Efficient Interaction with Frameworks

Performance case studies

Session 244

Philippe Hausler, Foundation
Donna Tom, TextKit
Improvements in Foundation

Bridges and how they affect your app

Strings, ranges, and text
Improvements in Foundation

Bridges and how they affect your app

Strings, ranges, and text
Improvements in Foundation

Bridges and how they affect your app

Strings, ranges, and text
Improvements in Foundation
Bridges and how they affect your app
Strings, ranges, and text
Improvements in Foundation
Improvements in Foundation

NSCalendar
Improvements in Foundation

NSCalendar

Internal locking improvements
Improvements in Foundation

NSCalendar

Internal locking improvements

NSOperation and NSOperationQueue
Improvements in Foundation

NSCalendar

Internal locking improvements

NSOperation and NSOperationQueue

Copy on write collections
Copy on Write Collections
There is a CoW level

What is copy on write?

How does it work?

How can I improve my code to work better and safer with it?

@implementation Container {
    NSMutableArray<Item *> *_elements;
}
- (NSArray<Item *> *)elements {
    return [_elements copy];
}
@end
Copy on Write Collections
There is a CoW level

What is copy on write?

How does it work?

How can I improve my code to work better and safer with it?

```c
@interface Container {
    NSMutableArray<Item *> *_elements;
}
@end

- (NSArray<Item *> *)elements {
    return [self._elements copy];
}
@end
```
Copy on Write Collections
Let’s milk this joke a bit more

```objective-c
a = [NSMutableArray new]
```

```objective-c
a
```
Copy on Write Collections
Let’s milk this joke a bit more

```objective-c
NSMutableArray *a = [[NSMutableArray alloc] init];
```
Copy on Write Collections

Let’s milk this joke a bit more

```objective-c
a = [NSArray new] // a
```
Copy on Write Collections
Let’s milk this joke a bit more

a = [NSMutableArray new]

b = [a copy]
Copy on Write Collections
Let’s milk this joke a bit more

```objectivec
a = [NSMutableArray new]

b = [a copy]
```
Copy on Write Collections
Let’s milk this joke a bit more

```swift
a = [NSMutableArray new]

b = [a copy]
```
Copy on Write Collections
Let’s milk this joke a bit more

```objective-c
a = [[NSMutableArray alloc] init];

b = [a copy];

[a addObject:@"A"];
```
Copy on Write Collections
Let’s milk this joke a bit more

```objective-c
a = [NSMutableArray new]
b = [a copy]
[a addObject:@"A"]
```

Diagram:
- `a` and `b` start as empty arrays.
- `a` is copied to `b`.
- `a` adds an object `A`.
- `b` remains unchanged.
Copy on Write Collections
Let’s milk this joke a bit more

```objective-c
a = [NSMutableArray new]
b = [a copy]
[a addObject:@"A"]
```
Copy on Write Collections
Let’s milk this joke a bit more

```objective-c
a = [[NSMutableArray alloc] init]
b = [a copy]
[a addObject:@"A"]
```
// Leveraging Copy-on-write, steer your code in the right direction

// WARNING: Don’t pass any NSMutablesArrays into here
@property (strong) NSArray<Item *> *items;
// Leveraging Copy-on-write, Steer your code in the right direction

// WARNING: Don’t pass any NSMutableArray into here
@property (strong) NSArray<Item *> *items;
// Leveraging Copy-on-write, Steer your code in the right direction

// WARNING: Don’t pass any NSMutables into here
@property (strong) NSMutableArray<Item *> *items;

// Copies are safer
@property (copy) NSArray<Item *> *items;
// Leveraging Copy-on-write, steer your code in the right direction

// WARNING: Don’t pass any NSMutables into here
@property (strong) NSArray<Item *> *items;

// Copies are safer
@property (copy) NSArray<Item *> *items;

-(NSArray<Item *> *)items {
    NSMutableArray *items = [[NSMutableArray alloc] init];
    [self buildItems:items];
    // WARNING: Don’t mutate this... it is declared as NSArray so it should be safe?
    return items;
}
// Leveraging Copy-on-write, Steer your code in the right direction

