Accessibility und CoreAnimation

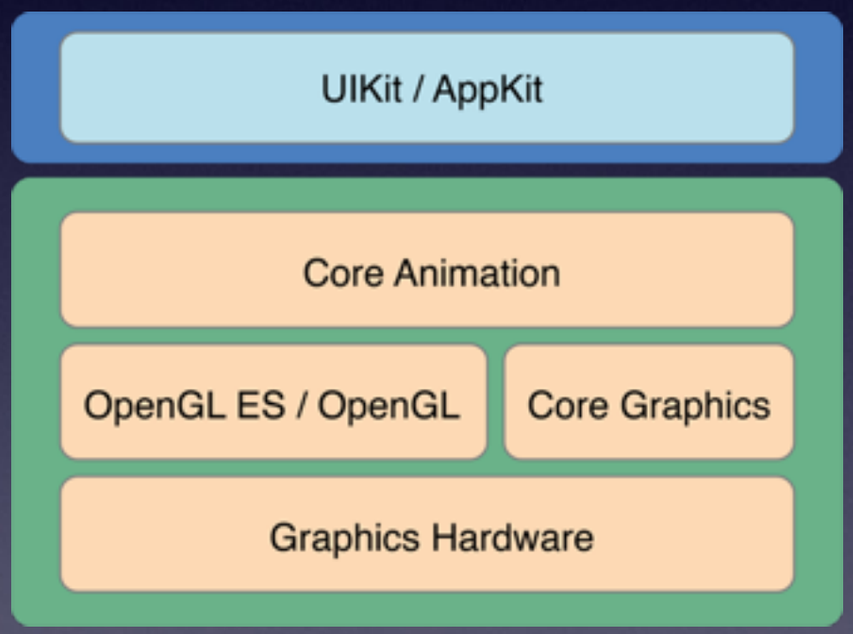
Bedienbare Controls für alle Benutzer

Warum Accessibility?

- benutzbare Oberflächen auch für Benutzer mit Behinderungen (blinde, taube oder in ihrer Bewegung eingeschränkte Menschen)
- gute Unterstützung in Mac OS und iOS bereits eingebaut
- Testbarkeit der UI

Demo

CoreAnimation



- Framework unterhalb von AppKit bzw. UIKit
- effiziente, animierte Darstellung von graphischen Inhalten
- einfach zu nutzen

