



**UNIVERSITÀ  
DI PARMA**

***APPLICAZIONI IOS: ANALISI DEI  
LINGUAGGI DI  
PROGRAMMAZIONE E SVILUPPI  
FUTURI DELLA PIATTAFORMA***

**Candidato:** Bertoli Federico  
**Relatore:** Prof. Alfieri Roberto

Anno Accademico 2015/2016

# Obiettivi

- Analizzare la piattaforma iOS nel suo complesso, attraverso la valutazione delle caratteristiche peculiari dei linguaggi utilizzati
- Valutare la sua possibile evoluzione
- Creare un applicazione in un campo di utilizzo reale per valutare il nuovo linguaggio (Swift)

# iOS: i linguaggi utilizzati

- **Objective-C**: nato negli anni '80 per opera di Brad Cox, utilizzato massivamente dall'azienda di Jobs NextStep per il suo sistema operativo e i frameworks. E' stato integrato in tutto l'ecosistema Apple dopo l'acquisizione di NextStep da parte della stessa e ne è il linguaggio principale ancora oggi
- **Swift**: presentato nel 2014 alla WWDC e creato principalmente da Chris Lattner, si propone come alternativa più semplice e sicura, portando allo stesso tempo caratteristiche non presenti in Objective-C

# Objective-C - Generalità

- Arricchisce il modello semantico del linguaggio C, aggiungendo i paradigmi di programmazione ad oggetti
- Ispirato a SmallTalk di Alan Kay, in particolare per la gestione dei messaggi
- Utilizza un runtime dinamico, che permette estensioni quali le categories e la riflessione

# Objective-C - Messaggi

- **Messaggi**: un metodo non viene chiamato direttamente, ma si invia un messaggio all'oggetto stesso:  
`[persona calcolaEta:dataDiNascita];`
- Questo è reso possibile da runtime del linguaggio che tiene traccia di tutti i metodi conosciuti tramite un dizionario contenente il nome degli stessi e la loro locazione in memoria
- A tempo di compilazione, viene sostituito il frammento di codice con  
`objc_msgSend(persona, @selector(calcolaEta:),dataDiNascita);`
- Particolarità: l'oggetto può decidere se accettare il messaggio od inoltrarlo; inoltre si possono inviare messaggi a nil

# Objective-C - Categories

- **Categories:** implementazioni di metodi aggiunti ad una classe a runtime
- Permettono l'estensione di classi esistenti senza ricompilazione e senza avere a disposizione il loro codice sorgente
- Immagine della tesi in cui estendo NSString

# Objective-C - Reflection e Deep Introspection

- TODO

# Objective-C - Utilizzo nei frameworks Cocoa

- Il linguaggio è ancora il più utilizzato e richiesto per la scrittura di applicazioni iOS e Mac OS
- Verrà mantenuto parallelamente a Swift per gli anni a venire
- Offre caratteristiche ancora non presenti in Swift (reflection, deep introspection)
- L'IDE di Apple, Xcode, al momento fornisce strumenti ulteriori al programmatore se viene utilizzato Objective-C



# Swift - Generalità

- Linguaggio completamente nuovo, ispirato ai più moderni linguaggi di programmazione
- Sintassi semplice, espressiva e potente
- Offre caratteristiche non presenti in Objective-C (type inference, closures semplificate, tuples, optionals, namespaces)
- Open source ed in continua evoluzione
- Utilizza lo stesso runtime di Objective-C, permettendo l'uso dei due linguaggi nella stessa applicazione

# Swift - Closures

- Presenti anche in Objective-C con il nome di Blocks, in Swift sono notevolmente semplificate e rese più versatili:

```
var numeri = [2,25,21,89,90]
numeri.map({
    (numero: Int) -> Int in
    let risultato = 3 * numero return risultato
})
```

# Swift - Closures

In caso di singolo statement:

```
let numeriInMap = numeri.map({ numero in 3 * numero })
```

Se unico argomento della funzione:

```
//ordino i numeri in modo crescente
```

```
let numeriOrdinati = numeri.sorted { $0 > $1 }
```

