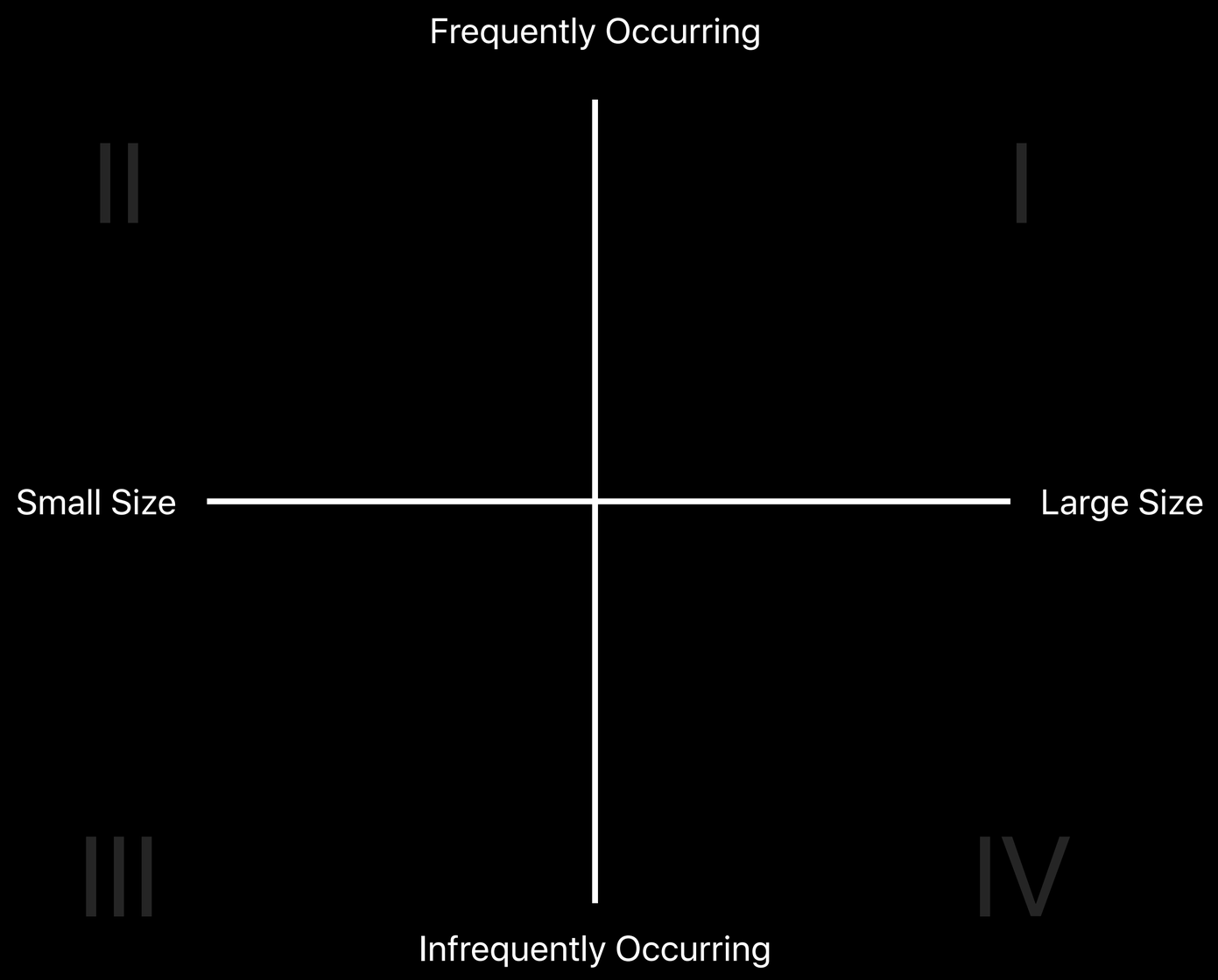
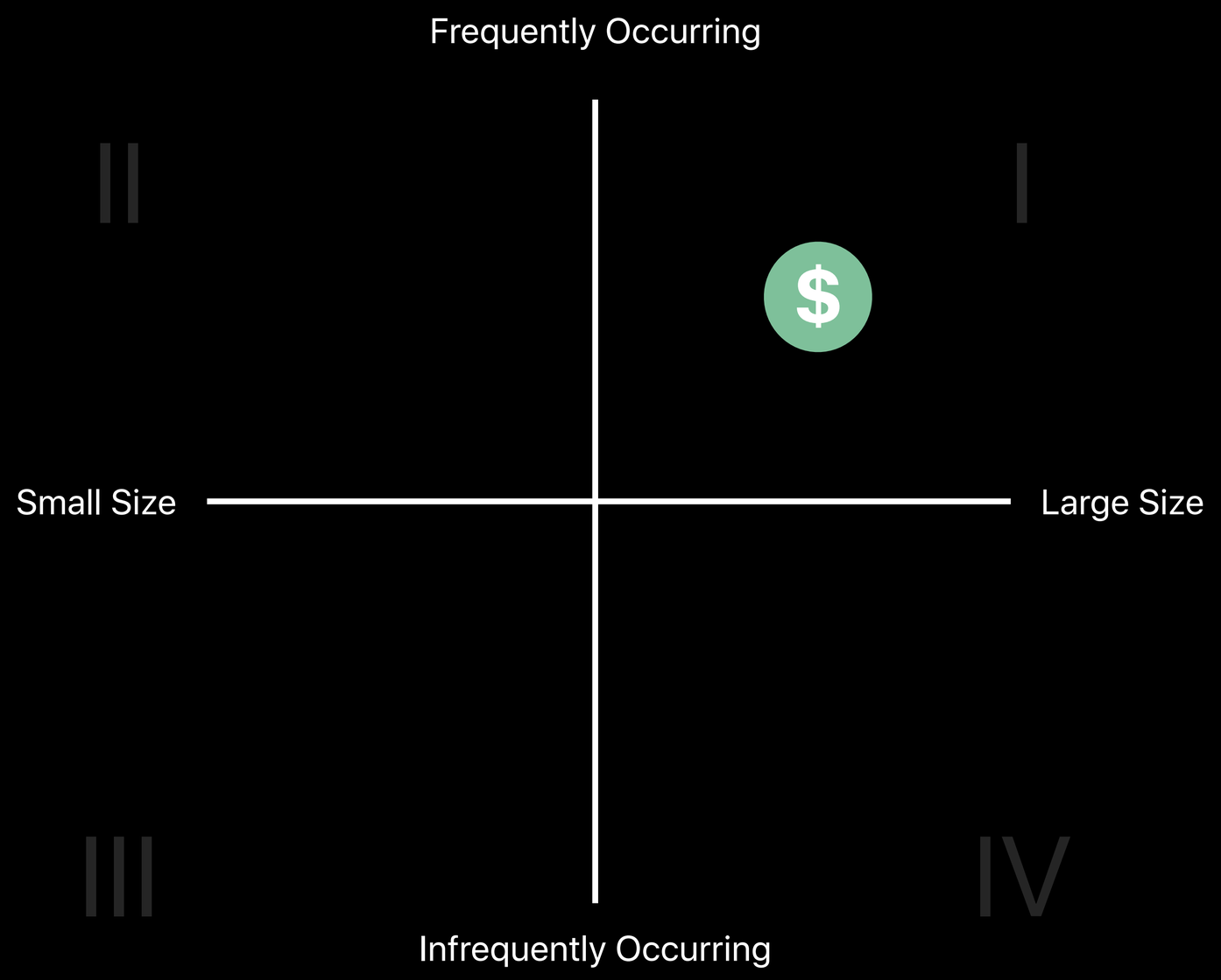
Efficient Interaction with Frameworks