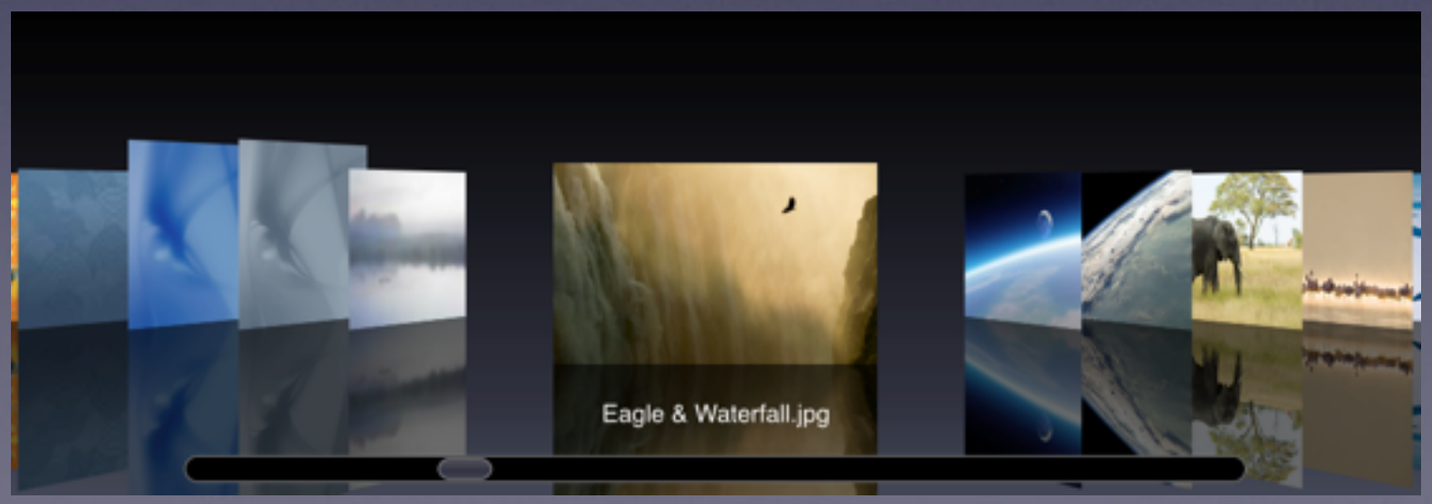
Demo

Die gute Nachricht

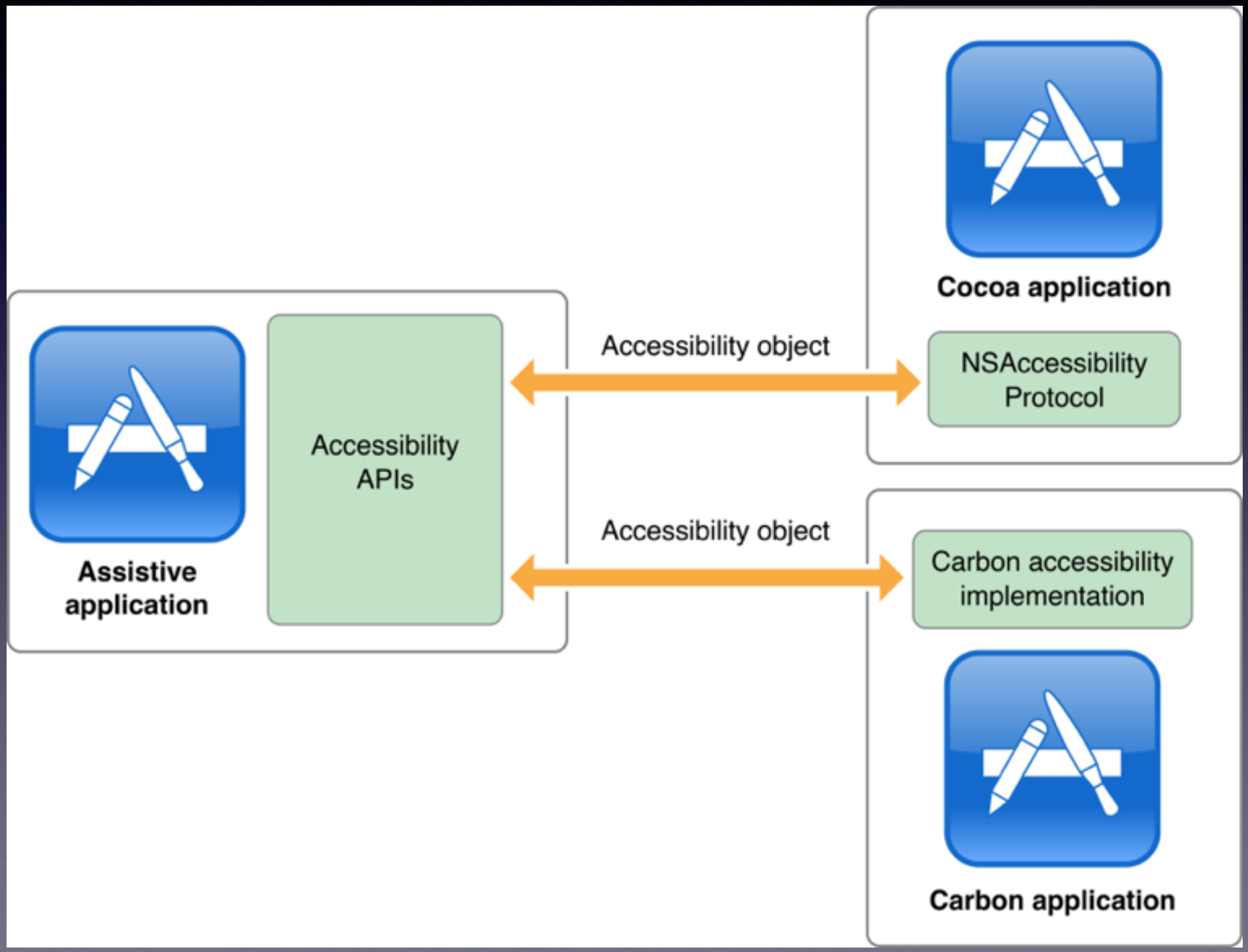
- Wenn immer möglich, die Standardcontrols benutzen
- Vorteil: Accessibility-Unterstützung bereits eingebaut
- notwendige Anpassungen sind klein und können größtenteils im IB erfolgen

Die schlechte Nachricht

- manchmal ist eine CALayer-UI notwendig (video- oder bildlastige UI)
- keine eingebaute Unterstützung für Accessibility in CALayer (NS/UIView-Äquivalent in CoreAnimation)

