating

common strategy:

- draw “unit” object at (0,0)
- translate, scale, rotate to
  final position/size/orientation

drawRect: can be *very* lazy

i.e., draw once, transform many times

implementation: “affine transform matrices”

transformation matrix

transformed coordinates

$$\begin{bmatrix} x' & y' & 1 \end{bmatrix} = \begin{bmatrix} x & y & 1 \end{bmatrix} \times \begin{bmatrix} a & b & 0 \\ c & d & 0 \\ t_x & t_y & 1 \end{bmatrix}$$

original coordinates

$$x' = ax + cy + t_x$$

$$y' = bx + dy + t_y$$

e.g.
$$\begin{bmatrix} x' & y' & 1 \end{bmatrix} = \begin{bmatrix} x & y & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ t_x & t_y & 1 \end{bmatrix}$$

$$= \begin{bmatrix} x + t_x & y + t_y & 1 \end{bmatrix}$$

```
struct CGAffineTransform {
  CGFloat a, b, c, d;
  CGFloat tx, ty;
}


/* The identity transform: [ 1 0 0 1 0 0 ]. */
extern const CGAffineTransform CGAffineTransformIdentity;

/* Return a transform which translates by `(tx, ty)':
     t' = [ 1 0 0 1 tx ty ] */
CGAffineTransform CGAffineTransformMakeTranslation(CGFloat tx, CGFloat ty);

/* Return a transform which scales by `(sx, sy)':
     t' = [ sx 0 0 sy 0 0 ] */
CGAffineTransform CGAffineTransformMakeScale(CGFloat sx, CGFloat sy);

/* Return a transform which rotates by `angle' radians:
     t' = [ cos(angle) sin(angle) -sin(angle) cos(angle) 0 0 ] */
CGAffineTransform CGAffineTransformMakeRotation(CGFloat angle)

/* Concatenate translation, scaling, rotation transforms to existing matrices. */
CGAffineTransform CGAffineTransformTranslate(CGAffineTransform t, CGFloat tx, CGFloat ty);
CGAffineTransform CGAffineTransformScale(CGAffineTransform t, CGFloat sx, CGFloat sy);
CGAffineTransform CGAffineTransformRotate(CGAffineTransform t, CGFloat angle);
```

for given graphics context (e.g., in drawRect:), change *current transform matrix* (CTM)

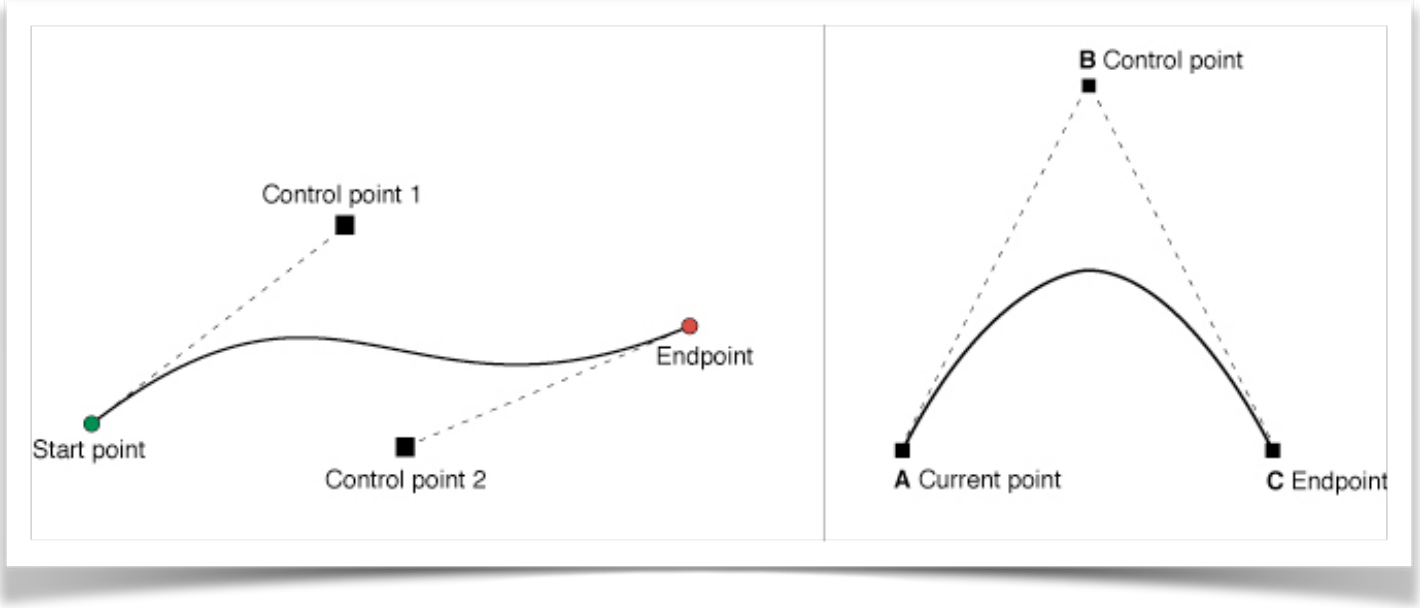
```
/* Scale the current graphics state's transformation matrix (the CTM) by
   `(sx, sy)'. */
void CGContextScaleCTM(CGContextRef c, CGFloat sx, CGFloat sy);

/* Translate the current graphics state's transformation matrix (the CTM) by
   `(tx, ty)'. */
void CGContextTranslateCTM(CGContextRef c, CGFloat tx, CGFloat ty);

/* Rotate the current graphics state's transformation matrix (the CTM) by
   `angle' radians. */
void CGContextRotateCTM(CGContextRef c, CGFloat angle);

/* Concatenate the current graphics state's transformation matrix (the CTM)
   with the affine transform `transform'. */
void CGContextConcatCTM(CGContextRef c, CGAffineTransform transform);
```