// WARNING: Don’t pass any NSMutables into here

@property (strong) NSArray<Item *> *items;

// Copies are safer

@property (copy) NSArray<Item *> *items;

- (NSArray<Item *> *)items {
    NSMutableArray *items = [[NSMutableArray alloc] init];
    [self buildItems:items];
    // The copy is completely safe here and also is nearly free so avoid bad things later
    return [items copy];
}
// Leveraging Copy-on-write, Steer your code in the right direction

// WARNING: Don’t pass any NSMutableArrays into here
@property (strong) NSArray<Item *> *items;

// Copies are safer
@property (copy) NSArray<Item *> *items;

-(NSArray<Item *> *)items {
    NSMutableArray *items = [@[NSMutableArray alloc] init];
    [self buildItems:items];
    // The copy is completely safe here and also is nearly free so avoid bad things later
    return [items copy];
}

// This will copy
aNSArray as? [Any]
Data
The best type for dealing with bytes

Data is its own slice

Indexing is only a few instructions in optimized builds

Appending is dramatically faster

Replacing regions is faster too
func findZeroByte(_ data: Data) -> Data.Index? {
    for index in data.indices {
        if data[index] == 0 { return index }
    }
    return nil
}
// Subscripting Data

```swift
func findZeroByte(_ data: Data) -> Data.Index? {
    for index in data.indices {
        if data[index] == 0 { return index }
    }
    return nil
}
```
// Leveraging Data, Don’t Believe the Lore
// Leveraging Data, Don’t Believe the Lore

var bytes: [UInt8] = [0xcf, 0xfa, 0xed, 0xfe]
// Leveraging Data, Don’t Believe the Lore

var bytes: [UInt8] = [0xcf, 0xfa, 0xed, 0xfe]
var bytes = Data(bytes: [0xcf, 0xfa, 0xed, 0xfe])
// Leveraging Data, Don’t Believe the Lore

var bytes: [UInt8] = [0xcf, 0xfa, 0xed, 0xfe]

var bytes = Data(bytes: [0xcf, 0xfa, 0xed, 0xfe])
// Leveraging Data, Don’t Believe the Lore

var bytes: [UInt8] = [0xcf, 0xfa, 0xed, 0xfe]
var bytes = Data(bytes: [0xcf, 0xfa, 0xed, 0xfe])

var buffer = malloc(250).assumingMemoryBound(to: UInt8.self)
defer { free(buffer) }
// Leveraging Data, Don’t Believe the Lore

var bytes: [UInt8] = [0xcf, 0xfa, 0xed, 0xfe]
var bytes = Data(bytes: [0xcf, 0xfa, 0xed, 0xfe])

var buffer = malloc(250).assumingMemoryBound(to: UInt8.self)
deferr { free(buffer) }

var buffer = Data(count: 250)
// Leveraging Data, Don’t Believe the Lore

var bytes: [UInt8] = [0xcf, 0xfa, 0xed, 0xfe]
var bytes = Data(bytes: [0xcf, 0xfa, 0xed, 0xfe])

var buffer = malloc(250).assumingMemoryBound(to: UInt8.self)
defer { free(buffer) }

var buffer = Data(count: 250)

var bytes: [UInt8] = [0xcf, 0xfa, 0xed, 0xfe]
var bytes = Data(bytes: [0xcf, 0xfa, 0xed, 0xfe])

var buffer = malloc(250).assumingMemoryBound(to: UInt8.self)
defer { free(buffer) }

var buffer = Data(count: 250)

var bytes: [UInt8] = [0xcf, 0xfa, 0xed, 0xfe]
var bytes = Data(bytes: [0xcf, 0xfa, 0xed, 0xfe])

var buffer = malloc(250).assumingMemoryBound(to: UInt8.self)
defer { free(buffer) }

var buffer = Data(count: 250)
// Leveraging Data, Don’t Believe the Lore

var bytes: [UInt8] = [0xcf, 0xfa, 0xed, 0xfe]
var bytes = Data(bytes: [0xcf, 0xfa, 0xed, 0xfe])

var buffer = malloc(250).assumingMemoryBound(to: UInt8.self)
defer { free(buffer) }

var buffer = Data(count: 250)

