Getting the Most Out of HealthKit

What's new and best practices

Session 209

Matthew Salesi iOS Software Engineer
Joefrey Kibuule iOS Software Engineer

Authorization

Authorization

Activity Rings

Authorization

Activity Rings

Health Records

Authorization

Activity Rings

Health Records

Handling Data

Authorization

Authorization

Overview

Authorization

Overview

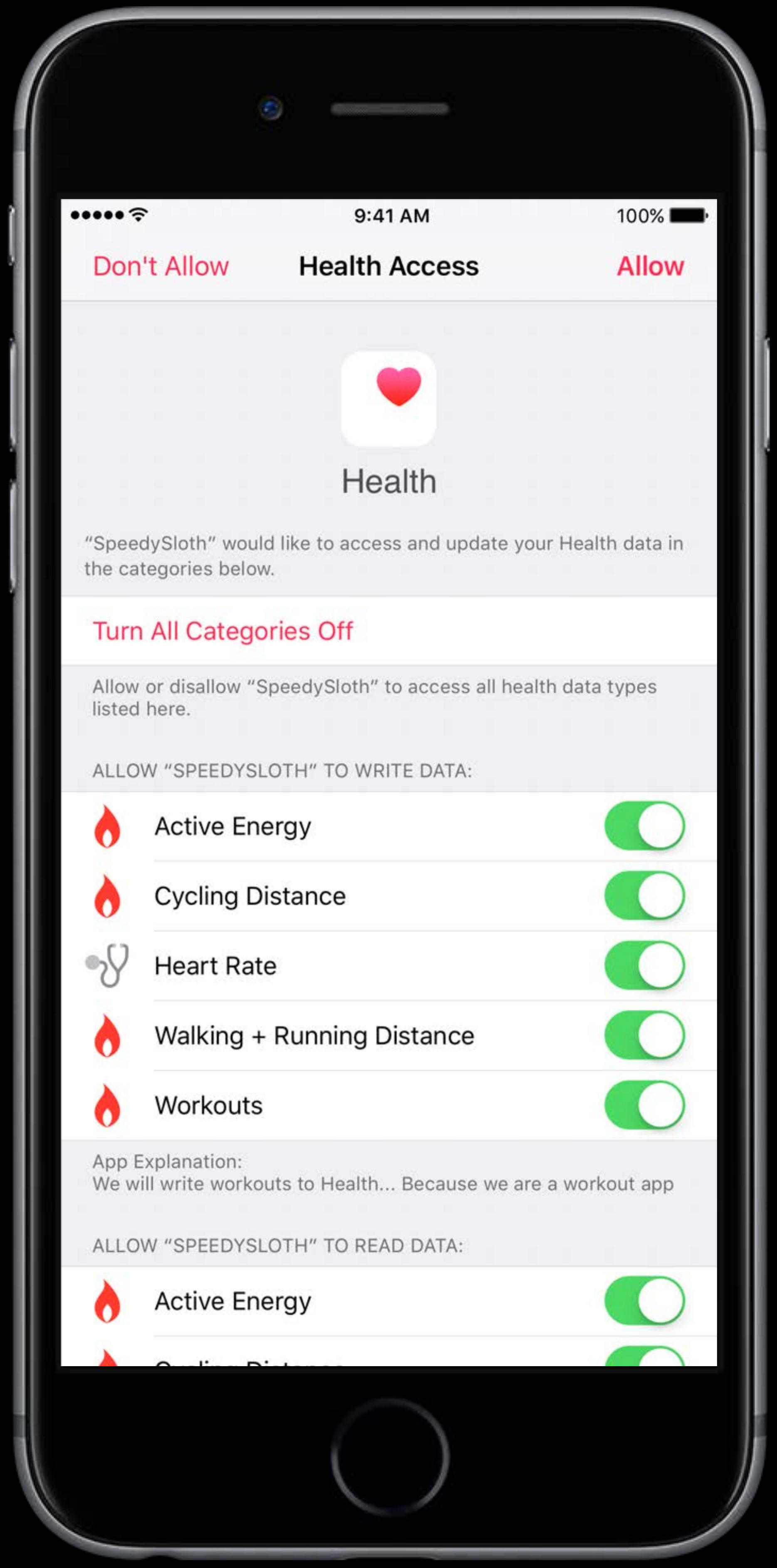
Users are in control

Authorization

Overview

Users are in control

iOS mediates authorization requests



“SpeedySloth” would like to access and update your Health data in the categories below.

Turn All Categories Off

Allow or disallow “SpeedySloth” to access all health data types listed here.

ALLOW “SPEEDYSLOTH” TO WRITE DATA:

Active Energy	✓
Cycling Distance	✓
Heart Rate	✓
Walking + Running Distance	✓
Workouts	✓

App Explanation:
We will write workouts to Health... Because we are a workout app

ALLOW “SPEEDYSLOTH” TO READ DATA:

Active Energy	✓

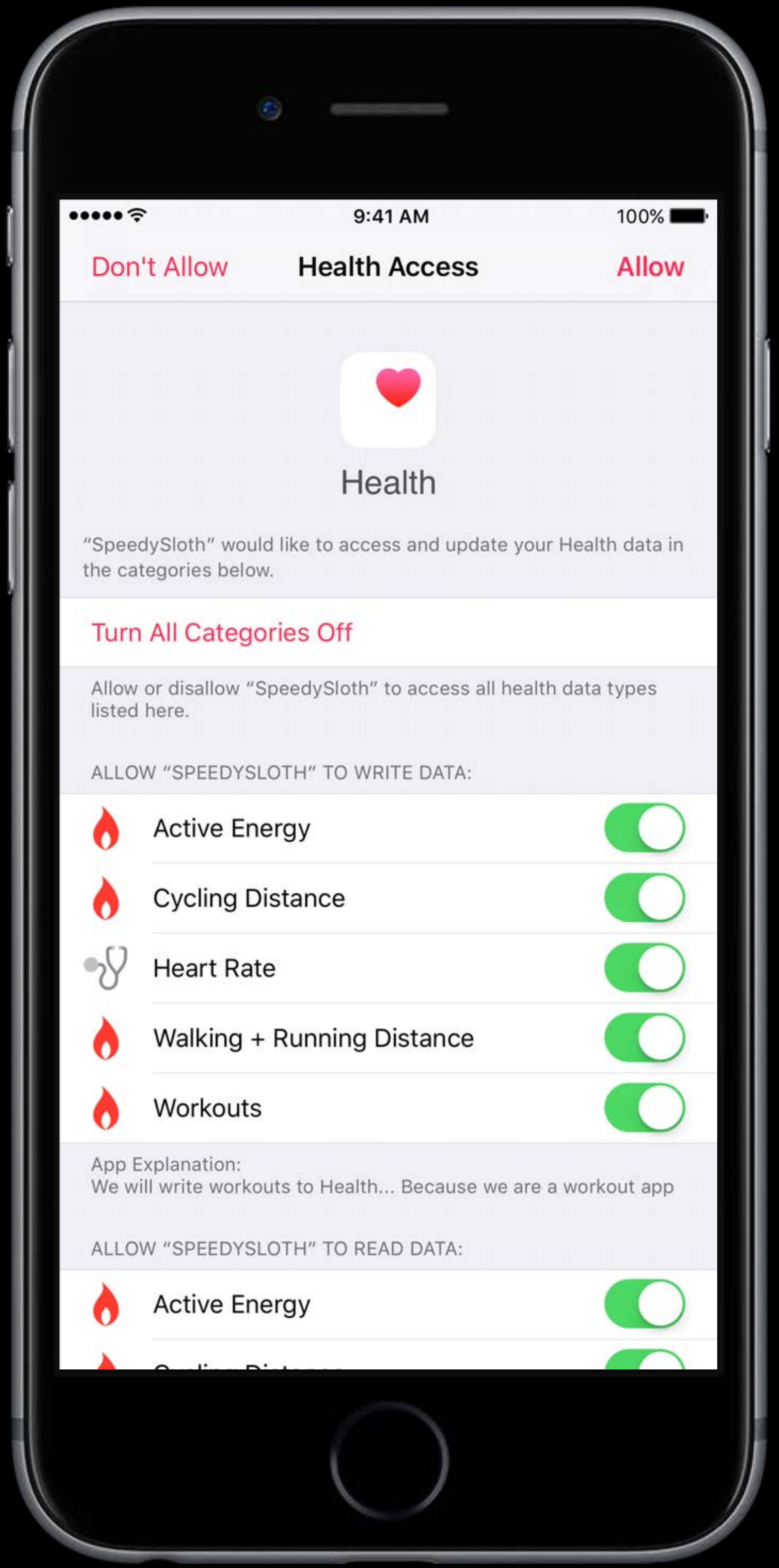
Authorization

Overview

Users are in control

iOS mediates authorization requests

Permissions may change at any time



9:41 AM 100%

Don't Allow | Health Access | Allow

Health

“SpeedySloth” would like to access and update your Health data in the categories below.

Turn All Categories Off

Allow or disallow “SpeedySloth” to access all health data types listed here.