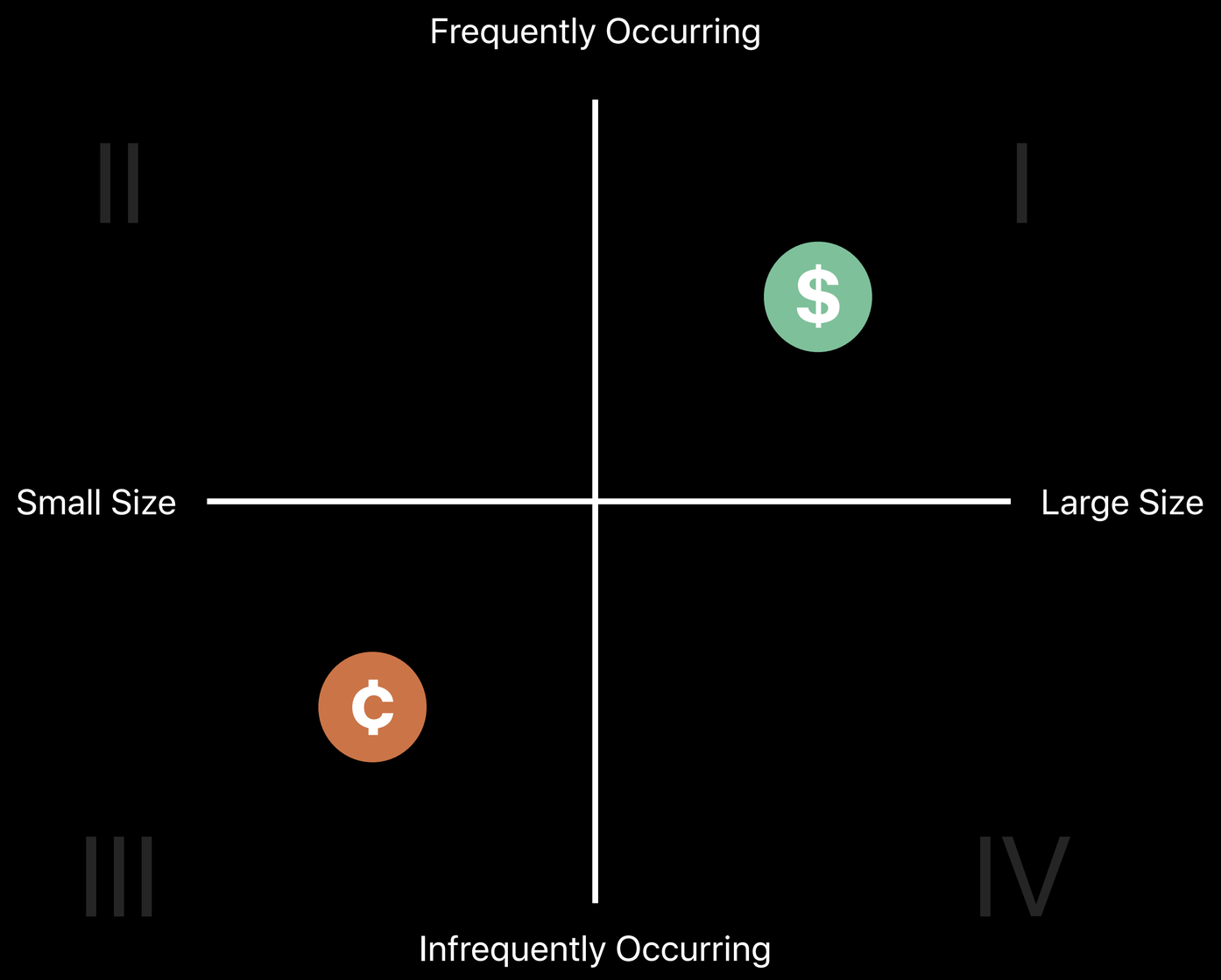
Performance case studies

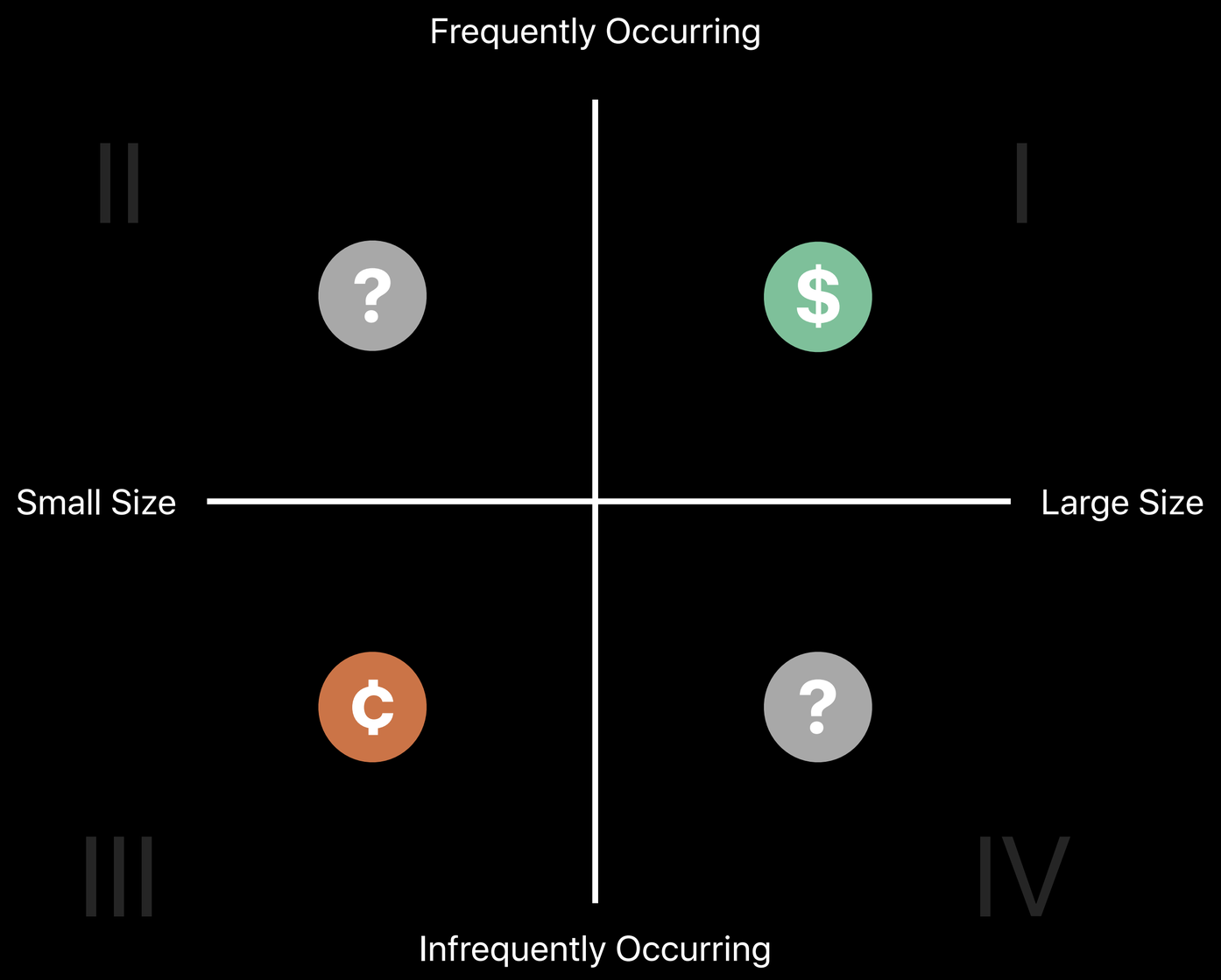
Session 244

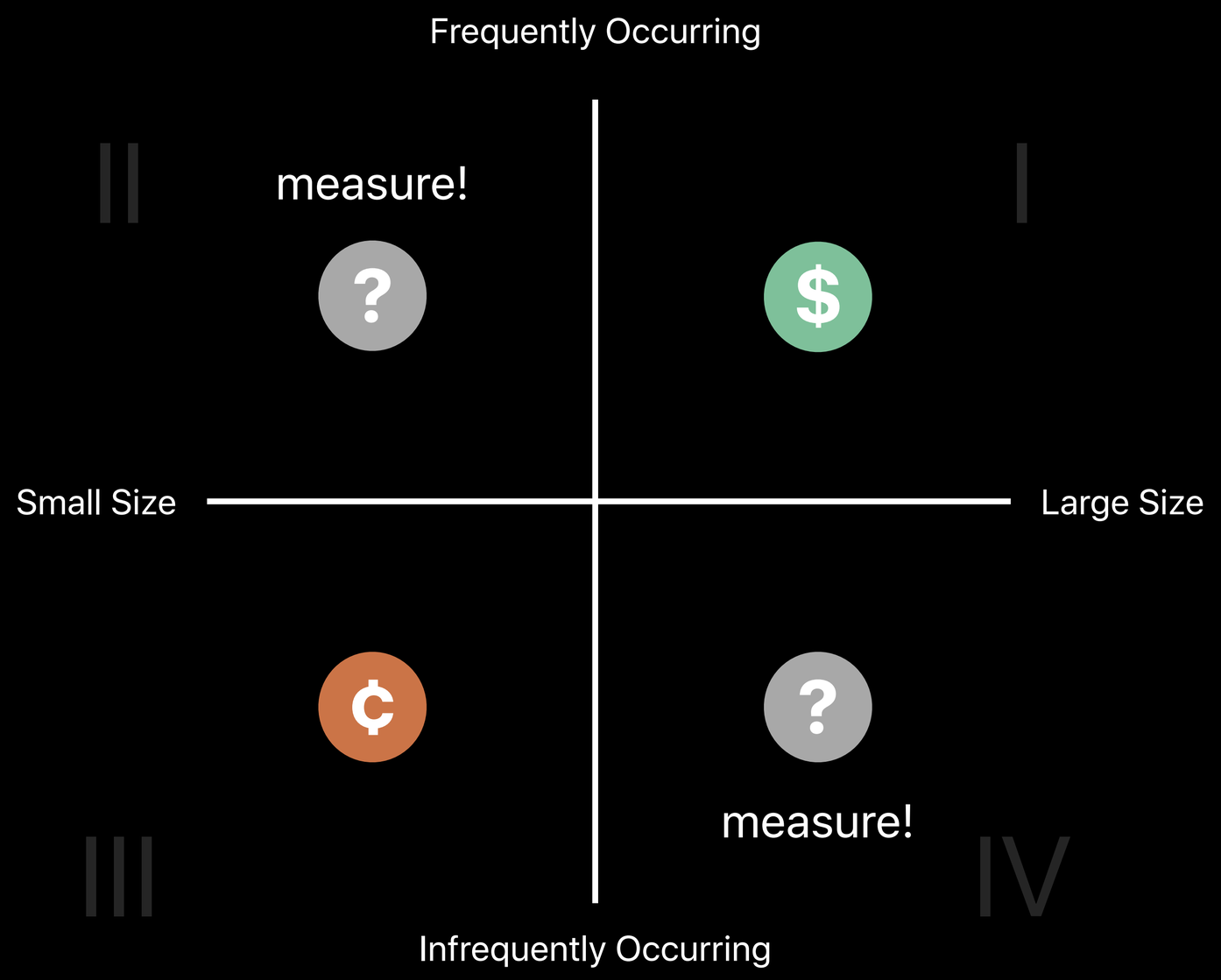
Philippe Hausler, Foundation
Donna Tom, TextKit


Frequently Occurring
II
I
Small Size
Large Size
III
IV
Infrequently Occurring


Frequently Occurring
II
I
$
Small Size
Large Size
III
IV
Infrequently Occurring


Frequently Occurring
II
I
$
Small Size
Large Size
¢
III
IV
Infrequently Occurring


Frequently Occurring
II
I
?
$
Small Size
Large Size
¢
?
III
IV
Infrequently Occurring


Frequently Occurring
II
measure!
?
I
$
Small Size
Large Size
¢
?
measure!
III
IV
Infrequently Occurring

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?

```
@implementation Container {
    NSMutableArray<Item *> *_elements;
}
- (NSArray<Item *> *)elements {
    return [_elements copy];
}
@end
```

Copy on Write Collections

Let's milk this joke a bit more

```
a = [NSMutableArray new]
```

Copy on Write Collections

Let's milk this joke a bit more

```
a = [NSMutableArray new]
```

Copy on Write Collections

Let's milk this joke a bit more

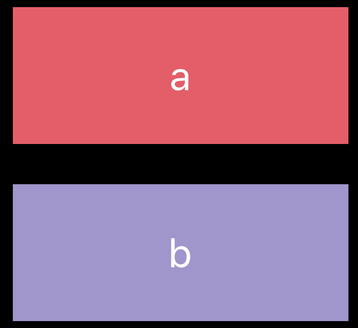
```
a = [NSMutableArray new]
```

Copy on Write Collections

Let's milk this joke a bit more

```
a = [NSMutableArray new]
```


```
b = [a copy]
```

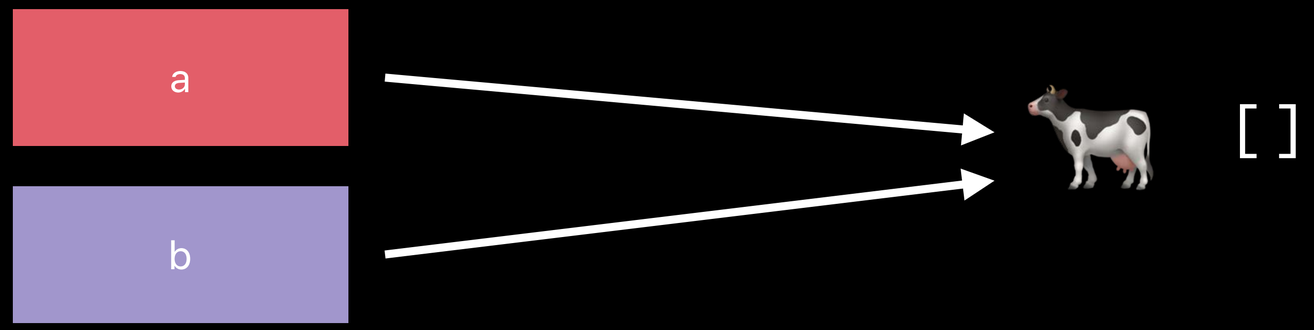


Copy on Write Collections

Let's milk this joke a bit more

```
a = [NSMutableArray new]
```


```
b = [a copy]
```

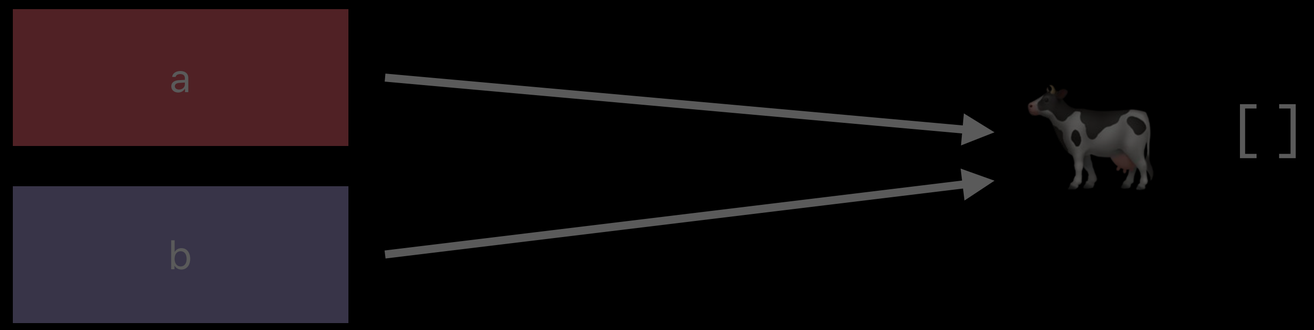


Copy on Write Collections

Let's milk this joke a bit more

```
a = [NSMutableArray new]
```


```
b = [a copy]
```

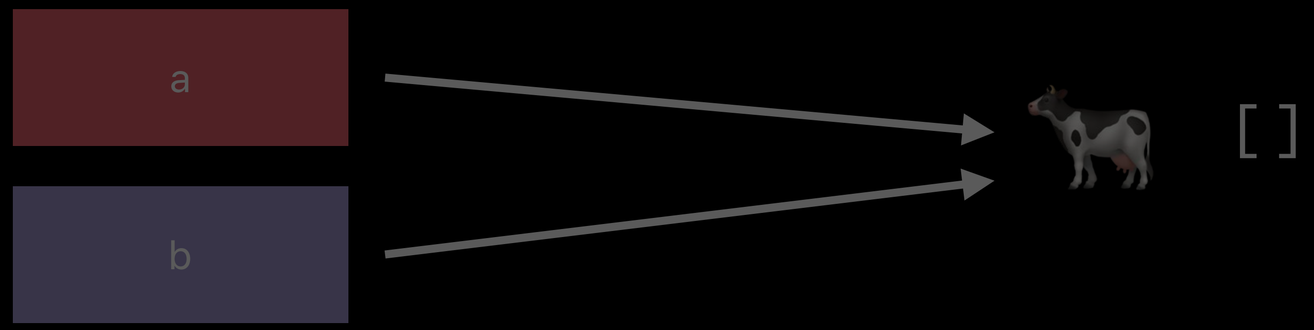


Copy on Write Collections

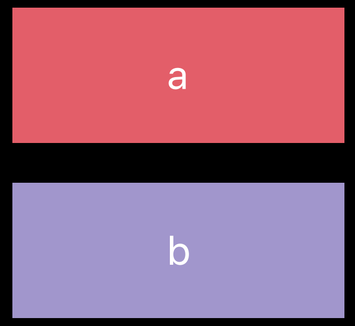
Let's milk this joke a bit more

```
a = [NSMutableArray new]
```


```
b = [a copy]
```




```
[a addObject:@"A"]
```

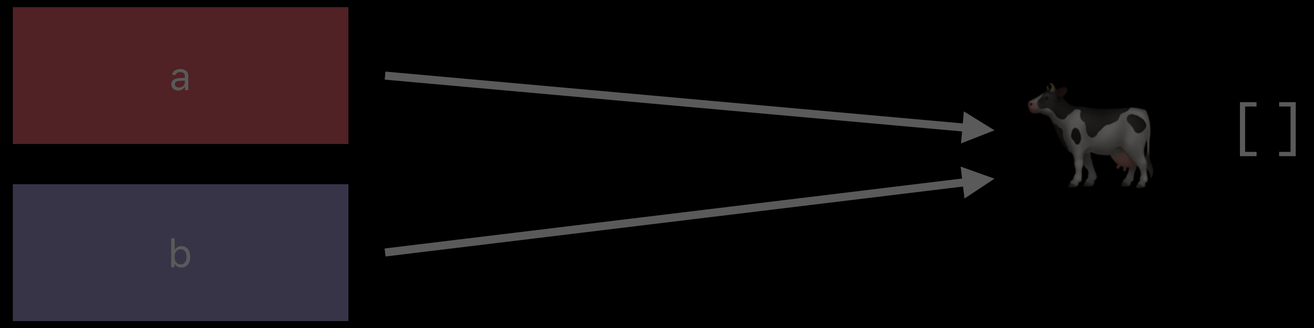


Copy on Write Collections

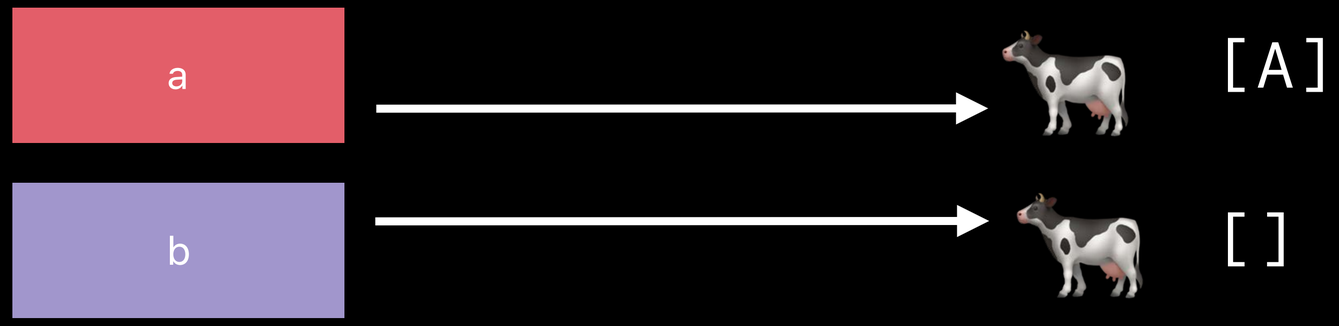
Let's milk this joke a bit more

```
a = [NSMutableArray new]
```


```
b = [a copy]
```

