Gero Herkenrath
Media Computing Group
RWTH Aachen University

Winter Semester 2013/2014

http://hci.rwth-aachen.de/iphone

RWTH AACHEN UNIVERSITY

Review

- Device restrictions
- Interaction paradigm changes

Objective-C

RWTH AACHEN UNIVERSITY

History

- Roots in the early 1980's
- Strongly influenced by Smalltalk
- Mainly used in NeXTSTEP, OS X, iPhone OS

Characteristics

- Strict superset of C
  - Can be mixed with C and C++
- Single inheritance
- Dynamic runtime
- Loosely typed (if you want it)
- Memory management
  - Automatic or manual reference counting

RWTH AACHEN UNIVERSITY

Syntax differences from C

- New types
  - Anonymous object (id)
  - Class
  - Selector
- Class definition
- Object messaging
- Properties
- Enumeration

RWTH AACHEN UNIVERSITY

New Types

```
// represents an objective-c class
Class aClass

// reference to an objective-c method
SEL someMethod

// reference to an instance of a specific objective-c class
Person *aPerson

// reference to an instance of any objective-c class
id aPerson

// boolean (YES or NO)
BOOL isAlive
```


RWTH AACHEN UNIVERSITY

Object Identifier: id

```
// id is defined as a pointer to an object
typedef struct objc_object {
    Class isa;
} *id;

// The Class type is defined as a pointer
typedef struct objc_class *Class;
```


RWTHAACHEN UNIVERSITY

Dynamic Typing

- id is completely unrestrictive
- The type of an object is only determined at runtime
- Allows for introspection

Object Messaging

```
// message without argument
[receiver message]

// message with single argument
[receiver message:argument]

// message with multiple arguments
[receiver message:arg1 argument2:arg2]

// receiving a return value
int status = [receiver message];

// message with a variable number of arguments
[receiver makeGroup:group, memberOne, memberTwo, memberThree];
```


RWTH AACHEN UNIVERSITY

Object Messaging Terminology

- Message expression
  - [receiver method:argument]
- Message
  - [receiver method:argument]
- Selector
  - [receiver method:argument]
- Method
  - The code selected by a message

RWTH AACHEN UNIVERSITY

Object Messaging

```
// message without argument
[rectangle draw]

// message with single argument
[rectangle setColor:blue]

// message with multiple arguments
[rectangle setStrokeColor:blue andThickness:2.0]

// receiving a return value
double length = [rectangle circumference];

// message with a variable number of arguments
[NSArray arrayWithObjects:one, two, three, nil];
```


RWTH AACHEN UNIVERSITY

Exercise

Java

```
Rectangle rect = Rectangle(someWidth, someHeight);
println(rect.getHeight());
boolean inside = rect.contains(x,y);
```

Objective-C

```
Rectangle *rect = [[Rectangle alloc] initWithWidth:someWidth
                                          andHeight:someHeight];
NSLog(@"%f",[rect height]);
BOOL inside = [rect containsPointWithX:x andY:y]
```


RWTH AACHEN UNIVERSITY

Identity vs. Equality

```
// test for object identity:
BOOL identical = @"Peter" == @"Peter";  //YES
     identical = @"Peter" == @"Peter "; //NO

// test for string equality: will return YES
NSString *aName = @"Peter";
NSMutableString *anotherName =
  [NSMutableString stringWithString:@"P"];

[anotherName appendString:@"eter"];

BOOL same = [aName isEqualToString:anotherName];   //YES
BOOL equal = (aName == anotherName);               //NO

// test for existence
BOOL exists = (object != nil);
```