# Swift - Optionals

- Tipo introdotto in Swift, rappresenta due possibilità:
  - La presenza di un valore, accessibile effettuando l'unwrapping dell'optional
  - L'assenza di valore alcuno

```
let possibileNumero = "123"  
let numeroConvertito = Int(possibileNumero)  
//numeroConvertito e' di tipo Int?, che si legge  
come "optional Int"
```

# Swift - Tuples

- Non presenti in Objective-C, permettono di raggruppare più valori, anche di tipi diversi, in un singolo valore composto
- Utilizzate principalmente come valori di ritorno dalle funzioni

```
let errore404http = (404, "Not found")
```

```
let (statusCode, statusMessage) = errore404http
```

# Swift - Playgrounds

The screenshot displays the Swift Playground environment with the following components:

- Code Editor:** Contains Swift code for a playground. Line 12 is selected.
- Results:** Shows the output of the selected code block, including a string, a color, and a rectangle.
- Timeline:** A graph titled "index" showing a linear progression from 0 to 10 over a time interval from 0 to 0.01 seconds. A red vertical line marks the current execution point at index 8.

```
1 // Playground - noun: a place where people can play
2
3 import Cocoa
4
5 var str = "Hello, playground"
6
7 let blue = NSColor.blueColor()
8
9 let r = NSRect(x: 0, y: 0, width: 200, height: 200)
10
11 for index in 1...10 {
12     index
13 }
```

**Results:**

- "Hello, playground"
- Color: `r 0.0 g 0.0 b 1.0 a 1.0` (Blue)
- Rectangle: `{x 0 y 0 w 200 h 200}`
- Execution count: (10 times)

**Timeline:**

The timeline graph shows a linear progression from 0 to 10 over a time interval from 0 to 0.01 seconds. A red vertical line marks the current execution point at index 8.

# Swift - Utilizzo nei frameworks Cocoa

- Ancora poco utilizzato a livello di frameworks
- Mancanza di ABI (Application Binary Interface) stabili
- Ancora in pieno sviluppo ma in via di definizione
- Per la versione 4 è previsto il blocco delle ABI, che porterà il linguaggio all'effettiva maturità

# Differenza di sintassi

Objective-C:

```
if (myDelegate != nil) {  
    if ([myDelegate respondsToSelector:  
        @selector(scrollViewDidScroll:)]) {  
        [myDelegate scrollViewDidScroll:myScrollView];  
    }  
}
```