ALLOW “SPEEDYSLOTH” TO WRITE DATA:

Active Energy	✓
Cycling Distance	✓
Heart Rate	✓
Walking + Running Distance	✓
Workouts	✓

App Explanation:
We will write workouts to Health... Because we are a workout app

ALLOW “SPEEDYSLOTH” TO READ DATA:

Active Energy	✓

Authorization

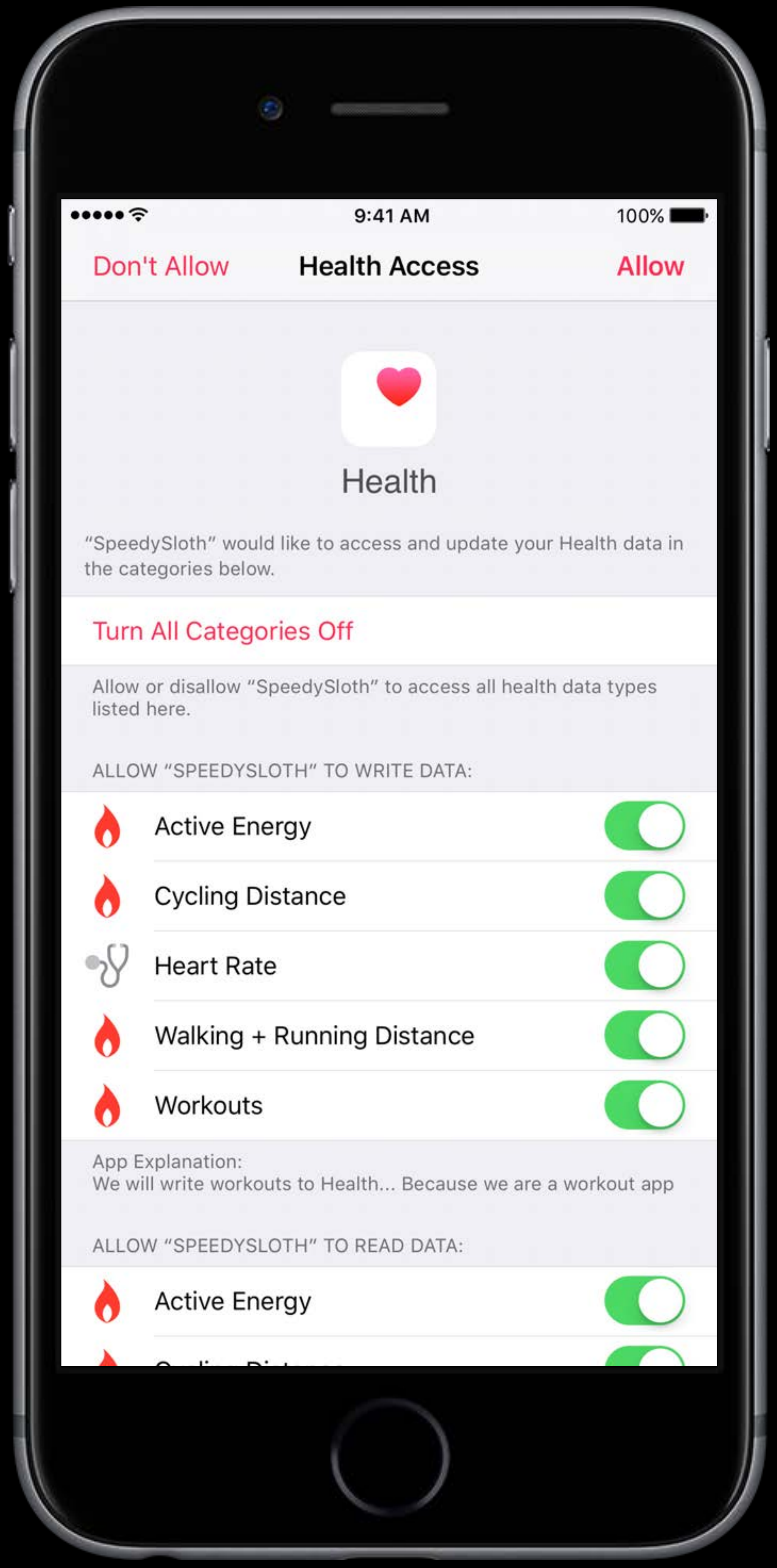
Overview

Users are in control

iOS mediates authorization requests

Permissions may change at any time

Read and write authorization
are independent


9:41 AM 100%

Don't Allow **Health Access** Allow

Health

“SpeedySloth” would like to access and update your Health data in the categories below.

Turn All Categories Off

Allow or disallow “SpeedySloth” to access all health data types listed here.

ALLOW “SPEEDYSLOTH” TO WRITE DATA:

Active Energy	✓
Cycling Distance	✓
Heart Rate	✓
Walking + Running Distance	✓
Workouts	✓

App Explanation:
We will write workouts to Health... Because we are a workout app

ALLOW “SPEEDYSLOTH” TO READ DATA:

Active Energy	✓

Authorization

Read/write permissions

Permissions	Can Query?	Can Save?

Authorization

Read/write permissions

Permissions	Can Query?	Can Save?
Read + Write	Yes	Yes

Authorization

Read/write permissions

Permissions	Can Query?	Can Save?
Read + Write	Yes	Yes
Read	Yes	No

Authorization

Read/write permissions

Permissions	Can Query?	Can Save?
Read + Write	Yes	Yes
Read	Yes	No
Write	Own Data Only	Yes

Authorization

Read/write permissions

Permissions	Can Query?	Can Save?
Read + Write	Yes	Yes
Read	Yes	No
Write	Own Data Only	Yes
None	No	No

Authorization

Usage descriptions

NEW

Authorization

Usage descriptions

NEW

Apps must explain how they will use Health data

Authorization

Usage descriptions

NEW

Apps must explain how they will use Health data

Statically declared in Info.plist

Authorization

Usage descriptions

NEW

Apps must explain how they will use Health data

Statically declared in Info.plist

- `NSHealthShareUsageDescription`

Authorization

Usage descriptions

NEW

Apps must explain how they will use Health data

Statically declared in Info.plist

- `NSHealthShareUsageDescription`
- `NSHealthUpdateUsageDescription`

```swift
// Requesting Authorization

import HealthKit

guard HKHealthStore.isHealthDataAvailable() else { return }

let healthStore = HKHealthStore()
let shareableTypes = Set([ ... ])
let readableTypes = Set([ ... ])

// Presents authorization sheet to user the first time this is called.
healthStore.requestAuthorization(toShare: shareableTypes,
                                    read: readableTypes) { success, error in
   // Handle authorization response here.
}
```

```swift
// Requesting Authorization

import HealthKit

guard HKHealthStore.isHealthDataAvailable() else { return }

let healthStore = HKHealthStore()
let shareableTypes = Set([ ... ])
let readableTypes = Set([ ... ])

// Presents authorization sheet to user the first time this is called.
healthStore.requestAuthorization(toShare: shareableTypes,
                                    read: readableTypes) { success, error in
   // Handle authorization response here.
}
```

```swift
// Requesting Authorization

import HealthKit

guard HKHealthStore.isHealthDataAvailable() else { return }

let healthStore = HKHealthStore()
let shareableTypes = Set([ ... ])
let readableTypes = Set([ ... ])

// Presents authorization sheet to user the first time this is called.
healthStore.requestAuthorization(toShare: shareableTypes,
                                    read: readableTypes) { success, error in
    // Handle authorization response here.
}
```

```swift
// Requesting Authorization

import HealthKit

guard HKHealthStore.isHealthDataAvailable() else { return }

let healthStore = HKHealthStore()
let shareableTypes = Set([ ... ])
let readableTypes = Set([ ... ])

// Presents authorization sheet to user the first time this is called.
healthStore.requestAuthorization(toShare: shareableTypes,
                                    read: readableTypes) { success, error in
   // Handle authorization response here.
}
```