let header = buffer.subdata(in: buffer.startIndex..<buffer.startIndex.advanced(by: 4))
// Leveraging Data, Don’t Believe the Lore

var bytes: [UInt8] = [0xcf, 0xfa, 0xed, 0xfe]

var bytes = Data(bytes: [0xcf, 0xfa, 0xed, 0xfe])  

var buffer = malloc(250).assumingMemoryBound(to: UInt8.self)
defer { free(buffer) }

var buffer = Data(count: 250)

let header = buffer.subdata(in: buffer.startIndex..<buffer.startIndex.advanced(by: 4))

let header = buffer[..<buffer.startIndex.advanced(by: 4)]  

// Leveraging Data, Don’t Believe the Lore
Bridges and How They Impact Your App
For whom the bridge tolls

```swift
NSArray *array = @[];
CFArrayGetCount((CFArrayRef)array);
let data = NSData()
let d = data as? Data
```
Bridges and How They Impact Your App
For whom the bridge tolls

```objective-c
NSArray *array = @[];
CFArrayGetCount((CFArrayRef)array);
let data = NSData();
let d = data as? Data
```

Toll-free bridging
Bridges and How They Impact Your App
For whom the bridge tolls

Toll-free bridging
• From a CF type to a NS type

```objective-c
NSArray *array = @[];
CFArrayGetCount((CFArrayRef)array);
let data = NSData()
let d = data as? Data
```
Bridges and How They Impact Your App
For whom the bridge tolls

Toll-free bridging
• From a CF type to a NS type
• From a NS type to a CF type

```c
NSArray *array = @[];
CFArrayGetCount((CFArrayRef)array);
let data = NSData()
let d = data as? Data
```
Bridges and How They Impact Your App
For whom the bridge tolls

Toll-free bridging
• From a CF type to a NS type
• From a NS type to a CF type
• Zero cost at cast

```swift
NSArray *array = @[];
CFArrayGetCount((CFArrayRef)array);
let data = NSData()
let d = data as? Data
```
Bridges and How They Impact Your App
For whom the bridge tolls

Toll-free bridging
• From a CF type to a NS type
• From a NS type to a CF type
• Zero cost at cast
• Extra cost at usage

```
NSArray *array = @[];
CFArrayGetCount((CFArrayRef)array);

let data = NSData()
let d = data as? Data
```
Bridges and How They Impact Your App
For whom the bridge tolls

Toll-free bridging
• From a CF type to a NS type
• From a NS type to a CF type
• Zero cost at cast
• Extra cost at usage

Swift bridges

NSArray *array = @[];
CFArrayGetCount((CFArrayRef)array);

let data = NSData()
let d = data as? Data
Bridges and How They Impact Your App
For whom the bridge tolls

Toll-free bridging
• From a CF type to a NS type
• From a NS type to a CF type
• Zero cost at cast
• Extra cost at usage

Swift bridges
• From a reference type to a struct

Swift code:

```swift
NSArray *array = @[];
CFArrayGetCount((CFArrayRef)array);

let data = NSData();
let d = data as? Data
```
Bridges and How They Impact Your App
For whom the bridge tolls

Toll-free bridging
- From a CF type to a NS type
- From a NS type to a CF type
- Zero cost at cast
- Extra cost at usage

Swift bridges
- From a reference type to a struct
- From a struct to a reference type

```swift
NSArray *array = @[];
CFArrayGetCount((CFArrayRef)array);

let data = NSData()
let d = data as? Data
```
Bridges and How They Impact Your App

For whom the bridge tolls

Toll-free bridging
- From a CF type to a NS type
- From a NS type to a CF type
- Zero cost at cast
- Extra cost at usage

Swift bridges
- From a reference type to a struct
- From a struct to a reference type
- Cost is paid in advance

```swift
NSArray *array = @[];
CFArrayGetCount((CFArrayRef)array);

let data = NSData()
let d = data as? Data
```
Bridges and How They Impact Your App
For whom the bridge tolls

Toll-free bridging
• From a CF type to a NS type
• From a NS type to a CF type
• Zero cost at cast
• Extra cost at usage

Swift bridges
• From a reference type to a struct
• From a struct to a reference type
• Cost is paid in advance
• Normal cost at usage