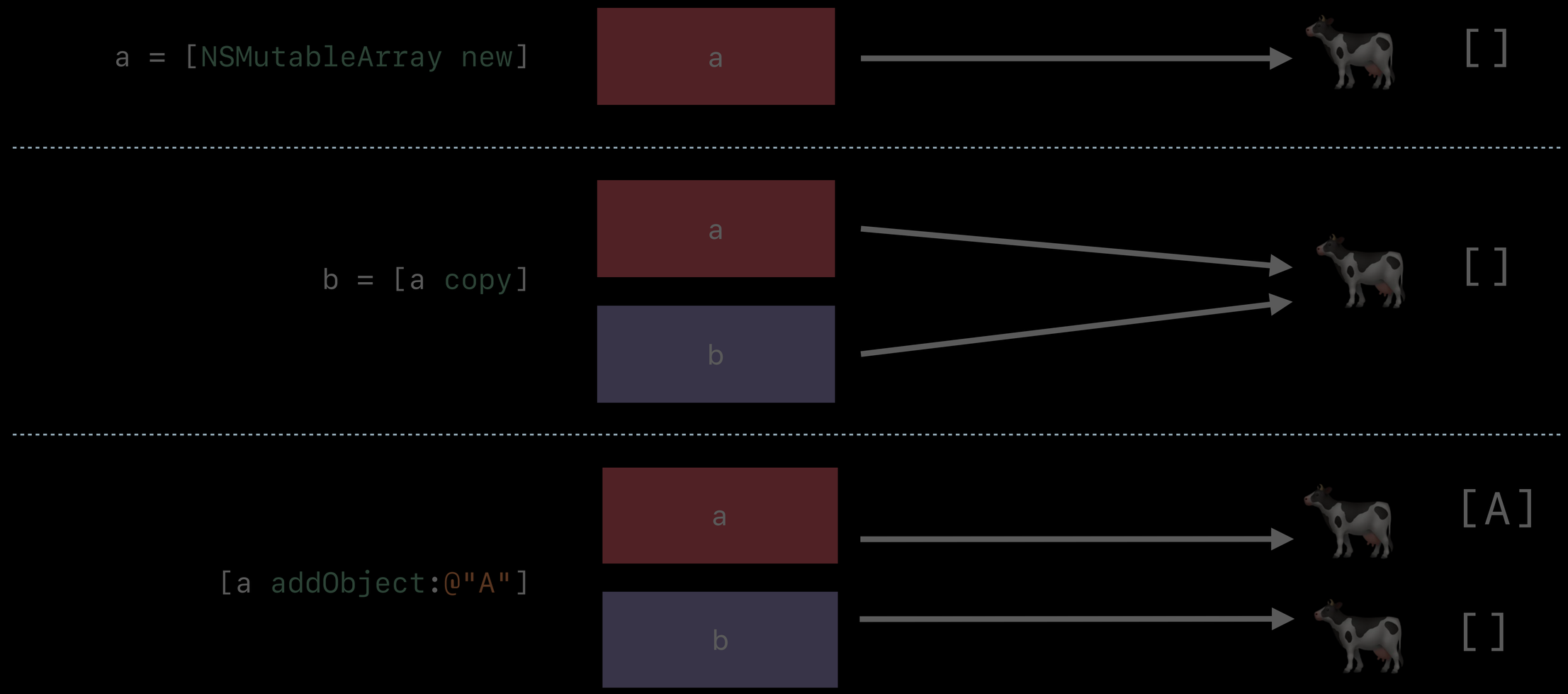



```
[a addObject:@"A"]
```



Copy on Write Collections

Let's milk this joke a bit more

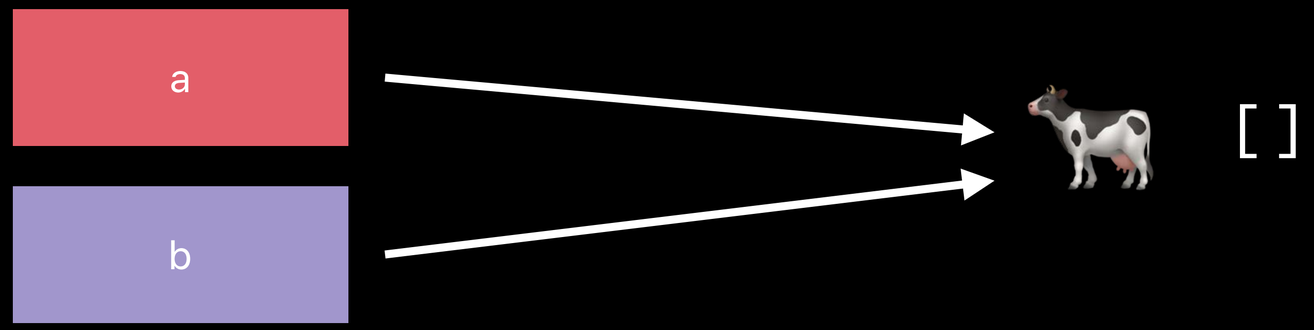


Copy on Write Collections

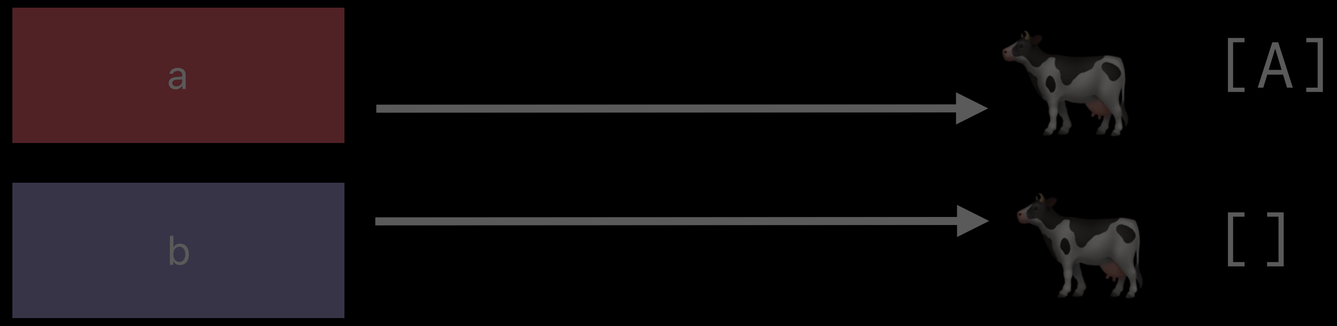
Let's milk this joke a bit more

```
a = [NSMutableArray new]
```


```
b = [a copy]
```




```
[a addObject:@"A"]
```



```
//Leveraging Copy-on-write, Steer your code in the right direction

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

```
//Leveraging Copy-on-write, Steer your code in the right direction

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

```
//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;
```

```
//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];
    // 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 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];
}
```

```
//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

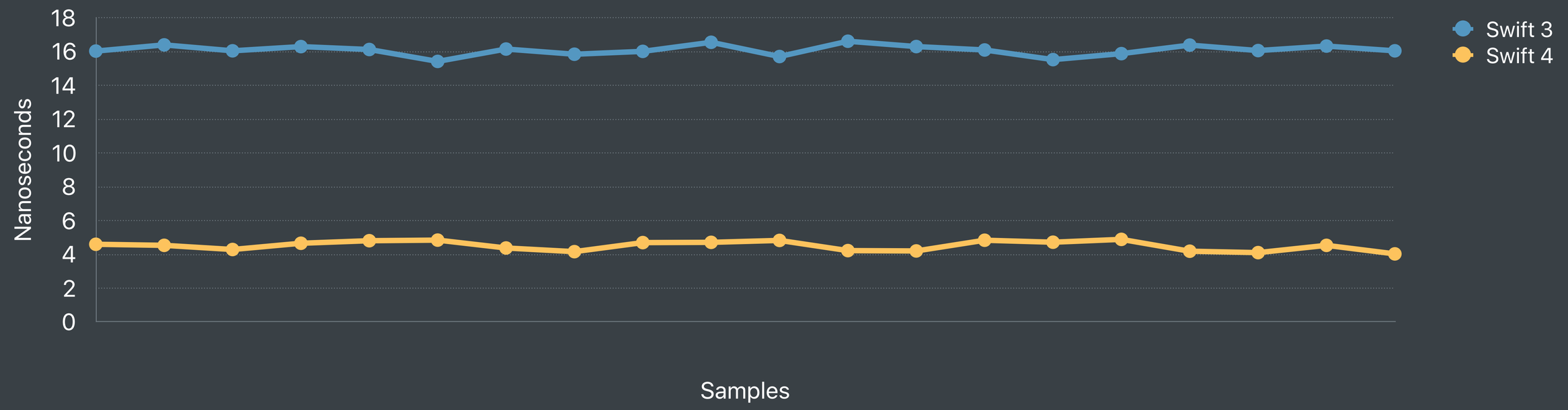
Indexing is only a few instructions in optimized builds

Appending is dramatically faster

Replacing regions is faster too

```swift
//Subscripting Data

func findZeroByte(_ data: Data) -> Data.Index? {
    for index in data.indices {
        if data[index] == 0 { return index }
    }
    return nil
}
```

```swift
//Subscripting Data

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

```swift
// Leveraging Data, Don’t Believe the Lore

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

```swift
// Leveraging Data, Don’t Believe the Lore

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

```swift
// Leveraging Data, Don't Believe the Lore

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

```swift
// 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) }
```

```swift
// 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)
```

```swift
// 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)
```

```swift
// 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))
```

```swift
// 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)]
```

Bridges and How They Impact Your App

For whom the bridge tolls

```
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

```
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

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

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

Toll-free bridging

- From a CF type to a NS type

Bridges and How They Impact Your App

For whom the bridge tolls

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

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

Toll-free bridging

- From a CF type to a NS type
- From a NS type to a CF type

Bridges and How They Impact Your App

For whom the bridge tolls

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

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

Toll-free bridging

- From a CF type to a NS type
- From a NS type to a CF type
- Zero cost at cast

Bridges and How They Impact Your App

For whom the bridge tolls

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

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

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

Bridges and How They Impact Your App

For whom the bridge tolls

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

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

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

Bridges and How They Impact Your App

For whom the bridge tolls

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

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

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

Bridges and How They Impact Your App

For whom the bridge tolls

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

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

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

Bridges and How They Impact Your App

For whom the bridge tolls

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

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

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

Bridges and How They Impact Your App

For whom the bridge tolls

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

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

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

```
// CF Bridging

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) {
    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;
}
```

```
// CF Bridging

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);
```

Small and Unknown Frequency

```swift
// 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
// 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
// Swift Bridging

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

```swift
// Swift Bridging

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

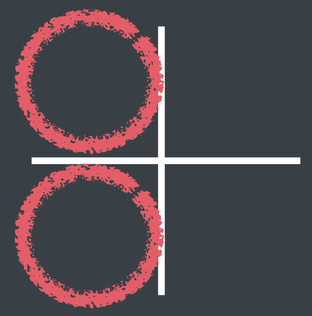
Usually small and infrequent...

```
// Swift Bridging

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

Usually small and infrequent...

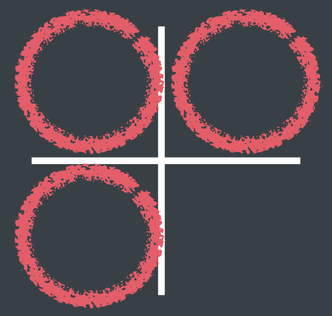
```swift
// Swift Bridging

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

Usually small and infrequent...



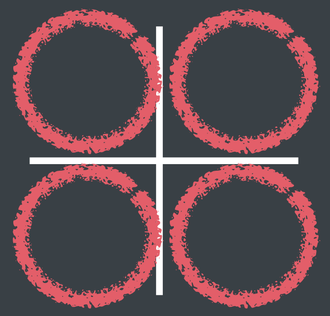
```swift
// Swift Bridging

