Speeding up conformances

SwiftLeeds 2024

Noah Martin | Co-Founder, Emerge Tools

emerge tools

Swift should be fast

Swift should be fast

- Fast is the first adjective used describing Swift on swift.org

> Swift is a **general-purpose** programming language that's **approachable** for newcomers and **powerful** for experts.
>
> It is **fast**, **modern**, **safe**, and a **joy** to write.

Swift should be fast

- Fast is the first adjective used describing Swift on swift.org
- Countless WWDC talks highlight how fast it is

Swift is a **general-purpose** programming language that's **approachable** for newcomers and **powerful** for experts.

It is **fast**, **modern**, **safe**, and a **joy** to write.

How fast is it really?

How fast is it really?

Pros

How fast is it really?

Pros

- Predictable, granular control, low overhead

How fast is it really?

Pros

- Predictable, granular control, low overhead

Cons

How fast is it really?

Pros

- Predictable, granular control, low overhead

Cons

- Abstractions hide details, easy to bloat

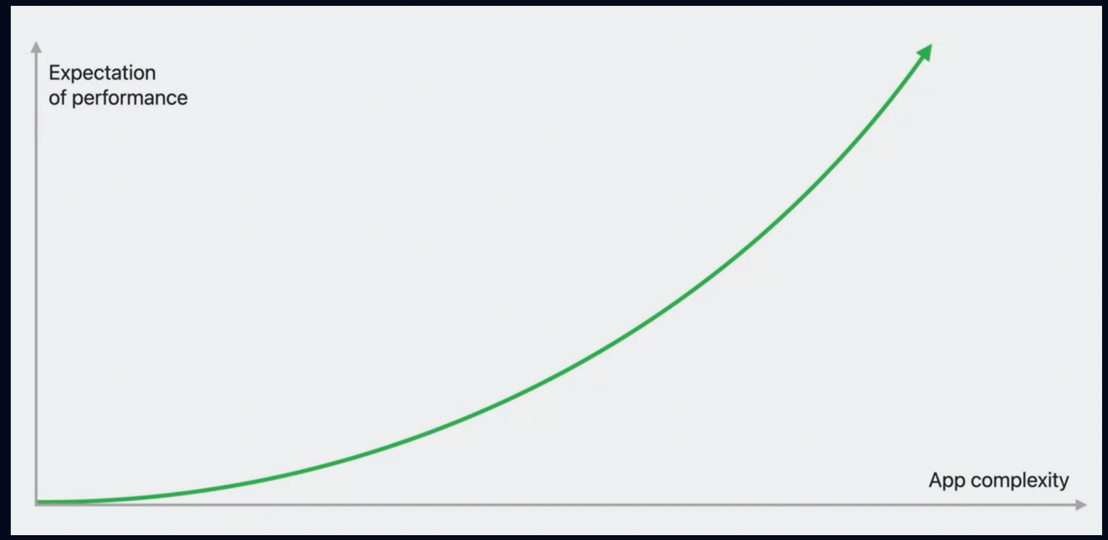
How fast is it really?

Pros

- Predictable, granular control, low overhead

Cons

- Abstractions hide details, easy to bloat



Demystify SwiftUI performance WWDC 23

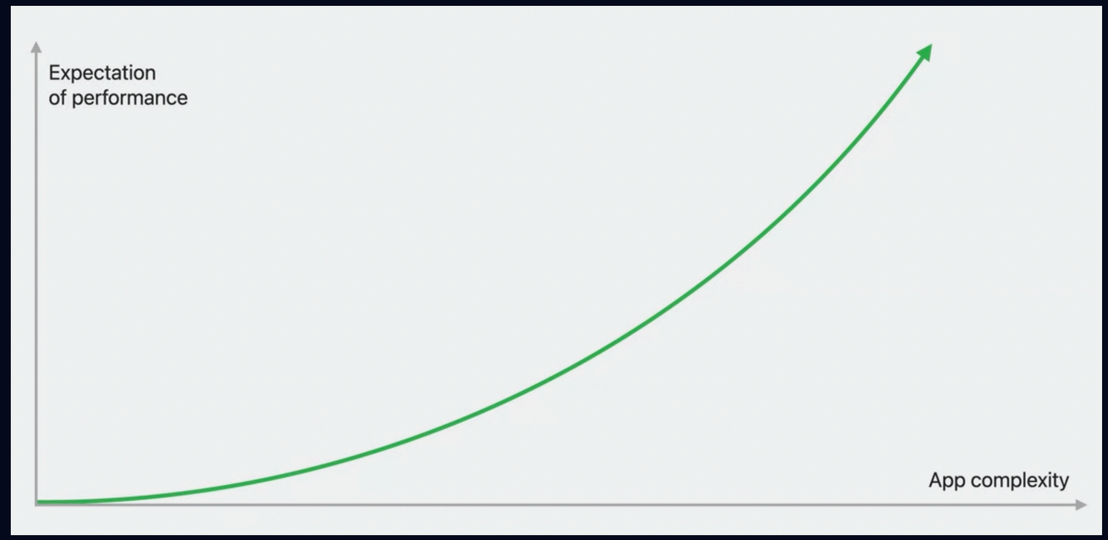
How fast is it really?

Pros

- Predictable, granular control, low overhead

Cons

- Abstractions hide details, easy to bloat



Demystify SwiftUI performance WWDC 23

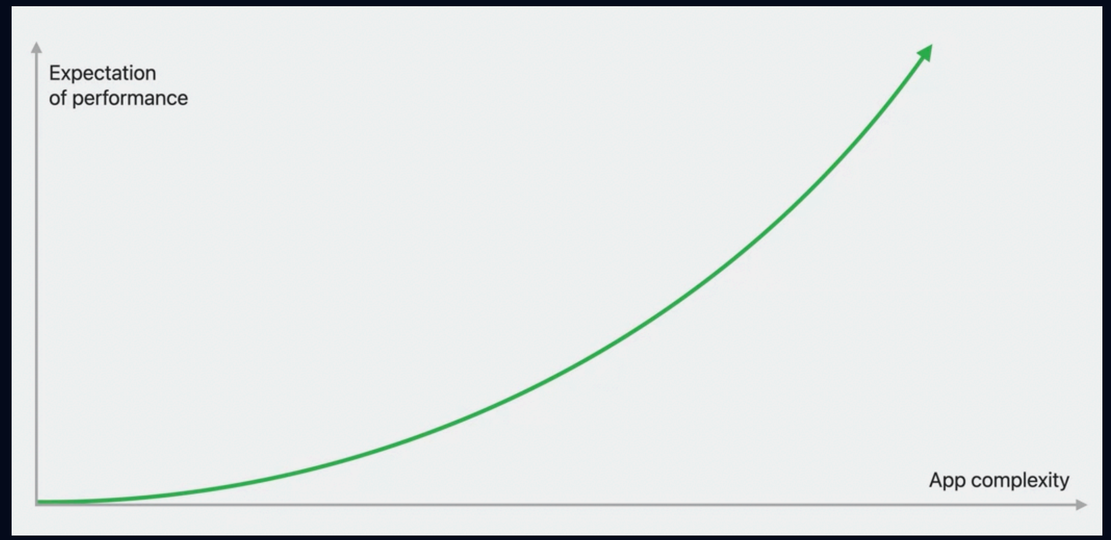
“Small issues can get amplified, and code that works well for a prototype might not work as well in production.”