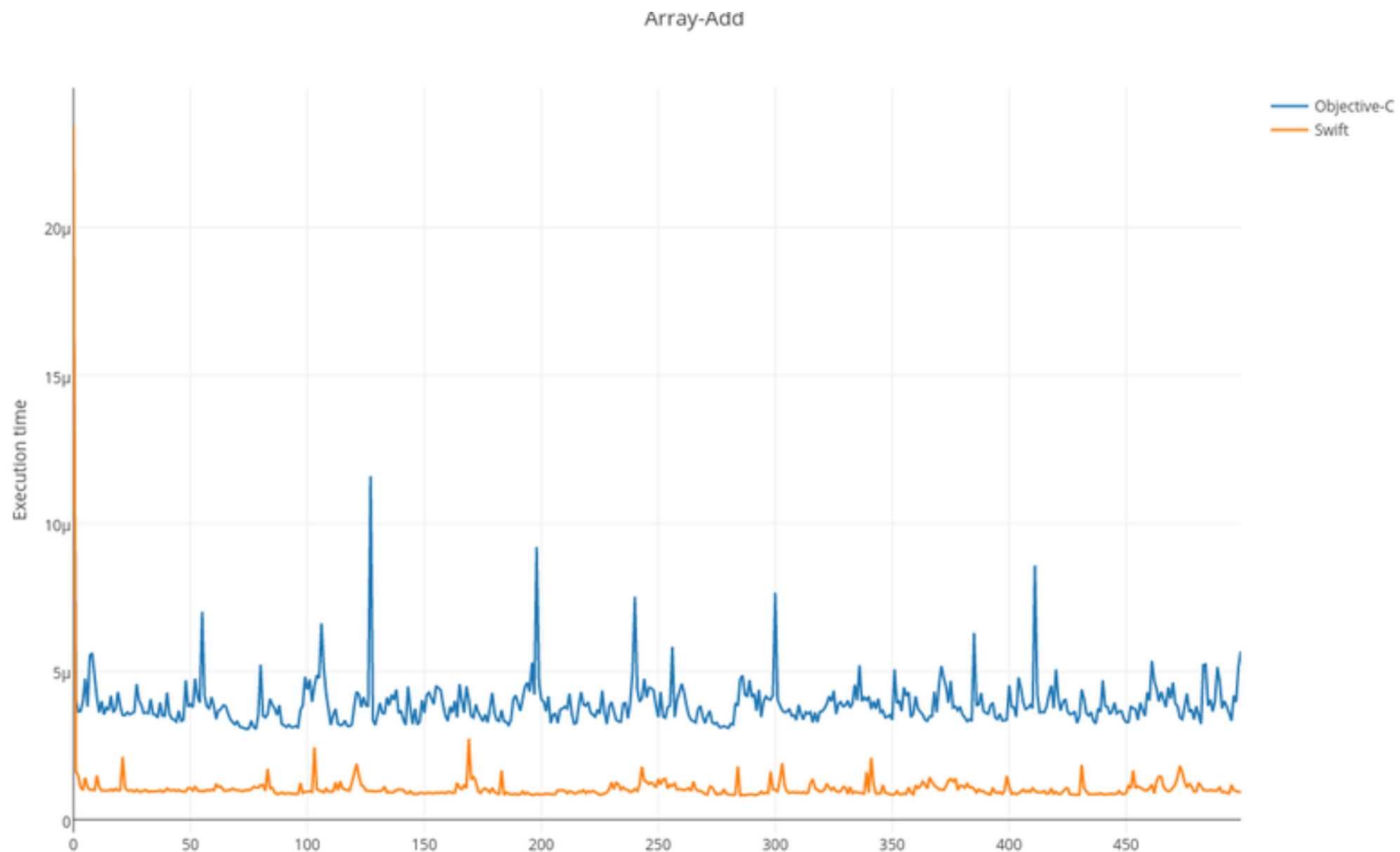
Swift:

```
myDelegate?.scrollViewDidScroll?(myScrollView)
```