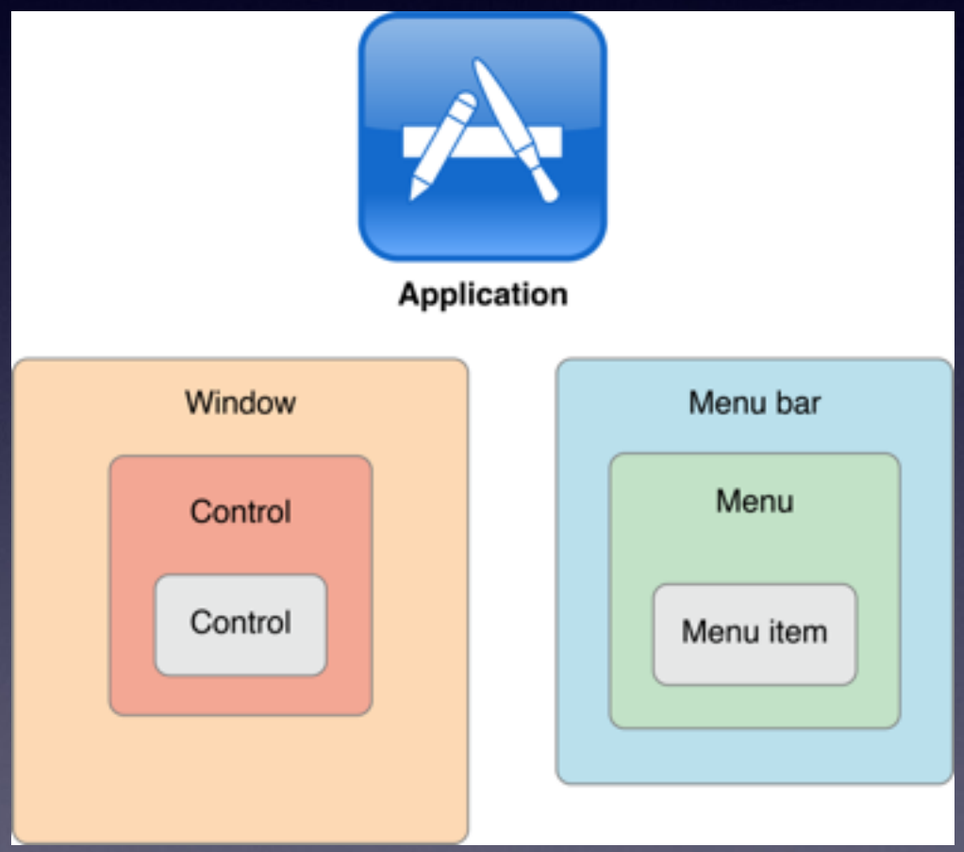


Accessibility in Cocoa



Don´t panic!

Struktur einer UI



- Übersetzung der UI in eine für das Betriebssystem auswertbare Form

NSAccessibility

- informelles Protokoll welches die Accessibility-Unterstützung bereitstellt (NSAccessibility.h)
- jedes Element der UI wird durch ein Accessibility-Objekt repräsentiert

NSAccessibility

```
@interface NSObject (NSAccessibility)

- (NSArray *)accessibilityAttributeNames;

- (id)accessibilityAttributeValue:(NSString *)attribute;

- (BOOL)accessibilityIsAttributeSettable:(NSString *)attribute;

- (void)accessibilitySetValue:(id)value forAttribute:
(NSString*)attribute;

...
```

NSAccessibility

```
...
- (NSArray *)accessibilityActionNames;
- (NSString *)accessibilityActionDescription:(NSString *)action;
- (void)accessibilityPerformAction:(NSString *)action;
...
```

NSAccessibility

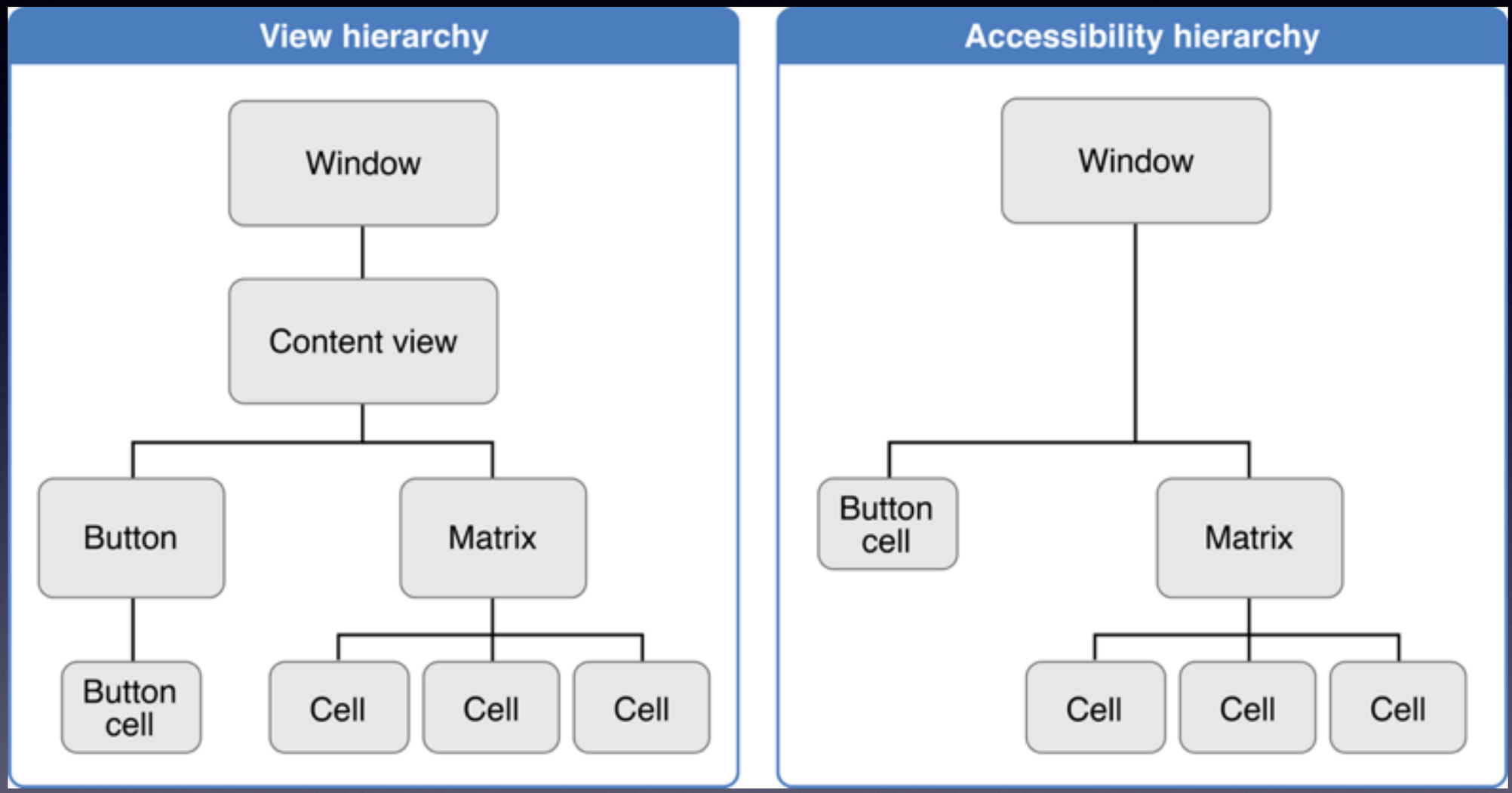
```
...
- (BOOL)accessibilityIsIgnored;

- (id)accessibilityHitTest:(NSPoint)point;

- (id)accessibilityFocusedUIElement;

@end
```