Key Takeaway

Write Less Code

Key Takeaway

Write Less Code



Detectives: Why is it slow? 🔎

Detectives: Why is it slow? 🔎

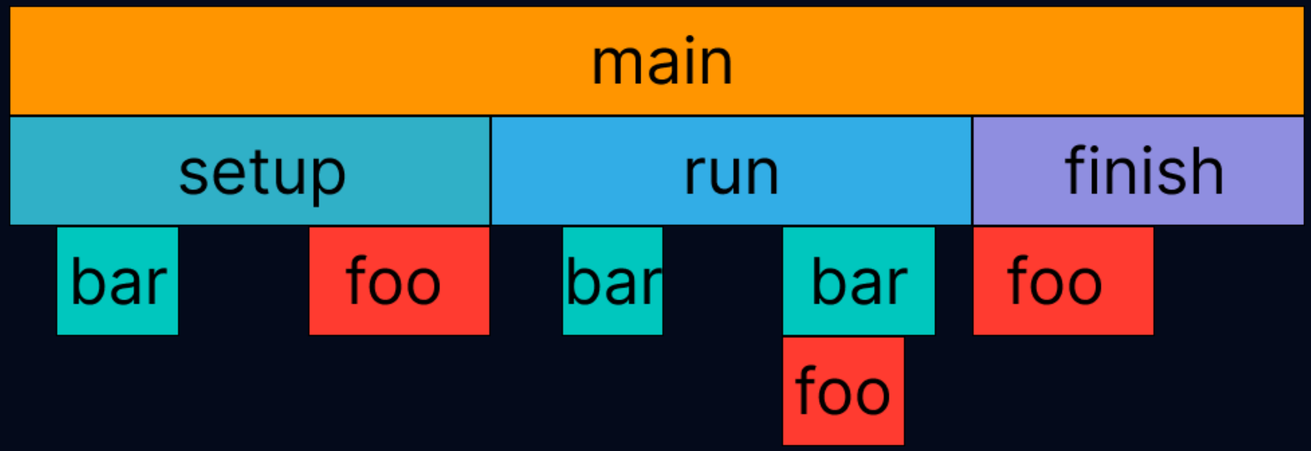
- Profile using ETTrace

Detectives: Why is it slow? 🔎

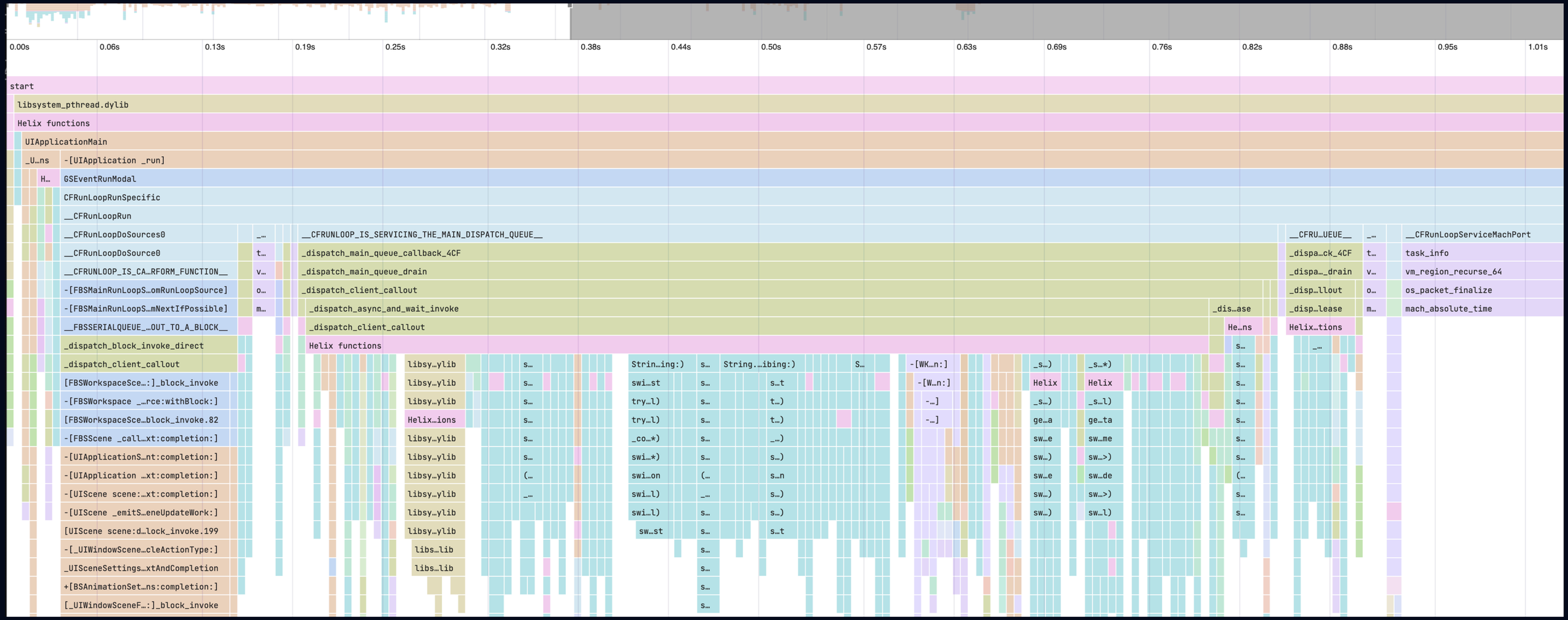
- Profile using ETTrace
- Visualize time spent in function calls using a flamechart

Detectives: Why is it slow? 🔎

- Profile using ETTrace
- Visualize time spent in function calls using a flamechart