NSArray *array = @[];
CFArrayGetCount((CFArrayRef)array);
let data = NSData()
let d = data as? Data
CFIndex CFArrayGetCount(CFArrayRef array) {
    CF_OBJC_FUNCDISPATCHV(CFArrayGetTypeID(), CFIndex, (NSArray *)array, count);
    return array->count;
}
CFIndex CFArrayGetCount(CFArrayRef array) {
    CF_OBJC_FUNCDISPATCHV(CFArrayGetTypeID(), CFIndex, (NSArray *)array, count);
    return array->count;
}
// CF Bridging

CFIndex CFArrayGetCount(CFArrayRef array) {
    CF_OBJC_FUNCDISPATCHV(CFArrayGetTypeID(), CFIndex, (NSArray *)array, count);
    return array->count;
}
// CF Bridging

CFIndex CFArrayGetCount(CFArrayRef array) {
    if (CF_IS_OBJC(CFArrayGetTypeID(), array)) {
        return [(NSArray *)obj count];
    }
    return array->count;
}
CFIndex CFArrayGetCount(CFArrayRef array) {
    if (object_getClass(array) != CFClasses[CFArrayGetTypeID()]) {
        return [(NSArray *)obj count];
    }
    return array->count;
}
// CF Bridging

NSArray *array = @[];
CFArrayGetCount((CFArrayRef)array);
// CF Bridging

NSArray *array = @[];
CFArrayGetCount((CFArrayRef)array);
// CF Bridging

NSArray *array = @[];
CFArrayGetCount((CFArrayRef)array);

Small and Unknown Frequency
// CF Bridging

NSArray *array = @[];
CFArrayGetCount((CFArrayRef)array);
// Swift Bridging

extension Data : _ObjectiveCBridgeable {

...  

    public static func _conditionallyBridgeFromObjectiveC(_ input: NSData, result: inout Data?) -> Bool {
        // We must copy the input because it might be mutable
        // just like storing a value type in ObjC
        result = Data(referencing: input)
        return true
    }

...  

}
// Swift Bridging

struct Data {
...
    public init(referencing reference: NSData) {
        _backing = _DataStorage(immutableReference: reference.copy() as! NSData)
        _sliceRange = 0..<reference.length
    }
...
}
```swift
// Swift Bridging

let data = NSData()
let d = data as? Data
```
// Swift Bridging

let data = NSData()
let d = data as? Data
let data = NSData()
let d = data as? Data
// Swift Bridging

let data = NSData()
let d = data as? Data

Usually small and infrequent...
let data = NSData()
let d = data as? Data
// Swift Bridging

let data = NSData()
let d = data as? Data

Usually small and infrequent...
// Swift Bridging

let data = NSData()
let d = data as? Data

Usually small and infrequent...
Strings, Ranges, and Text
Strings are everywhere
Invest in performance that matters to your users
String bridging
Ranges
Text layout and rendering
String bridging
Ranges
Text layout and rendering
Example 1: UILabel
Waffles
String Bridging
Example 1: UILabel

// Swift
var text = label.text
String Bridging

Example 1: UILabel

```swift
// Swift
var text = label.text
```

```swift
// Swift Interface - UIKit
open class UILabel : UIView {
    open var text : String?
}
```
String Bridging

Example 1: UILabel

// Swift Interface - UIKit
open class UILabel : UIView {
    open var text : String?
}

// Objective-C - UIKit
@interface UILabel : UIView

@property(nullable, nonatomic, copy) NSString *text;
@end

// Swift
var text = label.text
String Bridging
Example 1: UILabel

// Swift Interface - UIKit
open class UILabel : UIView {
    open var text : String?
}

// Objective-C - UIKit
@interface UILabel : UIView
@property(nullable, nonatomic, copy) NSString *text;
@end
String Bridging

