Programmare applicazioni per iPhone

Filippo Bergamasco
Università Ca' Foscari di Venezia
Dipartimento di Informatica

Overview

- Come iPhone ha cambiato il modo di concepire i dispositivi mobili
- AppStore
- iPhoneOS: Archiettura e panoramica dei componenti
- iPhone-SDK: Ambiente di sviluppo e tools integrati
- Objective-C e design patterns
- Piccola demo :)

iPhone

- Creato da Apple, lanciato nel 2007
- iPhone nasce come dispositivo atto a conigurare le caratteristiche di un **cellulare,** di un **iPod** e di un **Pda**

iPhone

- Introduce nuovi standard nell'interazione quotidiana con i dispositivi mobili
  - Fortemente orientato alla multimedialità
  - Ampio touch-screen capacitivo, accelerometri, Gps, Wifi
- Nuovi paradigmi per le interfacce utente
  - Tecnologia Multi-Touch
  - Auto-rotation
  - Controlli ottimizzati per l'utilizzo con le dita

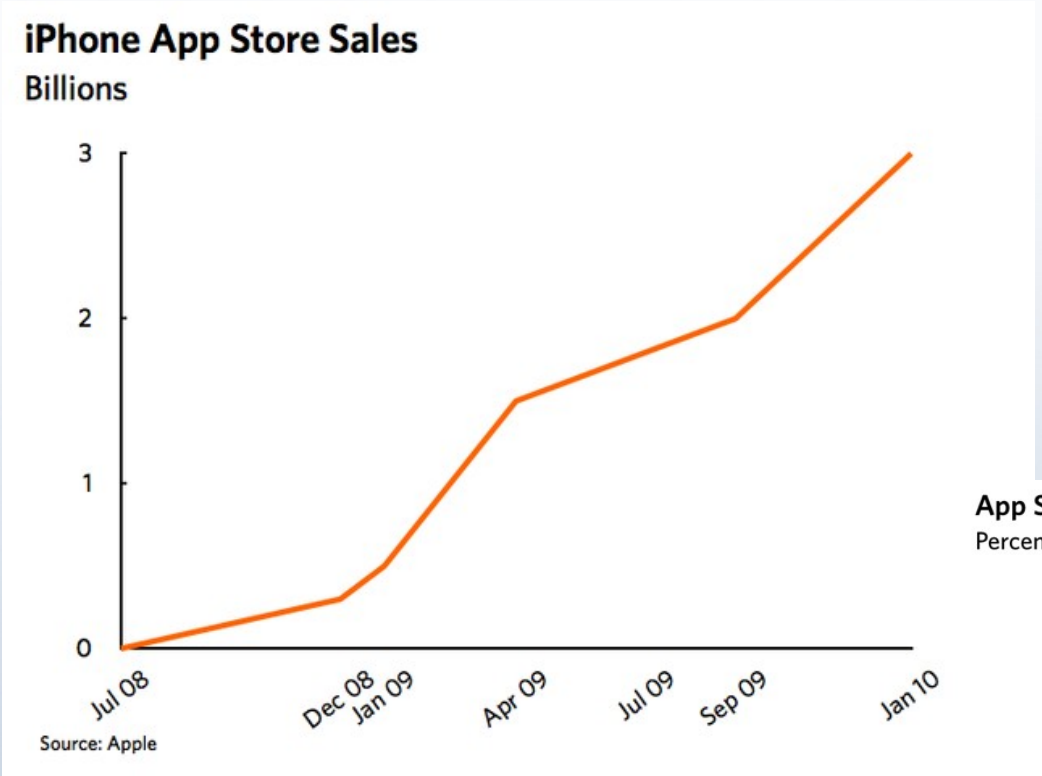
AppStore

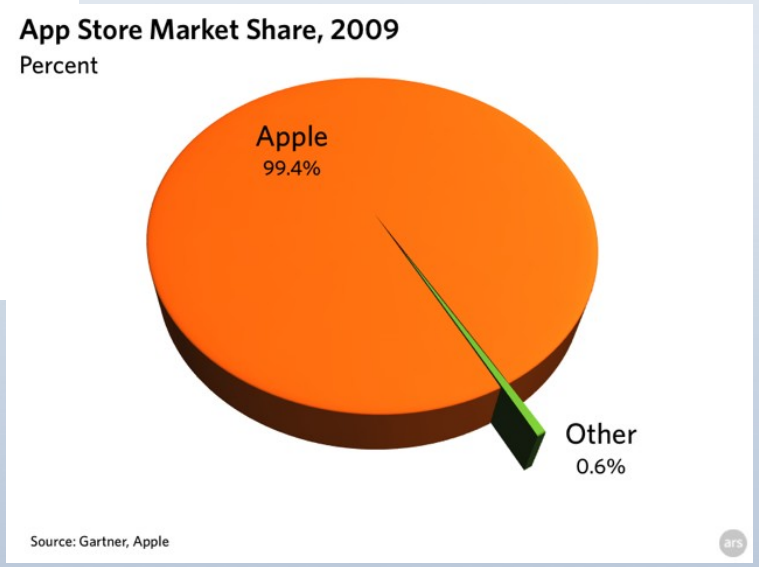
- Con iPhone nasce un nuovo modello di business per le applicazioni mobili
- AppStore offre una piattaforma **sicura, centralizzata** e **semplice** per la vendita e il content delivery delle applicazioni