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

Usually small and infrequent...



```swift
// Swift Bridging

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

Usually small and infrequent...

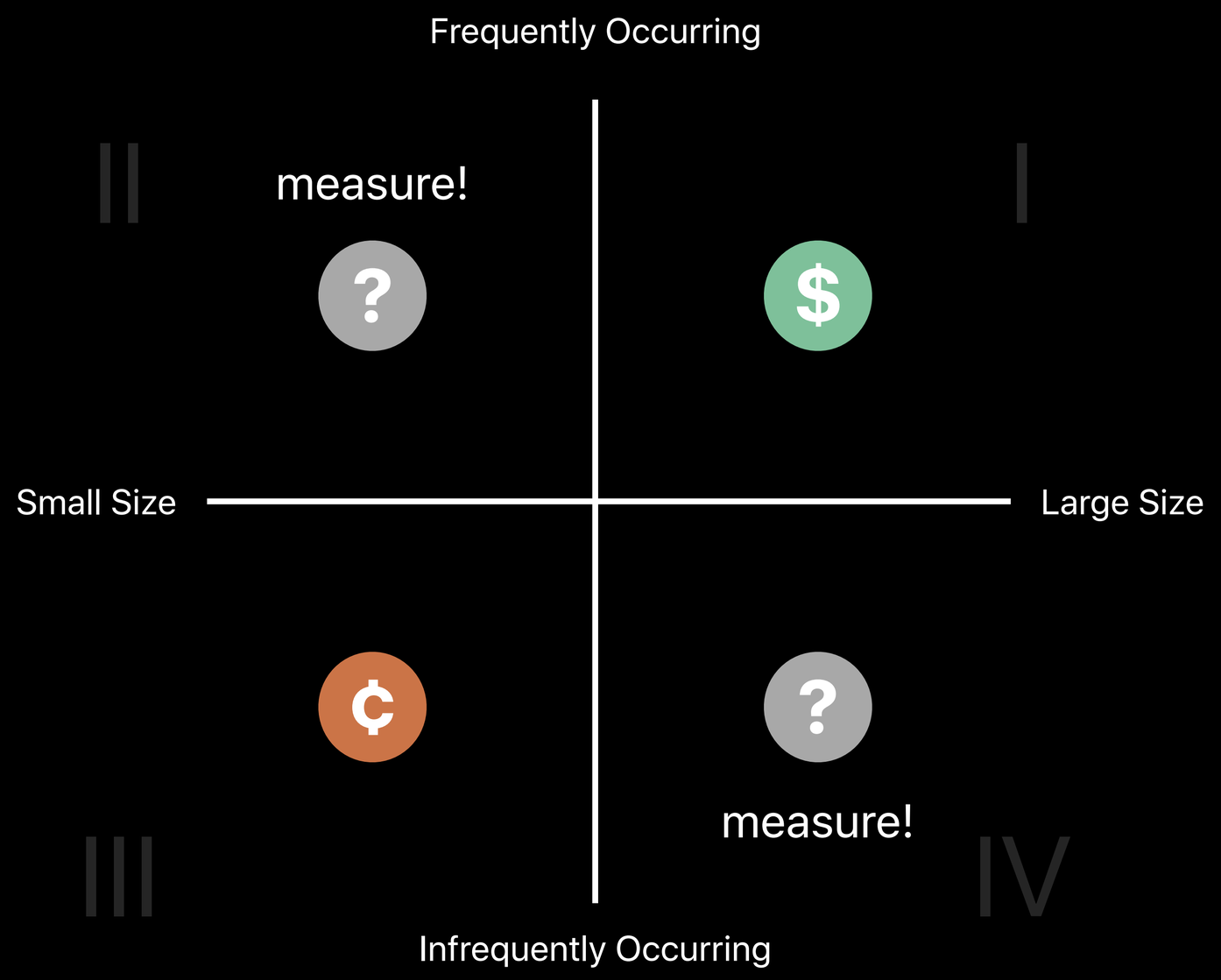


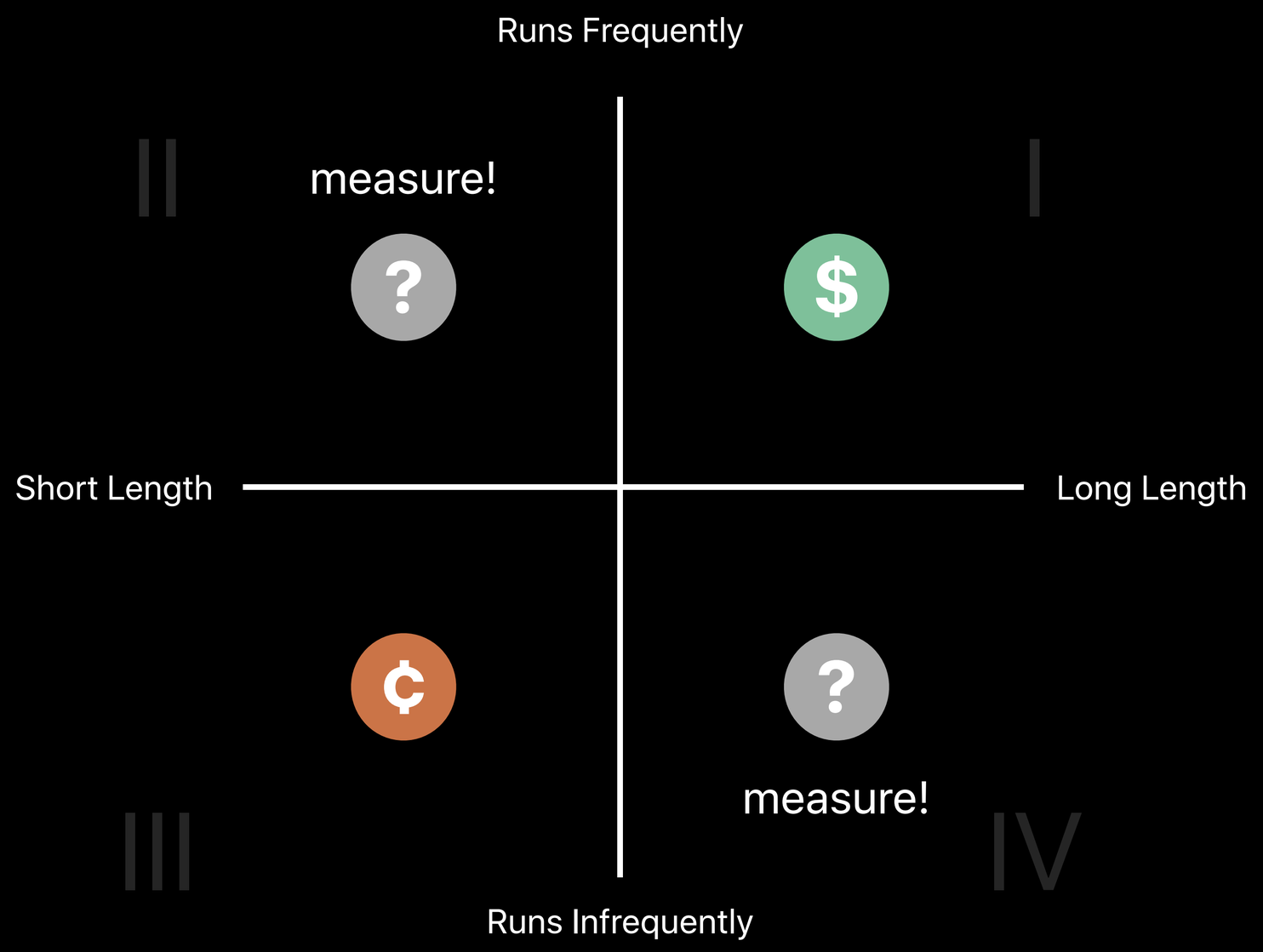
Strings, Ranges, and Text

Donna Tom, TextKit

Strings are everywhere

Invest in performance
that matters to your users


Frequently Occurring
II
measure!
?
I
$
Small Size
Large Size
¢
?
measure!
III
IV
Infrequently Occurring


Runs Frequently
II
measure!
?
I
$
Short Length
Long Length
¢
?
measure!
III
IV
Runs Infrequently

String bridging

Ranges

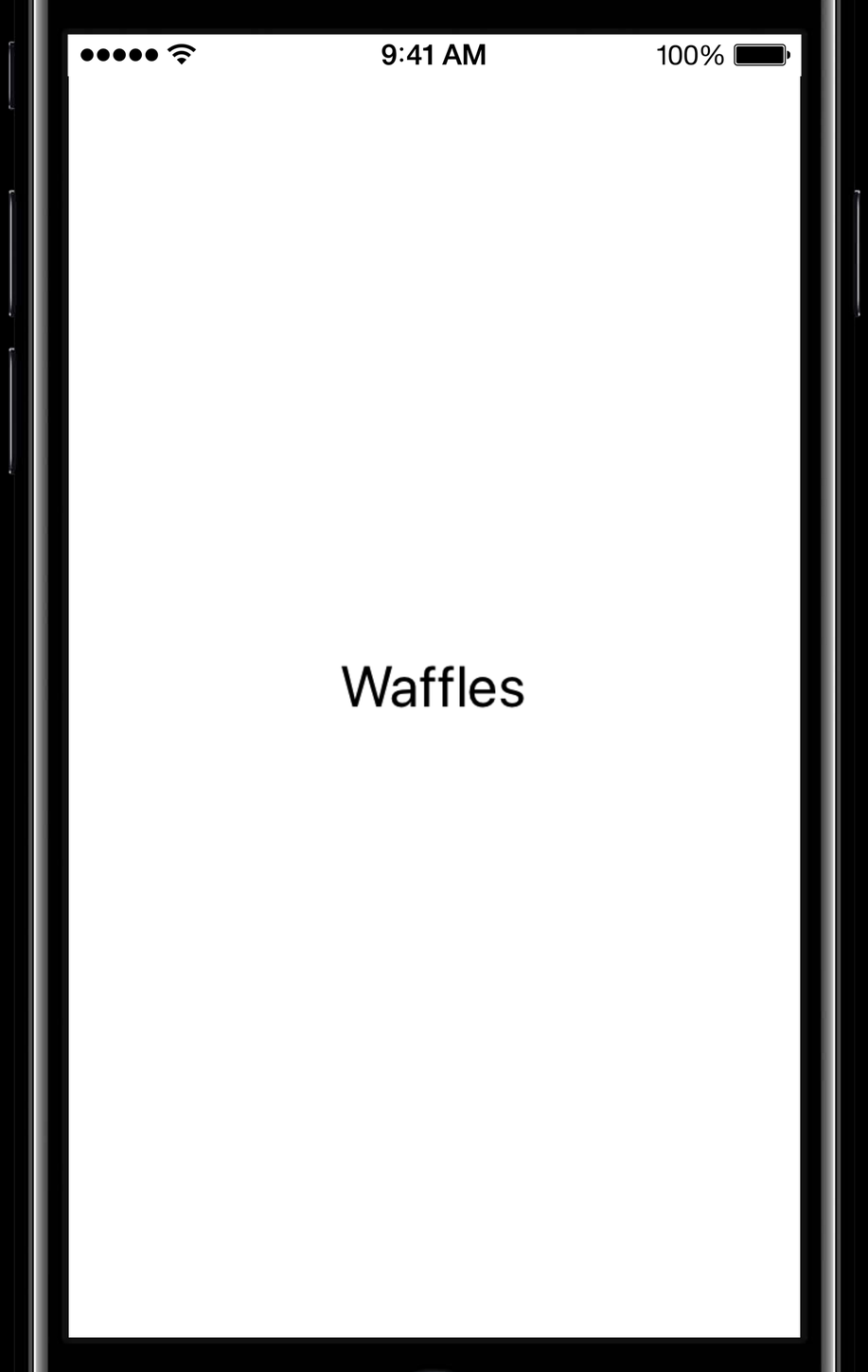
Text layout and rendering

String bridging

Ranges

Text layout and rendering

Example 1: UILabel


9:41 AM
100%
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
// Swift
var text = label.text
```

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

```objc
// Objective-C - UIKit

@interface UILabel : UIView

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

String Bridging

Example 1: UILabel

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

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