Accessibility Hierarchie

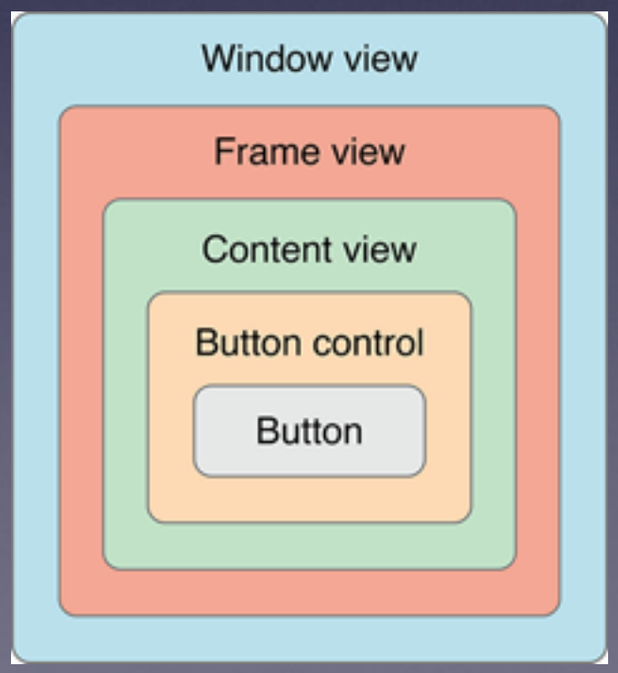


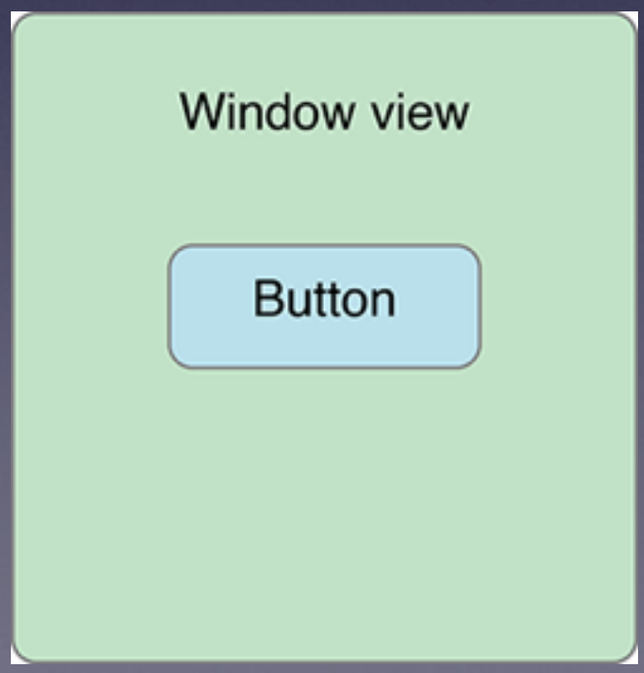
- die bereits vorhandene Viewhierachie kann verwendet werden
- u.U. wird diese vereinfacht

NSAccessibility

```
- (BOOL)accessibilityIsIgnored;
```

- YES in NSView
- Möglichkeit, Elemente zu überspringen:





Attribute

- Accessibility-Objekte werden durch Attribute beschrieben
- (Typ, Platz in der AX-Hierarchie, lokalisierte Beschreibung, Größe, Wert etc.)
- Attribute haben einen Namen und einen Wert

Attribute

- `- (NSArray*)accessibilityAttributeNames`
  - gibt ein Array aus NSStrings mit den unterstützen Attributen zurück
  - `NSAccessibilityRoleAttribute`, `NSAccessibilityValueAttribute`, `NSAccessibilityChildrenAttribute`, `NSAccessibilityTitleAttribute`, `NSAccessibilityParentAttribute` etc.
  - siehe `NSAccessibility.h`