- Sviluppatori possono vendere le proprie applicazioni secondo fasce di prezzo.
- Guadagni: 70% sviluppatore, 30% Apple.

AppStore

- Vantaggi per lo sviluppatore:
  - Ampia visibilità della propria applicazione
  - Discreta sicurezza anti-pirateria
  - Processo di vendita e delivery completamente gestito da Apple.
- Vantaggi per l'utente finale:
  - Applicazioni controllate prima della vendita (no malware)
  - Sistema centralizzato e semplice per l'installazione di nuovi contenuti

AppStore


iPhone App Store Sales
Billions
3
2
1
0
Jul 08
Dec 08
Jan 09
Apr 09
Jul 09
Sep 09
Jan 10
Source: Apple


App Store Market Share, 2009
Percent
Apple
99.4%
Other
0.6%
Source: Gartner, Apple

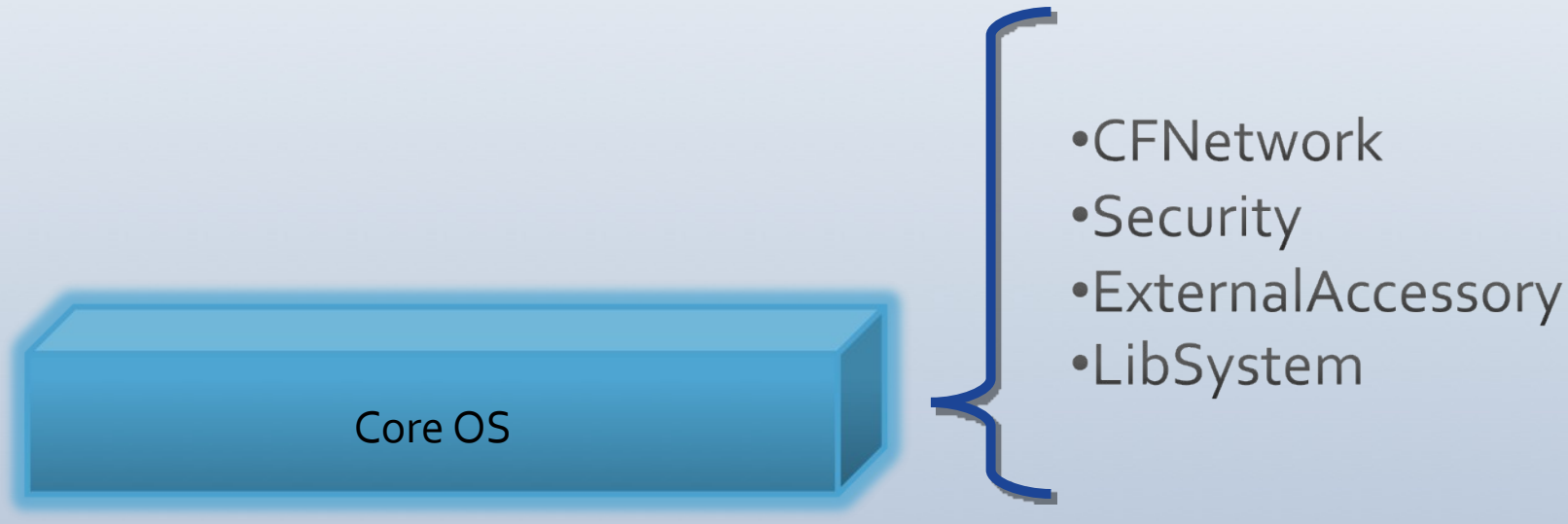
iPhoneOS

- Sistema operativo e tecnologie alla base dei dispositivi mobili iPhone, iPod Touch e iPad
- Versione alleggerita del Kernel di Mac OSX
- Sistema chiuso verso gli sviluppatori, le applicazioni possono utilizzare soltanto un set di API standard fornite con iPhone-SDK
- Vari livelli di astrazione per interagire con l'hardware del dispositivo