Authorization

watchOS

Authorization

watchOS

Shared with your iPhone app

Authorization
watchOS

Shared with your iPhone app

Approval requires interaction with iPhone

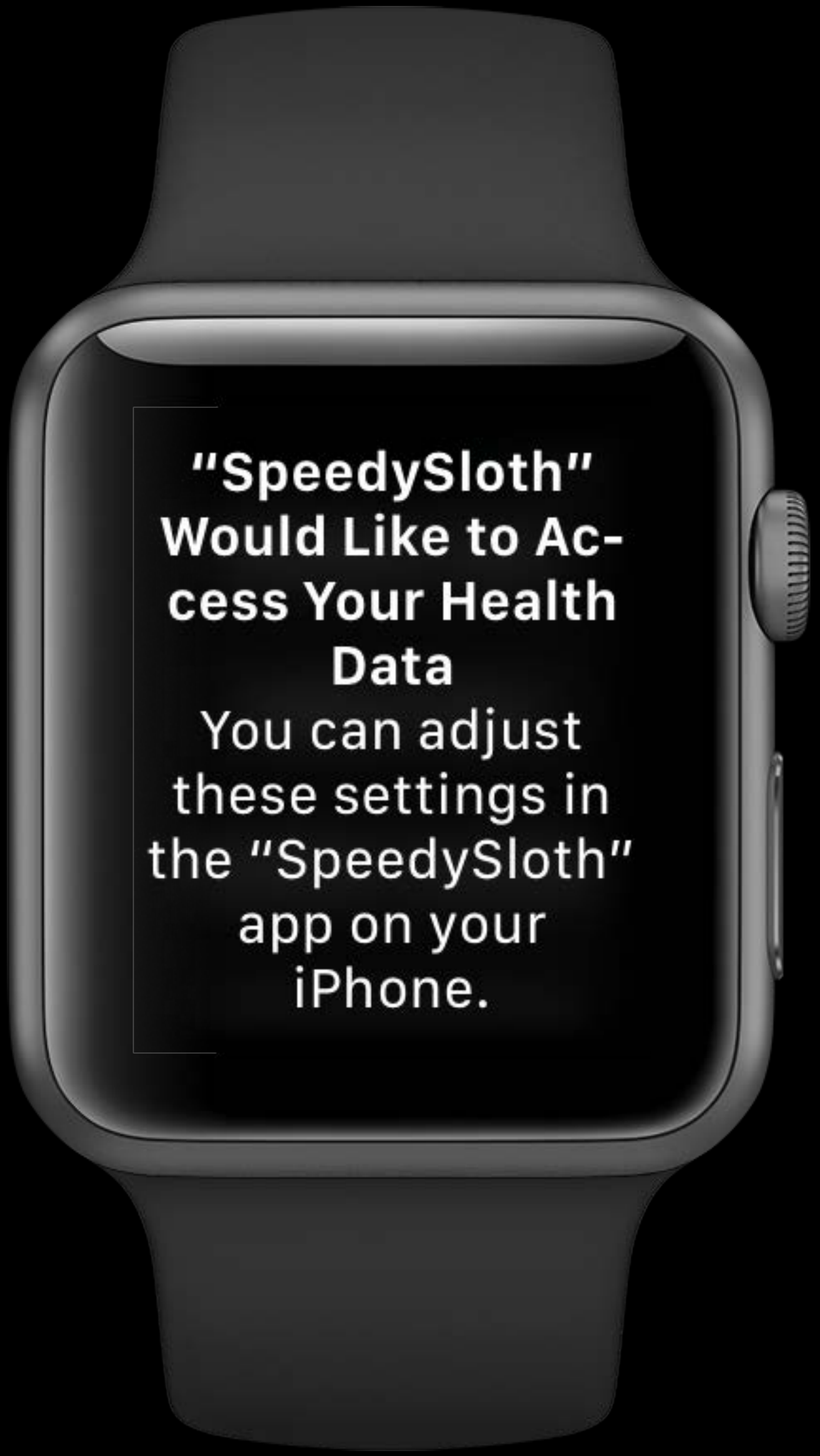
Authorization

watchOS

Shared with your iPhone app

Approval requires interaction with iPhone

- May not be easily accessible



“SpeedySloth” Would Like to Access Your Health Data

You can adjust these settings in the “SpeedySloth” app on your iPhone.

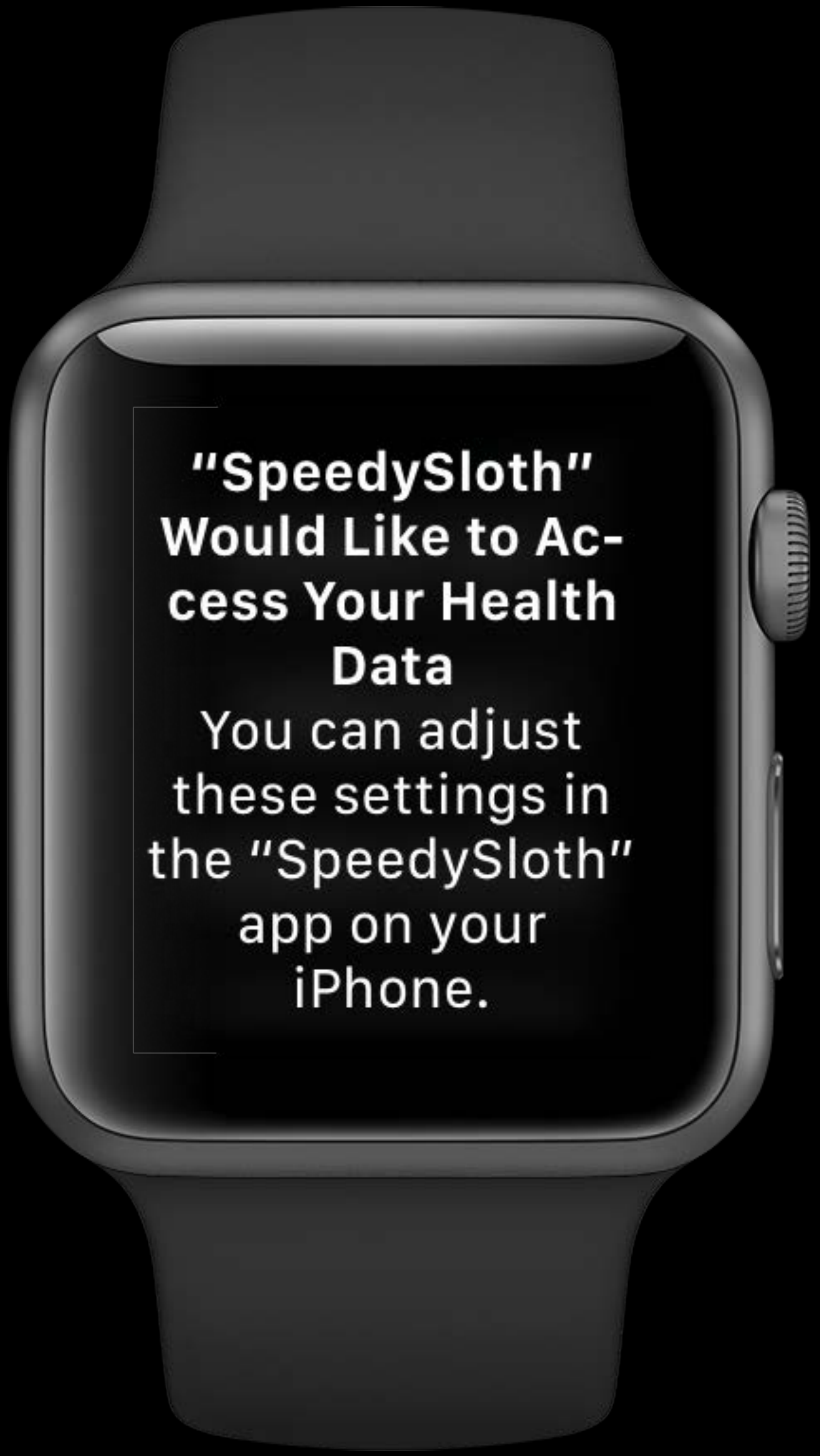
Authorization

watchOS

Shared with your iPhone app

Approval requires interaction with iPhone

- May not be easily accessible
- May be out of range



"SpeedySloth" Would Like to Access Your Health Data

You can adjust these settings in the "SpeedySloth" app on your iPhone.

Authorization

Best practices

Authorization

Best practices

✓

Make initial requests in sensible groupings of types

Authorization

Best practices

Make initial requests in sensible groupings of types

Option: Request all your app's types during on-boarding

Authorization

Best practices

Make initial requests in sensible groupings of types

Option: Request all your app's types during on-boarding

Reset authorization frequently during development

Authorization

Best practices

Make initial requests in sensible groupings of types

Option: Request all your app’s types during on-boarding

Reset authorization frequently during development

Test delaying/denying authorization

Authorization

Best practices

Make initial requests in sensible groupings of types

Option: Request all your app's types during on-boarding

Reset authorization frequently during development

Test delaying/denying authorization

Do what's best for the user

Activity Rings

10:09 AM
100%
September 2015
Activity
M T W T F S S
SEP
1 2 3 4 5 6
7 8 9 10 11 12 13
14 15 16 17 18 19 20
21 22 23 24 25 26 27
28 29 30
OCT
1 2 3 4
History
Workouts
Achievements
Sharing
Activity
10:09