```objc
// Objective-C - UIKit

@interface UILabel : UIView

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

bridge

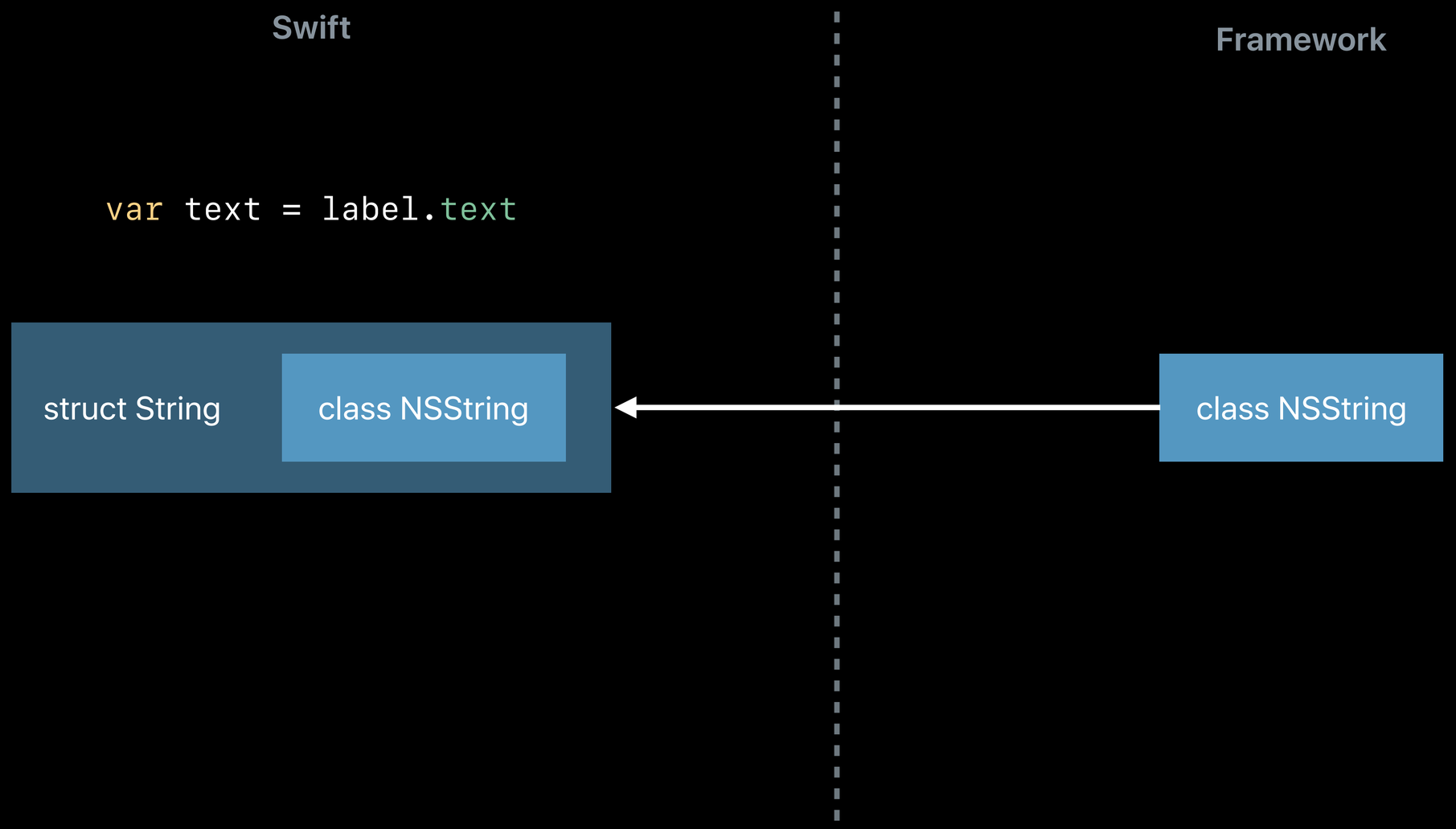
String Bridging

Example 1: UILabel

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

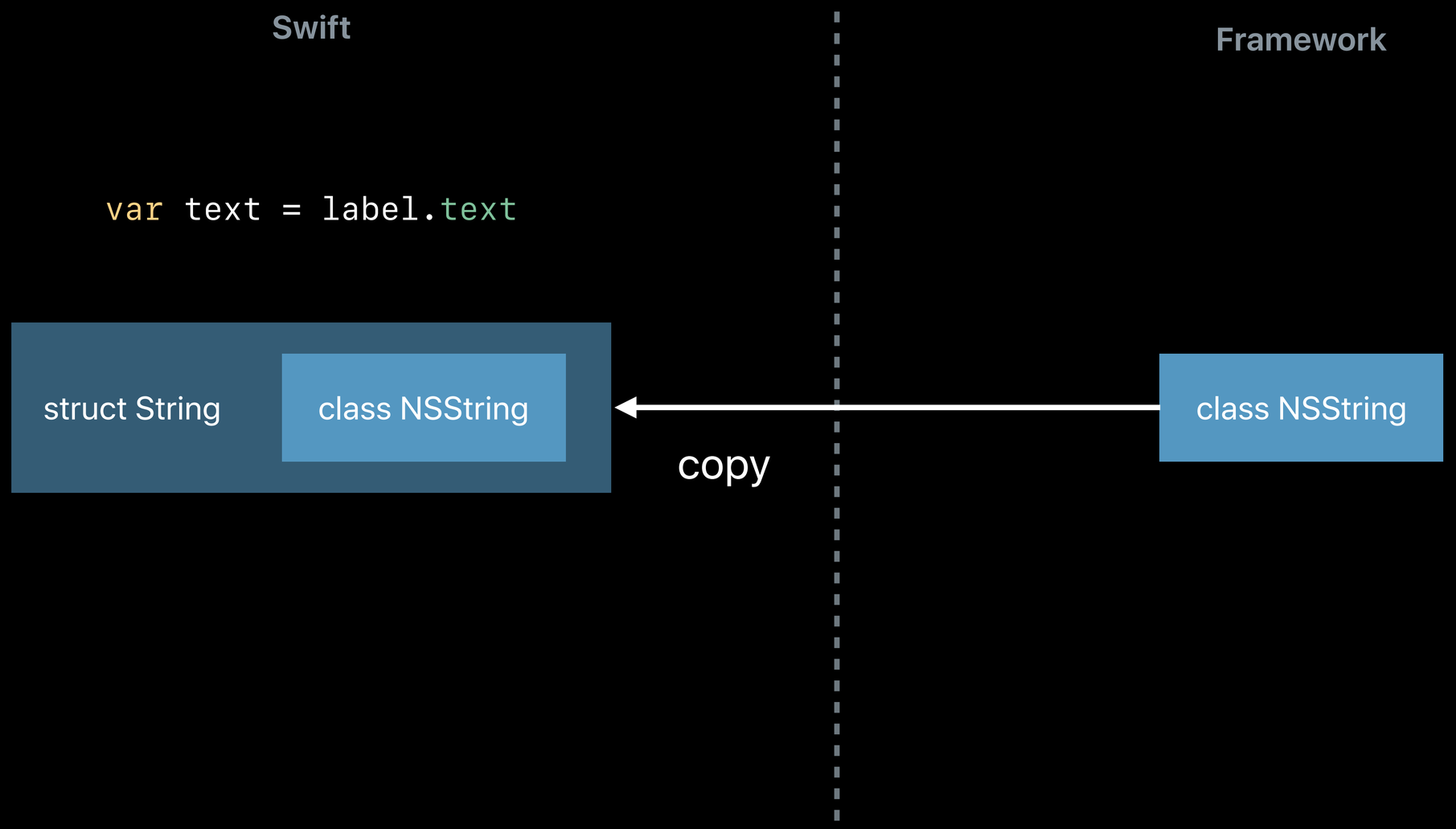
String Bridging

Example 1: UILabel

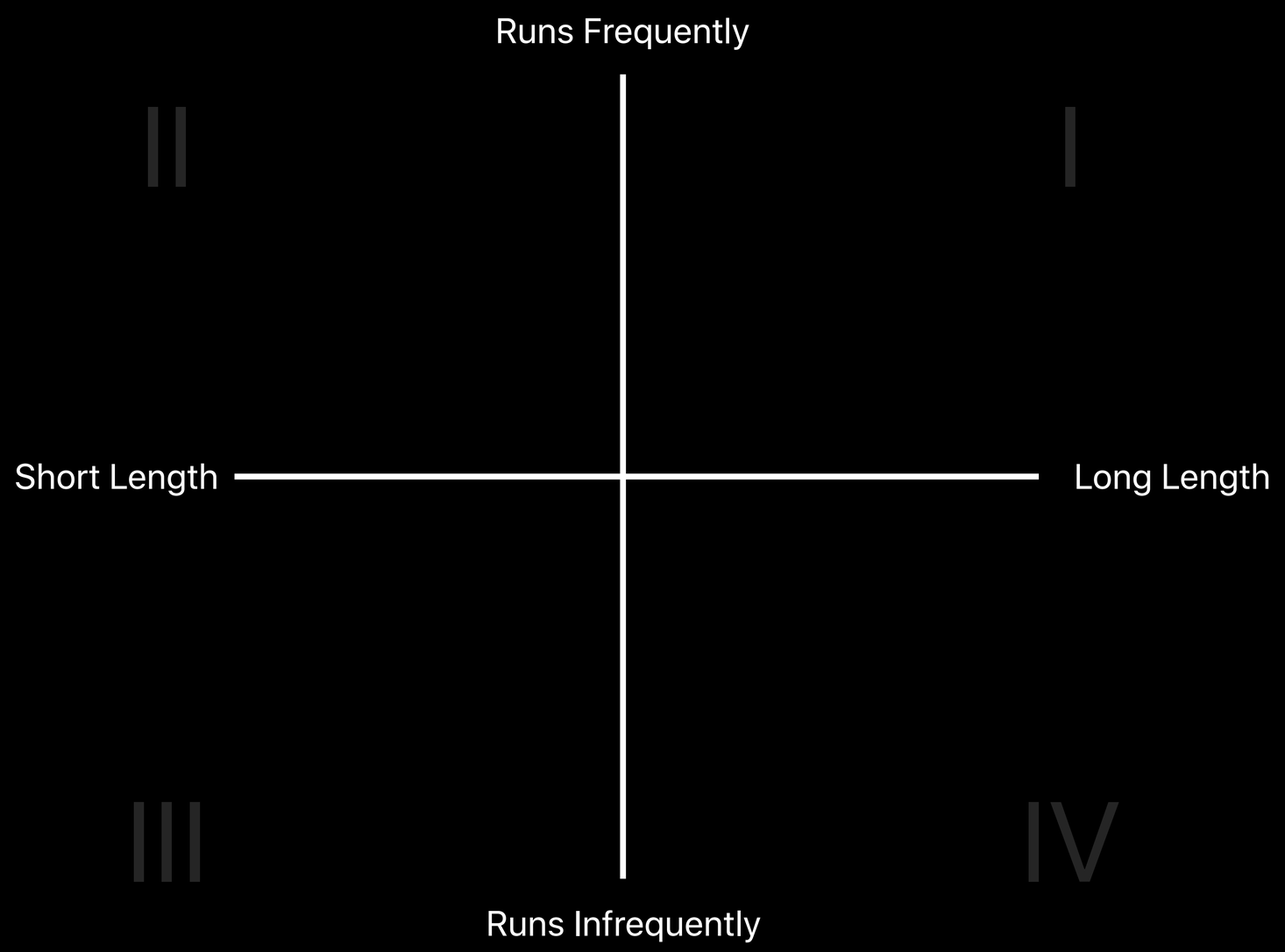


String Bridging

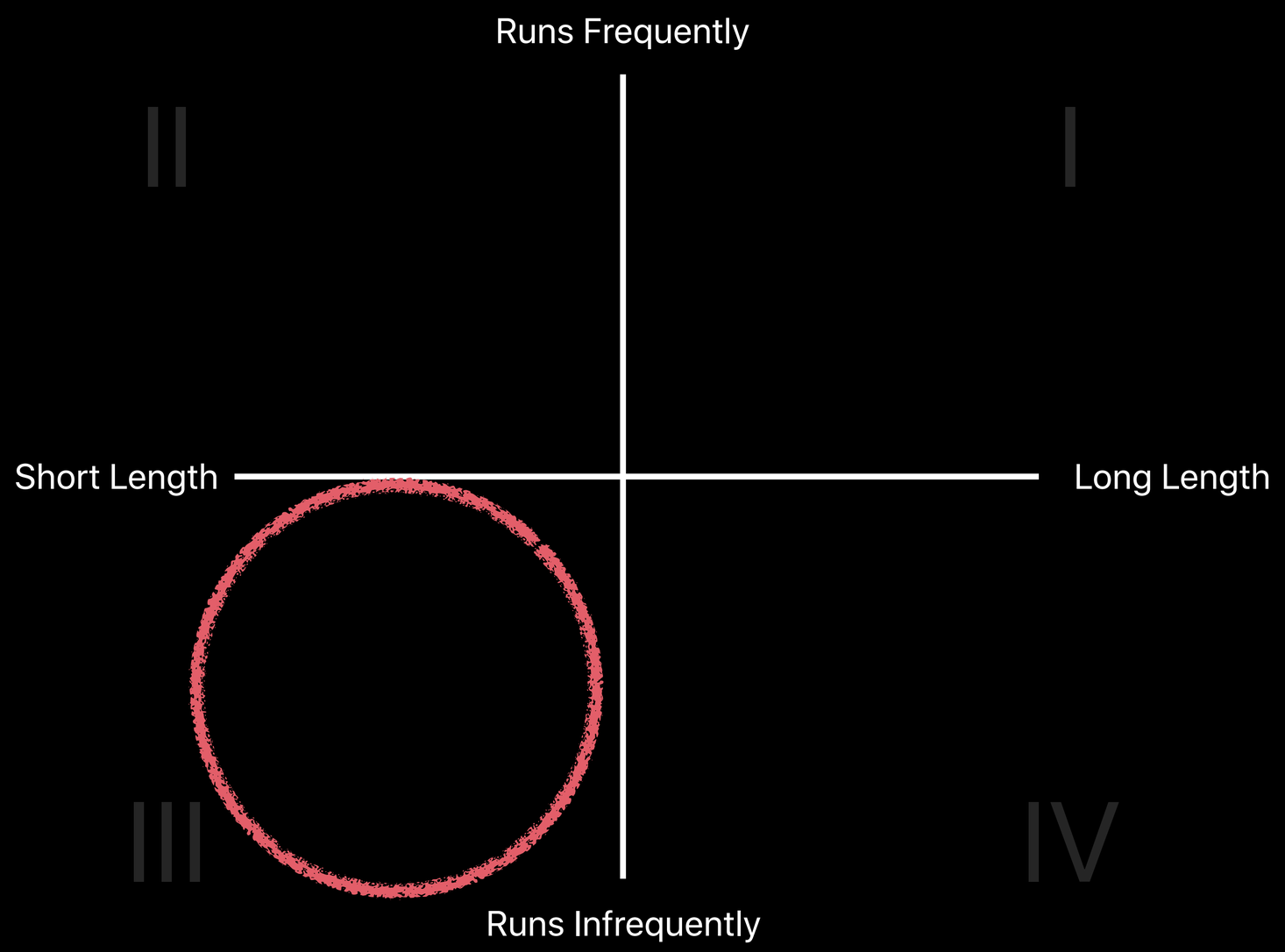
Example 1: UILabel



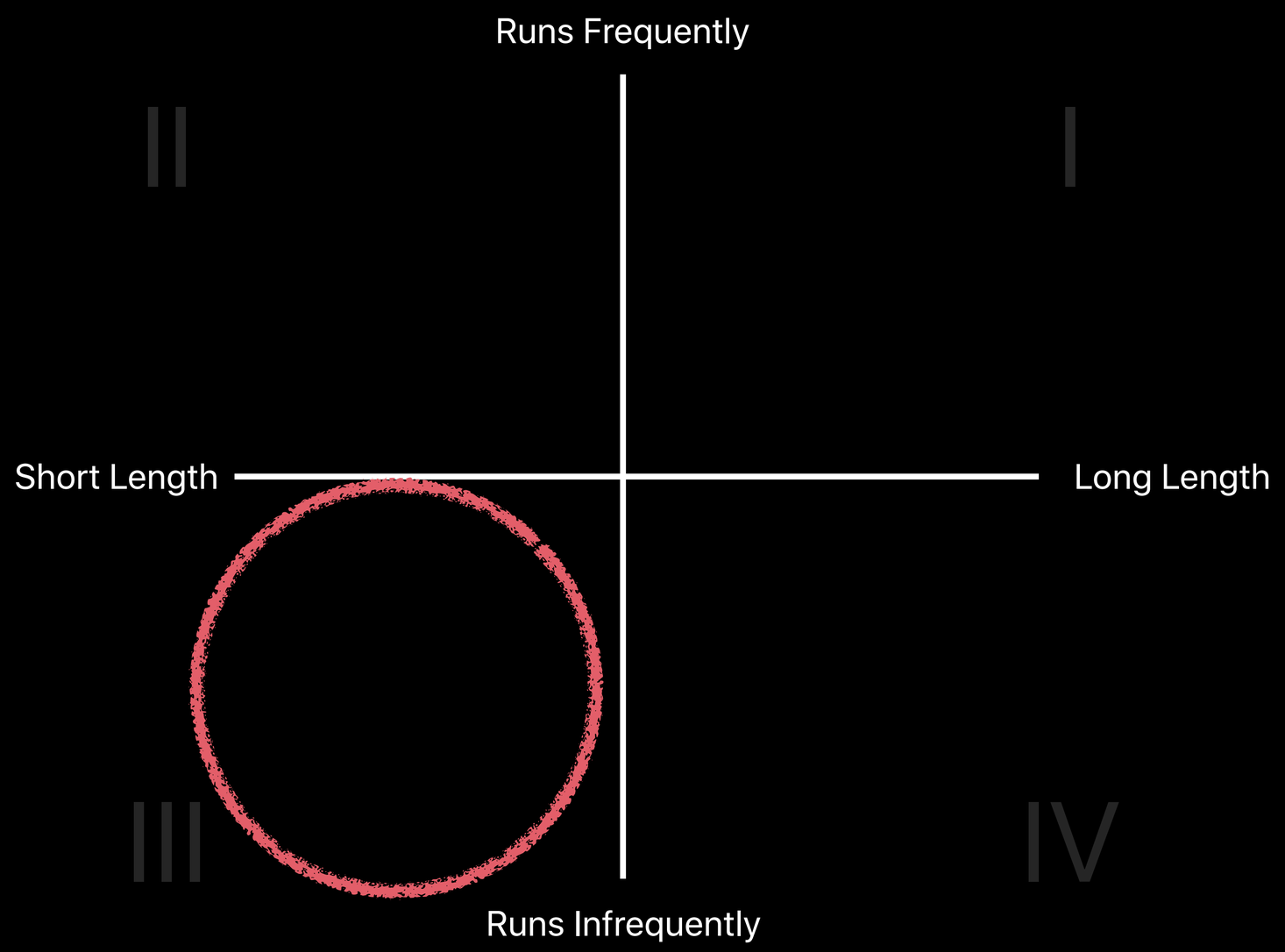
```
var text = label.text
```



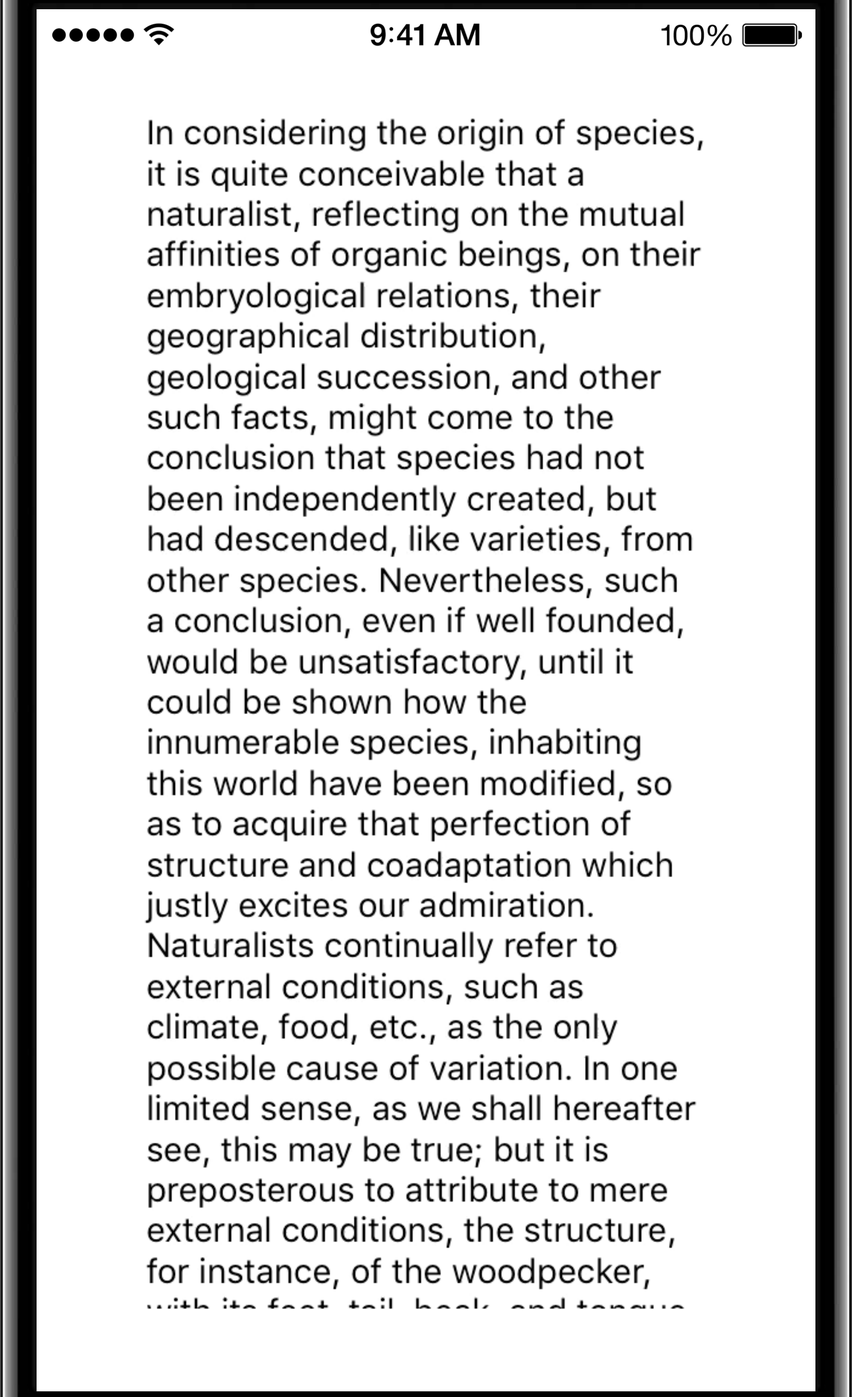
```
var text = label.text
```

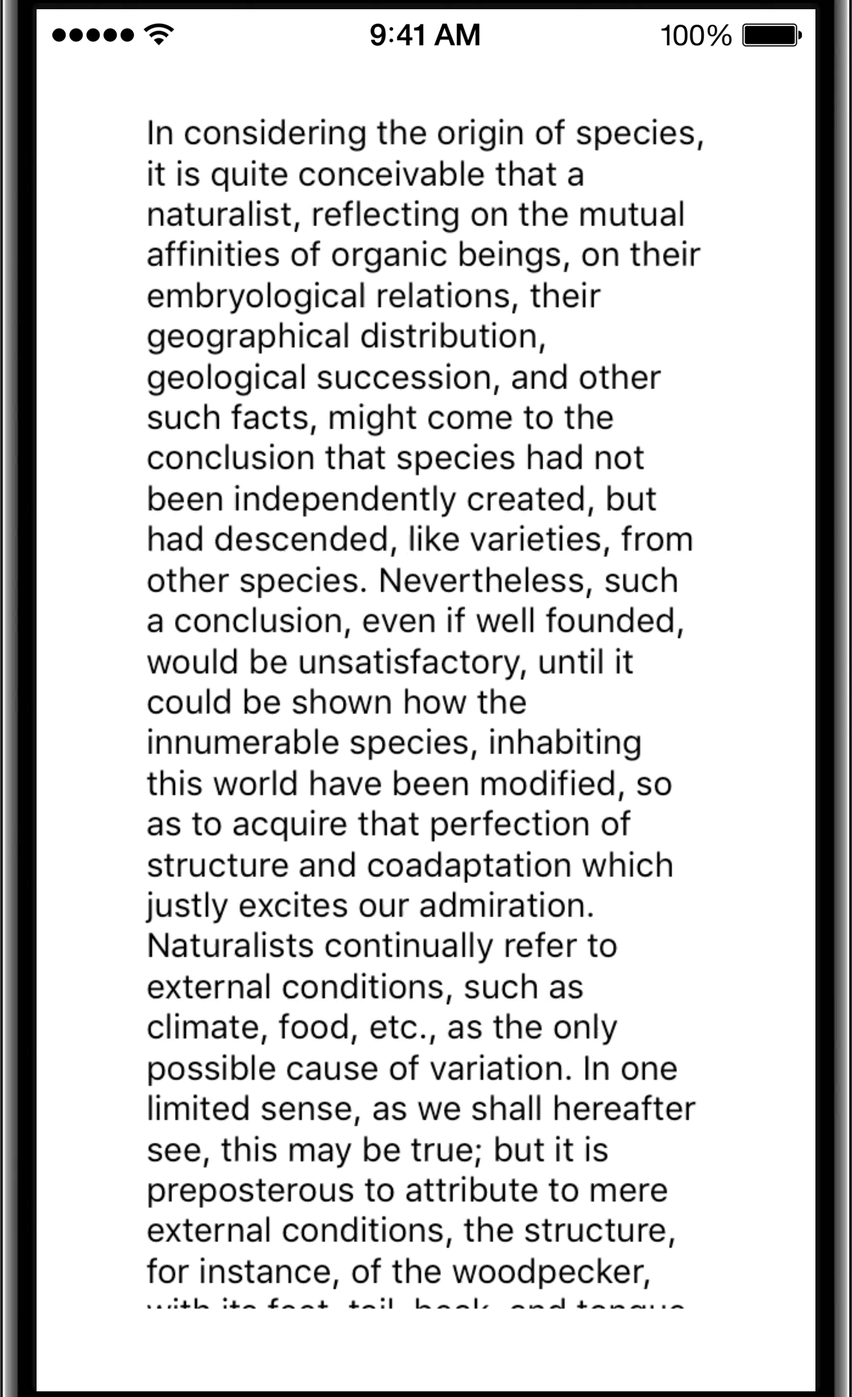


```
var text bel.text
```



Example 2: NSTextStorage


9:41 AM
100%
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,


9:41 AM
100%
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,

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
var text = textView.textStorage.string
```

```
// Swift Interface - UIKit
class NSTextStorage : NSMutableAttributedString
```

↓

```
open class NSMutableAttributedString :
                    NSAttributedString {
        open var string: NSString
```

String Bridging

Example 2: NSTextStorage

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

```objc
// Objective-C - UIKit

@interface NSTextStorage : NSMutableAttributedString
```

```objc
// Objective-C - Foundation

@interface NSMutableAttributedString :
                    NSAttributedString

@property(readonly, copy) NSString *string;
```

String Bridging

Example 2: NSTextStorage

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