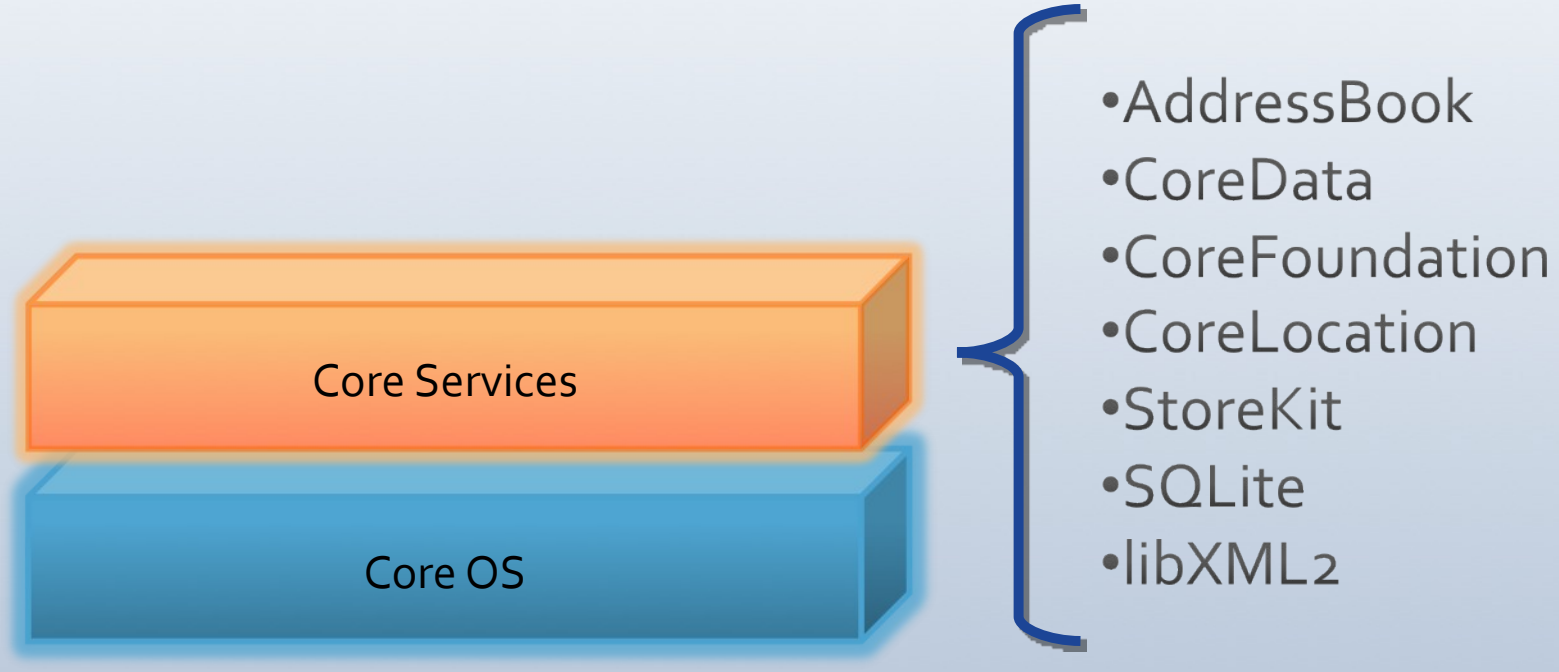
iPhoneOS

- Le applicazioni vengono eseguite nativamente dal dispositivo (no VMs, interpreti)
- Tranne poche eccezioni, può essere eseguita una sola applicazione per volta
- Applicazione = Una sola finestra, molte "view"
- Application **Sandbox**: iPhoneOS garantisce che ciascuna app sia eseguita in un un proprio container protetto.