RWTHAACHEN UNIVERSITY

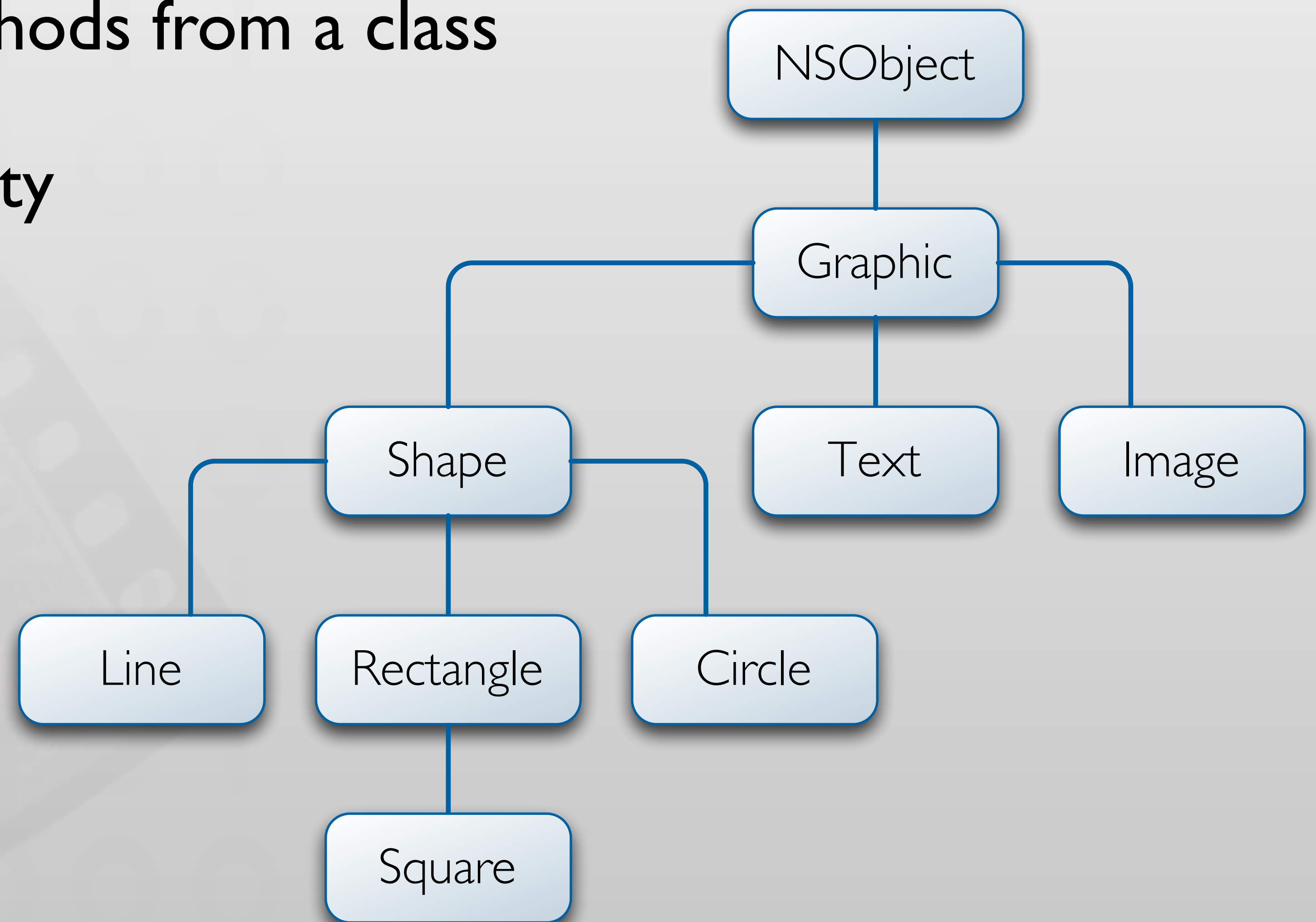
Selector

```
id object;

// create a selector for action:
SEL anAction = @selector(action:);

// check if the selector can be performed on object
if([object respondsToSelector:anAction]) {

  // perform the selector on object with a parameter
  [object performSelector:anAction withObject:self];
}
```


RWTH AACHEN UNIVERSITY

Class Inheritance

- Class definitions are additive
- Inherit variables and methods from a class
- Modify or add functionality



RWTH AACHEN UNIVERSITY

NSObject

- NSObject is root class
  - Similar to class Object in Java
- Provides functionality for
  - Memory management (allocation, release)
  - Object equality
  - Introspection
- Abstract class

RWTH AACHEN UNIVERSITY

Class Definition

Header File

contains public class declaration

Implementation File

contains method implementation

- To use a class, import the header file
  - The implementation file must also import its own header

RWTH AACHEN UNIVERSITY

Class Definition: Header

```
#import <Foundation/Foundation.h>
@interface Person : NSObject {
  NSString *name;
  int age;
}
// accessor for name
- (NSString *)name;
- (void)setName:(NSString *)newName;

// accessor for age
- (int)age;
- (void)setAge:(int)newAge;

// actions
-(BOOL)isAllowedToVote;
-(void)castBallot;
@end
```