NSAccessibilityRoleAttribute

- wichtigstes Attribut, beschreibt den Typ des UI-Elements
- NSAccessibilityTextFieldRole, NSAccessibilityButtonRole etc.
- fast immer die von Apple bereitgestellten Rollen benutzen!
- siehe NSAccessibility.h

Weitere wichtige Attribute

NSAccessibilityParentAttribute

- Elternobjekt der Accessibility-Hierarchie

NSAccessibilityChildrenAttribute

- Kindobjekte der Accessibility-Hierarchie

NSAccessibilityRoleDescriptionAttribute

- lokalisierte Beschreibung der Rolle
- NSAccessibilityRoleDescriptionForUIElement() Cocoa-Hilfsfunktion benutzen

Attribute

-(id)accessibilityAttributeValue:(NSString*)attribute

- gibt den Wert eines Attributes zurück
- zurückgebene Werte müssen je nach Typ das NSAccessibility-Protokoll unterstützen oder von einem der folgen Typen sein
- NSArray, NSAttributedString, NSData, NSDate, NSDictionary, NSNumber, NSString, NSURL oder NSValue

Attribute

- (BOOL)accessibilityIsAttributeSettable:(NSString*)attribute

- YES für beschreibbares Attribut zurückgeben (z.B. Selektion, Wert, Fokus etc.)

Attribute

- (void)accessibilitySetValue:(id)value forAttribute:(NSString*)attribute

parameterisierte Attribute

- Attribute, die einen Parameter benötigen
  - NSAccessibilityLineForIndexParameterizedAttribute, NSAccessibilityRangeForPositionParameterizedAttribute, NSAccessibilityBoundsForRangeParameterizedAttribute usw.
- Länge eines Strings, Range der Selektion, Bounds eines Wortes, Buchstaben etc.
  - - (NSArray *)accessibilityParameterizedAttributeNames
  - - (id)accessibilityAttributeValue:(NSString *)attribute forParameter:(id)parameter

Actions

- Aktionen die mit Accessibility-Objekten verknüpft sind
- generische Aktionen zur UI-Interaktion wie z.B. Mausklick, Tastaturdruck, Selektion
  - NSAccessibilityPressAction, NSAccessibilityIncrementAction, NSAccessibilityPickAction, NSAccessibilityCancelAction usw.
- von Apple zur Verfügung gestellte Aktionen benutzen

Actions

-(NSArray *)accessibilityActionNames

- Array mit NSStrings der unterstützen Aktionen

- (NSString*)accessibilityActionDescription:(NSString *)action

- lokalisierte Beschreibung der Aktion
- von Cocoa bereitgestellte Helper-Funktion NSAccessibilityActionDescription() benutzen

Actions

- (void)accessibilityPerformAction:(NSString *)action

- im Control für die normale Funktionalität bereitgestellte Funktionalität nutzen (z.B. Target/Action ausführen)

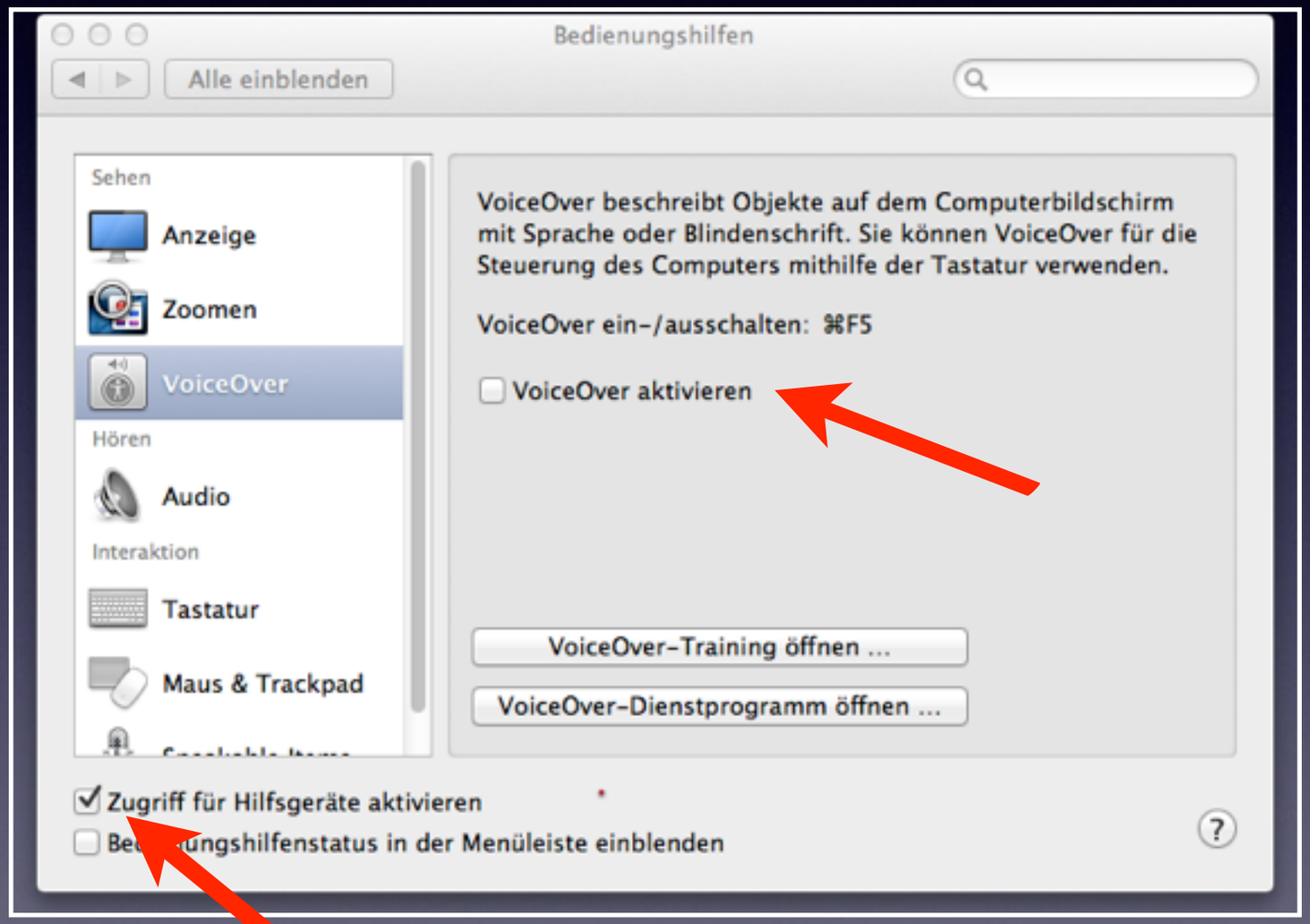
```
- (void)accessibilityPerformAction:(NSString*)action
{
    if ( [action isEqualToString:NSAccessibilityPressAction] )
        [self performPress];
    else
        [super accessibilityPerformAction:action];
}
```