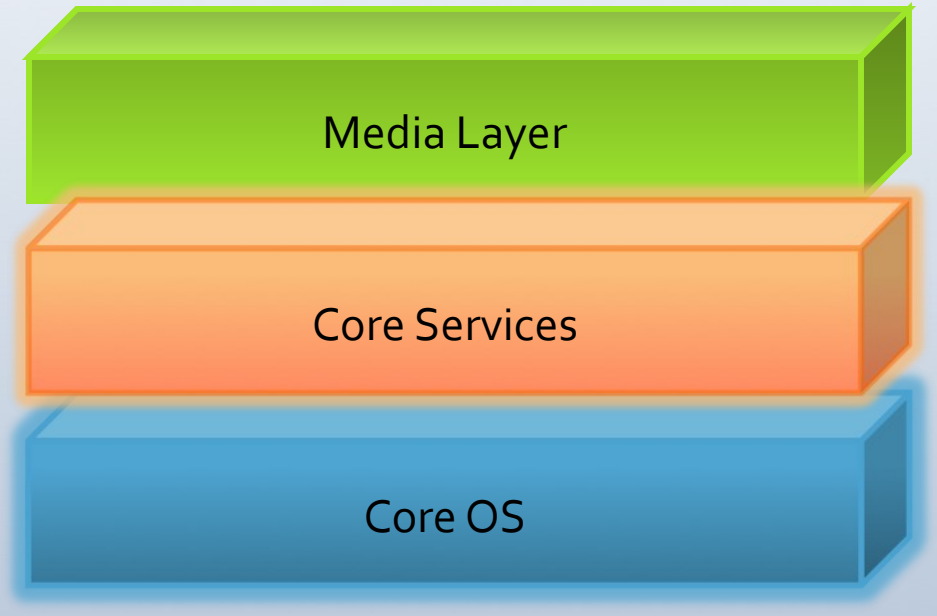
iPhoneOS: Livelli



iPhoneOS: Livelli

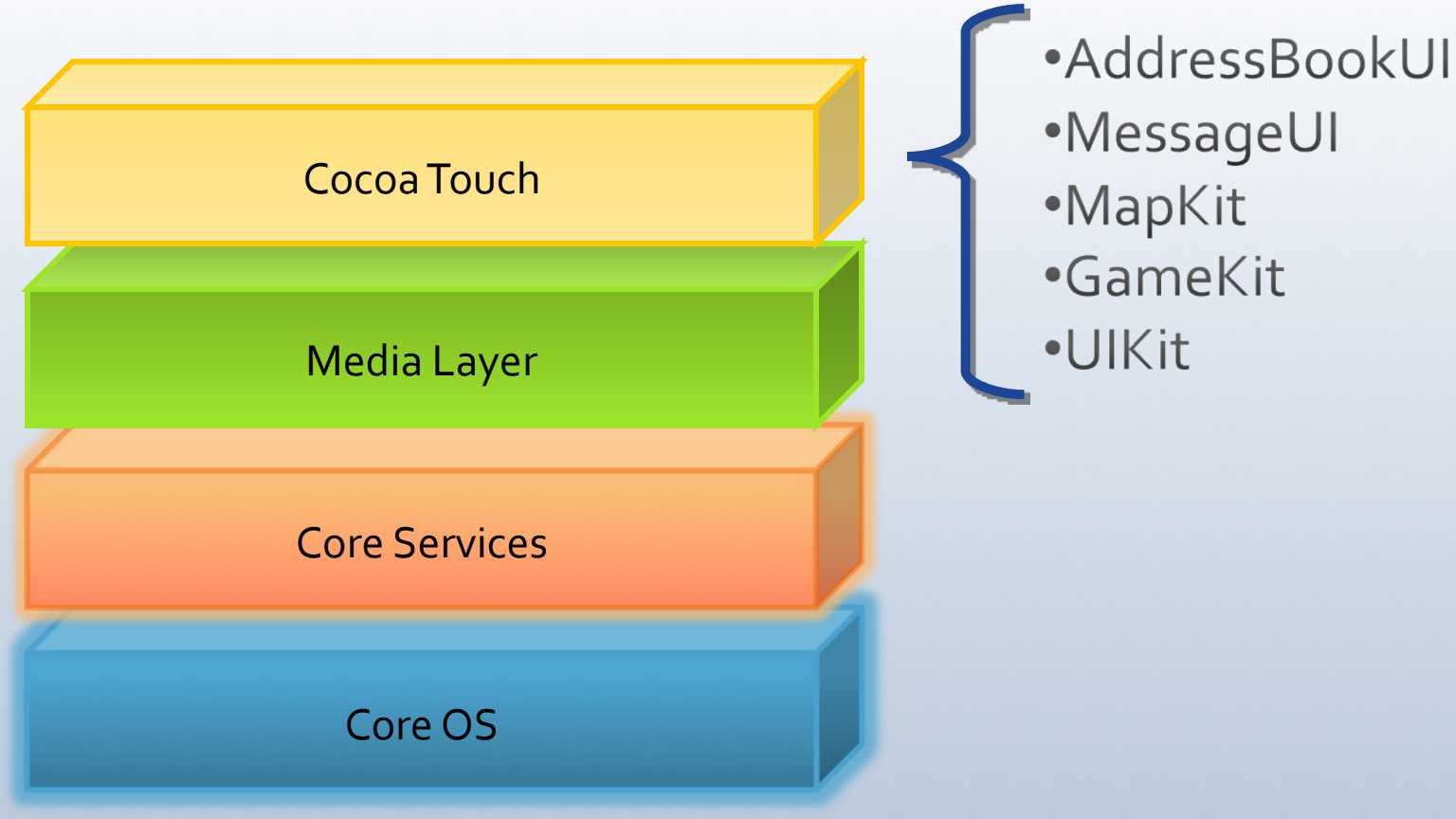


iPhoneOS: Livelli



- CoreGraphics
- QuartzCore
- OpenGL ES
- AVFoundation
- CoreAudio
- OpenAL
- MediaPlayer

iPhoneOS: Livelli

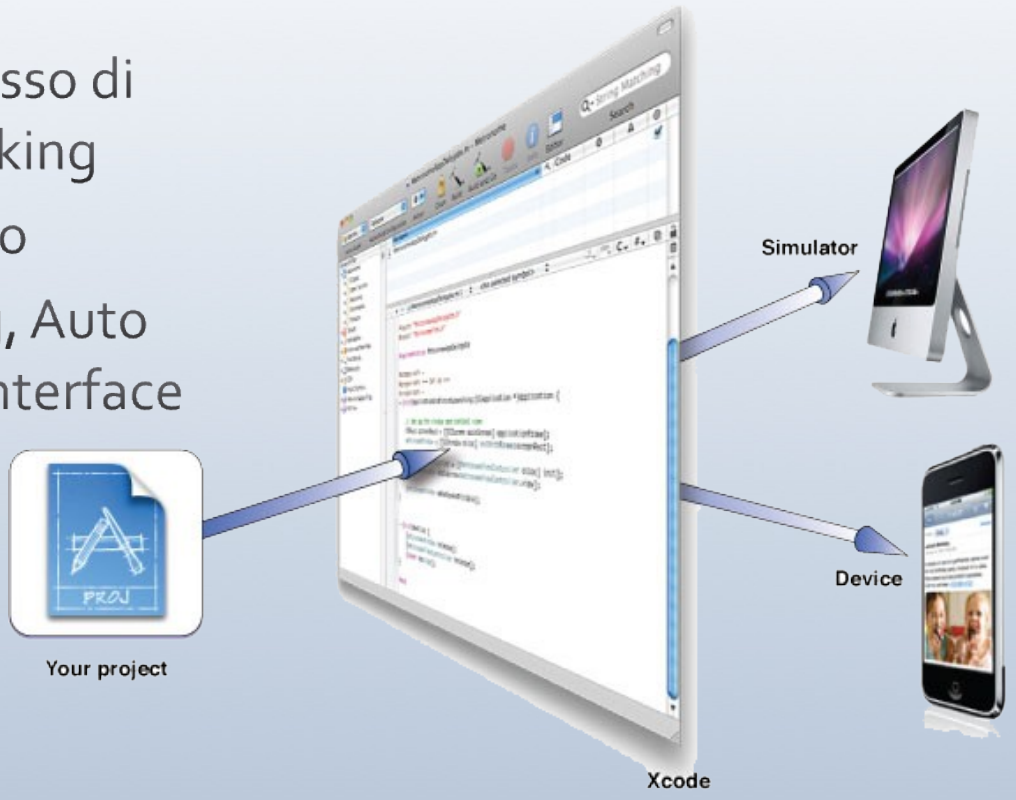


Sviluppare applicazioni

- Requisiti:
  - Mac OSX Intel (Leopard o superiore)
  - Iscrizione ad "Apple iPhone developer program"
  - Download e installazione di iPhone-SDK
- iPhone-SDK: fornisce i **compilatori,** le **librerie,** il **simulatore** e i **tools** necessari a sviluppare le vostre applicazioni.