Example 1: UILabel

var text = label.text
String Bridging
Example 1: UILabel

Swift

```swift
var text = label.text
```

Framework

struct String

class NSString

class NSString
String Bridging

Example 1: UILabel

```swift
var text = label.text
```

Swift

struct String

class NSString

Framework

class NSString

copy
var text = label.text
var text = label.text
var text = label.text
Example 2: NSTextStorage
In considering the origin of species, it is quite conceivable that a naturalist, reflecting on the mutual affinities of organic beings, on their embryological relations, their geographical distribution, geological succession, and other such facts, might come to the conclusion that species had not been independently created, but had descended, like varieties, from other species. Nevertheless, such a conclusion, even if well founded, would be unsatisfactory, until it could be shown how the innumerable species, inhabiting this world have been modified, so as to acquire that perfection of structure and coadaptation which justly excites our admiration. Naturalists continually refer to external conditions, such as climate, food, etc., as the only possible cause of variation. In one limited sense, as we shall hereafter see, this may be true; but it is preposterous to attribute to mere external conditions, the structure, for instance, of the woodpecker, with its feet, tail, beak, and tongue.
In considering the origin of species, it is quite conceivable that a naturalist, reflecting on the mutual affinities of organic beings, on their embryological relations, their geographical distribution, geological succession, and other such facts, might come to the conclusion that species had not been independently created, but had descended, like varieties, from other species. Nevertheless, such a conclusion, even if well founded, would be unsatisfactory, until it could be shown how the innumerable species, inhabiting this world have been modified, so as to acquire that perfection of structure and coadaptation which justly excites our admiration. Naturalists continually refer to external conditions, such as climate, food, etc., as the only possible cause of variation. In one limited sense, as we shall hereafter see, this may be true; but it is preposterous to attribute to mere external conditions, the structure, for instance, of the woodpecker, with its foot, tail, beak, and tongue,
String Bridging
Example 2: NSTextStorage

// Swift
var text = textView.textStorage.string
String Bridging
Example 2: NSTextStorage

// Swift
var text = textView.textStorage.string
String Bridging

Example 2: NSTextStorage

// Swift
```swift
var text = textView.textStorage.string
```

// Swift Interface - UIKit
```swift
open class NSMutableAttributedString : NSAttributedString {
    open var string: NSString
}
```
String Bridging
Example 2: NSTextStorage

// Swift
var text = textView.textStorage.string

// Objective-C - UIKit
@interface NSTextStorage : NSMutableAttributedString

// Objective-C - Foundation
@interface NSMutableAttributedString : NSAttributedString

@property(readonly, copy) NSString *string;

// Objective-C - UIKit
@interface NSMutableAttributedString : NSAttributedString

// Objective-C - Foundation
@interface NSAttributedString : NSAttributeString

@property(nonatomic, readonly, copy) NSAttributedString *string;

@property(nonatomic, readonly, copy) NSAttributedString *mutableAttributedString;
String Bridging
Example 2: NSTextStorage

// Swift
var text = textView.textStorage.string

// Objective-C - UIKit
@interface NSTextStorage : NSMutableAttributedString

// Objective-C - Foundation
@interface NSMutableAttributedString : NSAttributedString
@property(readonly, copy) NSString *string;

// Objective-C - UIKit
@interface NSTextStorage

// Objective-C - Foundation
@interface NSMutableAttributedString :
NSAttributedString

@interface NSAttributedString

@end

@end

@property(readonly, copy) NSString *string;
String Bridging
Example 2: NSTextStorage

```swift
var text = textView.textStorage.string
```
String Bridging
Example 2: NSTextStorage

Swift

```swift
var text = textView.textStorage.string
```

Framework

```
struct String
class NSString
```

```
class NSString
```

copy!
String Bridging
Example 2: NSTextStorage

```swift
var text = textView.textStorage.mutableString
```
String Bridging
Example 2: NSTextStorage

```swift
var text = textView.textStorage.mutableString
```

class NSMutableString

```
```

**Swift**

**Framework**

```
no copy!
```

```
class NSMmutableString
```

```
class NSMmutableString
```
var text = textView.textStorage.string
var text = textView.textStorage.string
var text = textView.textStorage.string