Class Definition: Implementation

```
#import "Person.h"

@implementation Person

- (int)age {
  return age;
}

- (void)setAge:(int)value {
  age = value;
}

//... and other methods

@end
```


RWTH AACHEN UNIVERSITY

Using a Class

- To create an instance of a class
  - Dynamically allocate memory
  - Initialize that memory with appropriate values

```
id anObject = [[SomeClass alloc] init];

- (id) init
{
  self = [super init];
  if (self) {
     // initializations
  }
  return self;
}
```


RWTH AACHEN UNIVERSITY

Class Instantiation

```
- (Person *)createPersonWithName:(NSString *)aName {

  // create a new instance of the class Person
  Person *aPerson = [[Person alloc] init];

  // use the accessors to assign property values
  [aPerson setName:aName];
  [aPerson setAge:21];

  // return the instance
  return aPerson;

}
```

Class Inheritance

```
@interface Student : Person {
   NSString *major;
}
@end

@implementation Student
// Constructor for a student
- (id) init
{
  self = [super init];
  if (self != nil) {
     // set attributes
     [self setMajor:@"Undefined"];
  }
  return self;
}
@end
```

Demo

Protocols

```
// MyServiceDelegate.h (or MyService.h)
@protocol MyServiceDelegate

// required protocol method
- (void)myService:(MyService *)myService didUpdateWithResult:
(id)object;

// optional methods
@optional
...
@end

// MyController.h
@interface MyController : NSObject <MyServiceDelegate> {}
@end
// For Objects:
id<MyServiceDelegate> anObject;
```


RWTH AACHEN UNIVERSITY

Review

- History
- Characteristics
- Naming conventions
  - How would you name a function that inserts a string into an existing string at a certain index?

RWTH AACHEN UNIVERSITY

Categories

```
// NSString+Moo.h
@interface NSString (Moo)
// define a new method for NSString
- (NSString *)stringWithMoo;
@end

// NSString+Moo.m
@implementation NSString (Moo)
// implement the new method for NSString
- (NSString *)stringWithMoo {
  return [self stringByAppendingString:@"Moo"];
}
@end

// extend NSString with the category
#import "NSString+Moo.h"

// log "FooMoo" to console
NSLog([@"Foo" stringWithMoo]);
```


RWTH AACHEN UNIVERSITY

Class Extensions

```
// Person.h
@interface Person : NSObject {
  // define instance variables...
  NSNumber *age;
  BOOL  pictureIsUploaded;
}
- (void)startPictureUpload;
@end
```

```
// Person.m
@implementation Person
...
@end
```


RWTH AACHEN UNIVERSITY

Class Extensions

```
// Person.h
@interface Person : NSObject {
  // define instance
variables...
  NSNumber *age;
}
@end
```

```
// Person.m
@interface Person ()
{
  BOOL pictureIsUploaded;
}
- (void)startPictureUpload;
@end


@implementation Person
...
@end
```


RWTH AACHEN UNIVERSITY

Class Extensions

```
// Person.h
@interface Person : NSObject {
  // define instance
variables...
  NSNumber *age;
}
@end
```

```
// Person.m
@interface Person ()
{
  BOOL  pictureIsUploaded;
}
@end



@implementation Person
...
@end
```

Demo

RWTHAACHEN UNIVERSITY

Properties

```
// Person.h
@interface Person : NSObject {
  int age;
}

// accessor for age
- (int)age;
- (void)setAge:(int)newAge;

@end
```

```
// Person.m
@implementation Person

- (int)age {
  return age;
}

- (void)setAge:(int)value {
  age = value;
}

@end
```


RWTH AACHEN UNIVERSITY

Properties

```
// Person.h
@interface Person : NSObject {
  int age;
}

// age property
@property(assign) int age;

@end
```

```
// Person.m
@implementation Person

@synthesize age;

@end
```