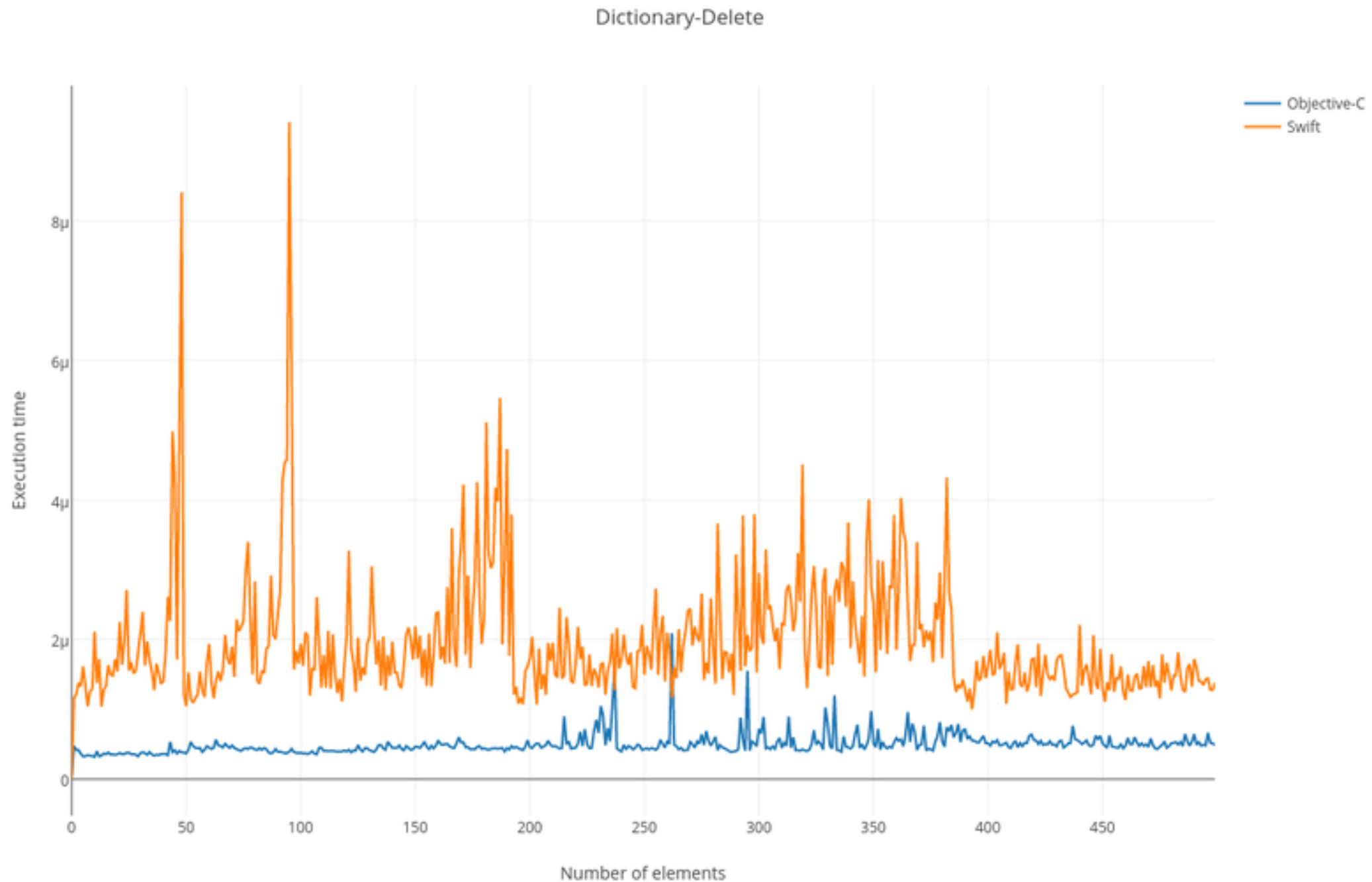


# Performance

Non esiste un chiaro vincitore, in base all'operazione effettuata un linguaggio è più veloce dell'altro