```objc
// Objective-C - UIKit

@interface NSTextStorage : NSMutableAttributedString
```

```objc
// Objective-C - Foundation

@interface NSMutableAttributedString :
                    NSAttributedString

@property(readonly, copy) NSString *string;
```

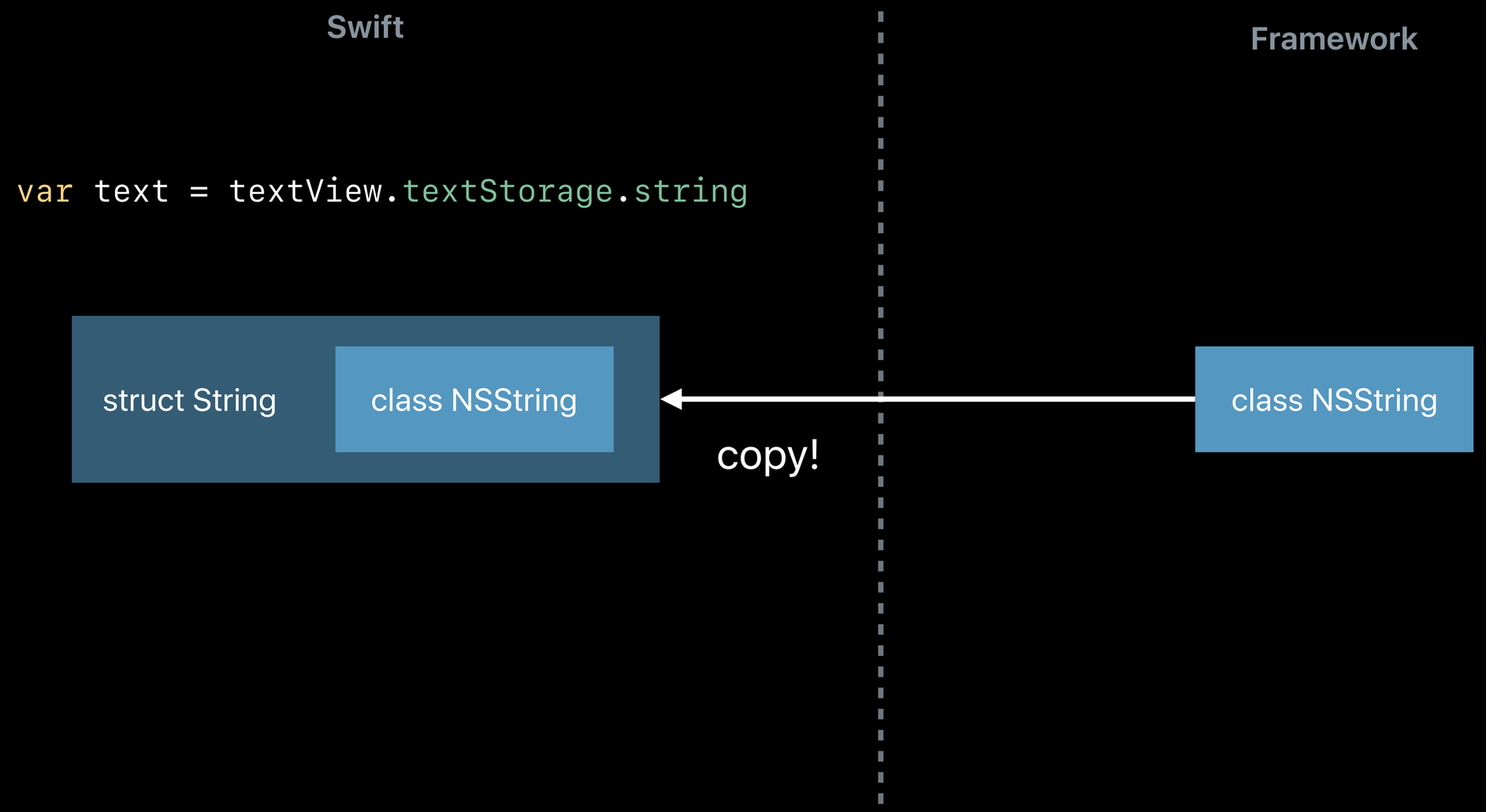
String Bridging

Example 2: NSTextStorage

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

String Bridging

Example 2: NSTextStorage



String Bridging

Example 2: NSTextStorage

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

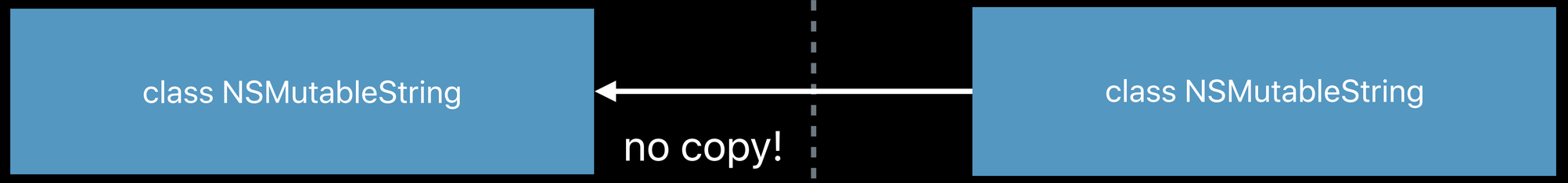
String Bridging

Example 2: NSTextStorage

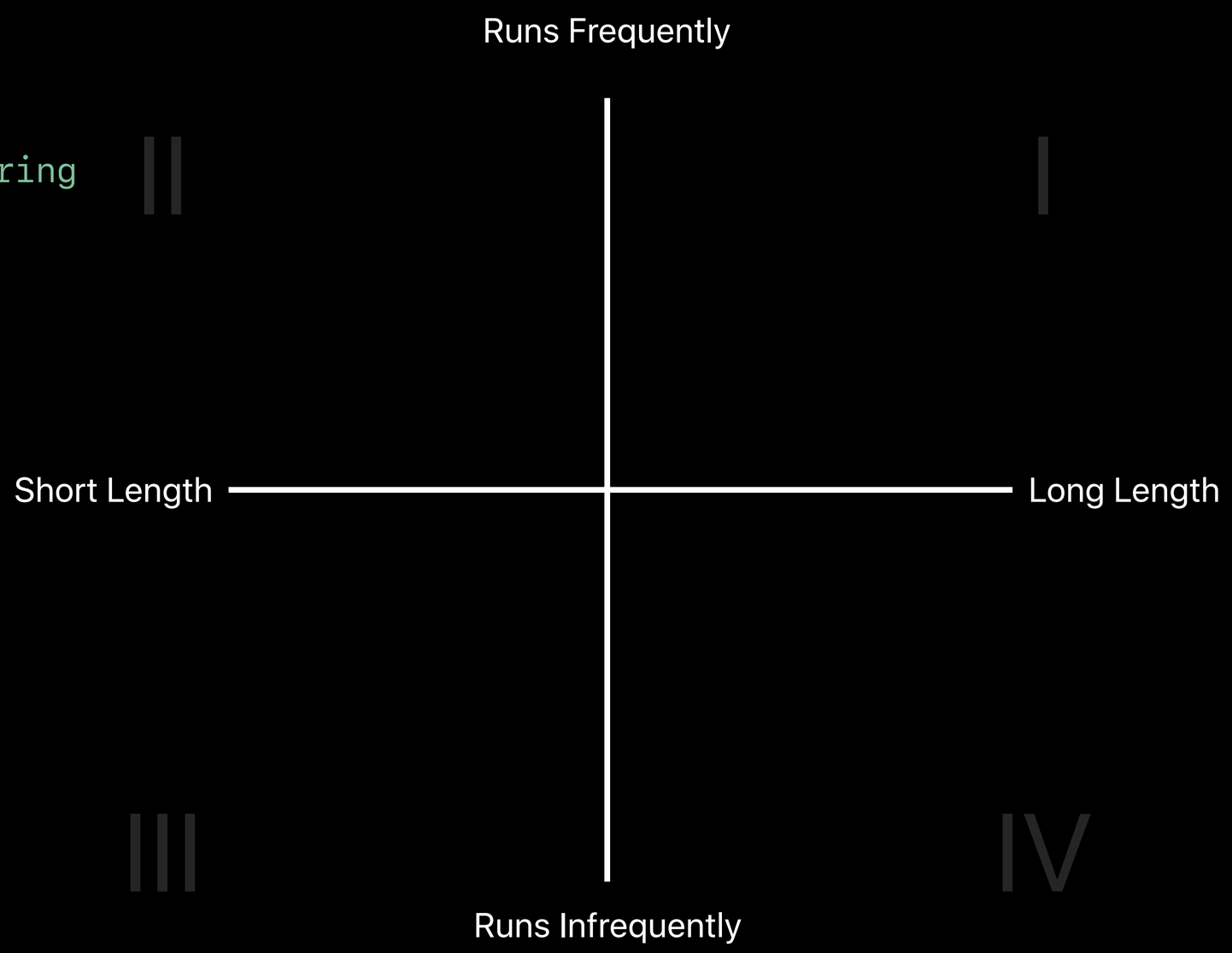
Swift

Framework

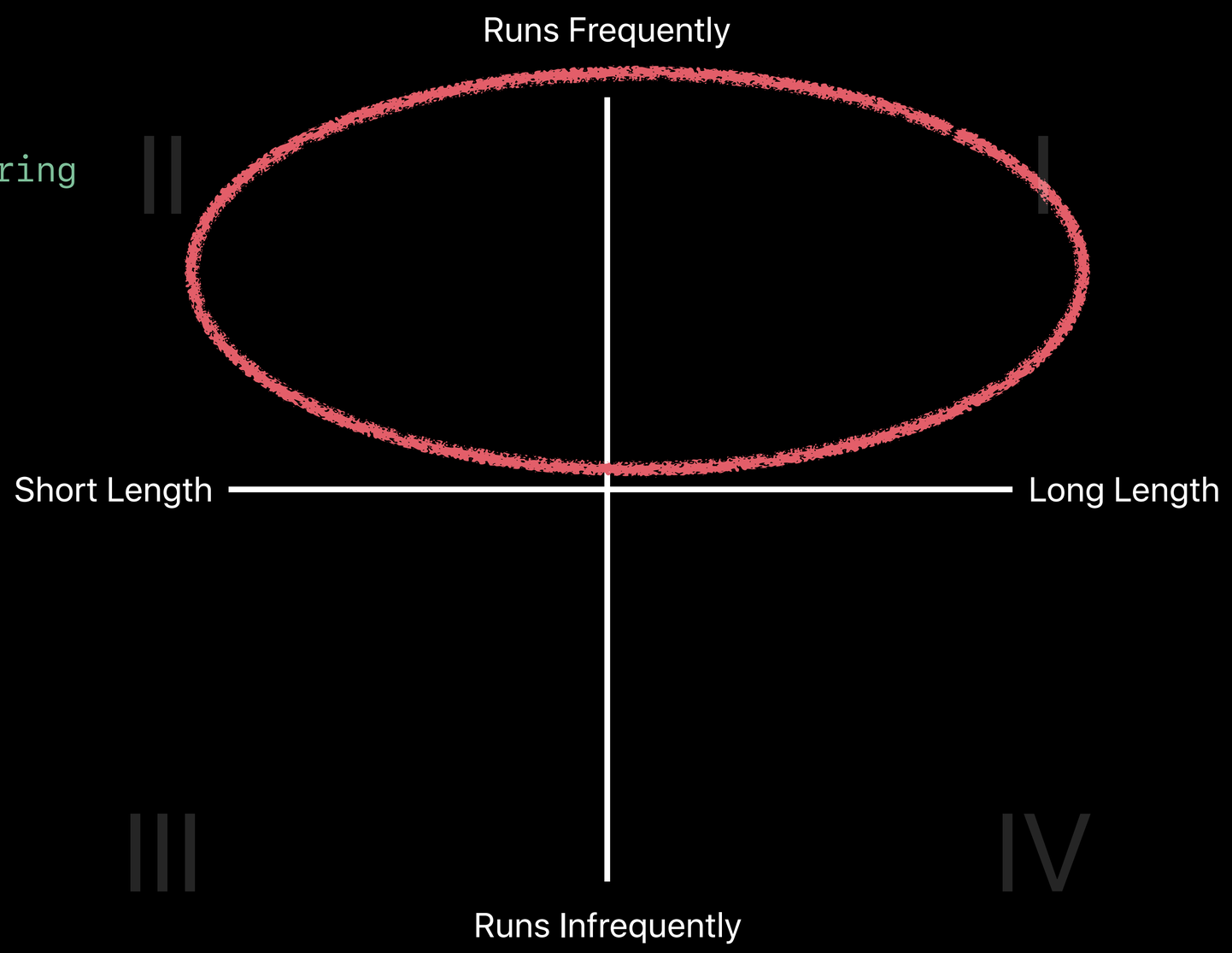
```
var text = textView.textStorage.mutableString
```



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

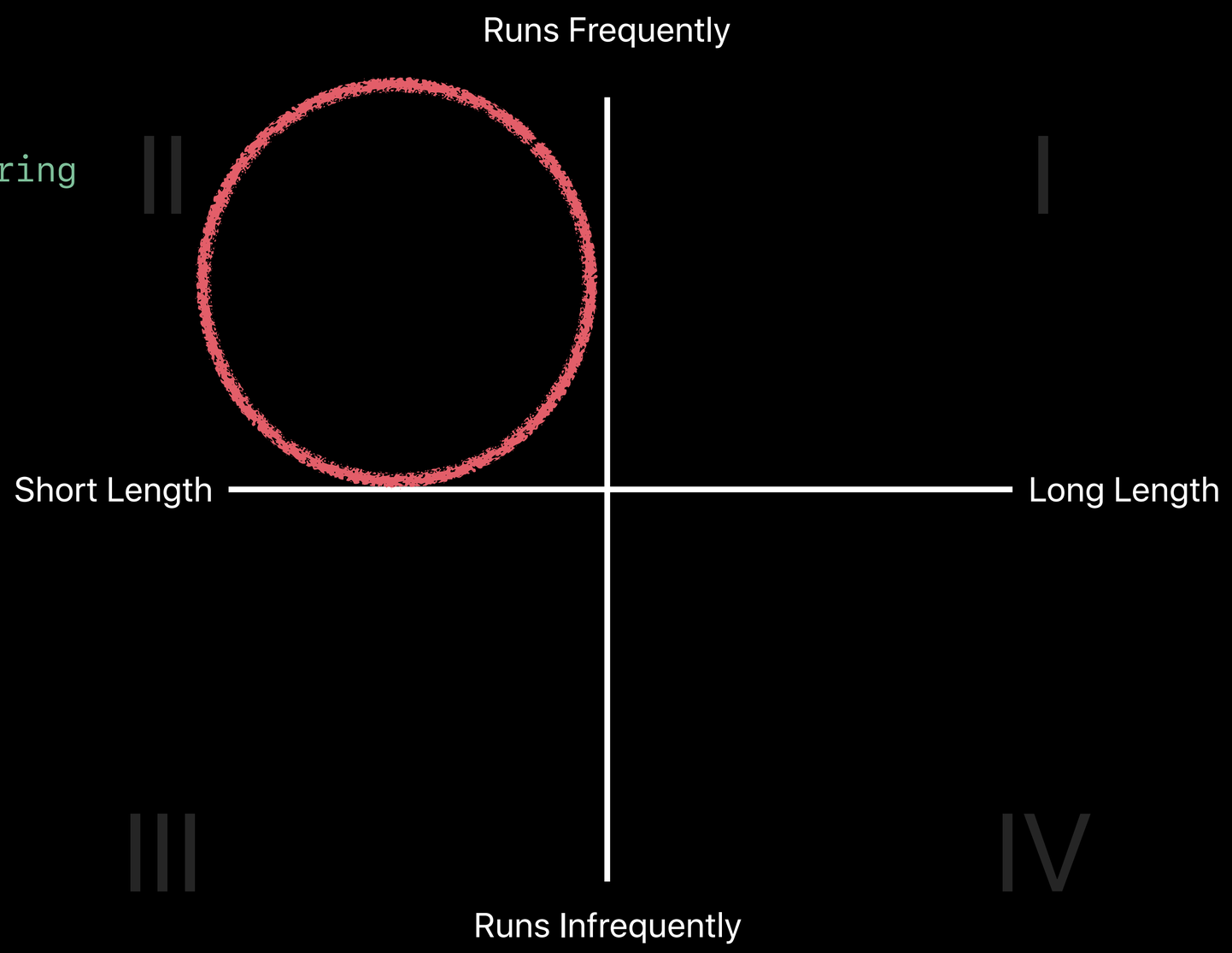


```
var text = textView.textStorage.string
```



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

1 KB?



```
var tex$textView.textStorage.string
```

1 KB?

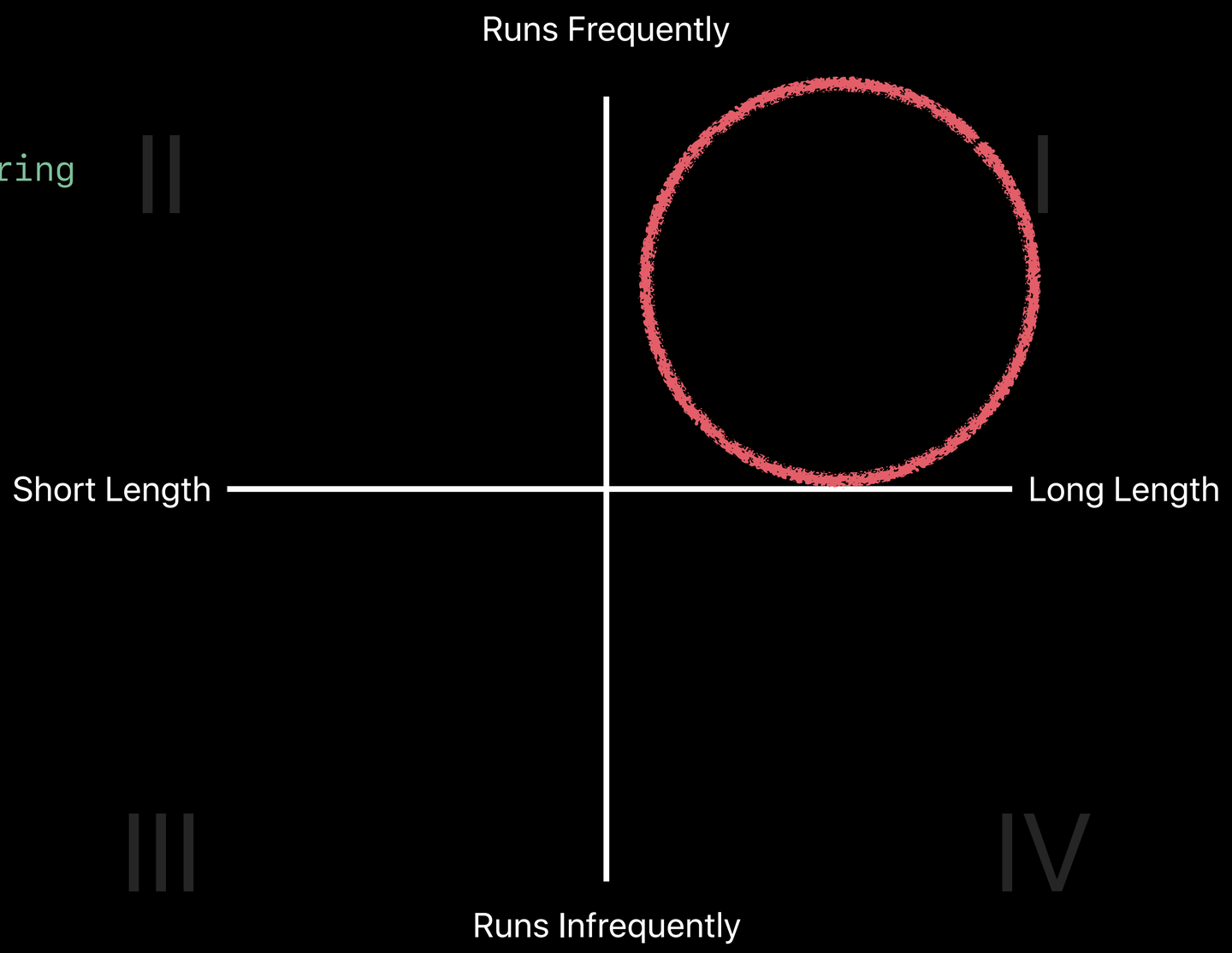
1 MB?

```
var te  extView.textStorage.string
```

1 KB?

1 MB?

1 GB?



String bridging

Ranges

Text layout and rendering

"Ranges 🤦🏻‍♀️"

"  "

Visible Components				♀	

Visible Components				♀	
Unicode Scalar Value	0x1F926	0x1F3FB	0x200D	0x2640	0xFE0F

Visible Components				♀	
Unicode Scalar Value	0x1F926	0x1F3FB	0x200D	0x2640	0xFE0F
Unicode Name	FACE PALM	EMOJI MODIFIER FITZPATRICK TYPE-1-2	ZERO WIDTH JOINER	FEMALE SIGN	VARIATION SELECTOR-16

Visible Components	🤦		🏻			♀	
Unicode Scalar Value	0x1F926		0x1F3FB		0x200D	0x2640	0xFE0F
Unicode Name	FACE PALM		EMOJI MODIFIER FITZPATRICK TYPE-1-2		ZERO WIDTH JOINER	FEMALE SIGN	VARIATION SELECTOR-16
UTF-16	0xD83E	0xDD26	0xD83C	0xDFFB	0x200D	0x2640	0xFE0F

Example 1:
Working with NSAttributedString

“What a 🎉💩!”

“What a 🎉💩!”

String Ranges

Example 1: Working with NSAttributedString

“What a 🎉💩!”