RWTH AACHEN UNIVERSITY

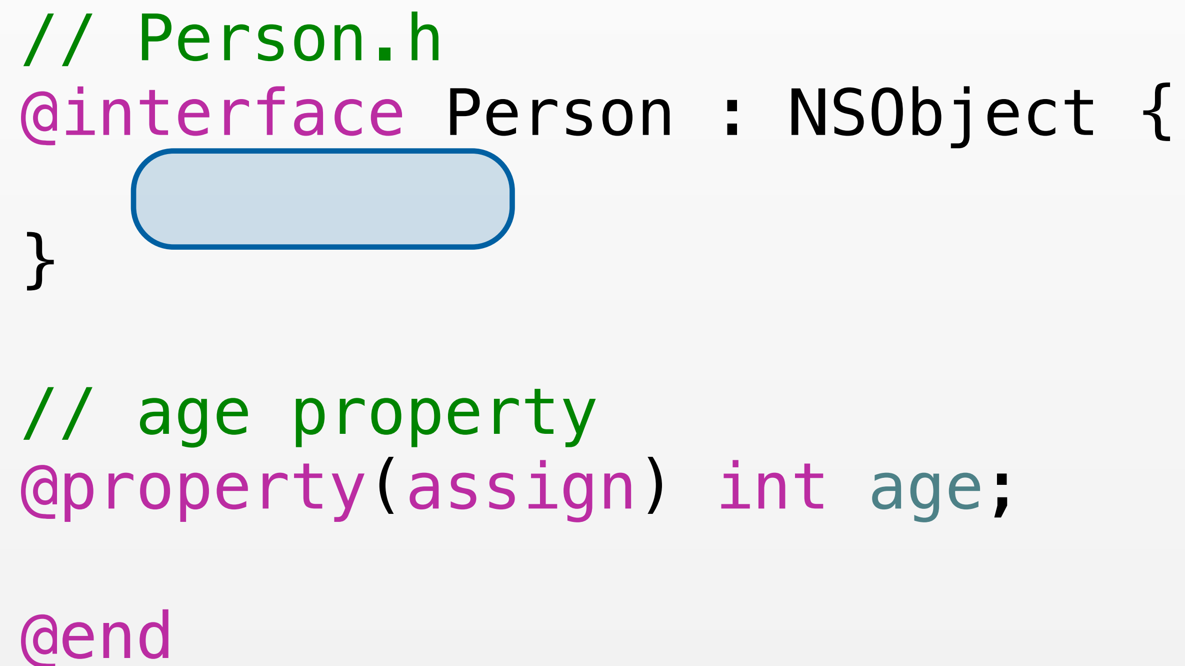
Properties

```
// Person.h
@interface Person : NSObject {
  int _age;
}

// age property
@property(assign) int age;

@end
```

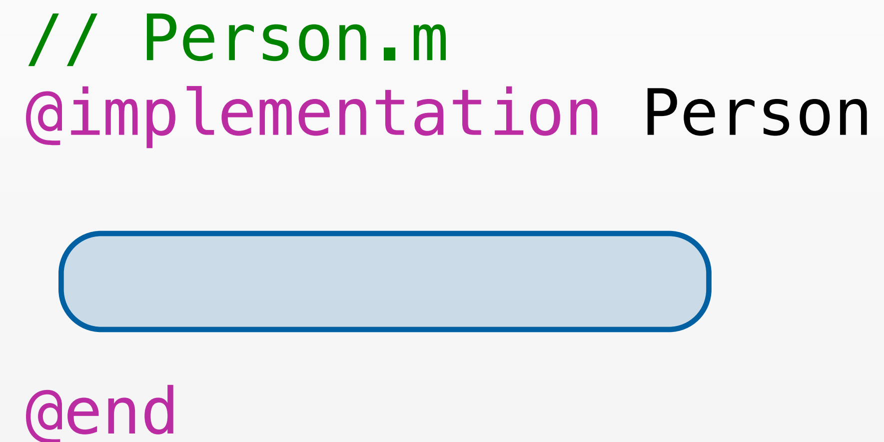
```
// Person.m
@implementation Person

@synthesize age;

@end
```


RWTH AACHEN UNIVERSITY

Properties



```
// Person.h
@interface Person : NSObject {

}

// age property
@property(assign) int age;

@end
```



```
// Person.m
@implementation Person


@end
```


RWTH AACHEN UNIVERSITY

Using Properties

```
// The statements below result in the exact same code
// access the value of a property
age = person.age;
age = [person age];

// change the value of a property
person.age = 12;
[person setAge:12];

document.author.name = @”Peter”;
[[document author] setName:@”Peter”];
```


RWTHAACHEN UNIVERSITY

Advanced Properties

```
// Imagine a class Rectangle
// A property does not necessarily have to relate to an ivar
@property (readonly) float circumference;


// The value is only calculated when required
- (float)circumference
{
  return 2*(width+height);
}
```