Sonstiges

- (id)accessibilityHitTest:(NSPoint)point
  - liefert das Accessibility-Objekt unter der angegebenen Bildschirmkoordinate zurück
    - Accessibility-Koordinaten sind immer im screenspace
- (id)accessibilityFocusedUIElement
  - liefert das Accessibility-Objekt, welches den Tastaturfokus hat zurück

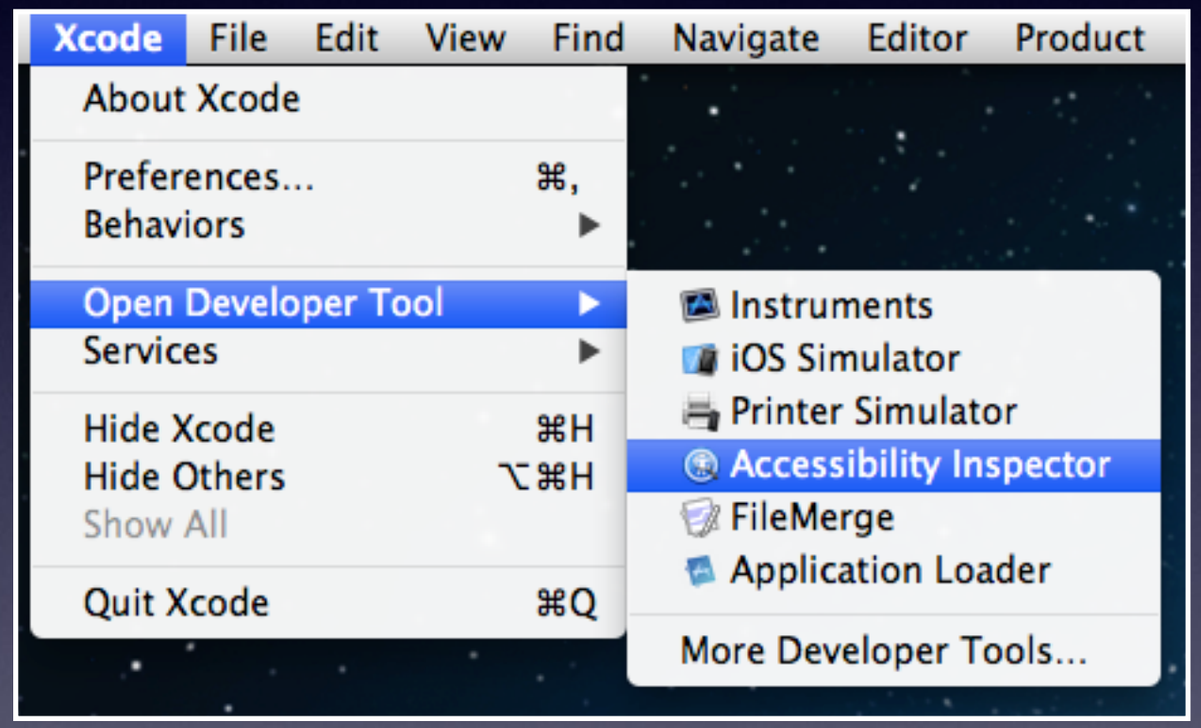
Workflow

- Bedienungshilfen aktivieren
- VoiceOver zum Testen



Workflow

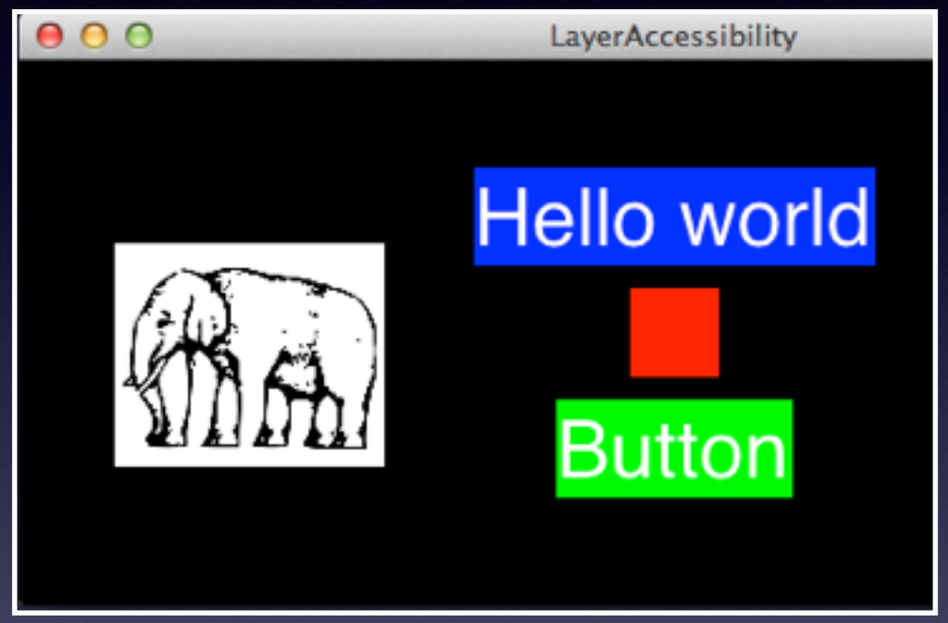
- Accessibility Inspector



Demo

CoreAnimation

- CALayer
- leichtgewichtige Basiskomponente
- implizit animiert
- Hierarchie ähnlich NS/UIView
- kann Bilder, Videos, einfachen Text oder Farben darstellen



CoreAnimation & NSAccessibility

- CALayer bietet keine eingebaute Unterstützung für Accessibility wie NS/UIView
- als Bestandteil des UIs trotzdem erforderlich

Mögliche Lösungen

- CALayer bzw. entspr. Unterklassen (CATextLayer, CAGradientLayer...) ableiten und NSAccessibility Protokoll implementieren
  - schlecht wiederverwendbar
  - DRY (NSAccessibilityParentAttribute, NSAccessibilityChildrenAttribute, NSAccessibilityPositionAttribute usw. müsten in jeder Unterklasse (CATextLayer, CAShapeLayer...) neu implementiert werden

Plan B

- Eigene NSAccessibility-konforme Hilfsklasse, welches je einen CALayer verwaltet