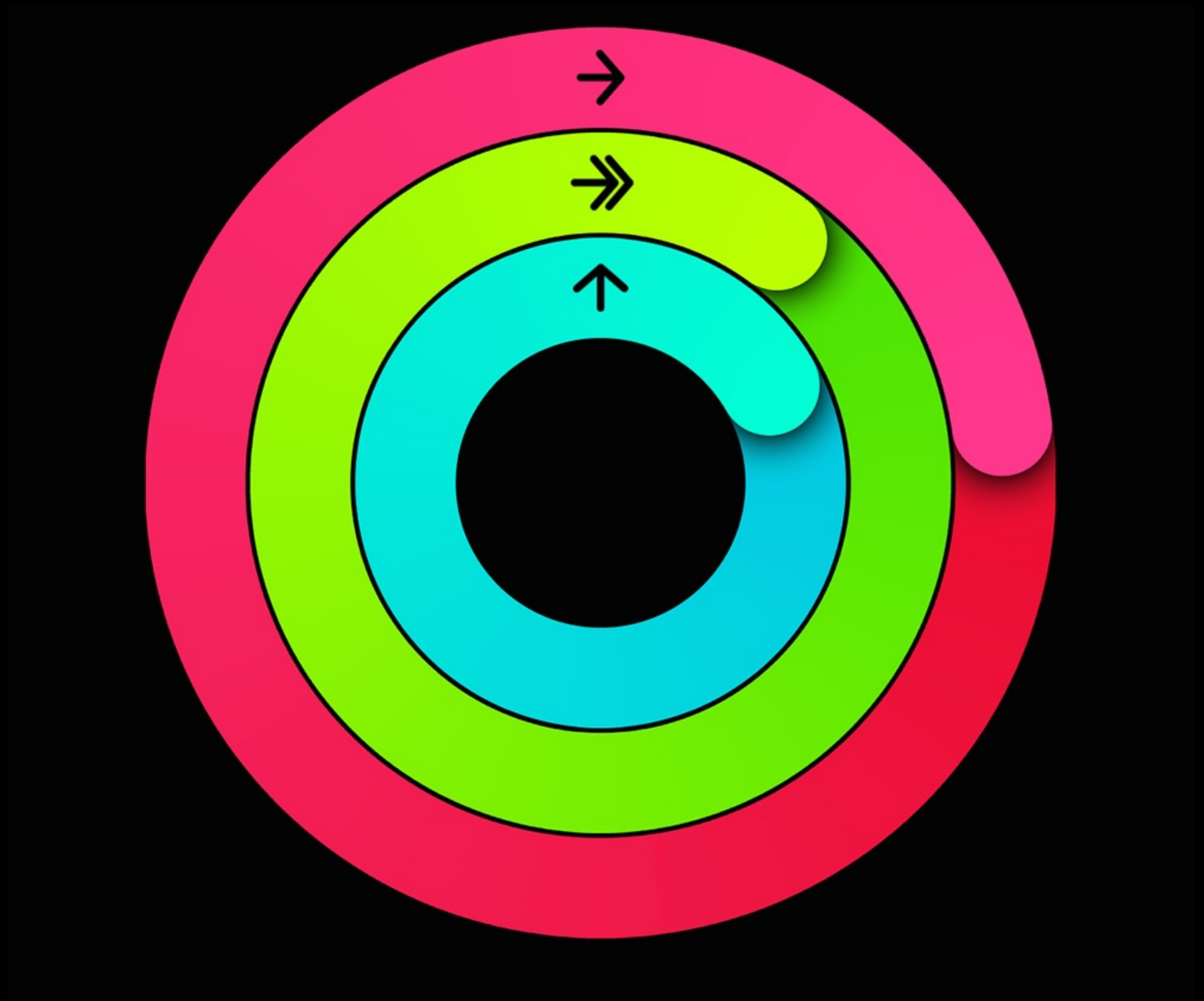
Activity Rings

HKActivitySummary

Activity Rings

HKActivitySummary

Daily activity values

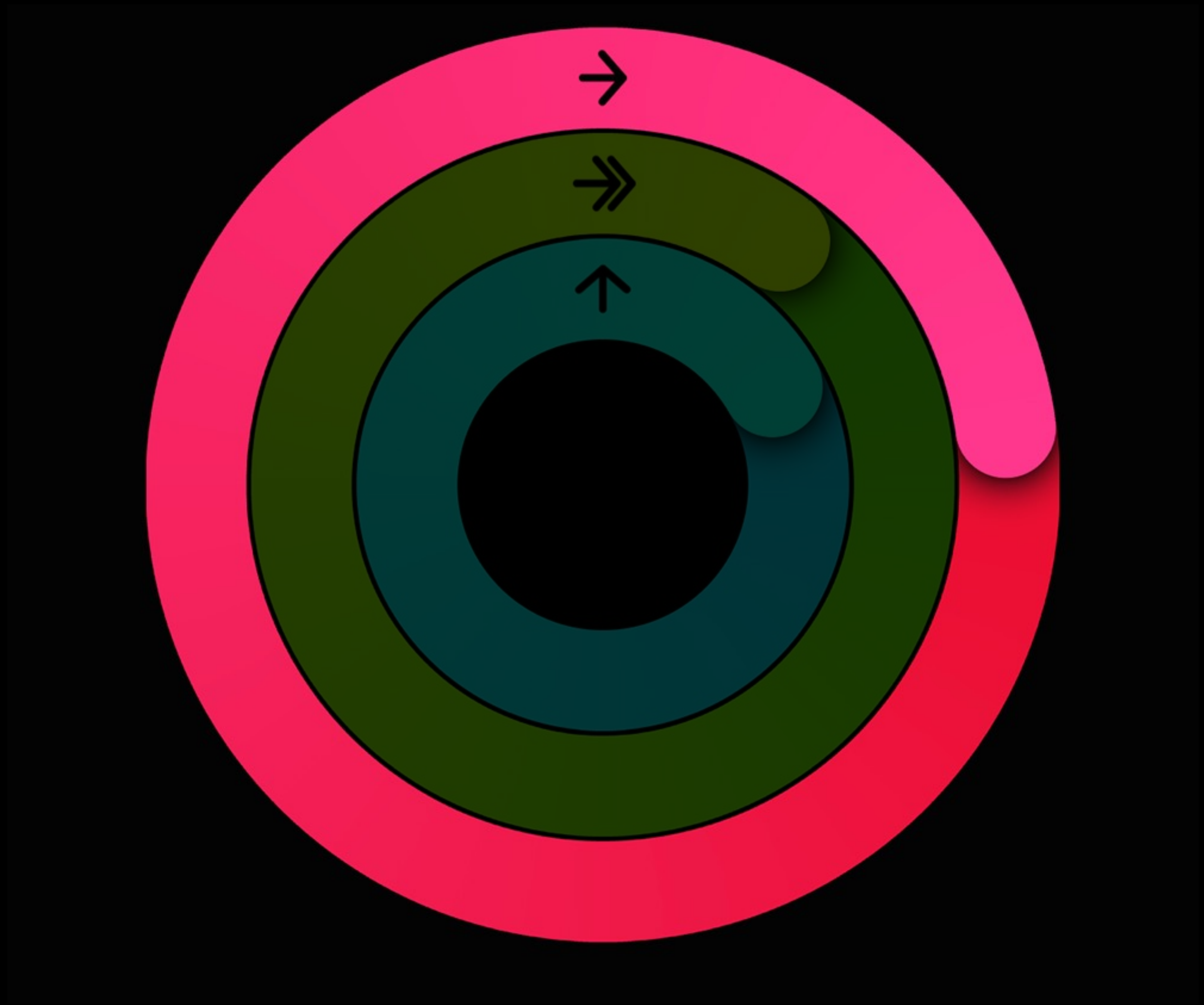


Activity Rings

HKActivitySummary

Daily activity values

- Move calories and goal



Activity Rings

HKActivitySummary

Daily activity values

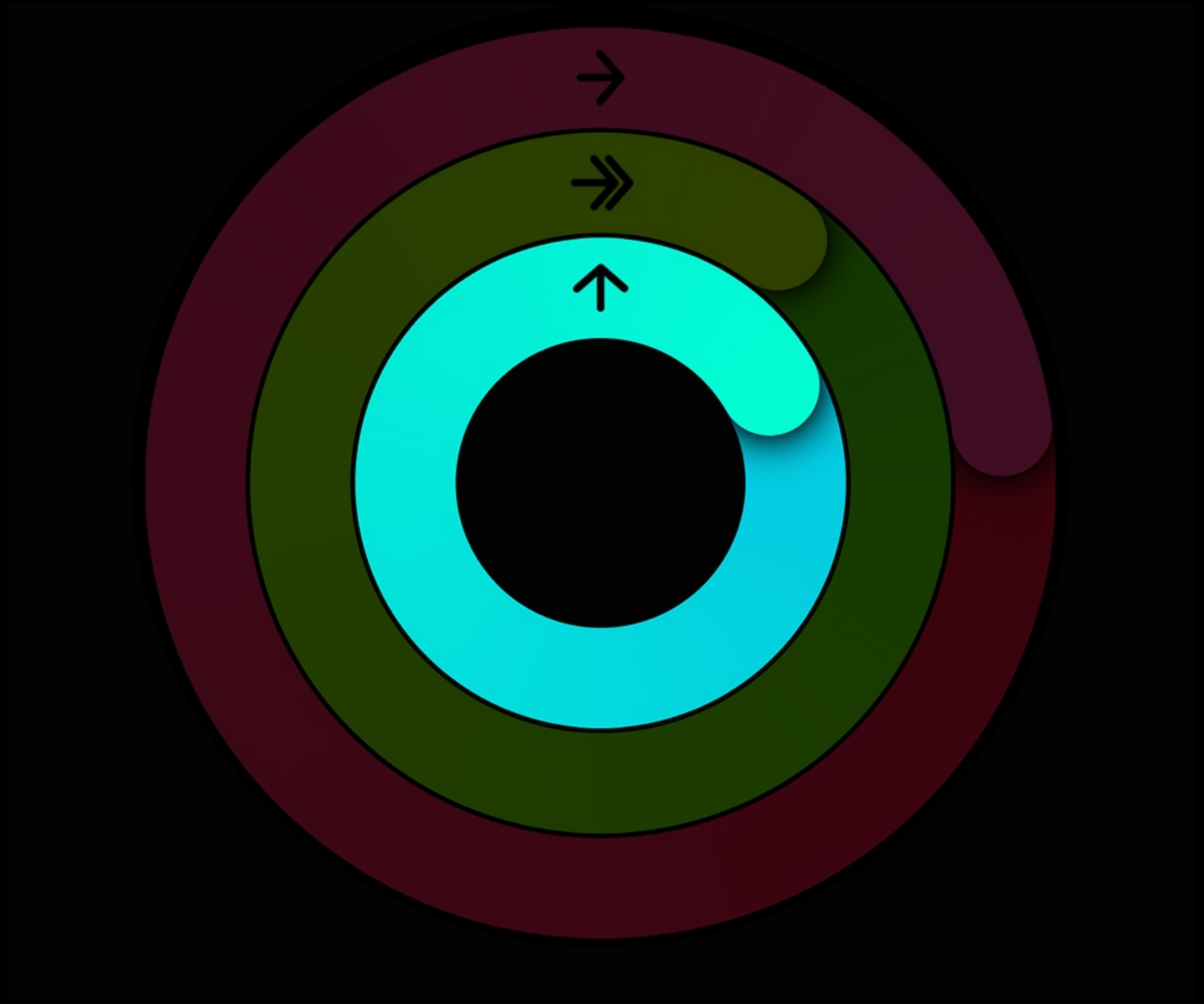
- Move calories and goal
- Exercise minutes and goal