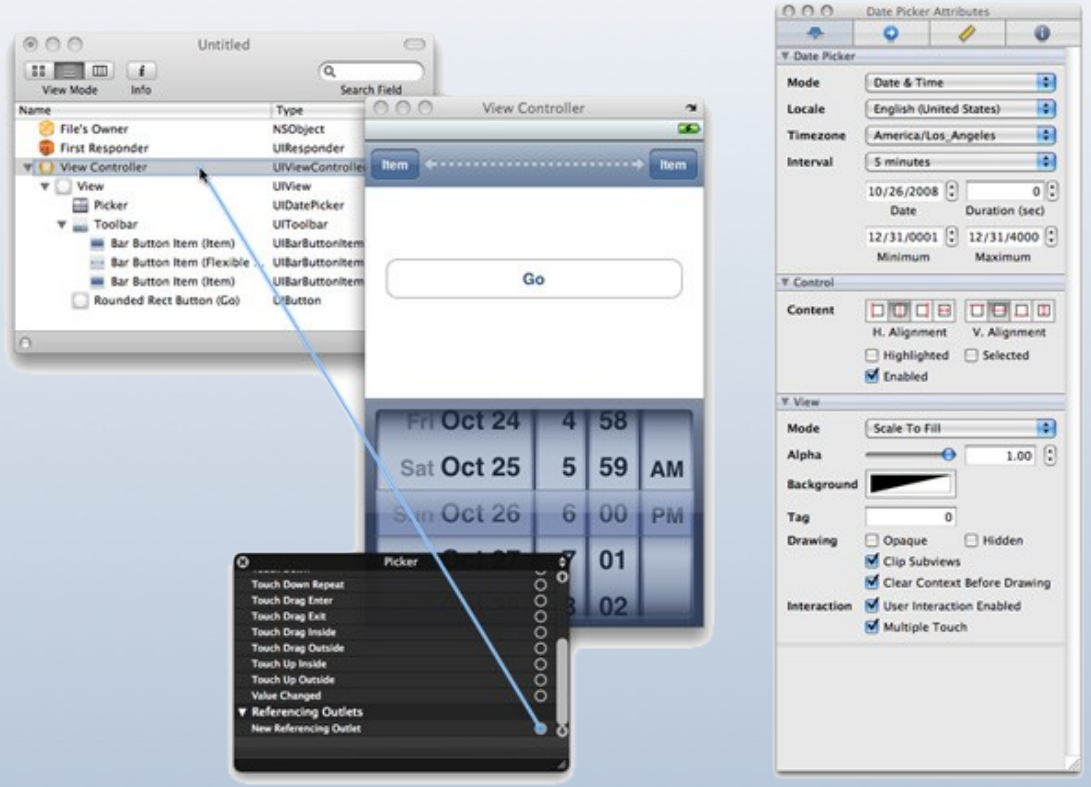
iPhone-SDK: XCode

- XCode è l'IDE standard fornito con l'SDK.
- Caratteristiche:
  - Gestione del processo di compilazione e linking
  - Analizzatore statico
  - Sintax highlighting, Auto completion, gdb interface