  - schwer zu warten, da eine parallele Hierarchie gepflegt werden muss (CALayer-Baum vs. Accessibility-Hilfsobjektbaum)
  - Anzahl der Hilfsklassen nimmt mit der Anzahl der UI-Objekte zu (Text, Bilder, Videos, Scroller, Buttons...)

Das Rad nicht neu erfinden

- Wie macht es AppKit?
  - NSView implementiert NSAccessibility
- CALayer-Hierarchie wiederverwenden
- CALayer ist ein KVC-kompatibler Kontainer

???

- CALayer-Instanzen können ähnlich wie dictionaries beliebige Werte via KVC speichern
- auch Blocks möglich
- pro accessibility-Attribut wird ein getter- und gegebenenfalls ein setter-Block im layer gespeichert

WTF?

- 1:1 Mapping zwischen NSAccessibilityAttributen und dem Handlerblockobjekt
- Generische Attribute werden in der Kategorie behandelt (NSAccessibilityParentAttribute, NSAccessibilityChildrenAttribute, NSAccessibilitySizeAttribute, NSAccessibilityPositionAttribute)

Code
CALayer Kategorie

```
- (id)accessibilityAttributeValue:(NSString*)anAttribute
{
   if ( [ anAttribute isEqualToString:NSAccessibilityChildrenAttribute ] ) {
    NSArray *sublayers = self.sublayers;

    NSArray *children = NSAccessibilityUnignoredChildren(sublayers);
    return children;
   }
  else if ( [ anAttribute isEqualToString:NSAccessibilityParentAttribute ] ) {
    id parent = [ self mm_accessibilityParent ];
        return NSAccessibilityUnignoredAncestor(parent);
   }

   ...
```