1 KB?
var textView.textStorage.string

1 KB?

1 MB?
String bridging
Ranges
Text layout and rendering
“Ranges”
<table>
<thead>
<tr>
<th>Visible Components</th>
<th>Unicode Scalar Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>🙈</td>
<td>0x1F926</td>
</tr>
<tr>
<td>🔄</td>
<td>0x1F3FB</td>
</tr>
<tr>
<td>🚦</td>
<td>0x200D</td>
</tr>
<tr>
<td>💃</td>
<td>0x2640</td>
</tr>
<tr>
<td>🌋</td>
<td>0xFE0F</td>
</tr>
<tr>
<td>Visible Components</td>
<td>Unicode Scalar Value</td>
</tr>
<tr>
<td>--------------------</td>
<td>----------------------</td>
</tr>
<tr>
<td>FACE PALM</td>
<td>0x1F926</td>
</tr>
<tr>
<td></td>
<td>0x1F3FB</td>
</tr>
<tr>
<td></td>
<td>0x200D</td>
</tr>
<tr>
<td></td>
<td>0x2640</td>
</tr>
<tr>
<td></td>
<td>0xFE0F</td>
</tr>
<tr>
<td>Visible Components</td>
<td>Unicode Scalar Value</td>
</tr>
<tr>
<td>--------------------</td>
<td>----------------------</td>
</tr>
<tr>
<td></td>
<td>0x1F926</td>
</tr>
<tr>
<td></td>
<td>0x1F3FB</td>
</tr>
<tr>
<td></td>
<td>0x200D</td>
</tr>
<tr>
<td></td>
<td>0x2640</td>
</tr>
<tr>
<td></td>
<td>0xFE0F</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Example 1:
Working with NSAttributedString
“What a 🎉💩!”
“What a 🎉💩!”
String Ranges
Example 1: Working with NSAttributedString

“What a 🎉💩!”

```swift
let string = "What a 🎉💩!"
let nsstring = string as NSString
let nsrange = nsstring.rangeOfString("💩")
var attributedString = NSMutableAttributedString(string: string)
attributedString.addAttribute(.backgroundColor,
    value: color,
    range: nsrange)
```
String Ranges
Example 1: Working with NSAttributedString

“What a 🎉💩!”

```swift
let string = "What a 🎉💩!"
let nsstring = string as NSString
let nsrange = nsstring.rangeOfString("💩")
var attributedString = NSMutableAttributedString(string: string)
attributedString.addAttribute(.backgroundColor,
                              value: color,
                              range: nsrange)
```
String Ranges
Example 1: Working with NSAttributedString

“What a 🎉💩!”

```swift
let string = "What a 🎉💩!
let nsstring = string as NSString
let nsrange = nsstring.rangeOfString("💩")
var attributedString = NSMutableAttributedString(string: string)
attributedString.addAttribute(.backgroundColor,
    value: color,
    range: nsrange)
```
String Ranges

Example 1: Working with NSAttributedString

“What a 🎉💩!”

```swift
let string = "What a 🎉💩!"
var attributedString = NSMutableAttributedString(string: string)

let backgroundRange = string.range(of: "💩")!
attributedString.addAttribute(.backgroundColor, value: color, range: NSRange(backgroundRange, in: string))
```
String Ranges
Example 1: Working with NSAttributedString

“What a 🎉💩!”

```swift
let string = "What a 🎉💩!"
var attributedString = NSMutableAttributedString(string: string)

let backgroundRange = string.range(of: "💩")!
attributedString.addAttribute(.backgroundColor, value: color, range: NSRange(backgroundRange, in: string))
```
Example 2:
Working with NSRegularExpression
// String Ranges: Working with NSRegularExpression

<html>
<body>
  <div>
    <span>Hello</span> <b>Swift</b><span>test</span></b>
  </div>
</body>
</html>
extension String {
    func rangeFromNSRange(nsRange : NSRange) -> Range<Index>? {
        guard nsRange.location != NSNotFound else { return nil }
        let from16 = utf16.startIndex.advanced(by: nsRange.location)
        let to16 = from16.advanced(by: nsRange.length)
        if let from = Index(from16, within: self),
            let to = Index(to16, within: self) {
            return from..<to
        }
        return nil
    }
}
import Foundation

func findTags(in string: String) -> [Range<String.Index>] {
    var found = [Range<String.Index>]()
    let re = try! NSRegularExpression(pattern: "<([a-z][a-z0-9]*)/>")
    for match in re.matches(in: string, range: NSRange(string.startIndex..<string.endIndex, in: string)) {
        found.append(Range(match.rangeAt(1), in: string)!)}
    return found}
import Foundation

func findTags(in string: String) -> [Range<String.Index>] {
    var found = [Range<String.Index>]()
    let re = try! NSRegularExpression(pattern: "<([a-z][a-z0-9]*)/>")
    for match in re.matches(in: string, range: NSRange(string.startIndex..<string.endIndex, in: string)) {
        found.append(Range(match.rangeAt(1), in: string)!
    }
    return found
}
String bridging