ExpandingFrame

look at RotatingView drawRect
motivate CGContextSave/RestoreGState
explore frame/bounds relationship

demo

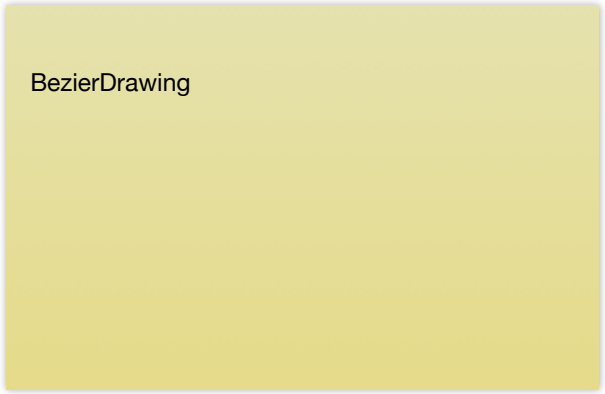
cubic & quadratic Bézier curves



Drawing Other Shapes

```objc
@interface UIBezierPath : NSObject<NSCopying, NSCoding> {

+ (UIBezierPath *)bezierPath;

- (void)moveToPoint:(CGPoint)point;
- (void)addLineToPoint:(CGPoint)point;
- (void)addCurveToPoint:(CGPoint)endPoint
          controlPoint1:(CGPoint)controlPoint1
          controlPoint2:(CGPoint)controlPoint2;
- (void)addQuadCurveToPoint:(CGPoint)endPoint
                controlPoint:(CGPoint)controlPoint;
- (void)addArcWithCenter:(CGPoint)center
                  radius:(CGFloat)radius
              startAngle:(CGFloat)startAngle
                endAngle:(CGFloat)endAngle
               clockwise:(BOOL)clockwise;
-(void)closePath;

+ (UIBezierPath *)bezierPathWithRect:(CGRect)rect;
+ (UIBezierPath *)bezierPathWithOvalInRect:(CGRect)rect;
+ (UIBezierPath *)bezierPathWithRoundedRect:(CGRect)rect
                                cornerRadius:(CGFloat)cornerRadius;

@property(nonatomic) CGPathRef CGPath;
@end
```

Bézier curves can be created, stored, and reused, independent of graphics context



demo

rects, curves, etc.

= **vector** graphics

$\Rightarrow$ infinite scaleability

raster graphics?

e.g., bitmaps, JPG, PNG

UIKit: load with UIImage

imageNamed:

Returns the image object associated with the specified filename.

```
+ (UIImage *)imageNamed:(NSString *)name
```

Parameters

name

The name of the file. If this is the first time the image is being loaded, the method looks for an image with the specified name in the application's main bundle.

Return Value

The image object for the specified file, or nil if the method could not find the specified image.

Discussion

This method looks in the system caches for an image object with the specified name and returns that object if it exists. **If a matching image object is not already in the cache, this method loads the image data from the specified file, caches it, and then returns the resulting object.**

On a device running iOS 4 or later, the behavior is identical if the device's screen has a scale of 1.0. If the screen has a scale of 2.0, this method first searches for an image file with the same filename with an @2x suffix appended to it. For example, if the file's name is button, it first searches for button@2x. If it finds a 2x, it loads that image and sets the scale property of the returned UIImage object to 2.0. Otherwise, it loads the unmodified filename and sets the scale property to 1.0.

caveat emptor: imageNamed cache can be quite aggressive!