RWTH AACHEN UNIVERSITY

(Fast) Enumeration

```
// we have an array of Person objects
NSArray *people = ...;

// declare person variable outside the for loop
Person *person;

// fast enumeration of the people array
for(person in people) {

  // do something with all persons in the array
  [person castBallot];
}
```


RWTH AACHEN UNIVERSITY

Memory Management

Memory Management

	Java	C	Core Foundation	Cocoa / UIKit
Garbage collection	✓			
Malloc/free		✓	✓	
Retain/Release			✓	✓
ARC				✓

Object Lifecycle

- Objective-C uses reference counting
  - [object retain] increases retain counter
  - [object release] decreases retain counter
  - object is deallocated if the retain counter reaches 0
- Object is initialized with retain count of 1
  - [NSObject alloc]
  - [NSObject copy]

RWTH AACHEN UNIVERSITY

Basic Rules

- You own an object you create
- You take ownership of an object by calling *retain*
- You must relinquish ownership of unused objects you own (by calling *release* or *autorelease*)

RWTH AACHEN UNIVERSITY

The Retain Count

```
  NSNumber *age = [[NSNumber alloc] initWithInt:42];
                  //retain count = 1
...

[age retain];  //retain count = 2
...

[age release]; //retain count = 1
...

[age release]; //retain count = 0 -> Object deallocated
```


RWTH AACHEN UNIVERSITY

Clean up

```
@interface Person : NSObject
@property(retain) NSNumber *age;
@end

- (id)init
{
  NSNumber *someAge = [[NSNumber alloc] initWithInt:42];
  self.age = someAge;
   [someAge release]
}

- (void)dealloc
{
  // With synthesized setters, you set the object
  // to nil to release it
  // If delegate would be just a simple ivar,
  // we would call [_delegate release];
  self.age = nil;
}
```


RWTH AACHEN UNIVERSITY

Retain/release for Instance Variables

```
// Person.h
@interface Person {
  NSString *name;
}
@end

// Person.m
@implementation Person
- (void)setName:(NSString *)aName {

   if(name == aName) return; // prevent unnecessary assignments

   // always retain before releasing
   [aName retain];
   [name release];

   name = aName; // forget the old name and assign the new name
}
// cleanup (called when this object is released)
- (void)dealloc
{
   [name release];
   [super dealloc];
}
@end
```

Property Types

```
// c type property
@property(assign) int age;

// retained objective-c type property
@property(retain) NSString *name;

// copied objective-c type property
@property(copy) NSMutableString *address;

// read-only objective-c type property
@property(readonly) NSDate *birthday;

// retained, not-thread-safe objective-c type property
@property(retain,nonatomic) NSString *name;
```


RWTH AACHEN UNIVERSITY

Autorelease

```
- (Person*)studentWithName:(NSString*)aName{
  Person* aStudent = [[Person alloc] init];
  aStudent.type = @"Student";
  aStudent.name = aName;
  return aStudent;
}
```


RWTH AACHEN UNIVERSITY

Autorelease

```
- (Person*)studentWithName:(NSString*)aName{
  Person* aStudent = [[Person alloc] init];
  aStudent.type = @"Student";
  aStudent.name = aName;
  return [aStudent autorelease];
  // Released once the program returns to the runloop
}
```

Autorelease vs. Retain/Release

- Use autorelease for all temporary variables
  - except when performance is critical
- Always use autorelease for returned variables
  - return [object autorelease]
  - [NSSomething somethingWith...] is always autoreleased
- Use retain/release for static or instance variables

RWTH AACHEN UNIVERSITY

Autorelease Pools

```
- (void)loadImages
{
  //Assume imageNames contains the names of the Images to load
  NSArray *imageNames;
  for (NSString *imageName in imageNames) {
     // imageNamed returns an autoreleased reference
     UIImage *anImage = [UIImage imageNamed:imageName];
     //Do sth with this image, creating more autoreleased
objects
  }
}
```


RWTH AACHEN UNIVERSITY

Autorelease Pools

```
- (void)loadImages
{
  //Assume imageNames contains the names of the Images to load
  NSArray *imageNames;
  for (NSString *imageName in imageNames) {
     @autoreleasepool {
        // imageNamed returns an autoreleased reference
        UIImage *anImage = [UIImage imageNamed:imageName];
        //Do sth with this image,
        //creating more autoreleased objects
     }
  }
}
```