Activity Rings

HKActivitySummary

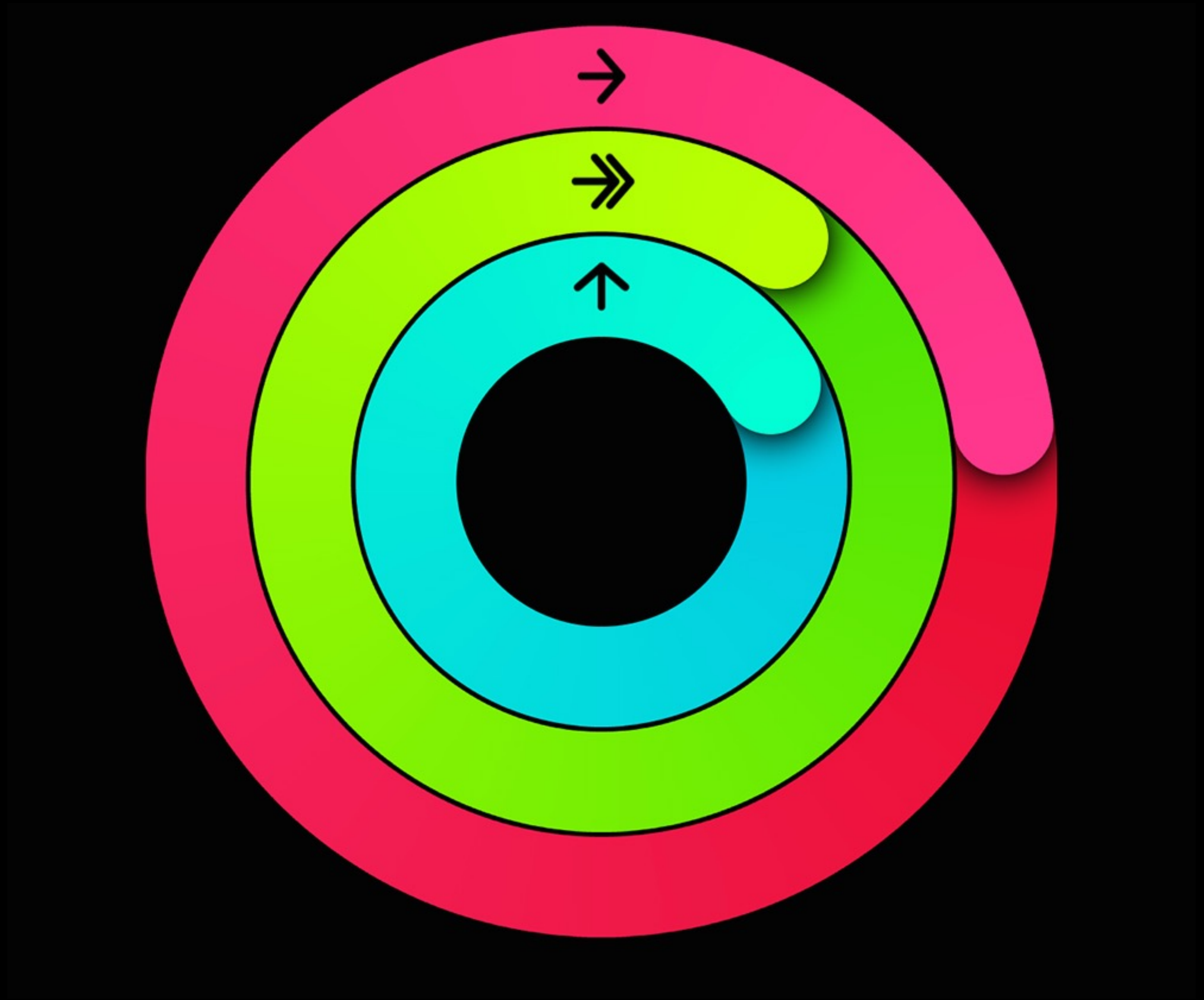
Daily activity values

- Move calories and goal
- Exercise minutes and goal
- Stand hours and goal



Activity Rings

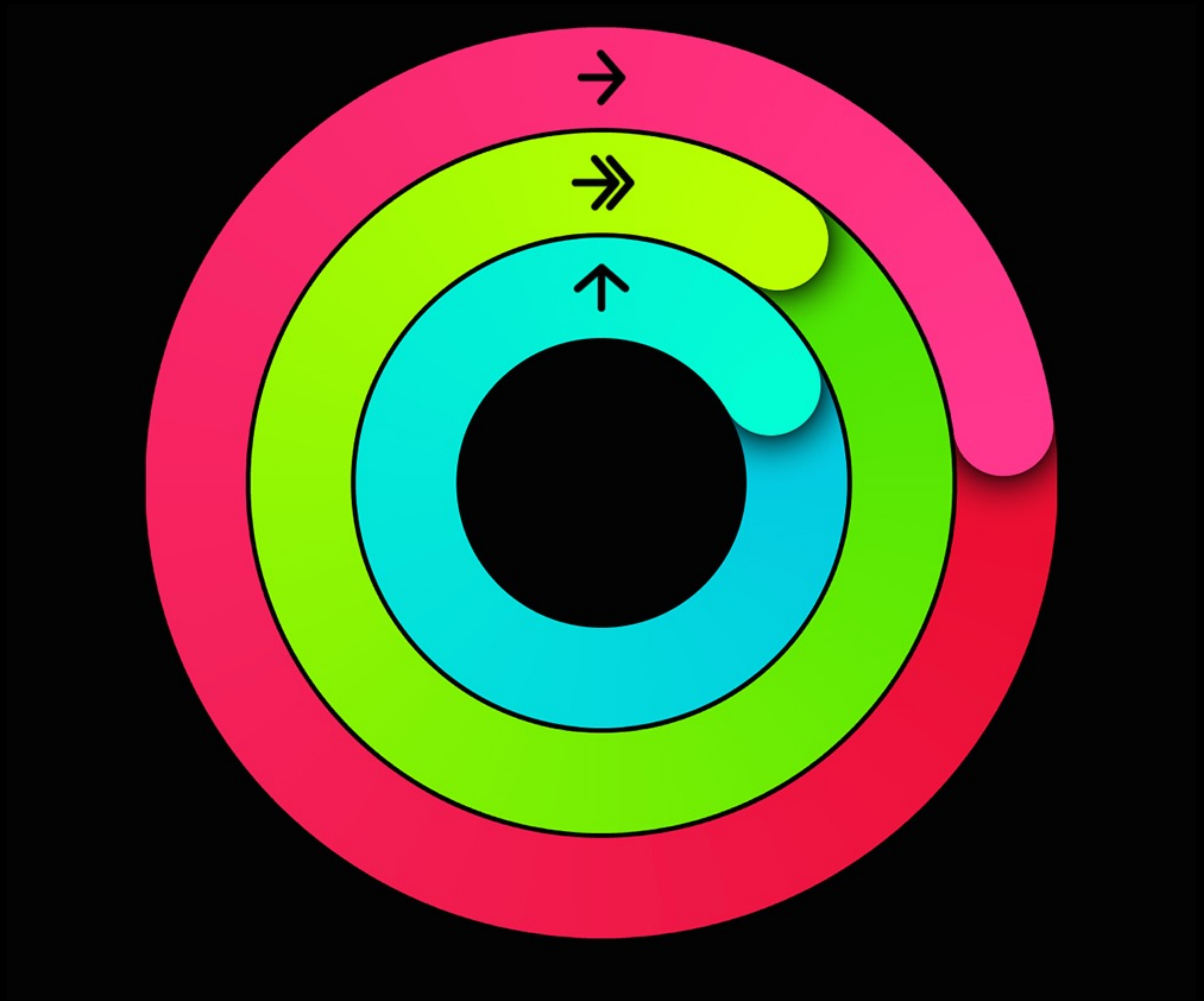
HKActivitySummary



Activity Rings

HKActivitySummary

Distinct type for authorization

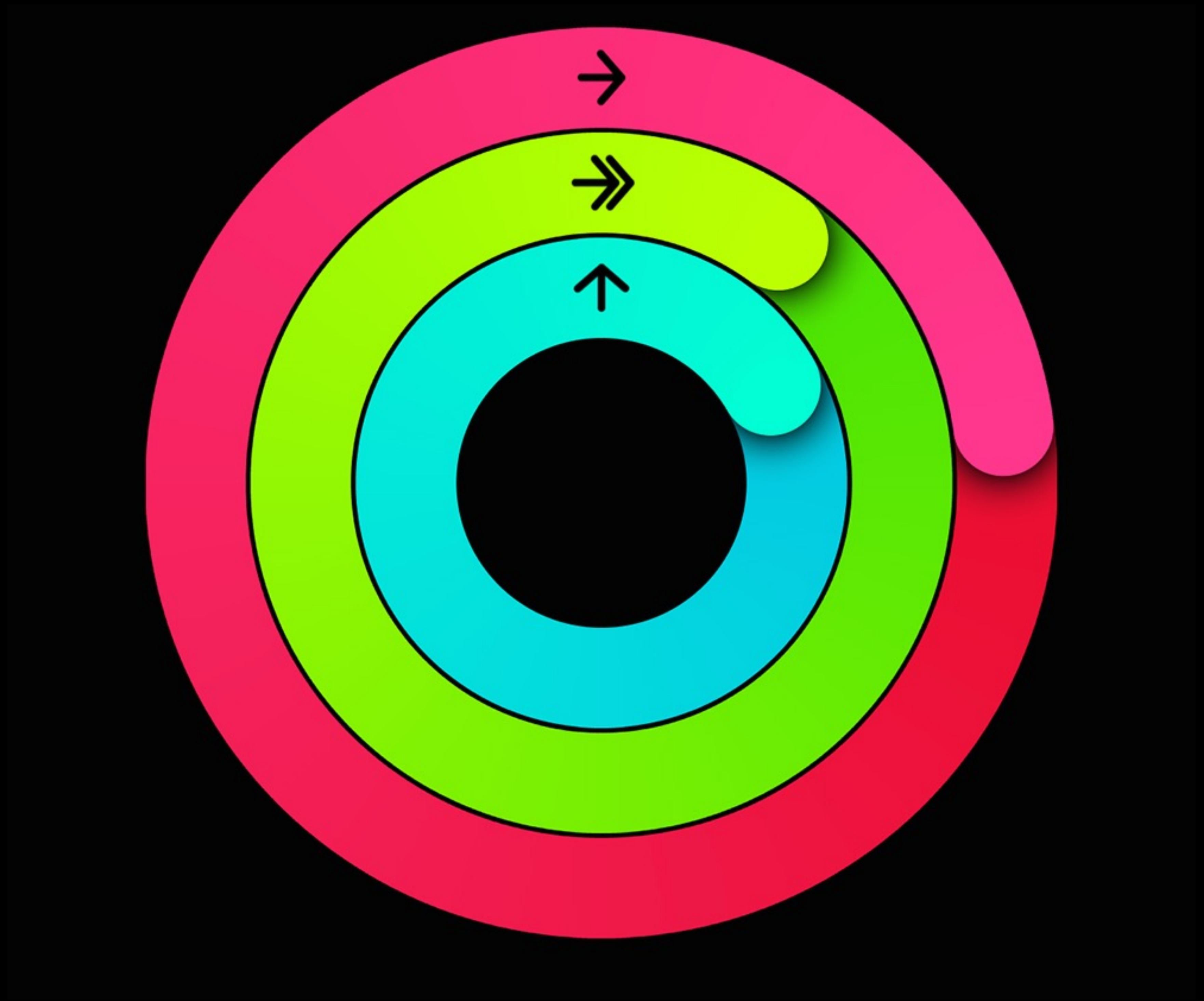


Activity Rings

HKActivitySummary

Distinct type for authorization

Daily aggregated totals



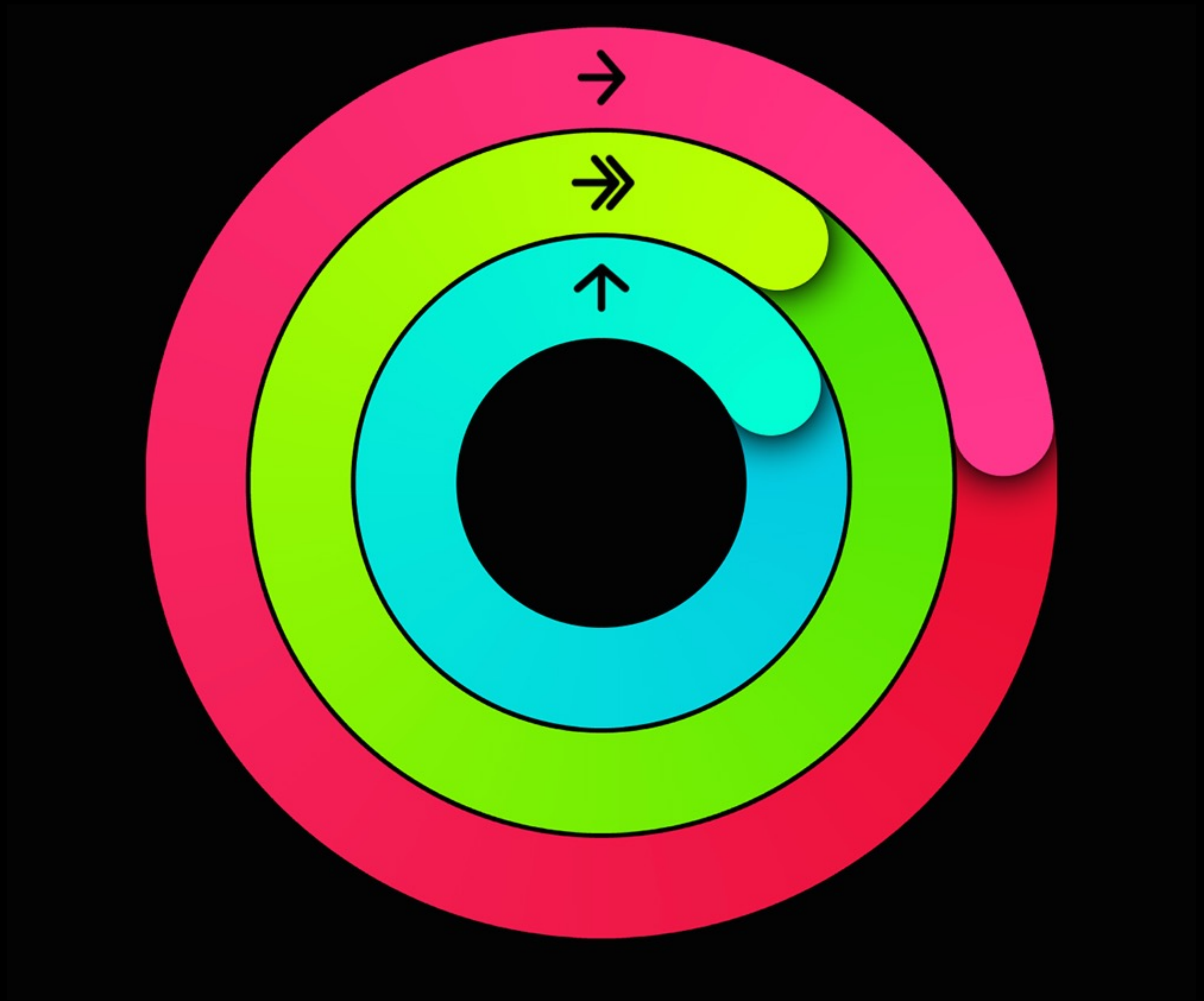
Activity Rings

HKActivitySummary

Distinct type for authorization

Daily aggregated totals

Day specified as `DateComponents`



```swift
// HKActivitySummaryQuery

import HealthKit

let healthStore = HKHealthStore()

let calendar = Calendar.current()
var components = calendar.components([.era, .year, .month, .day], from:Date())
components.calendar = calendar

let predicate = Predicate(format: "%K = %@", HKPredicateKeyPathDateComponents, components)

let query = HKActivitySummaryQuery(predicate: predicate) { query, summaries, error in
   guard let todayActivitySummary = summaries?.first else { return }
   // Display todayActivitySummary's data.
}
healthStore.execute(query)
```

```swift
// HKActivitySummaryQuery

import HealthKit

let healthStore = HKHealthStore()

let calendar = Calendar.current()
var components = calendar.components([.era, .year, .month, .day], from:Date())
components.calendar = calendar

let predicate = Predicate(format: "%K = %@", HKPredicateKeyPathDateComponents, components)

let query = HKActivitySummaryQuery(predicate: predicate) { query, summaries, error in
   guard let todayActivitySummary = summaries?.first else { return }
   // Display todayActivitySummary's data.
}
healthStore.execute(query)
```

```swift
// HKActivitySummaryQuery

import HealthKit

let healthStore = HKHealthStore()

let calendar = Calendar.current()
var components = calendar.components([.era, .year, .month, .day], from:Date())
components.calendar = calendar

let predicate = Predicate(format: "%K = %@", HKPredicateKeyPathDateComponents, components)

let query = HKActivitySummaryQuery(predicate: predicate) { query, summaries, error in
   guard let todayActivitySummary = summaries?.first else { return }
   // Display todayActivitySummary's data.
}
healthStore.execute(query)
```