(Google: “imageNamed cache”)

raster graphics problem: scaling → pixelation

UIView `contentStretch` property defines which parts of an image are “stretched”

demo

```
image = UIImage(named:@"image.png")
```

❶ UIKit (subview)

```
let imageView = UIImageView(image: image)
view.addSubview(imageView)
```

❷ CG (drawing)

```
func drawRect(rect: CGRect) {
    image.drawAtPoint(CGPoint(x: 0, y: 0))
}
```

❸ CA (compositing)

```
let layer = CALayer()
layer.contents = image.CGImage
view.layer.addSublayer(layer)
```

Image Drawing Options

ImageDrawing
run all three versions (view, layer, drawRect) --- may want to keep num_image < 100 for the last
run through instruments: memory allocations (num_images=10000 for view, layer version), and time profiler (esp. interesting for drawRect based)

demo

more reading (Xcode library):

- View Programming Guide for iOS
- Quartz 2D Programming Guide

§ Animation

UIKit / CoreAnimation

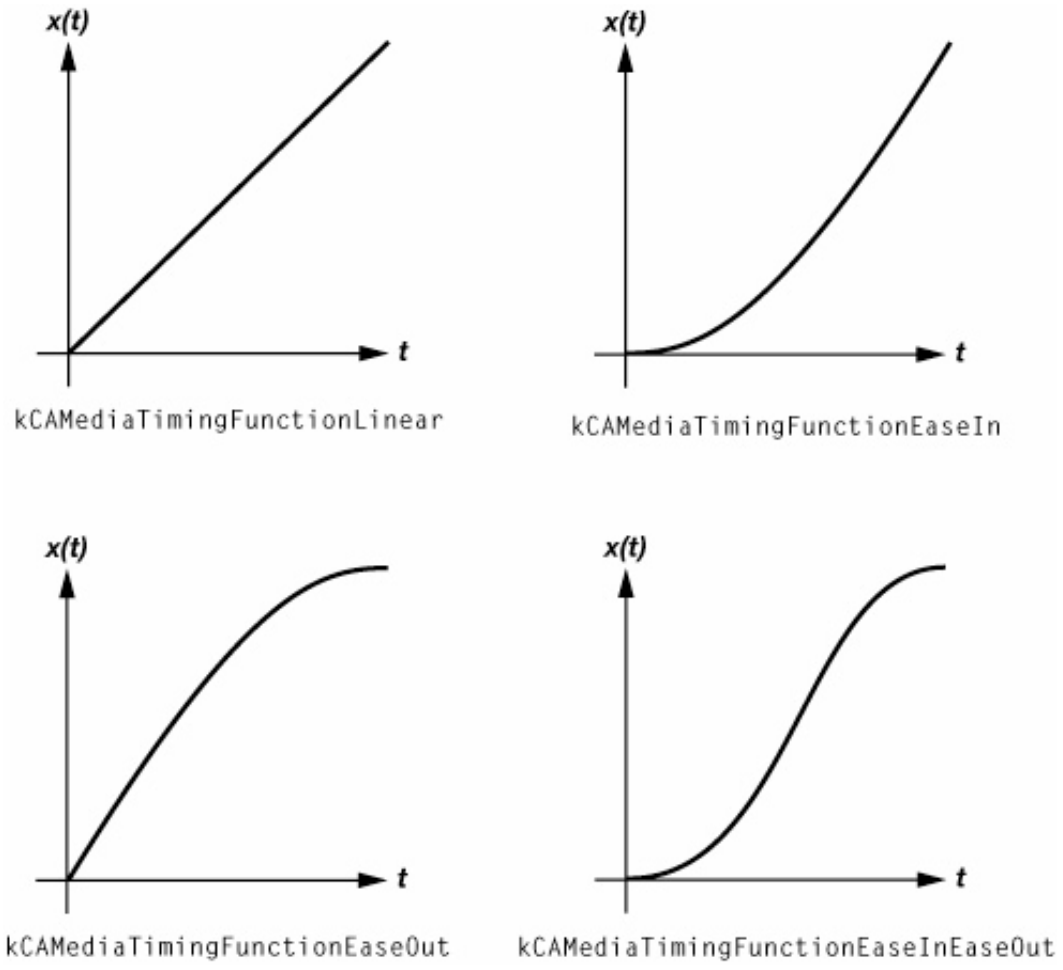
typically, use “magic” UIView / CALayer animation mechanism

i.e., provide “new” view properties in animation block along with time frame

— core animation does the rest

note: CA updates happen in a separate thread!

animated parameters are updated according to a specified *interpolation curve* (aka timing function)



timing functions