# Performance



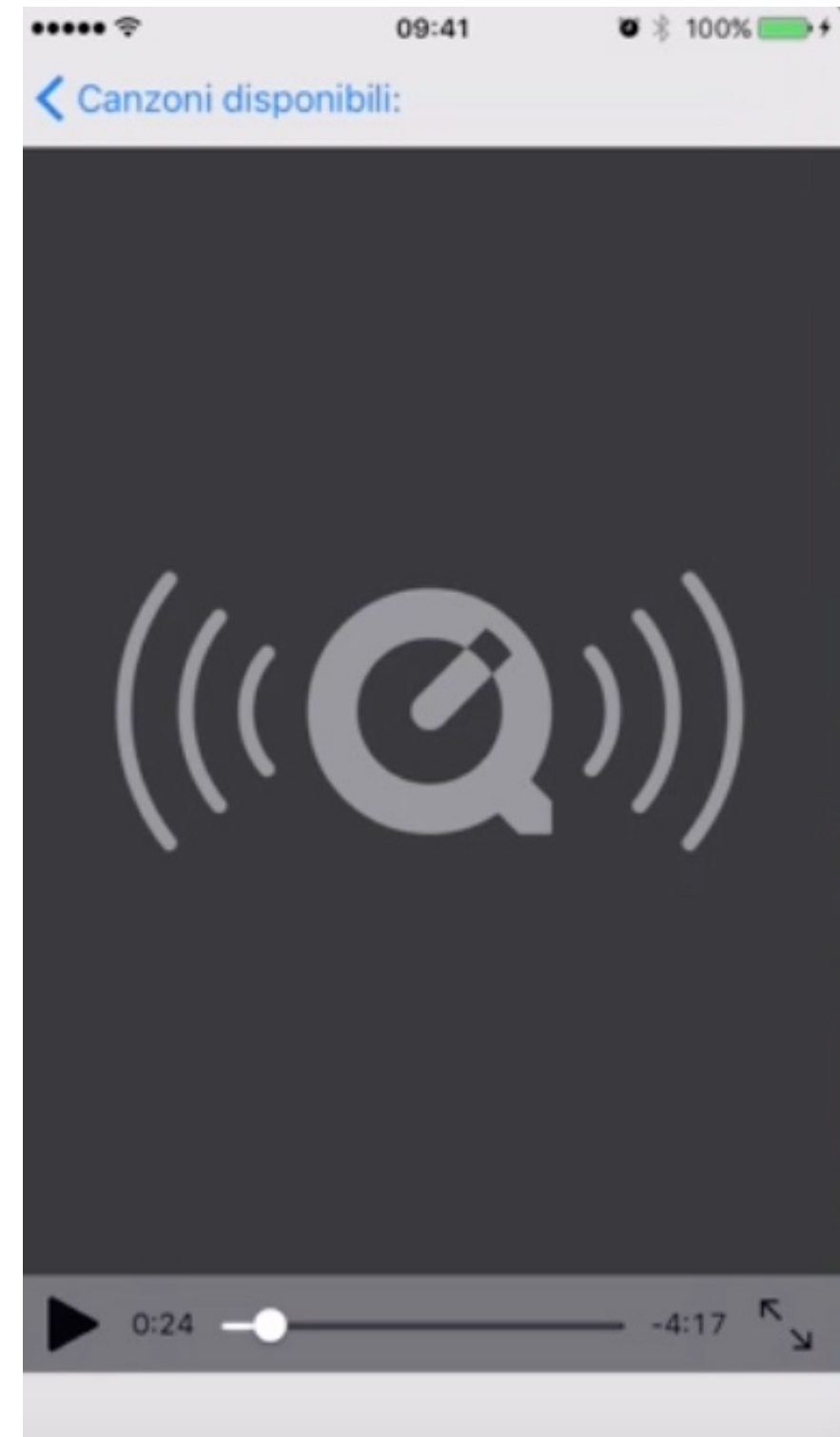
# L'applicazione

- E' stata creata un'applicazione in Swift per l'accesso a dati multimediali
- E' stato utilizzato un Raspberry Pi come web server Apache per questi dati
- L'app apre e mostra foto, mostra video a schermo intero ed in modalità picture-in-picture su iPad e permette l'ascolto di audio in streaming anche in background
- Utilizza CocoaPods per l'integrazione di frameworks esterni