Detectives: Why is it slow? 🔍



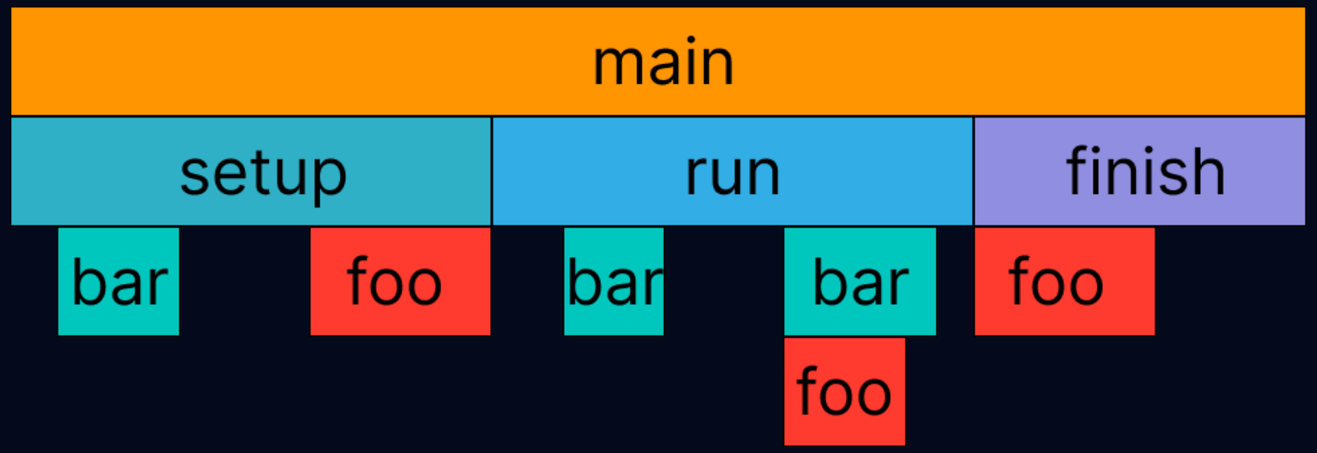
Detectives: Why is it slow? 🔎

Detectives: Why is it slow? 🔎

Inverting flamecharts

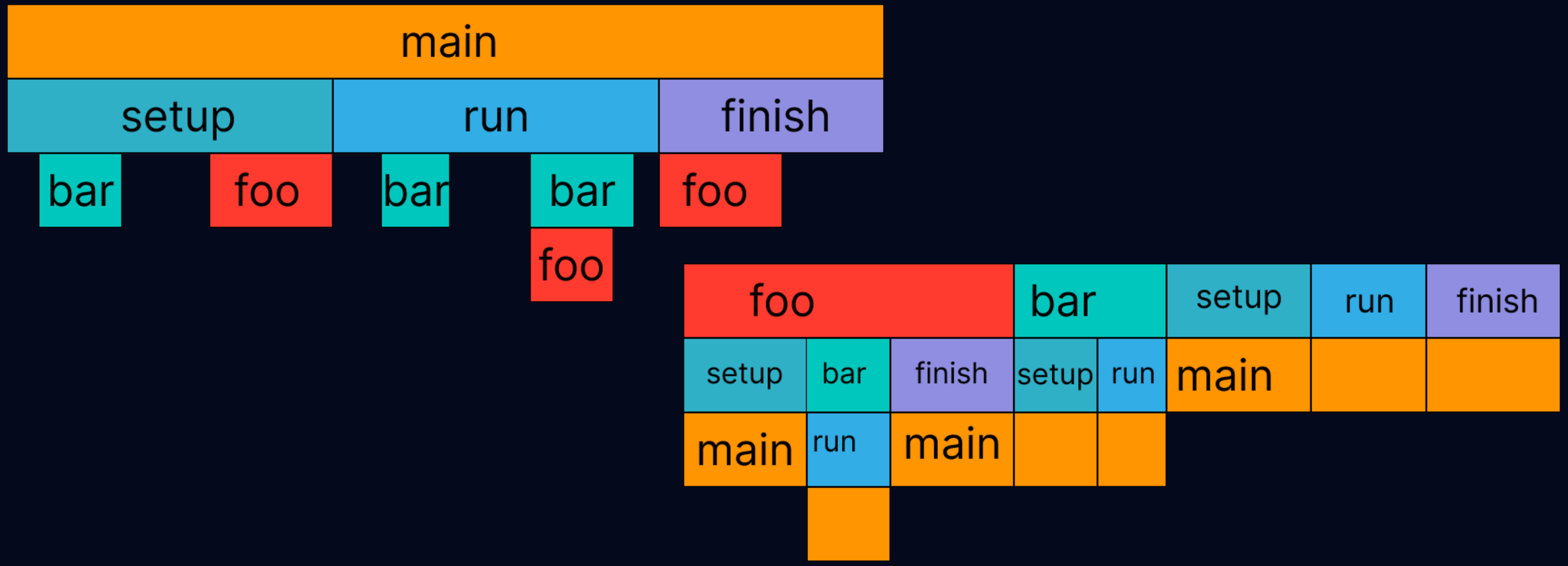
Detectives: Why is it slow? 🔎

Inverting flamecharts



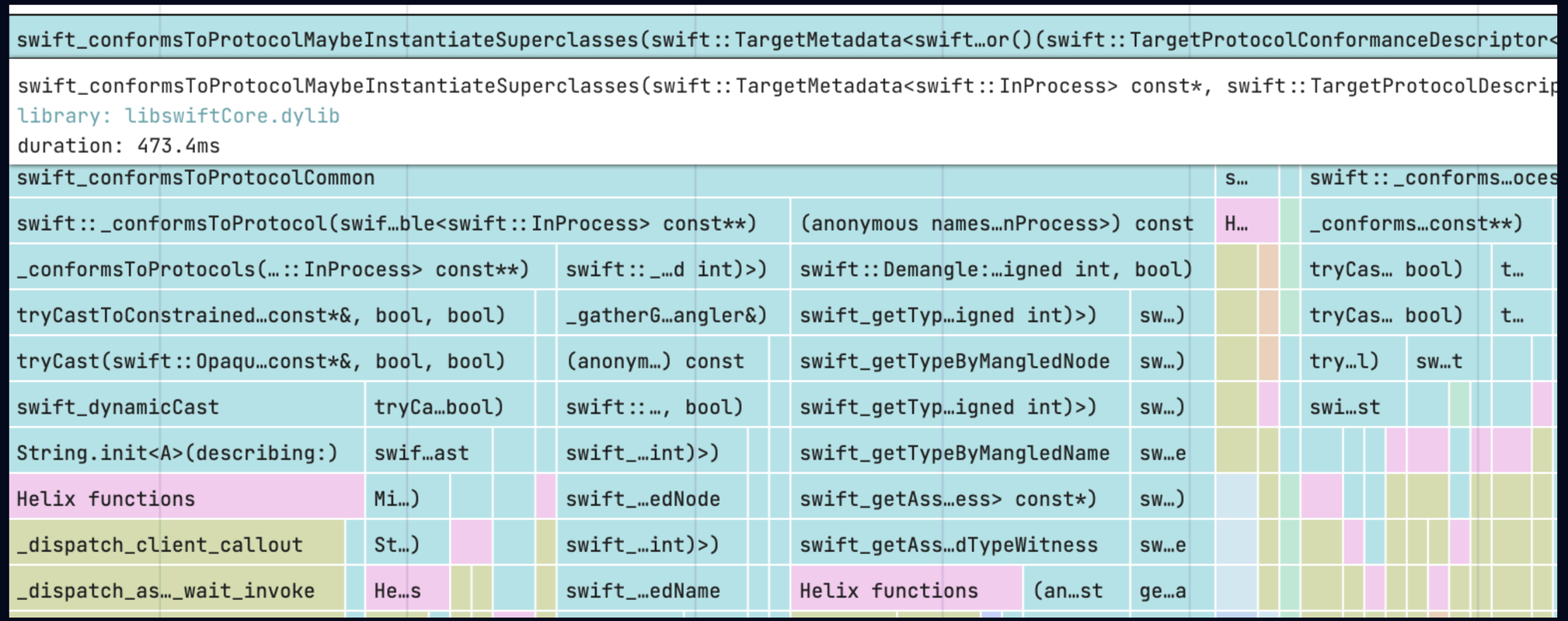
Detectives: Why is it slow? 🔎

Inverting flamecharts



Detectives: Why is it slow? 🔍

Detectives: Why is it slow? 🔎



Detectives: Why is it slow? 🔎

Clue #1

```
swift_conformsToProtocol
```

Detectives: Why is it slow? 🔎

Swift 5.4 Release Notes

Protocol conformance checks at runtime are significantly faster, thanks to a faster hash table implementation for **caching previous lookup results**. In particular, this speeds up common runtime as? and as! casting operations.

What is a conformance check?

What is a conformance check?

Conformances are defined

```
class MyClass: MyProtocol { }
```

What is a conformance check?

Conformances are defined

```
class MyClass: MyProtocol { }
```

And they are checked

```
myVar as? MyProtocol
```

What is a conformance check?

Conformances are defined

```
class MyClass: MyProtocol { }
```

And they are checked

```
myVar as? MyProtocol
```

This is a runtime operation

```
cast(myVar, MyProtocol.self)
```

Summary

- **Found what is slow**
  - Protocol conformance checks
- Why is it so slow?
- How can we make it faster?

Detectives: Why is it slow? 🔎

Detectives: Why is it slow? 🔎

- PR: Reimplement protocol conformance cache with a hash table #33487

Clue #2

Detectives: Why is it slow? 🔎

- PR: Reimplement protocol conformance cache with a hash table #33487
- Uses more efficient cache data structure

Clue #2

Detectives: Why is it slow? 🔎

- PR: Reimplement protocol conformance cache with a hash table #33487
- Uses more efficient cache data structure
- 13x speedup

Clue #2

Detectives: Why is it slow? 🔎

- PR: Reimplement protocol conformance cache with a hash table #33487
- Uses more efficient cache data structure
- 13x speedup
- But this is still a “cache” - a cache miss can be slow

Clue #2

Testing the cache

Testing the cache

- App with 50k protocol conformances

Testing the cache

- App with 50k protocol conformances
- On app launch check if 50 classes conform to a protocol, repeat 3 times

Testing the cache

- App with 50k protocol conformances
- On app launch check if 50 classes conform to a protocol, repeat 3 times

```
let p = cls as? MyProtocol
```

Testing the cache

- App with 50k protocol conformances
- On app launch check if 50 classes conform to a protocol, repeat 3 times

```
let p = cls as? MyProtocol
```

- First time: 71ms

Testing the cache

- App with 50k protocol conformances
- On app launch check if 50 classes conform to a protocol, repeat 3 times

```
let p = cls as? MyProtocol
```

- First time: 71ms
- Second time and third time: < 0.1ms

Cache works but the uncached state is slow!

Cache works but the uncached state is slow!

Fewer conformances

Fewer conformances

- App with 10000 protocol conformances

Fewer conformances

- App with 10000 protocol conformances
- Again 50x conformance check

Fewer conformances

- App with 10000 protocol conformances
- Again 50x conformance check

```
let p = cls as? MyProtocol
```

Fewer conformances

- App with 10000 protocol conformances
- Again 50x conformance check

```
let p = cls as? MyProtocol
```

- Now takes 15ms, less than a quarter of the time!