Ranges

Text layout and rendering
Text is hard
40 iOS localizations
40 iOS localizations
35 macOS localizations
40 iOS localizations
35 macOS localizations
39 watchOS localizations
40 iOS localizations
35 macOS localizations
39 watchOS localizations
40 tvOS localizations
40 iOS localizations
35 macOS localizations
39 watchOS localizations
40 tvOS localizations
More than 300 other languages
Example:
A Tale of Two Labels
Millennium Park
Loop
Millennium Park
Loop

Tribune Tower
Near North Side

Union Park
West Loop

Rookery Building
Robie House
Hyde Park
Burnham Park
Museum Campus
Union Station
Loop
大同
虎跳峡
丽江
龙门石窟
洛阳
Runs Frequently

Short Length

Long Length

Runs Infrequently

measure!
Postmortem
Example: A Tale of Two Labels

Initial conditions qualified for fast rendering
Postmortem
Example: A Tale of Two Labels

Initial conditions qualified for fast rendering

Input change forced rendering to slower path
Postmortem
Example: A Tale of Two Labels

Initial conditions qualified for fast rendering
Input change forced rendering to slower path
App used older layout practices
What You Can Do

Higher-level strategies
What You Can Do

Higher-level strategies

Use standard label controls
What You Can Do
Higher-level strategies

Use standard label controls

3x faster rendering with NSTextField in macOS 10.13
What You Can Do
Higher-level strategies

Use modern layout practices
What You Can Do
Lower-level tips

Set rendering attributes for attributed strings

```swift
let attributes: [NSAttributedStringKey : Any] = [
    .font: UIFont.systemFont(ofSize:.systemFontSize),
    .paragraphStyle: NSParagraphStyle.default,
    .foregroundColor: UIColor.darkText
]

let myString = NSAttributedString(string: "Hello", attributes: attributes)
```
What You Can Do

Lower-level tips

Specify alignment and writing direction if known

```swift
// Only do this if you’re absolutely sure your text doesn’t have mixed writing directions
var myParagraphStyle = NSMutableParagraphStyle()
myParagraphStyle.baseWritingDirection = .leftToRight
myParagraphStyle.alignment = .left
```
What You Can Do

Lower-level tips

Use clipping line break mode for single line labels

// Only do this if you’re sure your text doesn’t require wrapping
var myParagraphStyle = NSMutableParagraphStyle()
myParagraphStyle.lineBreakMode = .byClipping
More Information

https://developer.apple.com/wwdc17/244
## Related Sessions

<table>
<thead>
<tr>
<th>Session</th>
<th>Year</th>
</tr>
</thead>
<tbody>
<tr>
<td>Understanding Swift Performance</td>
<td>WWDC 2016</td>
</tr>
<tr>
<td>What’s New in Cocoa</td>
<td>WWDC 2017</td>
</tr>
<tr>
<td>What’s New in Foundation</td>
<td>WWDC 2017</td>
</tr>
<tr>
<td>Modernizing Grand Central Dispatch Usage</td>
<td>WWDC 2017</td>
</tr>
<tr>
<td>Cocoa Development Tips</td>
<td>WWDC 2017</td>
</tr>
<tr>
<td>Writing Energy Efficient Apps</td>
<td>WWDC 2017</td>
</tr>
<tr>
<td>Labs</td>
<td></td>
</tr>
<tr>
<td>--------------</td>
<td>---</td>
</tr>
<tr>
<td><strong>Cocoa Lab</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Technology Lab B</strong></td>
<td></td>
</tr>
</tbody>
</table>