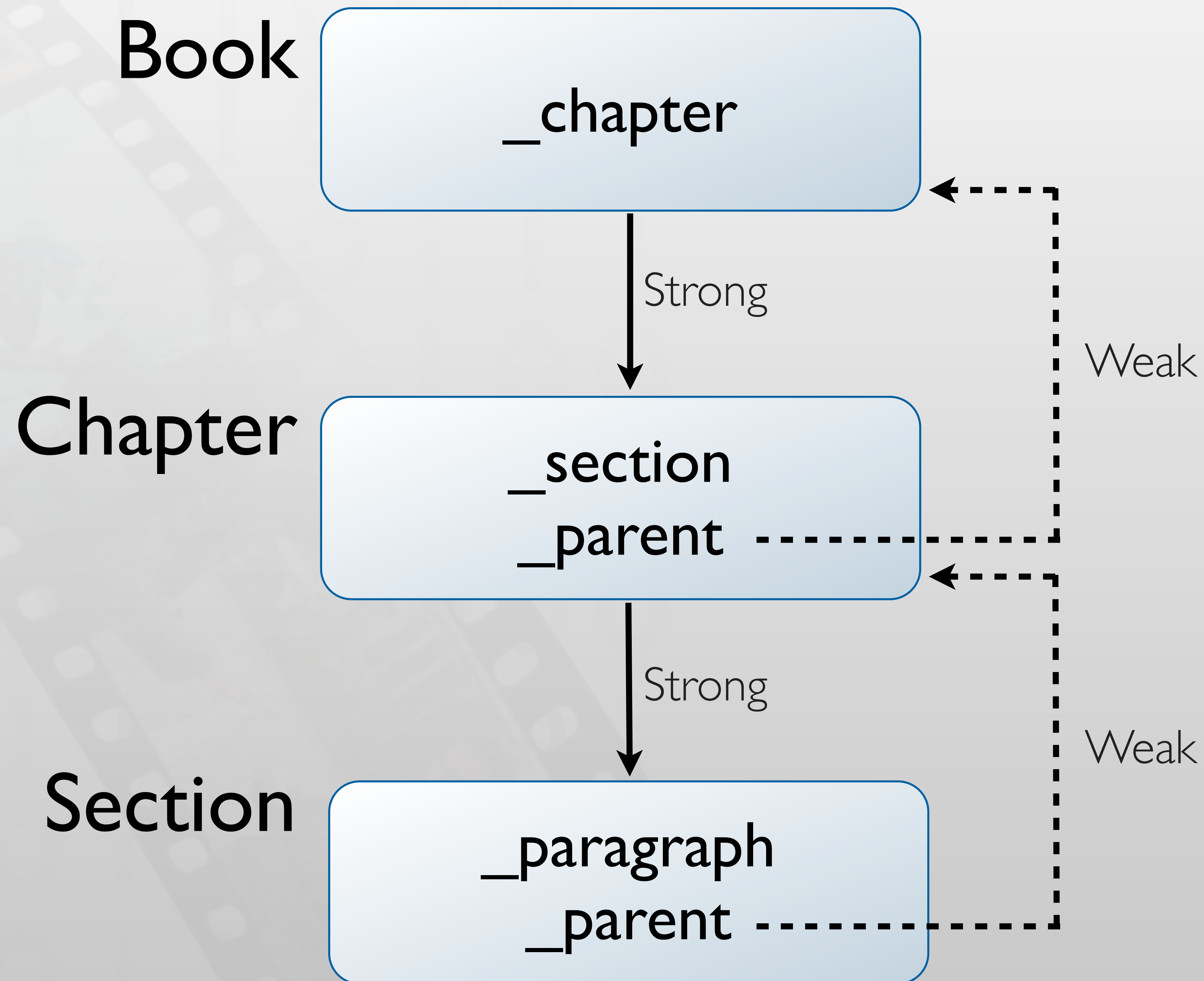

RWTH AACHEN UNIVERSITY

Automatic Reference Counting (ARC)

- Inserts retain/release calls at compile time
- No garbage collection!
- Introduces new lifetime qualifiers

RWTH AACHEN UNIVERSITY

References & Ownership



RWTHAACHEN UNIVERSITY

Lifetime Qualifiers

```
// Used like const

__strong NSString *iron; // default

__weak NSString *lead; // zeroing

__unsafe_unretained NSString *magnesium; // non-zeroing

__autoreleasing NSString *helium; // target is autoreleased
```