Dependencies

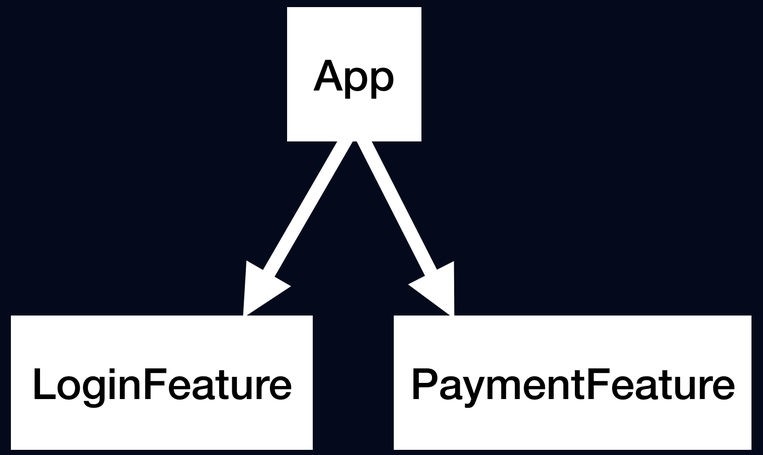
- Code that doesn’t run affects your code

Dependencies

- Code that doesn’t run affects your code
- Violates typical assumptions about module dependencies

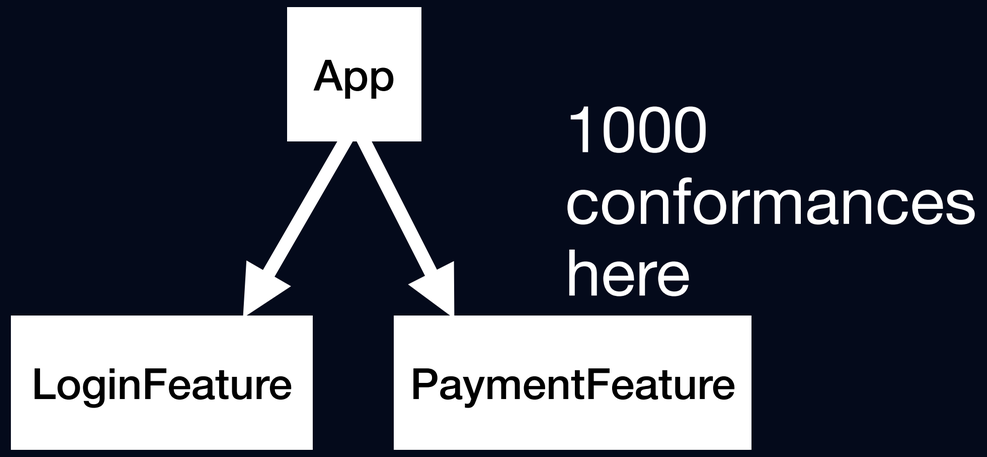
Dependencies

- Code that doesn’t run affects your code
- Violates typical assumptions about module dependencies



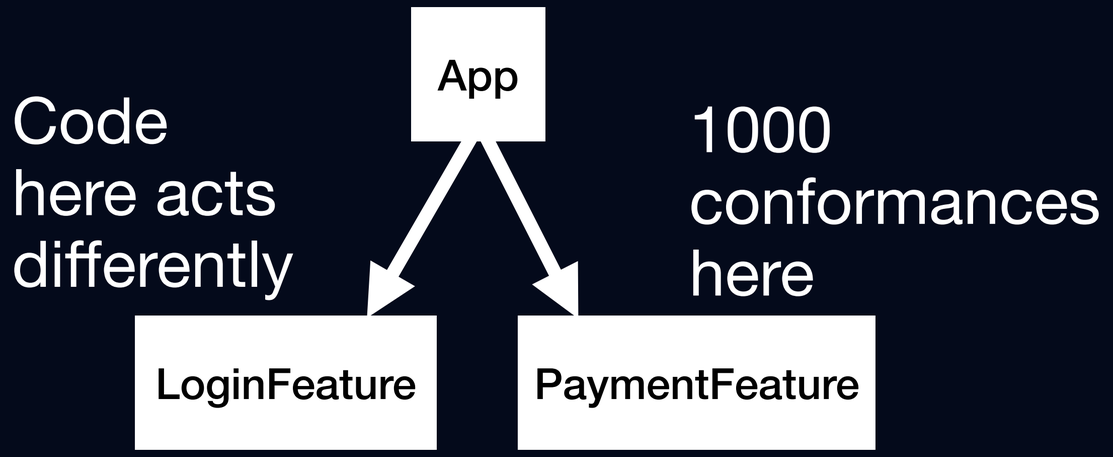
Dependencies

- Code that doesn’t run affects your code
- Violates typical assumptions about module dependencies

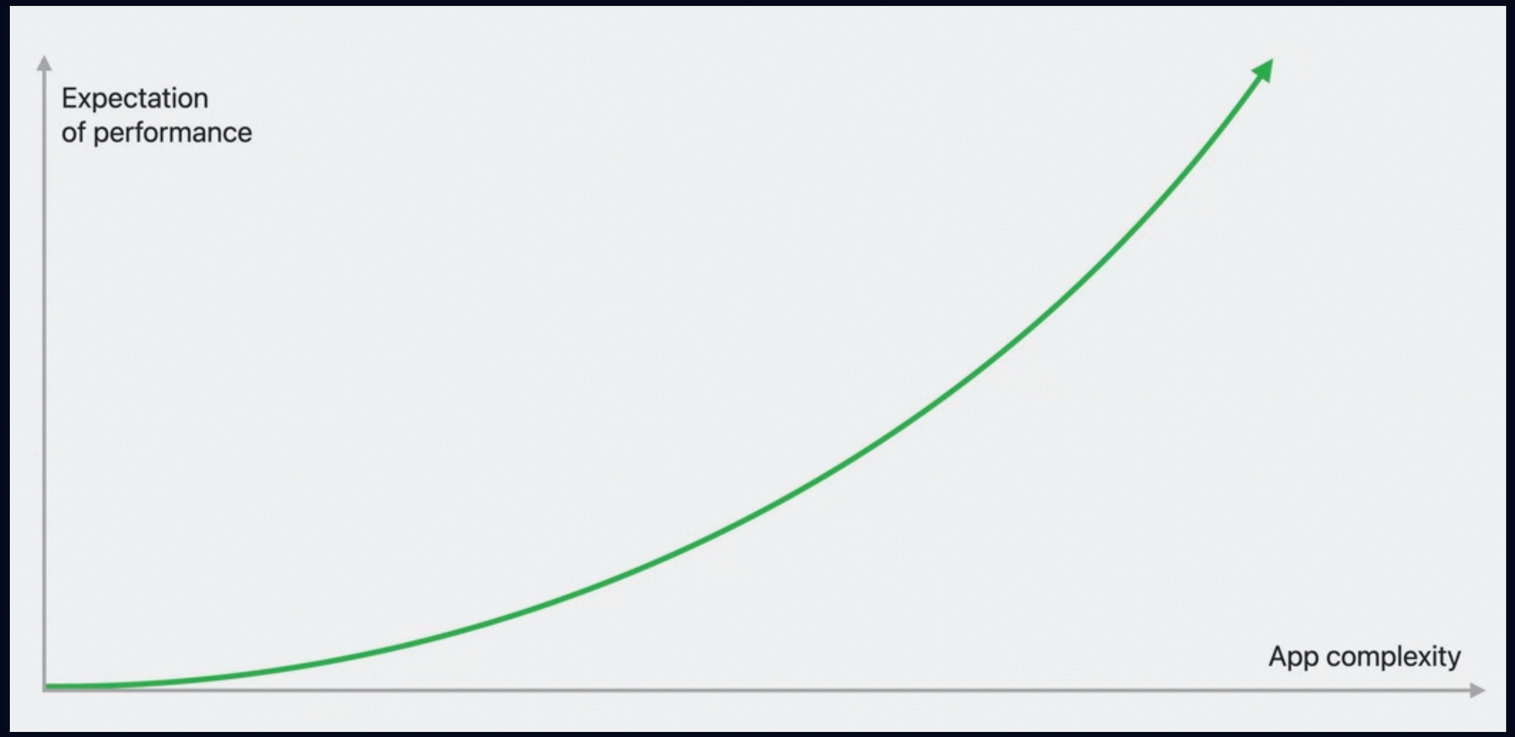


Dependencies

- Code that doesn’t run affects your code
- Violates typical assumptions about module dependencies



Dependencies



Complexity in unrelated areas of the app affect each other

Key difficulties

Abstractions hide details

Key difficulties

Abstractions hide details

Unpredictable/unexpected behavior

What is a conformance?