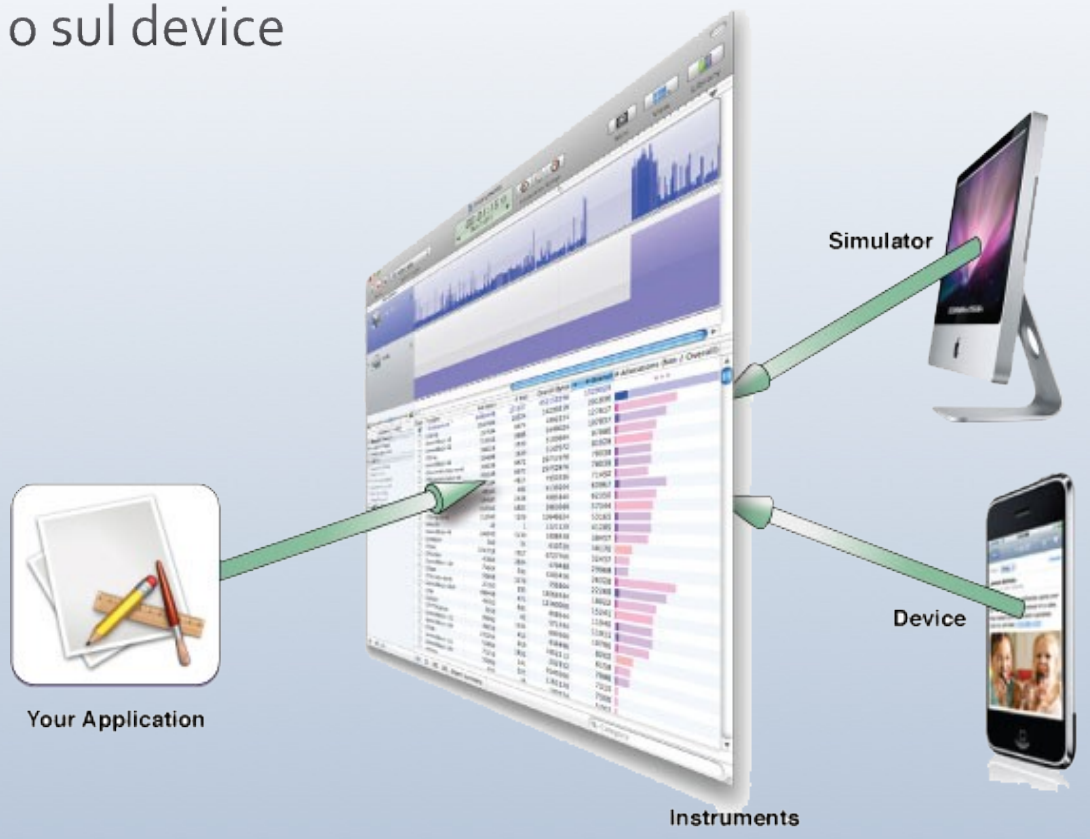
iPhone-SDK: Interface builder

- Sistema di design rapido delle interfacce utente mediante UIKit



iPhone-SDK:Instruments

- Analisi e reporting real-time delle prestazioni di un applicazione in esecuzione sul simulatore o sul device



iPhone-SDK: frameworks

- iPhone-SDK fornisce le librerie necessarie ad interagire con le funzionalità di iPhoneOS
- Interfacce in **C** per i componenti di basso livello (CoreOS, Core Services), **Objective-C** per quelli di alto livello (Cocoa Touch e parte di media layer)
- Programmazione Object Oriented e forte utilizzo di design patterns classici come MVC e delegation

Objective-C

- Linguaggio di programmazione ispirato a SmallTalk sviluppato da Brad Cox verso la metà degli anni 80.
- Caratteristiche:
  - Estende il linguaggio C fornendo il supporto per la programmazione Object Oriented
  - Può usare la tipizzazione dinamica, cerca di posticipare il più possibile le "decisioni" a run-time anziché a compile-time.
  - Necessita di un run-time system che viene inglobato nelle applicazioni in fase di linking

Objective-C

- Caratteristiche:
  - Permette la riflessione
  - Permette l'ereditarietà multipla di specifica ma non di implementazione
  - Supporta le eccezioni
- Objective-C 2.0
  - Supporto per la garbage collection (non per iPhoneOS)
  - Enumerazione veloce delle collezioni
  - Proprietà

Objective-C

- Obj-C richiede che **interfaccia** e **implementazione** di una classe siano definiti in blocchi di codice differenti:

```
@interface NomeDellaClasse : NomeDellaSuperclasse
{
      //variabili d'istanza
      int a;

}
//metodi di classe
+ metodoA
+ metodoB

//metodi di istanza
-metodoC: (int)p;
-metodoD
@end
```

NomeDellaClasse.h

NomeDellaClasse.m

```
@implementation NomeDellaClasse

+ metodoA {
      …
}
+ metodoB {
      …
}

-metodoC: (int)p {
      …
}
-Metodo D {
      …
}

@end
```

Objective-C

- Ogni oggetto appartiene al tipo **id**

```
typedef struct objc_object {
      Class isa;
} *id;

typedef struct objc_class *Class;
```

- Un oggetto è univocamente identificato dal suo indirizzo
- Ogni oggetto possiede un riferimento alla propria classe (isa pointer)

Objective-C

- Un oggetto può essere istanziato in questo modo

```
id o;
o = [[NomeClasse alloc] init];
```

- alloc inizializza a zero tutte le variabili di istanza tranne isa che collega l'oggetto con la sua classe
- Init è il costruttore di default disponibile per ciascuna classe

Objective-C

- Si può interagire con un oggetto inviando dei messaggi:

```
[oggetto nomeDelMetodo:nomeparametro];
```

- Il sistema a run-time cerca il metodo di istanza più appropriato da invocare per l'oggetto
- Grazie alla tipizzazione dinamica è sintatticamente corretto:
  - Inviare un messaggio a nil
  - Inviare il messaggio "nomeDelMetodo" anche se oggetto non lo implementa.