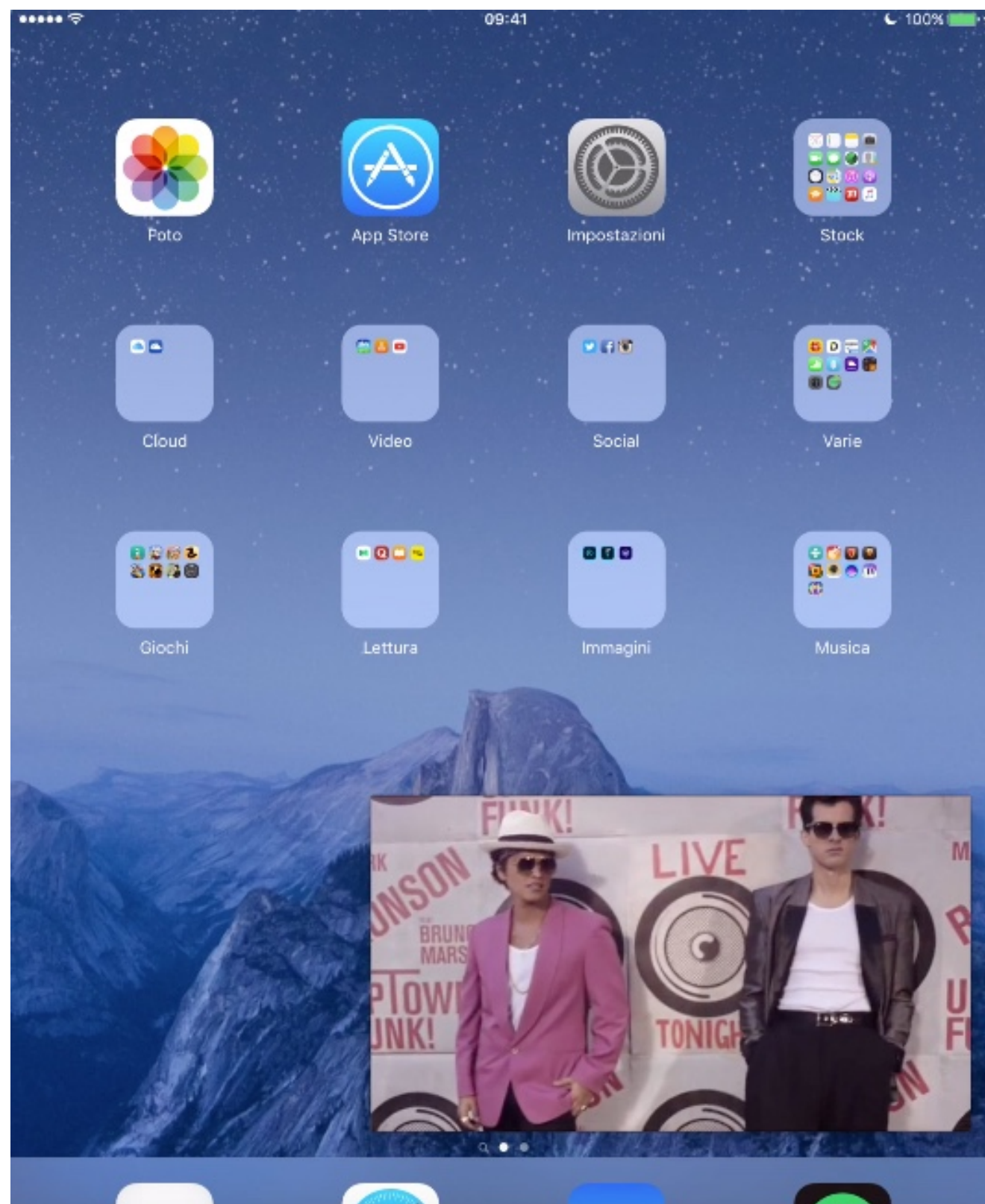
# L'applicazione



# L'applicazione



# L'applicazione



# Possibili implementazioni future

- Rendere l'applicazione uno strumento per l'accesso ai dati multimediali su un Raspberry Pi generico
- Migliorare l'interfaccia (copertine per gli album, anteprima delle immagini e dei video nella lista)
- Migliorare la gestione delle navigazioni e del passaggio dei dati
- Aggiungere un meccanismo unificato per le chiamate di rete
- Pubblicazione su App Store

# L'evoluzione della piattaforma

- Versioni modificate di iOS sono state utilizzate per watchOS e tvOS
- E' cominciata nel 2013 la transizione verso sistema operativo e dispositivi a 64-bit, giunta ormai alla conclusione
- Swift sta evolvendo grazie alla community open source e l'adozione del linguaggio aumenta rapidamente



# Conclusioni

- Swift è un linguaggio che sta guadagnando sempre più trazione, ma affinché cominci a prendere il sopravvento è necessario attendere la stabilità delle ABI, che porteranno gli sviluppatori di frameworks (interni ed esterni) a considerarlo per lo sviluppo in quanto linguaggio stabile.
- Objective-C rimane un linguaggio molto conosciuto e performante, con caratteristiche non ancora importate da Swift. La sintassi particolare e l'ereditarietà del C lo rendono poco appetibile ai neofiti della piattaforma e della programmazione in generale.
- Il consiglio per chi si approccia al mondo iOS è comunque quello di utilizzare Swift (dalla versione 3 e successivi) per la sua sintassi espressiva e concisa, valutando attentamente la presenza di frameworks aggiornati che supportino il linguaggio. Si renderà probabilmente necessario utilizzare files in Objective-C, ma l'interoperabilità tra i due porta comunque un vantaggio se si utilizza Swift.

Grazie per  
l'attenzione.