What is a conformance?

- Listed in binary section named `__TEXT/__swift5_proto`

What is a conformance?

- Listed in binary section named `__TEXT/__swift5_proto`
- Each entry is a type/protocol pair

What is a conformance?

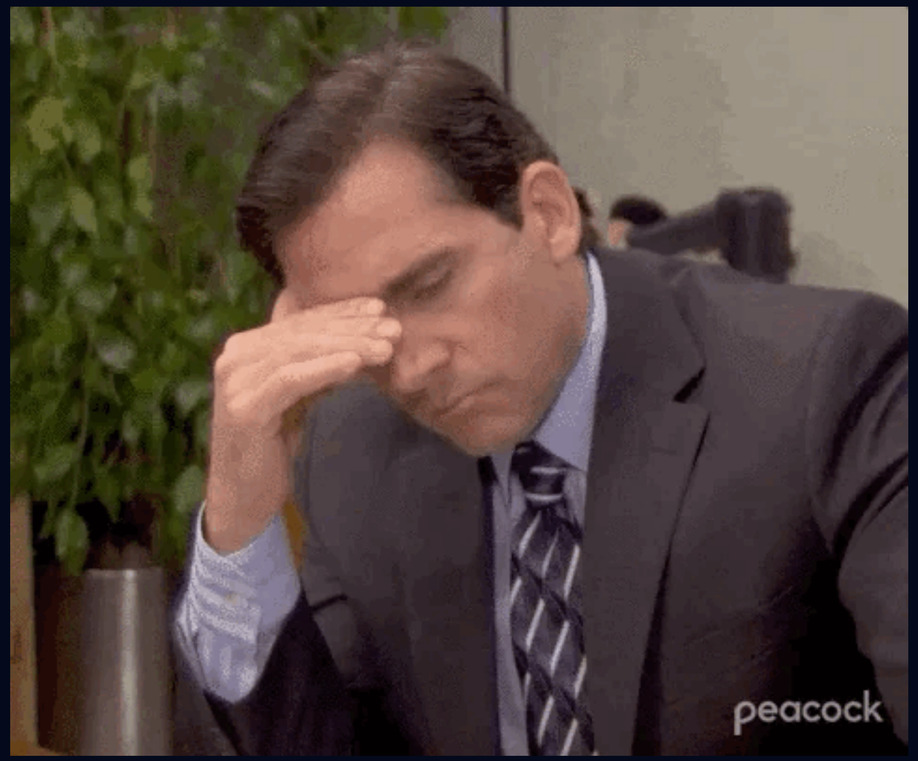
- Listed in binary section named `__TEXT/__swift5_proto`
- Each entry is a type/protocol pair
- Layout is defined in swift compiler and part of the ABI

What is a conformance?

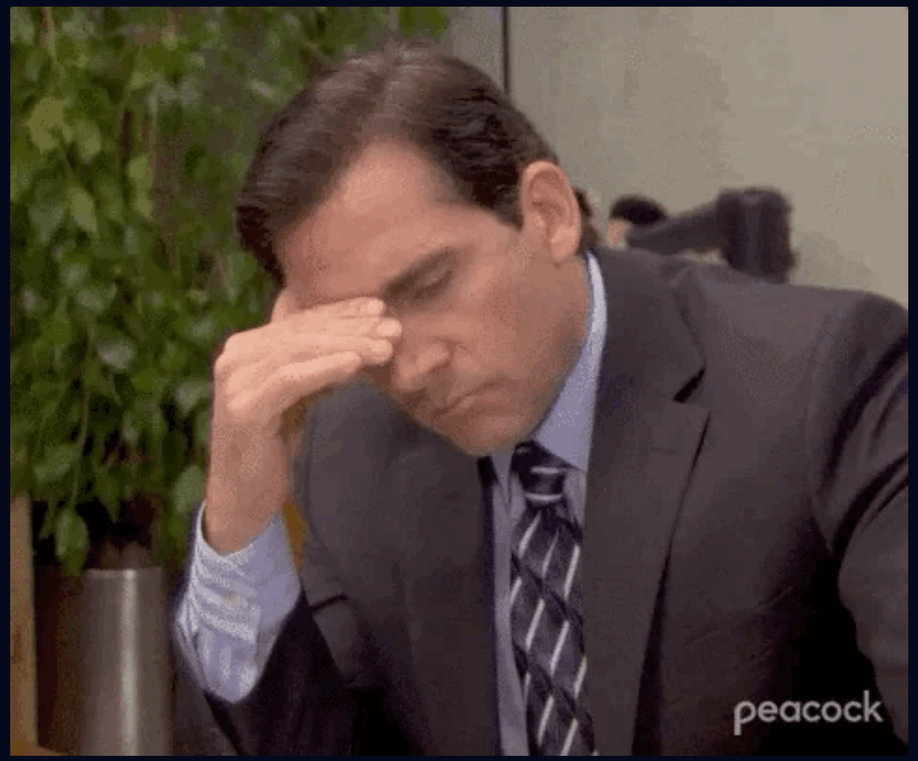
- Listed in binary section named `__TEXT/__swift5_proto`
- Each entry is a type/protocol pair
- Layout is defined in swift compiler and part of the ABI

```
struct ProtocolConformanceDescriptor {
  let protocolDescriptor: Int32
  var nominalTypeDescriptor: Int32
  let protocolWitnessTable: Int32
  let conformanceFlags: ConformanceFlags
}
```

Why does this matter?



Why does this matter?

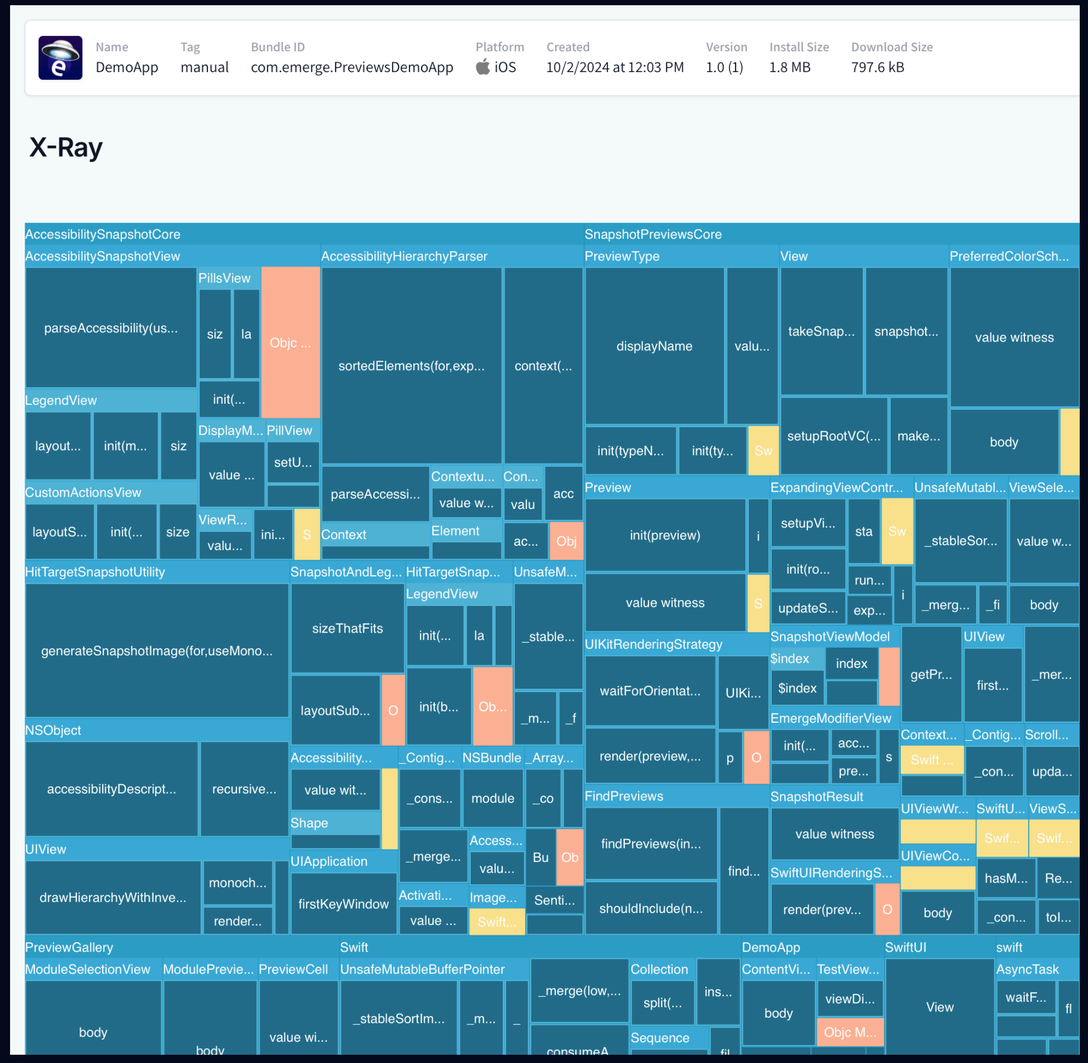


Why does this matter?