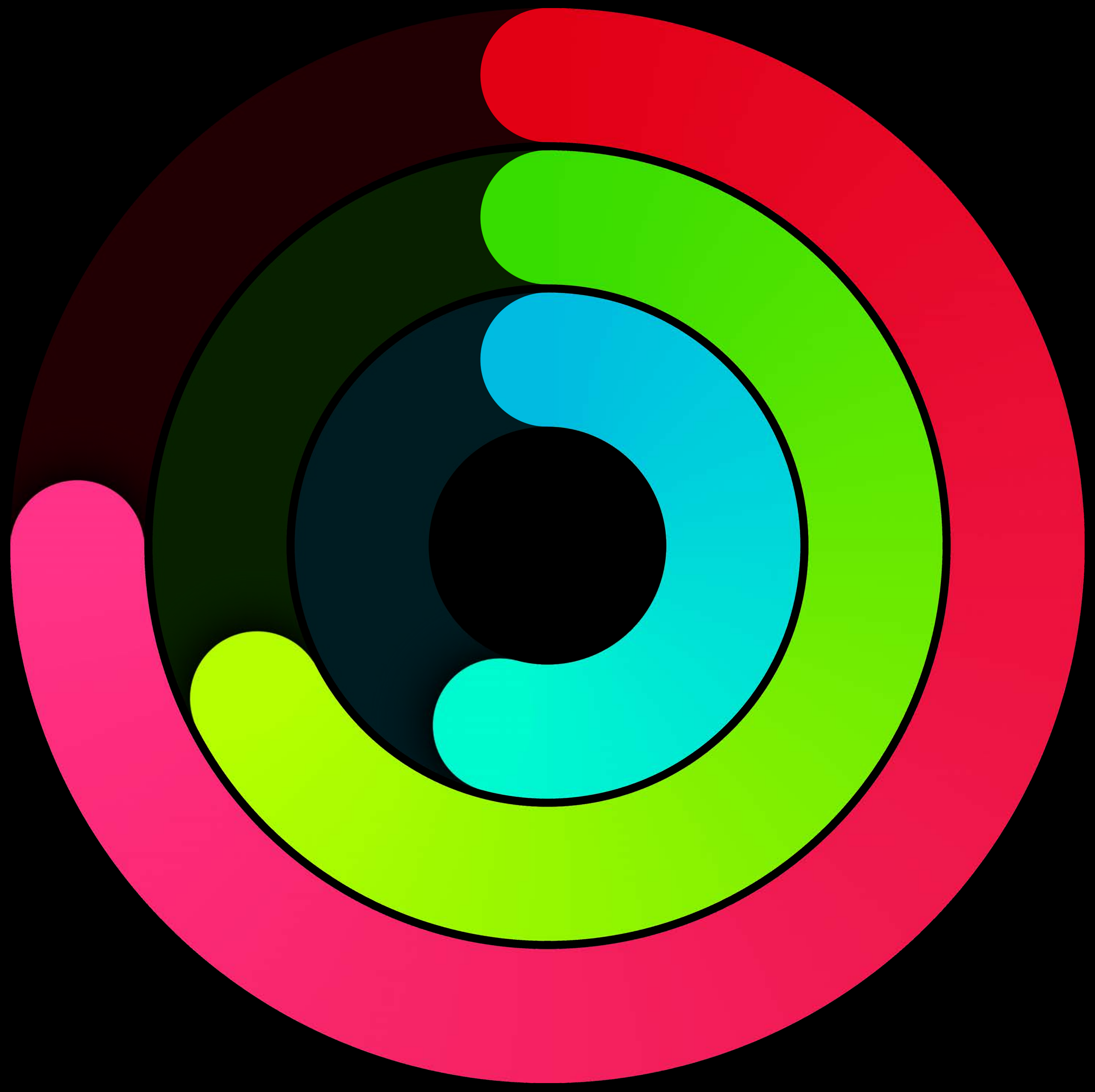
```swift
// HKActivitySummaryQuery

import HealthKit

let healthStore = HKHealthStore()

let calendar = Calendar.current()
var components = calendar.components([.era, .year, .month, .day], from:Date())
components.calendar = calendar

let predicate = Predicate(format: "%K = %@", HKPredicateKeyPathDateComponents, components)

let query = HKActivitySummaryQuery(predicate: predicate) { query, summaries, error in
   guard let todayActivitySummary = summaries?.first else { return }
   // Display todayActivitySummary's data.
}
healthStore.execute(query)
```