RWTH AACHEN UNIVERSITY

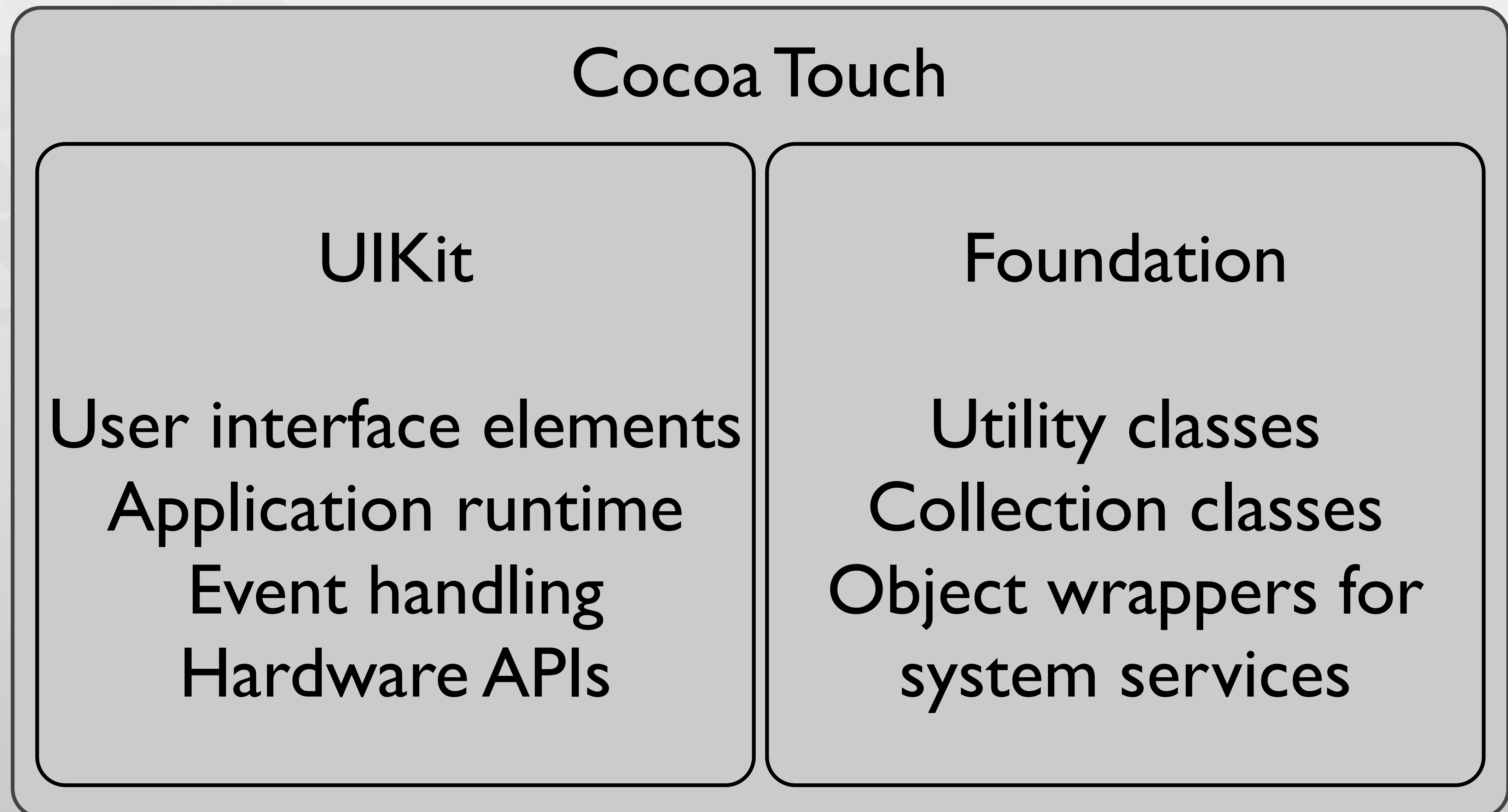
Property Types with ARC

```
// Describes an owning relationship to the destination object
@property(strong) NSString *name;

// There is only a weak relationship to the destination object
// If the destination object is deallocated, the property value
// is automatically set to nil.
@property(weak) NSMutableString *address;
```