You can build cool things with it!

Why does this matter?

You can build cool things with it!



Why does this matter?

```
func allPreviews() -> [any PreviewProvider.Type] {
  // TODO
}
```

Why does this matter?

```swift
func allPreviews() -> [any PreviewProvider.Type] {
  // TODO
}
```

Similar to `objc_copyClassList` but for Swift

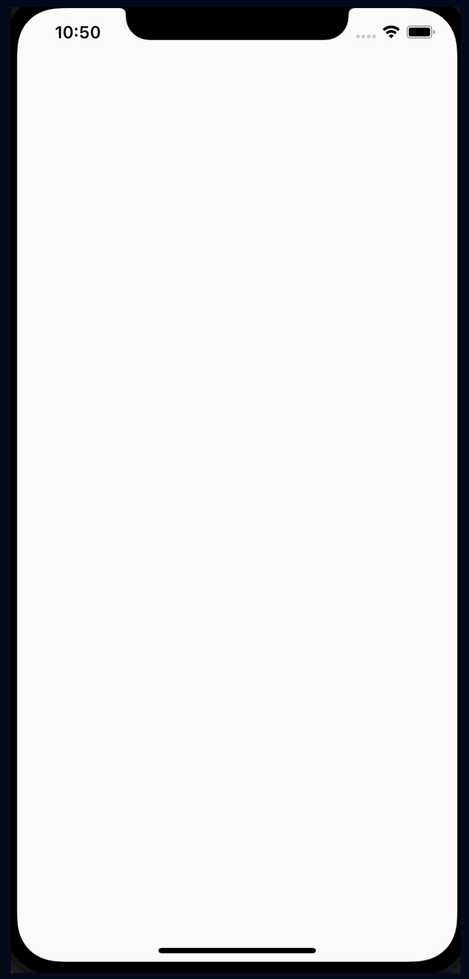
Why does this matter?

```
func allPreviews() -> [any PreviewProvider.Type] {
  // TODO
}
```

Similar to `objc_copyClassList` but for Swift

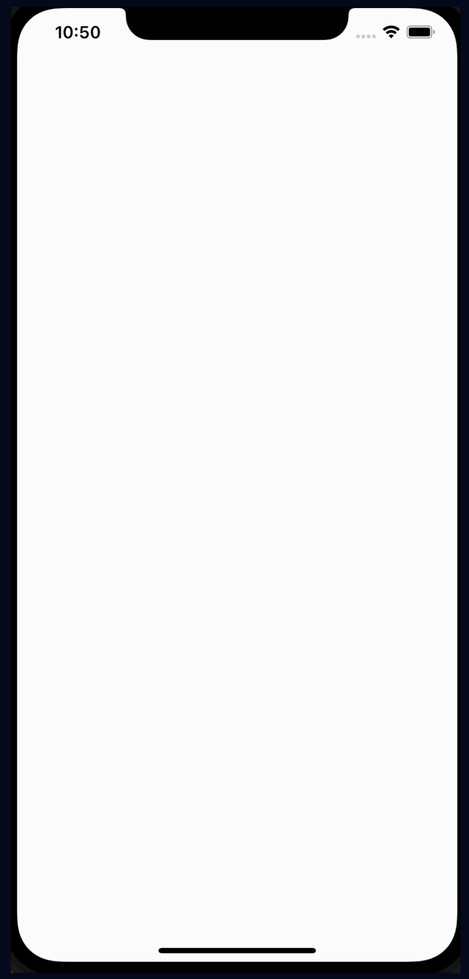
Used for EmergeTools Snapshot Testing

Why does this matter?



https://github.com/EmergeTools/SnapshotPreviews-iOS

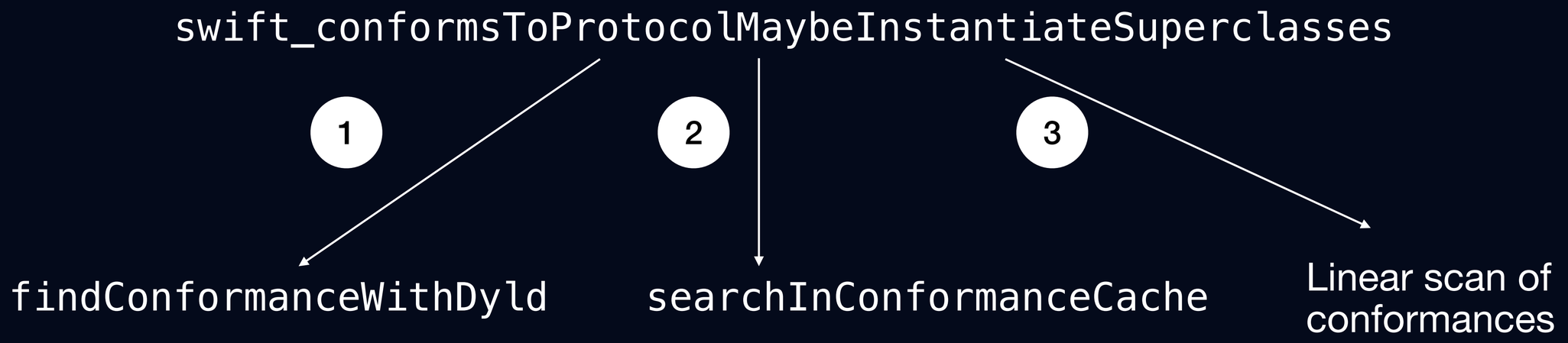
Why does this matter?



https://github.com/EmergeTools/SnapshotPreviews-iOS

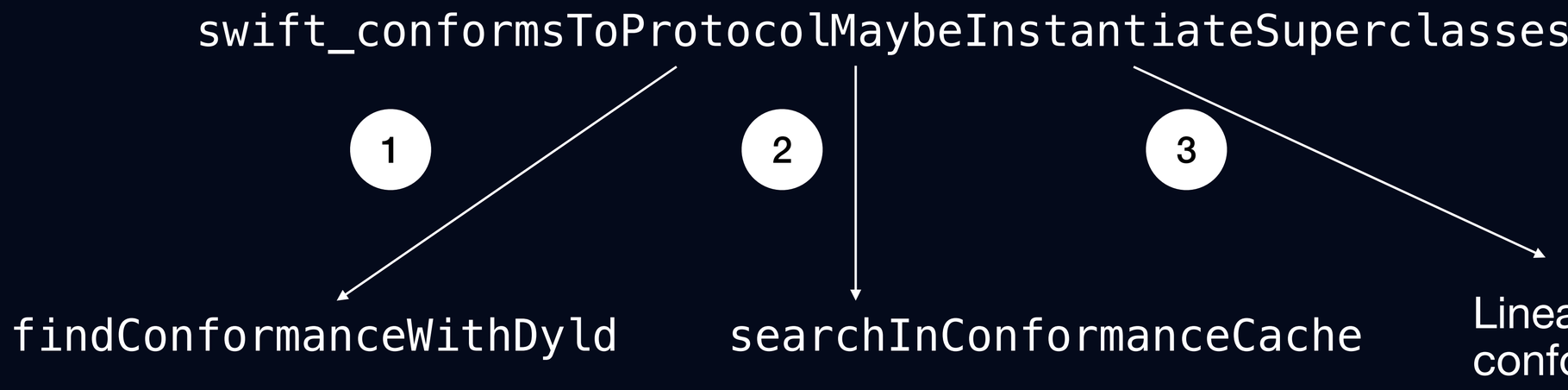
Detectives: Why more types are slower

github.com/apple/swift



Detectives: Why more types are slower

github.com/apple/swift



Clue #3

When O(n) becomes O(n^2)