```
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 🎉💩!"

```
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 🎉💩!"

```
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

NEW

"What a 🎉💩!"

```
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

NEW

"What a 🎉💩!"

```
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<span>test</span></b>
      </div>
   </body>
</html>
```

```swift
// String Ranges: Working with NSRegularExpression

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
    }
}
```

```swift
// Improved String Ranges: Working with NSRegularExpression

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
}
```

NEW

```swift
// Improved String Ranges: Working with NSRegularExpression

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
}
```

NEW

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

Line Breaking Cursor Positioning Bidirectional Dynamic Type Shaping Metrics
Attributes Screen Size Precomposed Characters Ligature RTL Stroke Fringing
Hyphenation Writing Direction Widows Glyph Dilation Grapheme Cluster Gamma
Decomposed Characters Even-Odd Script Text Matrix Uncached
Tightening Orphans Tracking Truncation Spacing Orientation Glyph Bounds
Font Leading Pattern Fill LTR Font Smoothing Locale Flippedness Fonts Margin
Kerning Shadow Em Non-Zero Language Unicode Glyph Bounds Emoji
Anti-Aliasing Glyph Substitution Line Height Ascenders Clipping Legibility
Linear Blending Selection Letterpress Optical Alignment Exclusion Paths Encoding
Letterpress Bounding Boxes Accessibility Descenders Attachments Baselines

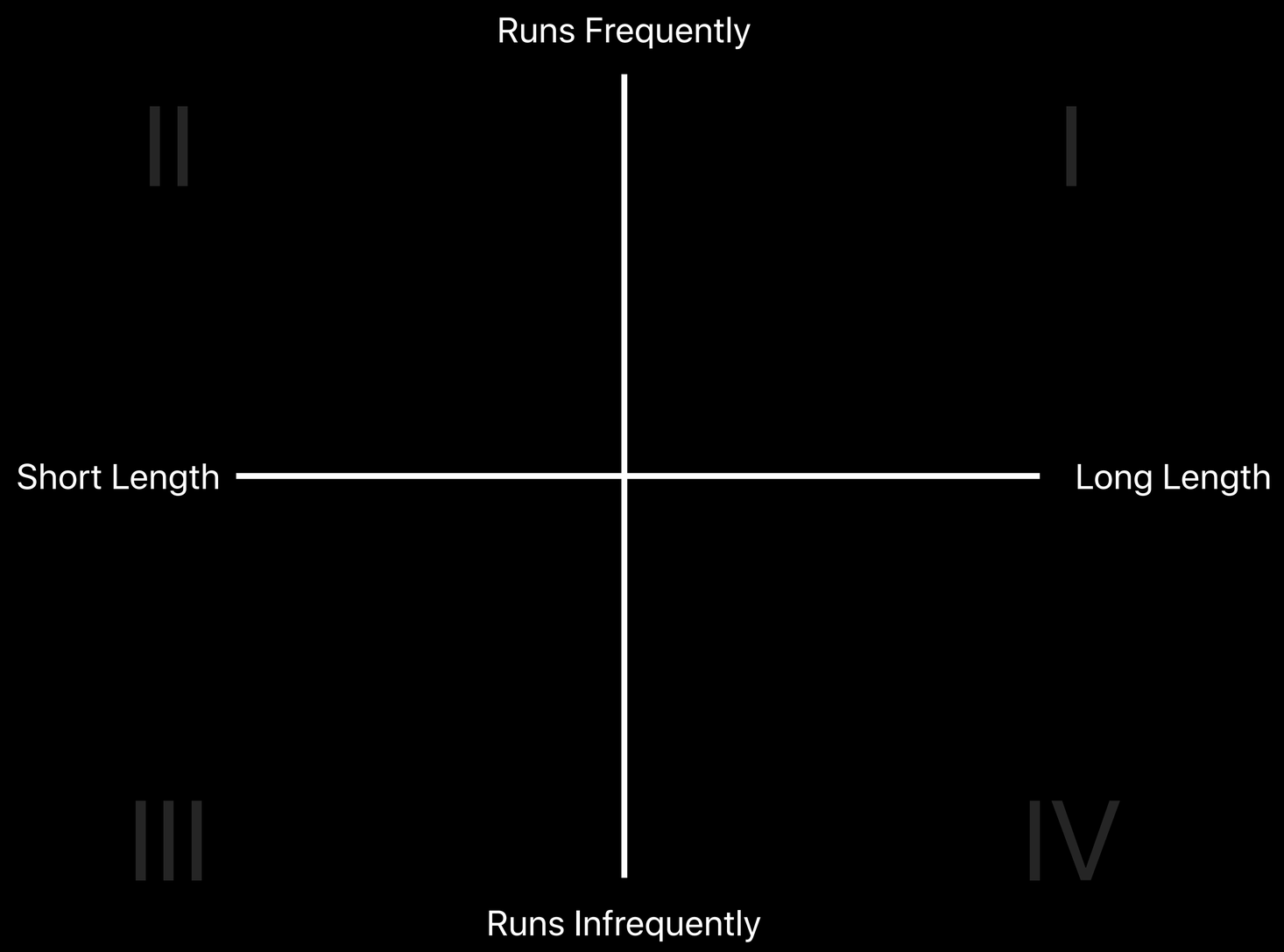
Example:
A Tale of Two Labels

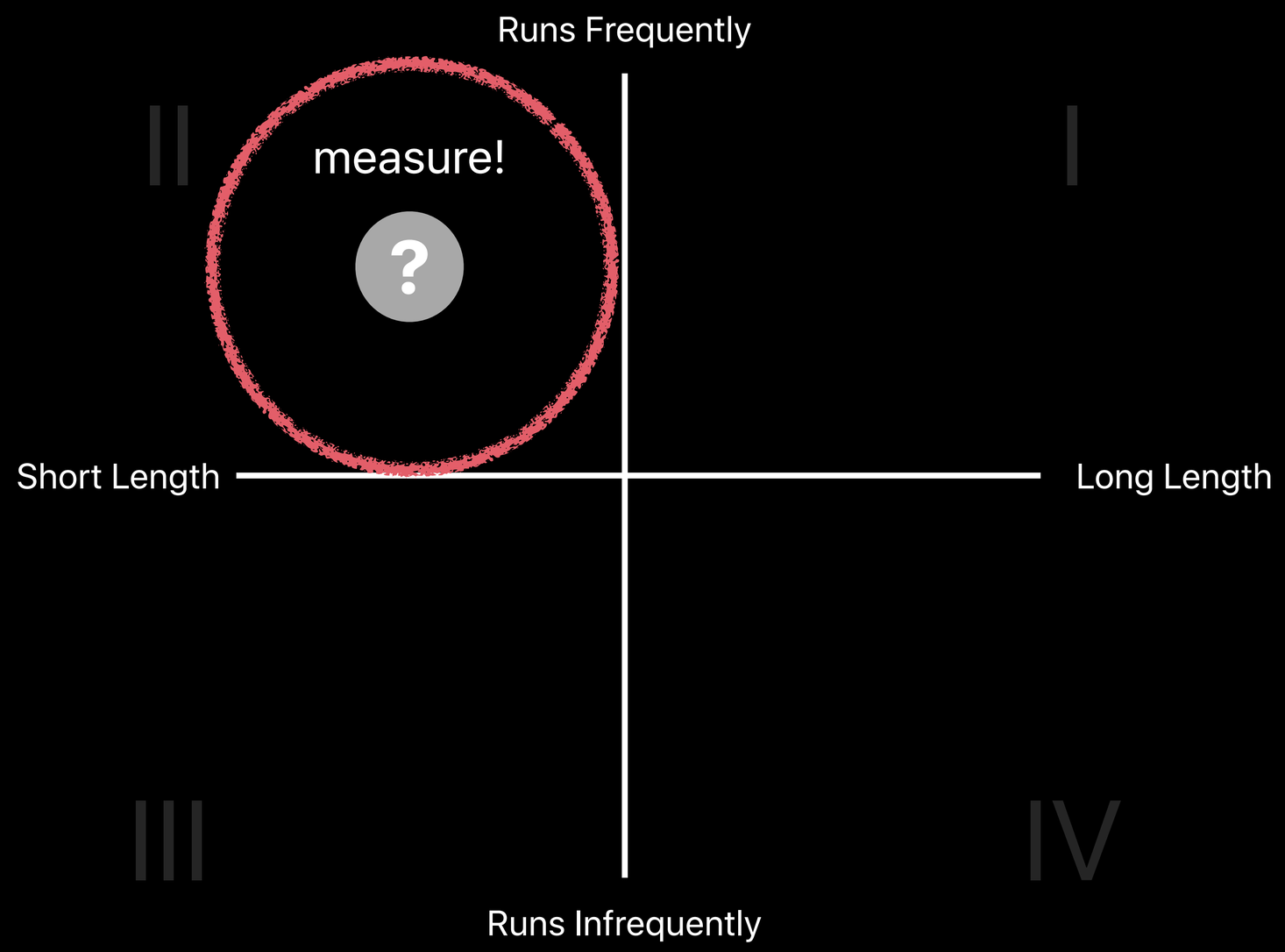

Millennium Park
Loop


Millennium Park
Loop
Tribune Tower
Near North Side
Union Park
West Loop
Rookery Building


Millennium Park
Loop
Tribune Tower
Near North Side
Union Park
West Loop
Rookery Building


长城
北京
紫禁城
北京
天安门广场


Runs Frequently
II
I
Short Length
Long Length
III
IV
Runs Infrequently


Runs Frequently
II
I
measure!
?
Short Length
Long Length
III
IV
Runs Infrequently

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

```
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

```
// 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
```

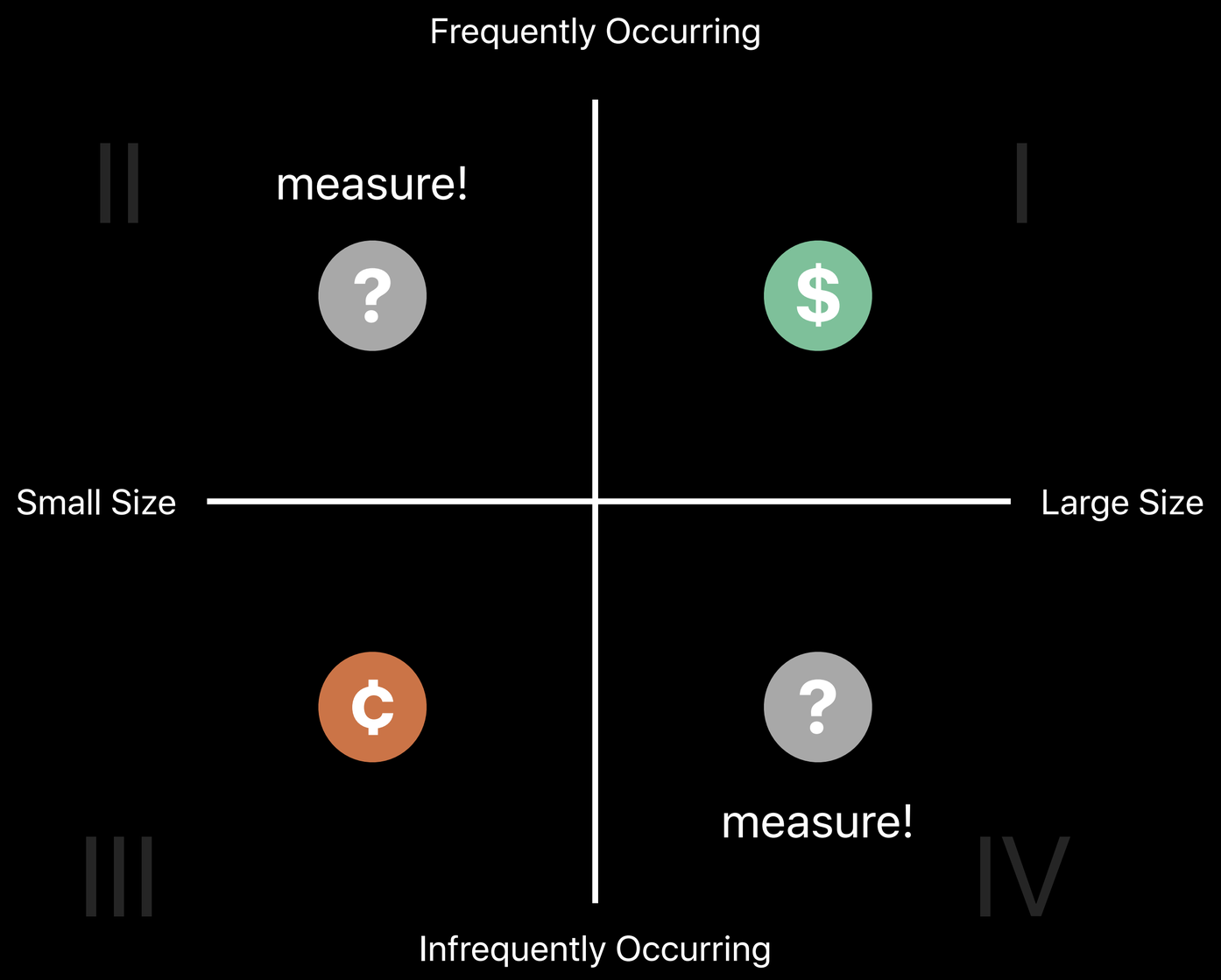
What You Can Do

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
```

Summary


Frequently Occurring
II
measure!
?
I
$
Small Size
Large Size
¢
?
measure!
III
IV
Infrequently Occurring

More Information

https://developer.apple.com/wwdc17/244

Related Sessions

Understanding Swift Performance	WWDC 2016
What's New in Cocoa	WWDC 2017
What's New in Foundation	WWDC 2017
Modernizing Grand Central Dispatch Usage	WWDC 2017
Cocoa Development Tips	WWDC 2017
Writing Energy Efficient Apps	WWDC 2017

Labs

Cocoa Lab	Technology Lab B	Fri 1:50PM–3:20PM

WWDC17