RWTH AACHEN UNIVERSITY

Cocoa Touch

Cocoa Touch Architecture

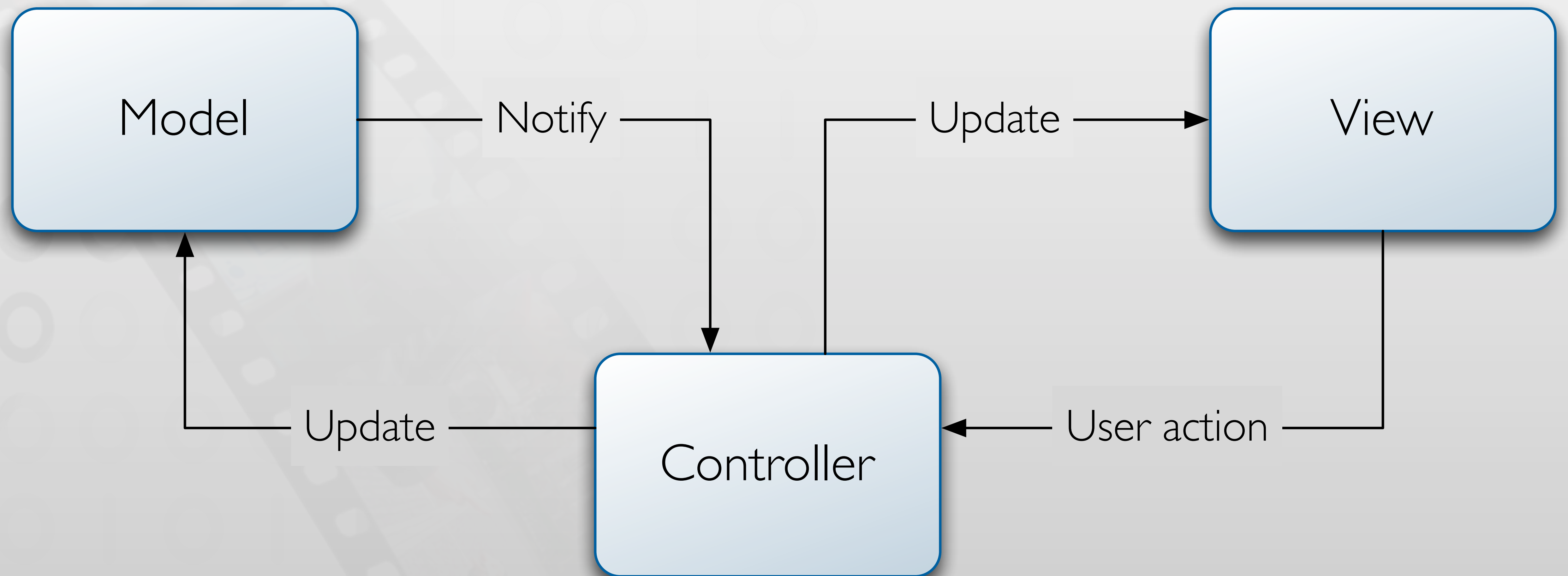


RWTH AACHEN UNIVERSITY

Characteristics

- Strictly follows established software development patterns
  - Model-View-Controller
  - Loose coupling: target/action, delegate
- Retain/release memory management
- Consistently object-oriented
  - Wrapper classes for low-level functionality

Model-View-Controller



Loose Coupling: Target/Action

- View generates events from user input
- Controller tells view to trigger a method for a certain event
- From then on the view calls this method, directly

```
// tell the view to trigger the action: method on the controller for given
event
[view addTarget:controller action:@selector(action:)
 forControlEvents:UIControlEventTouchUpInside];

// action method (the calling view is passed along as a parameter)
- (void)action:(id)sender {
...
}
```

Loose Coupling: Delegate

- View generates events from user input
- Controller tells view to relay specific events to its delegate

```
// make controller the delegate of the textField
textField.delegate = controller;

// implement delegate methods (e.g. for NSApplication)
- (void)textFieldDidEndEditing:(UITextField *)aTextField {
...
}
```


RWTH AACHEN UNIVERSITY

Summary

- Objective-C
- Cocoa Touch
- Basic datatypes

- Reading Assignment:
  - Getting Started: Learning Objective-C: A Primer
  - Transitioning to ARC Release Notes
  - Reference: NSString, NSArray, NSDictionary, UIView