Code
CALayer Kategorie

```
...
  else if ( [anAttribute isEqualToString:NSAccessibilityPositionAttribute] ) {
   NSView *view = [self mm_containingView];
   CGPoint pointInView = [view.layer convertPoint:self.frame.origin
                                        fromLayer:self.superlayer ];
   NSPoint windowPoint = [view convertPoint:NSPointFromCGPoint(pointInView)
                                      toView:nil ];
   return [NSValue valueWithPoint:[[view window] convertBaseToScreen:windowPoint]];
  }
    else if ( [anAttribute isEqualToString:NSAccessibilitySizeAttribute] ) {
   NSView *view = [self mm_containingView];

   return [NSValue valueWithSize:[view convertSizeFromBacking:self.bounds.size]];
  }
  else if ( [anAttribute isEqualToString:NSAccessibilityWindowAttribute] ) {
   return [[self mm_accessibilityParent]
accessibilityAttributeValue:NSAccessibilityWindowAttribute];
  }
...
```

Code
CALayer Kategorie

```
...else {
   NSSet *customAttributes = [self valueForKey:kCustomAccessibilityAttributeNamesKey];
   if ( [customAttributes containsObject:anAttribute] ) {
    id (^attributeGetter)(void) =  [self mm_getterForAttribute:anAttribute];
    if ( attributeGetter ) {
        return attributeGetter();
    }
   }
   return nil;
}
```

Boilerplate #1

- Der Rootlayer benötigt einen Link zurück in sein NSView
- notwendig für die Konvertierung zwischen Layer- und Bildschirmkoordinaten

Boilerplate #2

- der Layer speichert eine weak-Referenz auf den View mittels objc_setAssociatedObject mit Parameter

OBJC_ASSOCIATION_ASSIGN

- zusätzliche Kategorie für NSView:

```
@implementation NSView (MMLayerAccessibility)

- (void)setAccessiblityEnabledLayer:(CALayer*)layer
{
    objc_setAssociatedObject(layer, kMMLayerAccessibilityParentViewKey, self, OBJC_ASSOCIATION_ASSIGN);
    [ self setLayer:layer ];
    [ self setWantsLayer:YES ];
}

@end
```

Boilerplate die letzte

- Das umgebende view implementiert lediglich die notwendigsten NSAccessibility-Methoden

NSView code

```
- (NSArray*)accessibilityAttributeNames
{
   static NSMutableArray *attributes = nil;

   if ( !attributes ) {
    attributes = [[super accessibilityAttributeNames] mutableCopy];
    NSArray *appendedAttributes = @[NSAccessibilityChildrenAttribute];

    for ( NSString *attribute in appendedAttributes ) {
        if ( ![attributes containsObject:attributes] ) {
            [attributes addObject:attribute];
        }
    }
   }
   return attributes;
}
```

NSView code

```
- (id)accessibilityAttributeValue:(NSString *)attribute
{
   if ( [attribute isEqualToString:NSAccessibilityChildrenAttribute] ) {
    return NSAccessibilityUnignoredChildren(@[self.layer]);
   }
   return [ super accessibilityAttributeValue:attribute];
}
```

NSView code

```
- (id)accessibilityHitTest:(NSPoint)aPoint
{
   NSPoint windowPoint = [ [ self window ] convertScreenToBase:aPoint ];
     CGPoint localPoint = NSPointToCGPoint([ self convertPoint:windowPoint
                                                      fromView:nil ] );

   CALayer *presentationLayer = [ self.layer presentationLayer ];
   CALayer *hitLayer = [ presentationLayer hitTest:localPoint ];
   return hitLayer ? NSAccessibilityUnignoredAncestor( [hitLayer modelLayer] ) : self;
}
```

Layer mit Attributen versehen

```
@interface CALayer (NSAccessibility)

- (void)setReadableAccessibilityAttribute:(NSString*)attribute withBlock:(id(^)(void))handler;

- (void)setWritableAccessibilityAttribute:(NSString*)attribute readBlock:(id(^)(void))getter writeBlock:(void(^)(id
value))setter;

- (void)removeAccessibilityAttribute:(NSString*)attribute;

- (void)setParameterizedAccessibilityAttribute:(NSString*)parameterizedAttribute withBlock:(id(^)(id))handler;

- (void)setAccessibilityAction:(NSString*)actionName withBlock:(void(^)(void))handler;

@end
```

Layer mit Attributen versehen

```
- (CALayer*)createDemoLayer
{
    CALayer *layer = [CALayer layer];
    ...
    [layer setReadableAccessibilityAttribute:NSAccessibilityRoleAttribute withBlock:^id{
     return NSAccessibilityColorWellRole;
    }];
    [layer setReadableAccessibilityAttribute:NSAccessibilityDescriptionAttribute withBlock:^id{
     return NSLocalizedString(@"Choose a color", @"color well ax description");
    }];
    [layer setAccessibilityAction:NSAccessibilityPressAction withBlock:^{
                        NSLog(@"action pressed");
    }];
    return layer;
}
```

Demo

- Accessibility ist kein Hexenwerk
- grössere Benutzerbasis
- In den Standardviews sowohl in AppKit als auch UIKit eingebaut

- WWDC-Session Videos
  - 2010: Session 100, 2011: Session 127, 2012: Session 203
- Apple-Samplecode (Dicey, AccessibilityUIExamples)
- https://github.com/mmllr/MMLayerAccessibility