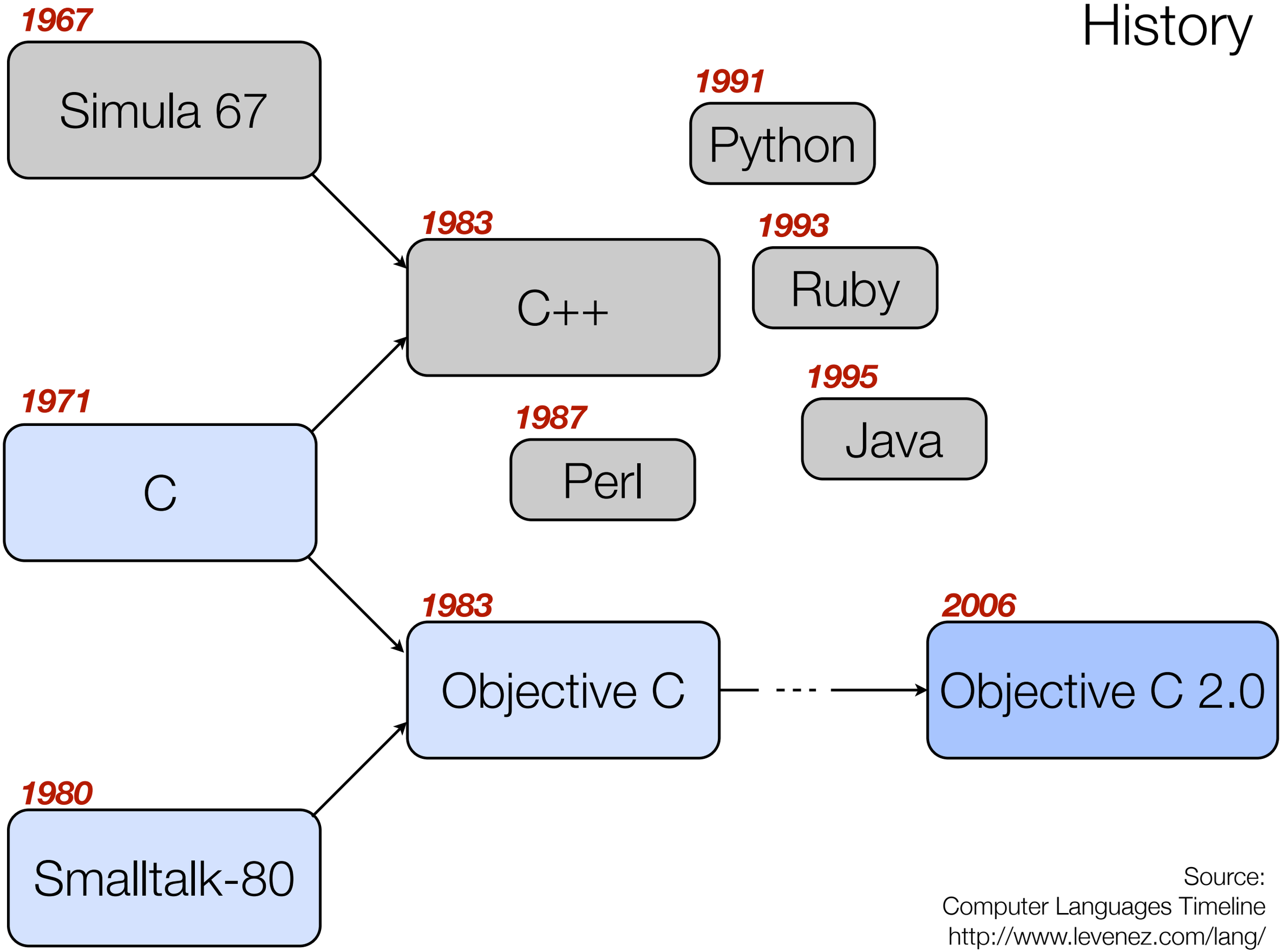
Objective C Crash Course

Introduction to iPhone Development
IAP 2009 ❄

iphonedev.csail.mit.edu

edward benson / eob@csail.mit.edu

History



Source:
Computer Languages Timeline
http://www.levenez.com/lang/

Overview

- Primitives & Strings
- Objects
- Messages
- Properties
- Memory Management

Primitives

The usual suspects

`int, float, ... (C Types)`

It's own boolean *(ObjC forked before C99)*

`BOOL`

Takes values NO=0 and YES=1

Some special types

`id, Class, SEL, IMP`

nil is used for a null instance pointer.