When O(n) becomes O(n^2)

- Linear scan is O(n) - it gets slower as your app gets bigger

When O(n) becomes O(n²)

- Linear scan is O(n) - it gets slower as your app gets bigger
- This happens every time a conformance is checked

When O(n) becomes O(n²)

- Linear scan is O(n) - it gets slower as your app gets bigger
- This happens every time a conformance is checked
- Assume X% of conformances in your app are used

When O(n) becomes $O(n^2)$

- Linear scan is O(n) - it gets slower as your app gets bigger
- This happens every time a conformance is checked
- Assume X% of conformances in your app are used

```
usedTypes = sample(allTypes, X)
for type in usedTypes:
    for conformance in allConformances:
        if conformance.type == type:
            // Conformance found
```

When O(n) becomes O(n^2)

- Linear scan is O(n) - it gets slower as your app gets bigger
- This happens every time a conformance is checked
- Assume X% of conformances in your app are used

X% of types eventually used

```
usedTypes = sample(allTypes, X)
for type in usedTypes:
    for conformance in allConformances:
        if conformance.type == type:
            // Conformance found
```

When O(n) becomes $O(n^2)$

- Linear scan is O(n) - it gets slower as your app gets bigger
- This happens every time a conformance is checked
- Assume X% of conformances in your app are used

```
usedTypes = sample(allTypes, X)
for type in usedTypes:
    for conformance in allConformances:
        if conformance.type == type:
            // Conformance found
```

X% of types eventually used

Inner loop

When O(n) becomes O(n²)

- Increase the used percentage by removing unused conformances

When O(n) becomes O(n²)

- Increase the used percentage by removing unused conformances
- Every unused conformance is wasted time when doing the linear scan

When O(n) becomes O(n²)

- Increase the used percentage by removing unused conformances
- Every unused conformance is wasted time when doing the linear scan
- Dead code detection tools like Periphery and Reaper

Summary

- **Found what is slow**
- **Why is it so slow?**
  - Using all conformances is $O(n^2)$
- How can we make it faster?

How big of a problem is this really?

How big of a problem is this really?

- Large apps have over 100k conformances

How big of a problem is this really?

- Large apps have over 100k conformances
  - Tip: Avoid codegen protocol conformances, Decodable not Codable

How big of a problem is this really?

- Large apps have over 100k conformances
  - Tip: Avoid codegen protocol conformances, Decodable not Codable
- Conformance checks happen a lot

How big of a problem is this really?

- Large apps have over 100k conformances
  - Tip: Avoid codegen protocol conformances, Decodable not Codable
- Conformance checks happen a lot
  - ```
    class Test<T: Decodable> {}
    let _ = Test<Int>.self
    ```
    ```

How big of a problem is this really?

- Large apps have over 100k conformances
  - Tip: Avoid codegen protocol conformances, Decodable not Codable
- Conformance checks happen a lot
  - ```
    class Test<T: Decodable> {}
    let _ = Test<Int>.self
    ```
  - “\(myVar)”
    ```

How big of a problem is this really?