Obj-C++

- È possibile mescolare codice obj-c con codice c++ con alcune limitazioni:
  - Una classe c++ non può estendere una classe obj-c, viceversa.
  - Le classi Obj-c non possono contenere variabili di istanza di classi c++ che non abbiano un costruttore di default. (in caso solo puntatori a classi)
  - Gestione delle eccezioni separate
  - Etc.

Memory Management

- Tutti gli oggetti sono istanziati nello heap.
- iPhoneOS non prevede il supporto per il garbage collection.
- Tutte le classi fornite con iPhone-SDK ereditano dalla classe NSObject che implementa un semplice sistema di reference counting

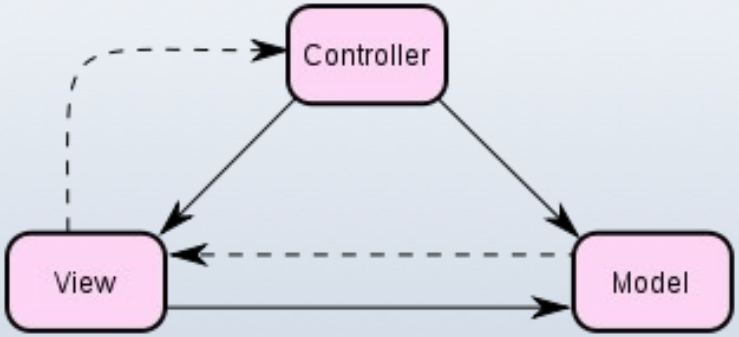
```
id o;
[o retain];
[o release];
```

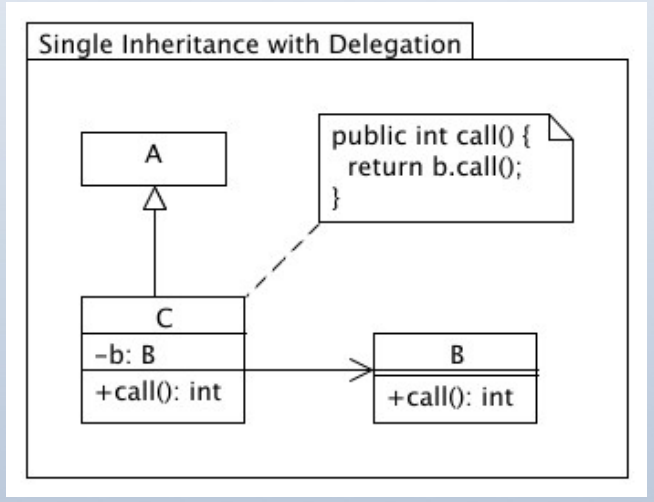
Memory Management

- Regola generale per la gestione della memoria:
  - Si possiede un oggetto quando:
    - Si è i creatori dell'oggetto (si invia il messaggio alloc)
    - Si è inviato un messaggio che contiene la dicitura "copy"
    - Si è preso possesso dell'oggetto inviando il messaggio **retain**
  - Se si possiede un oggetto è necessario inviare il messaggio **release** o **autorelease** al termine del suo utilizzo
- Il reference counting garantisce che l'oggetto sarà automaticamente distrutto se privo di riferimenti.

Design Patterns

- Le librerie di alto livello in iPhone-SDK fanno ampio uso di due patterns architetturali fondamentali:
  1. Delegation
  2. Model-View-Controller





Delegation

- Pattern molto utilizzato nell'ottica della gestione dell'interfaccia utente event-based.
- Si verifica quando un oggetto, invece di svolgere esso stesso un task, ne delega l'esecuzione ad un altro oggetto.
  - Può essere realizzato in generale per composition o ereditarietà multipla.
- Utili per definire il comportamento di componenti pre-esistenti all'interno dei frameworks

Model-View-Controller

- Pattern classico per modellare il funzionamento delle interfacce utente
- Si basa sulla suddivisione dei compiti fra **Model** (implementa la logica applicativa), **View** (gui) e **Controller** che mette in relazione modello e view con le azioni dell'utente
- Interface builder permette di costruire delle view e collegarle ai controllers attraverso **outlets** e **actions**

Piccola demo

Grazie per l'attenzione