Strings

*Always use (NSString *) instead of C Strings unless you know what you're doing!!*

Inline

```
@"This is an inline string";
```

Assigned

```
NSString *str = @"This is assigned to a variable";
```

If you accidentally leave out the @, expect to crash!

Overview

- Primitives & Strings
- **Objects**
- Messages
- Properties
- Memory Management

Objects - Typing

Every object is of type

`id`

This is a pointer to the instance data of the object.

```
id person;
```

Of course, you can also declare a more specific type.

```
Person * person;
```

Objects - Declaring

Objects have:

methods and **instance variables**.

```
- (void)save;
```

```
BOOL complete;
```

Objects are defined by:

interface and **implementation**

```
.h
```

```
.m
```

Objects - Interface

New Class *Parent Class*

```
@interface TodoItem : NSObject {
  int dbkey;
  BOOL complete;
  int priority;
  NSString * title;
  NSDate * due;
}

+ (TodoItem *)loadFromDatabase:(int)key;

- (BOOL)saveToDatabase;

@end
```

Instance Variables

Class Method (+)

Instance Method (-)

Objects - Interface

This defines a contract that states:

```
@interface TodoItem : NSObject {
   int dbkey;
   NSString * title;
   NSDate * due;
   BOOL complete;
   int priority;
}

+ (TodoItem *)loadFromDatabase:(int)key;

- (BOOL)saveToDatabase;

@end
```

The class will respond to

`loadFromDatabase`

An instance will respond to

`saveToDatabase`

Within the class will be variables

`(int)dbkey, (int)priority, (BOOL)complete, (NSString*)title, (NSDate*)due`

OBTW *"NS" is from the NextStep days*

Objects - Implementation

```
#import "TodoItem.h"

@implementation TodoItem

+ (TodoItem *)loadFromDatabase:(int)key {
    // Your code here
}

- (BOOL)saveToDatabase {
    // Your code here
}

@end
```

pretty straightforward..

Objects - Protocols

Objective C's Protocols are similar to Java's Interfaces

```
@protocol AbleToEatSandwich
- (void)eatSandwich:(Sandwich *)sandwich;
@end
```

Declaring the Protocol

```
@interface Person <AbleToEatSandwich>
...
@end
```

Implementing the Protocol

```
id<AbleToEatSandwich> sandwichEater = ...;

[sandwichEater eatSandwich:sandwich];
```

Using the Protocol

Overview

- Primitives & Strings
- Objects
- **Messages**
- Properties
- Memory Management

Messages

Method Calling v. Message Passing

Messages

With no arguments

[object message];

Messages

With no arguments

[object message];

With 1 arguments

[object message:value];

Messages

With no arguments

[object message];

With 1 arguments

[object message:value];

With 2 arguments

[object message:value arg2:value];

Messages

With no arguments

```
[aPerson init];
```

With 1 arguments

```
[aPerson initWithFirst:@"Ted"];
```

With 2 arguments

```
[aPerson initWithFirstAndLast:@"Ted" last:@"Benson"];
```

You can send messages to **classes**

```
[Person alloc];
```

You can **nest** messages

```
Person* p = [[Person alloc] initWithName:@"Ted"];
```

equal to

```
Person* p = [Person alloc];
[p initWithName:@"Ted"];
```

Defining Methods

To Call

```
[aPerson initWithFirstAndLast:@"Ted" last:@"Benson"];
```

To Define

```
- (id)initWithFirstAndLast:(NSString*)firstName
                      last:(NSString*)lastName;
```

Overview

- Primitives & Strings
- Objects
- Messages
- **Properties**
- Memory Management

Properties

```
@interface TodoItem : NSObject {

  int dbkey;
  BOOL complete;
  int priority;
  NSString * title;
  NSDate * due;
}

@end
```

These all need getters and setters.