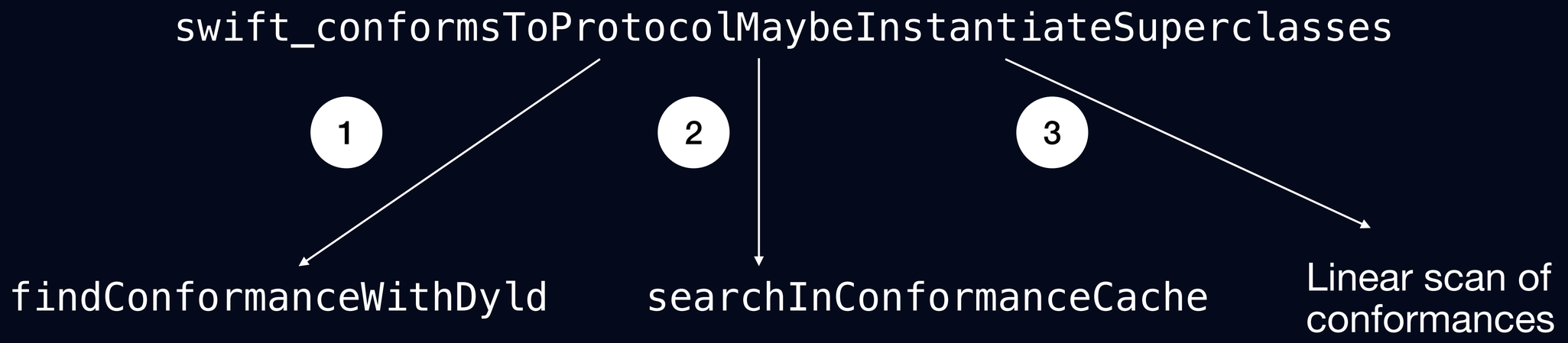
- Large apps have over 100k conformances
  - Tip: Avoid codegen protocol conformances, Decodable not Codable
- Conformance checks happen a lot
  - ```
    class Test<T: Decodable> {}
    let _ = Test<Int>.self
    ```
  - `“\(myVar)”`
  - `Mirror`
    ```

iOS 16 to the rescue

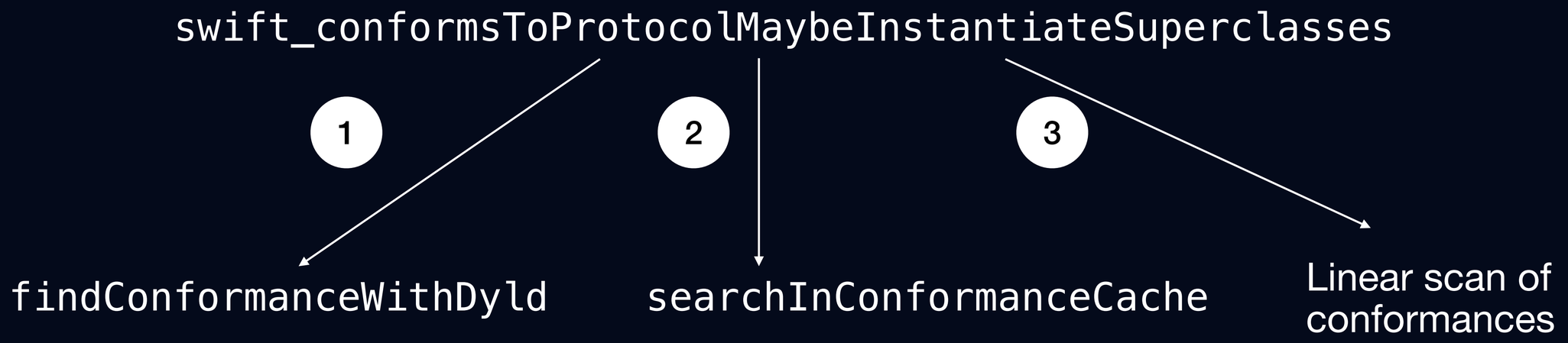
iOS 16 Release Notes

with apps like Lyft or Airbnb launching almost twice as fast thanks to improvement in the dynamic linker.

iOS 16 to the rescue



iOS 16 to the rescue



dyld cache

iOS 16 to the rescue

- dyld (dynamic linker) bootstraps app launch

iOS 16 to the rescue

- dyld (dynamic linker) bootstraps app launch
- iOS 16 added dyld conformance cache

iOS 16 to the rescue

- dyld (dynamic linker) bootstraps app launch
- iOS 16 added dyld conformance cache
- Pre-computed on app first launch to enable O(1) lookups

iOS 16 to the rescue

- dyld (dynamic linker) bootstraps app launch
- iOS 16 added dyld conformance cache
- Pre-computed on app first launch to enable O(1) lookups
  - It’s a hash table

iOS 16 to the rescue

- dyld (dynamic linker) bootstraps app launch
- iOS 16 added dyld conformance cache
- Pre-computed on app first launch to enable O(1) lookups
  - It’s a hash table
- Linear scan is done one time - O(n) operation before your code runs

iOS 16 to the rescue

- dyld (dynamic linker) bootstraps app launch
- iOS 16 added dyld conformance cache
- Pre-computed on app first launch to enable O(1) lookups
  - It’s a hash table
- Linear scan is done one time - O(n) operation before your code runs
- Cache is saved to disk

iOS 16 to the rescue

- Optimizations are open source in dyld

iOS 16 to the rescue

- Optimizations are open source in dyld
- OptimizerSwift.cpp

iOS 16 to the rescue

- Optimizations are open source in dyld
- OptimizerSwift.cpp

https://github.com/apple-oss-distributions/dyld

> The shared cache Swift optimizations are designed to speed up protocol conformance lookups.
>
> This optimization builds a number of hash tables to speed up these lookups … This saves both time and memory.

Still work to be done

Still work to be done

- Cache is computed on first launch, but not used on first launch

Still work to be done

- Cache is computed on first launch, but not used on first launch
- Each update causes a new first launch

Still work to be done

- Cache is computed on first launch, but not used on first launch
- Each update causes a new first launch
- Cache often disabled: unit testing, simulator, running from Xcode

Still work to be done

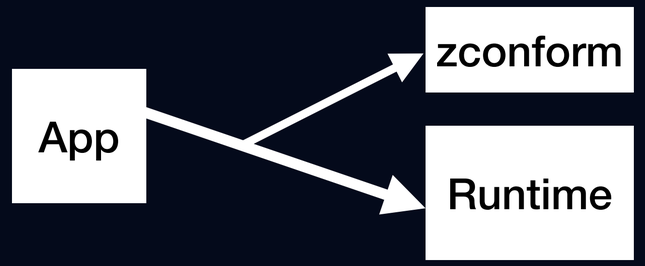
- Cache is computed on first launch, but not used on first launch
- Each update causes a new first launch
- Cache often disabled: unit testing, simulator, running from Xcode
- dyld is for Apple platforms, not server side Swift

zconform

- Drop in replacement for Swift runtime that makes apps faster

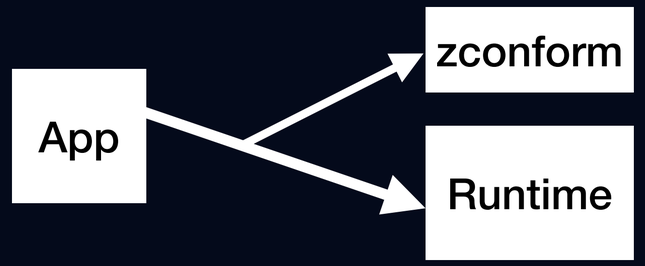
zconform

- Drop in replacement for Swift runtime that makes apps faster



zconform

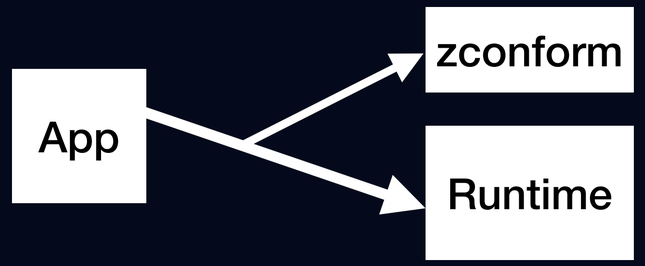
- Drop in replacement for Swift runtime that makes apps faster



- Builds a cache of conformances on app launch

zconform

- Drop in replacement for Swift runtime that makes apps faster



- Builds a cache of conformances on app launch
- Not production ready, but can be used for tests as a CI optimization, and POC for improvements in the OS

https://github.com/EmergeTools/zconform

How we use zconform at Emerge

- Snapshot tests involve repeatedly launching the app

How we use zconform at Emerge

- Snapshot tests involve repeatedly launching the app
- Each launch is $O(n^2)$ - cache is disabled due to env vars

How we use zconform at Emerge

- Snapshot tests involve repeatedly launching the app
- Each launch is $O(n^2)$ - cache is disabled due to env vars
- Snapshots should be fast, but this adds significant overhead

How we use zconform at Emerge

- Snapshot tests involve repeatedly launching the app
- Each launch is $O(n^2)$ - cache is disabled due to env vars
- Snapshots should be fast, but this adds significant overhead
- Use zconform to remove the overhead

What you can do

What you can do

- Delete unnecessary protocols with only one conforming type

What you can do

- Delete unnecessary protocols with only one conforming type
- String(describing:) -> ObjectIdentifier

What you can do

- Delete unnecessary protocols with only one conforming type
- String(describing:) -> ObjectIdentifier
- Prefer generic functions over casts

What you can do

- Delete unnecessary protocols with only one conforming type
- String(describing:) -> ObjectIdentifier
- Prefer generic functions over casts

```
func logEvent(_ event: Event) {
  if let severity = (event as? EventSeverityProviding)?.severity {
    sendToServer("Received log \(event.description)", severity: severity)
  } else {
    sendToServer("Received log \(event.description)")
  }
}
```

What you can do

- Delete unnecessary protocols with only one conforming type
- String(describing:) -> ObjectIdentifier
- Prefer generic functions over casts

```
func logEvent<T: Event & EventSeverityProviding>(_ event: T) {
  sendToServer("Received log \(event.description)", severity: event.severity)
}
func logEvent(_ event: Event) {
  sendToServer("Received log \(event.description)")
}
```

Summary

- **Found what is slow**
- **Why is it so slow?**
- **How can we make it faster?**
  - iOS 16 helped
  - zconform
  - Do fewer conformance checks

Ongoing conversation

Ongoing conversation

- Uber engineer started a forum thread about protocols earlier this year

https://forums.swift.org/t/pitch-speed-up-protocol-conformance-checks-on-first-launch-using-bloom-filters/70225/29

Ongoing conversation

- Uber engineer started a forum thread about protocols earlier this year
- “the cadence of our updates mean that the majority of users will have a poor experience once a week”

https://forums.swift.org/t/pitch-speed-up-protocol-conformance-checks-on-first-launch-using-bloom-filters/70225/29

Ongoing conversation

- Uber engineer started a forum thread about protocols earlier this year
- “the cadence of our updates mean that the majority of users will have a poor experience once a week”
- “Many engineering teams may be wasting time trying to understand why their application is taking so much time”

https://forums.swift.org/t/pitch-speed-up-protocol-conformance-checks-on-first-launch-using-bloom-filters/70225/29

Ongoing conversation

- Uber engineer started a forum thread about protocols earlier this year
- “the cadence of our updates mean that the majority of users will have a poor experience once a week”
- “Many engineering teams may be wasting time trying to understand why their application is taking so much time”
- Could build a faster lookup into Swift to avoid the linear scan

https://forums.swift.org/t/pitch-speed-up-protocol-conformance-checks-on-first-launch-using-bloom-filters/70225/29

Ongoing conversation

- My preference is to prefer simple solutions, reduce complexity

Ongoing conversation

- My preference is to prefer simple solutions, reduce complexity
- The linear scan is here to stay for ABI stability

Ongoing conversation

- My preference is to prefer simple solutions, reduce complexity
- The linear scan is here to stay for ABI stability
- The dyld cache does solve this problem, it only needs to be updated to not be skipped on first launch

Ongoing conversation

- My preference is to prefer simple solutions, reduce complexity
- The linear scan is here to stay for ABI stability
- The dyld cache does solve this problem, it only needs to be updated to not be skipped on first launch
- Still issues with non-Apple platforms

Concluding Thoughts

Concluding Thoughts

- Complexity -> poor performance

Concluding Thoughts

- Complexity -> poor performance
- Understanding performance requires low level details

Concluding Thoughts

- Complexity -> poor performance
- Understanding performance requires low level details
- Reduce usage of `as?` and amount of protocol conformances

Concluding Thoughts

- Complexity -> poor performance
- Understanding performance requires low level details
- Reduce usage of `as?` and amount of protocol conformances
- Get in touch @sond813