Properties

```
@interface TodoItem : NSObject {
  int dbkey;
  NSString * title;
}

@property (readonly) int dbkey;
@property (nonatomic, retain) NSString *title;

@end
```

```
#import "TodoItem.h"

@implementation TodoItem

@synthesize title, dbkey;

@end
```

You are still responsible for cleaning up memory for this object!

Property Attributes

@property (attributes) type name;

Writability

readwrite (default)
readonly

Setter Semantics

assign (default)
retain
copy

Atomicity

nonatomic
(no “atomic” attribute
but this is the default)

Source

http://developer.apple.com/documentation/Cocoa/Conceptual/ObjectiveC/Articles/chapter_5_section_3.html

Overview

- Primitives & Strings
- Objects
- Messages
- Properties
- **Memory Management**

If you're coming from a Python/Java/C# background, this is where things can get tricky

Memory Management

No garbage collection!
You have to manage reference counts yourself

Object Lifecycle

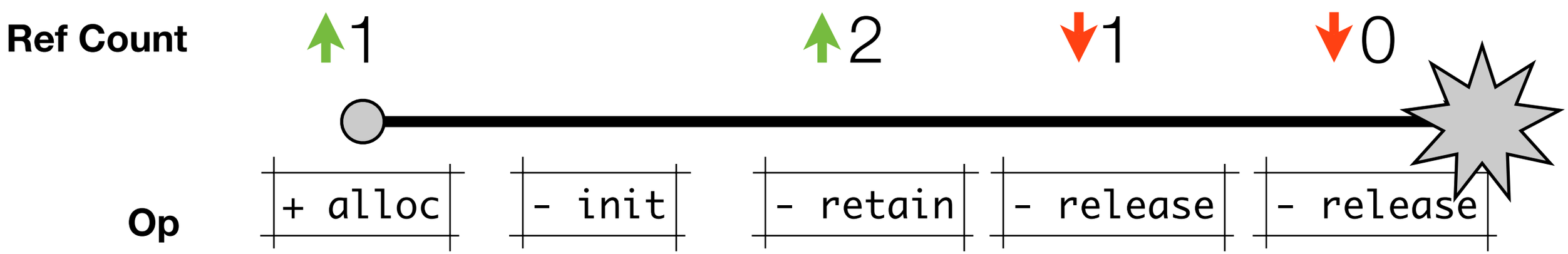


Image Credit: blog.tice.de

Memory Management - Creating an Object

Almost always follows the pattern

```
TodoItem *item = [[TodoItem alloc] init];
```

`+ alloc`	Allocates the memory
`- init`	Performs the initialization

So `init` is your Constructor...

```
- initWithName:(NSString*)name
```

etc

Constructors

```
@interface TodoItem : NSObject {
  NSString * title;
}

- (id)initWithTitle:(NSString*)aTitle;

@end
```

```
@implementation TodoItem

- (id)initWithTitle:(NSString*)aTitle {
     if (self = [super init]) {
        self.title = aTitle;
     }
     return self;
}

@end
```

This notation uses properties. We'll get to that in a minute..

When is an object destroyed?

When it's retain count reaches 0

Then the deconstructor `- dealloc` is called

*Never call **dealloc** yourself -- this is always called automatically for you.*

Destructors

```
@implementation TodoItem

- (void)dealloc {
    //  Clean up & release objects we own
    [title release];

    //  Call dealloc on super
    [super dealloc];
}

@end
```

Autorelease

- If you have a method that **creates** an object, you can't `release` it before returning the object, or else it might disappear before the caller `retain`s it.

- Every application has one or more NSAutoReleasePool objects that manage that retain counts of objects associated with them.

```
[someObject autorelease];
```

- **autorelease** is an alternative to **release** that essentially says "I'm done with this object, but don't let me be the one to **dealloc** it just yet."

Great Objective C Resources

- Cocoa Dev Central
  http://cocoadevcentral.com/d/learn_objectivec/

- The Objective-C 2.0 Programming Language
  http://developer.apple.com/documentation/Cocoa/Conceptual/ObjectiveC/ObjC.pdf

- Stanford's CS 193
  http://www.stanford.edu/class/cs193p/cgi-bin/index.php

- BYU's CocoaHeads Chapter
  http://cocoaheads.byu.edu/resources