Activity Rings

HKActivityRingView and WKInterfaceActivityRing

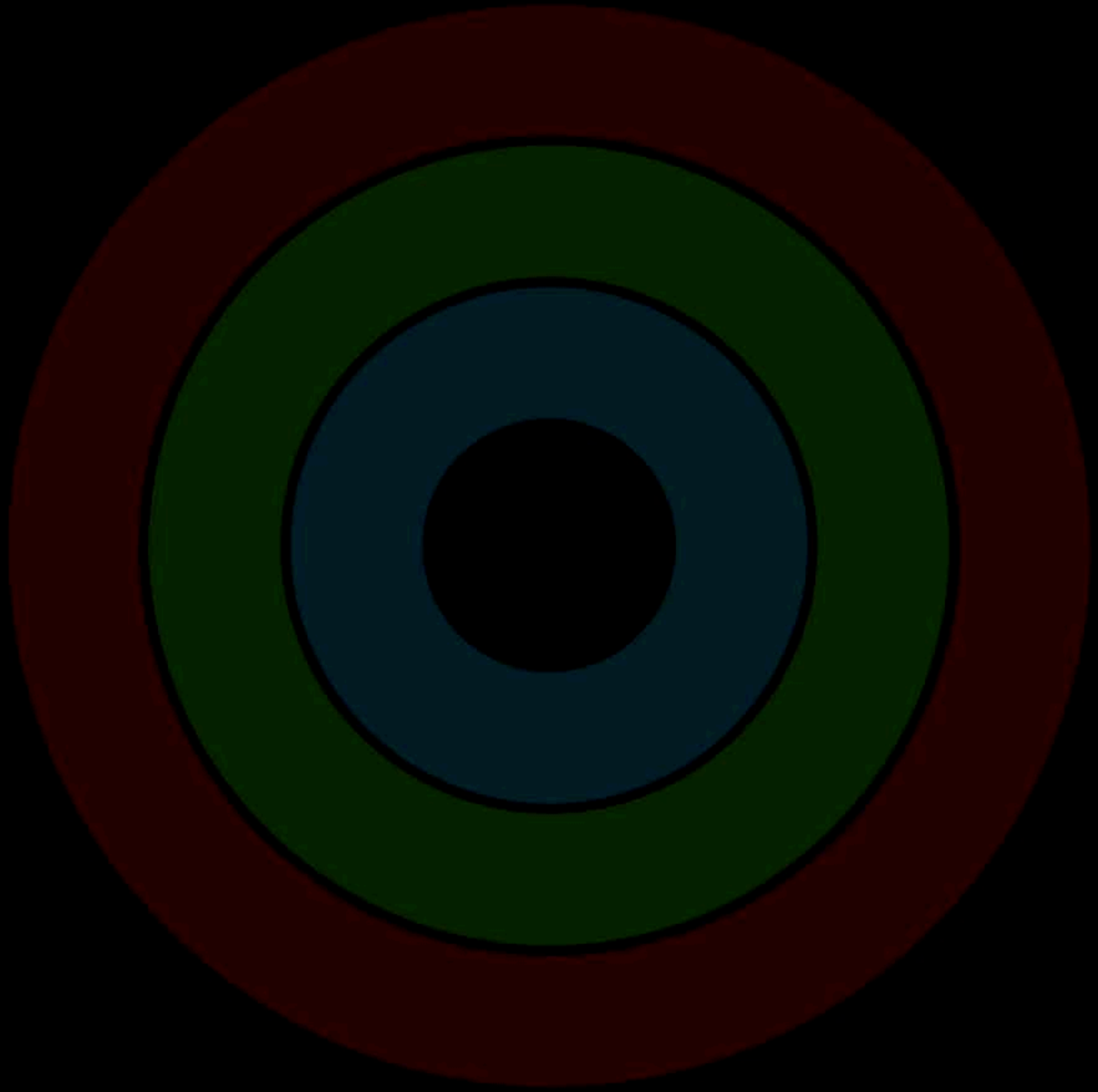
Activity Rings

HKActivityRingView and WKInterfaceActivityRing



Activity Rings

HKActivityRingView and WKInterfaceActivityRing



Activity Rings

Tips

Activity Rings

Tips

Rings look best on black background

Activity Rings

Tips

Rings look best on black background

Construct an HKActivitySummary
to represent another user's rings

Activity Rings

Tips

Rings look best on black background

Construct an HKActivitySummary
to represent another user's rings

Provide era, year, month, and day in your
DateComponents

Activity Rings

Tips

Rings look best on black background

Construct an HKActivitySummary to represent another user's rings

Provide era, year, month, and day in your DateComponents

Demo

Authorization and activity rings

Health Records

Jeff Kibuule
iOS Software Engineer

Health Records

Current experience

Health Records

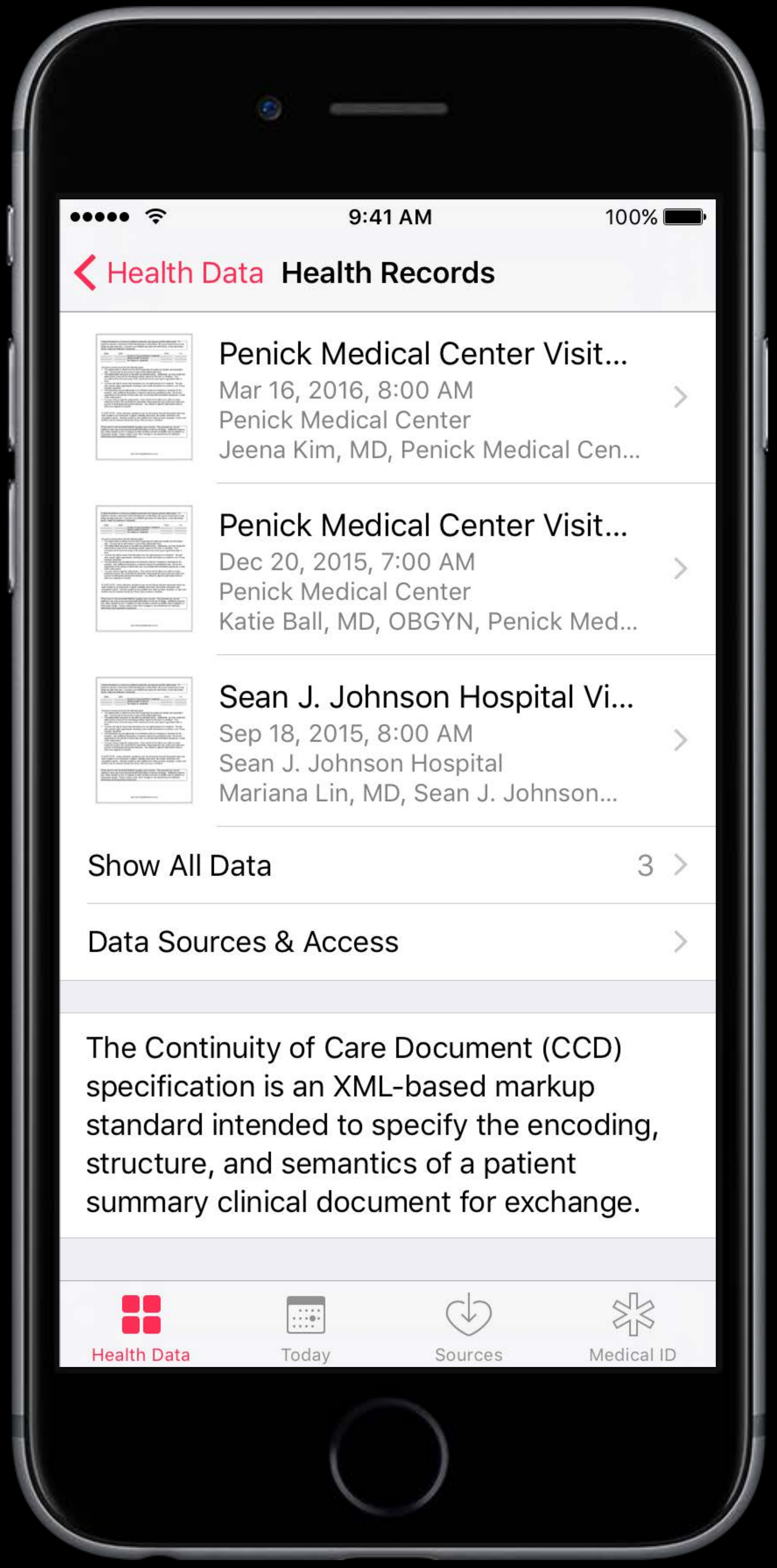
Current experience

Health Records

Current experience

Health Records

Current experience



9:41 AM 100%

Health Data **Health Records**

Penick Medical Center Visit...
Mar 16, 2016, 8:00 AM
Penick Medical Center
Jeena Kim, MD, Penick Medical Cen...

Penick Medical Center Visit...
Dec 20, 2015, 7:00 AM
Penick Medical Center
Katie Ball, MD, OBGYN, Penick Med...

Sean J. Johnson Hospital Vi...
Sep 18, 2015, 8:00 AM
Sean J. Johnson Hospital
Mariana Lin, MD, Sean J. Johnson...

Show All Data 3

Data Sources & Access

The Continuity of Care Document (CCD) specification is an XML-based markup standard intended to specify the encoding, structure, and semantics of a patient summary clinical document for exchange.

Health Records

Overview

NEW

Health Records

NEW

Overview

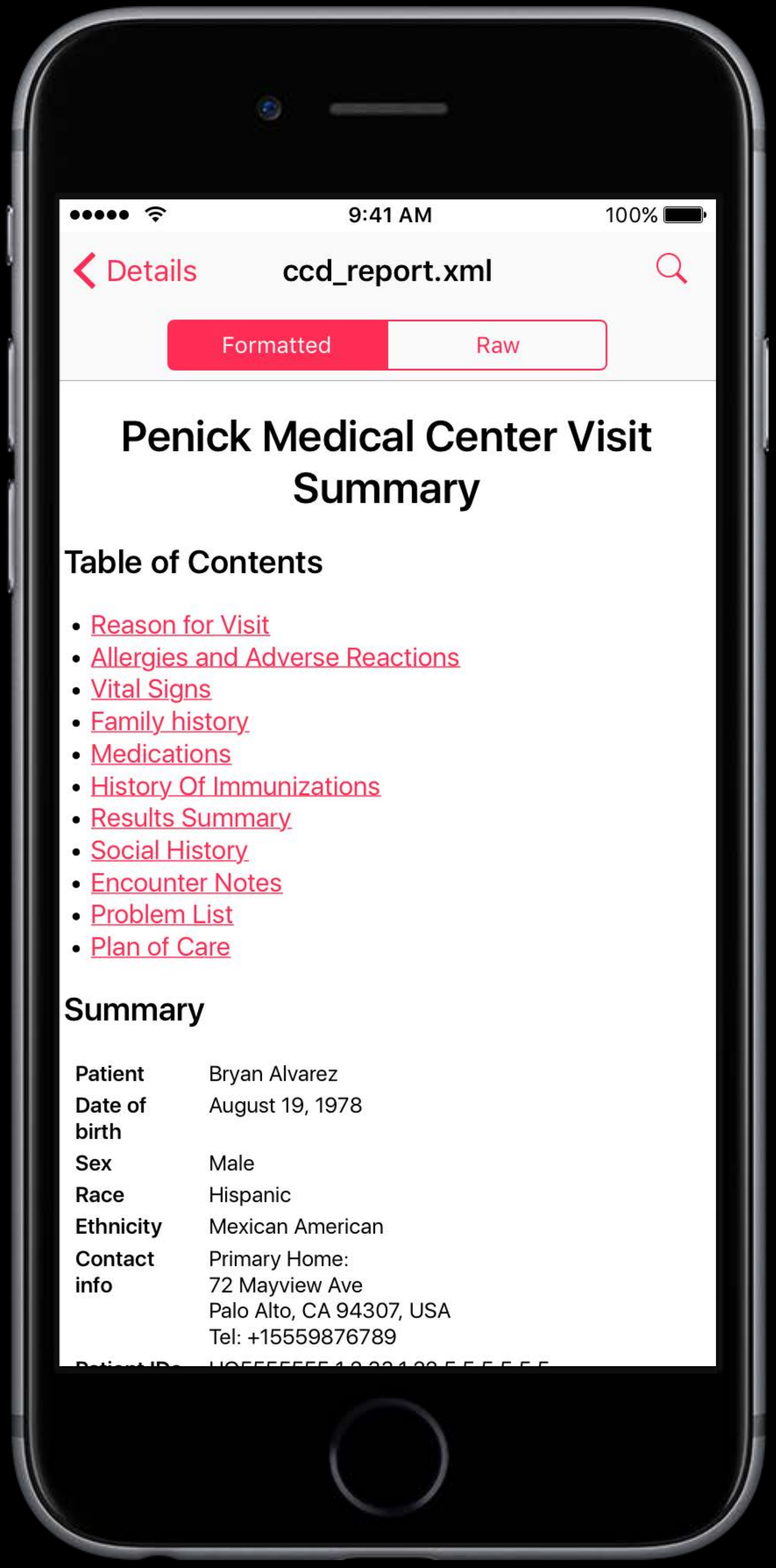
Represent different types of patient visits

Health Records

Overview

NEW

Represent different types of patient visits



Penick Medical Center Visit Summary

Table of Contents

- Reason for Visit
- Allergies and Adverse Reactions
- Vital Signs
- Family history
- Medications
- History Of Immunizations
- Results Summary
- Social History
- Encounter Notes
- Problem List
- Plan of Care

Summary

Patient	Bryan Alvarez
Date of birth	August 19, 1978
Sex	Male
Race	Hispanic
Ethnicity	Mexican American
Contact info	Primary Home: 72 Mayview Ave Palo Alto, CA 94307, USA Tel: +15559876789

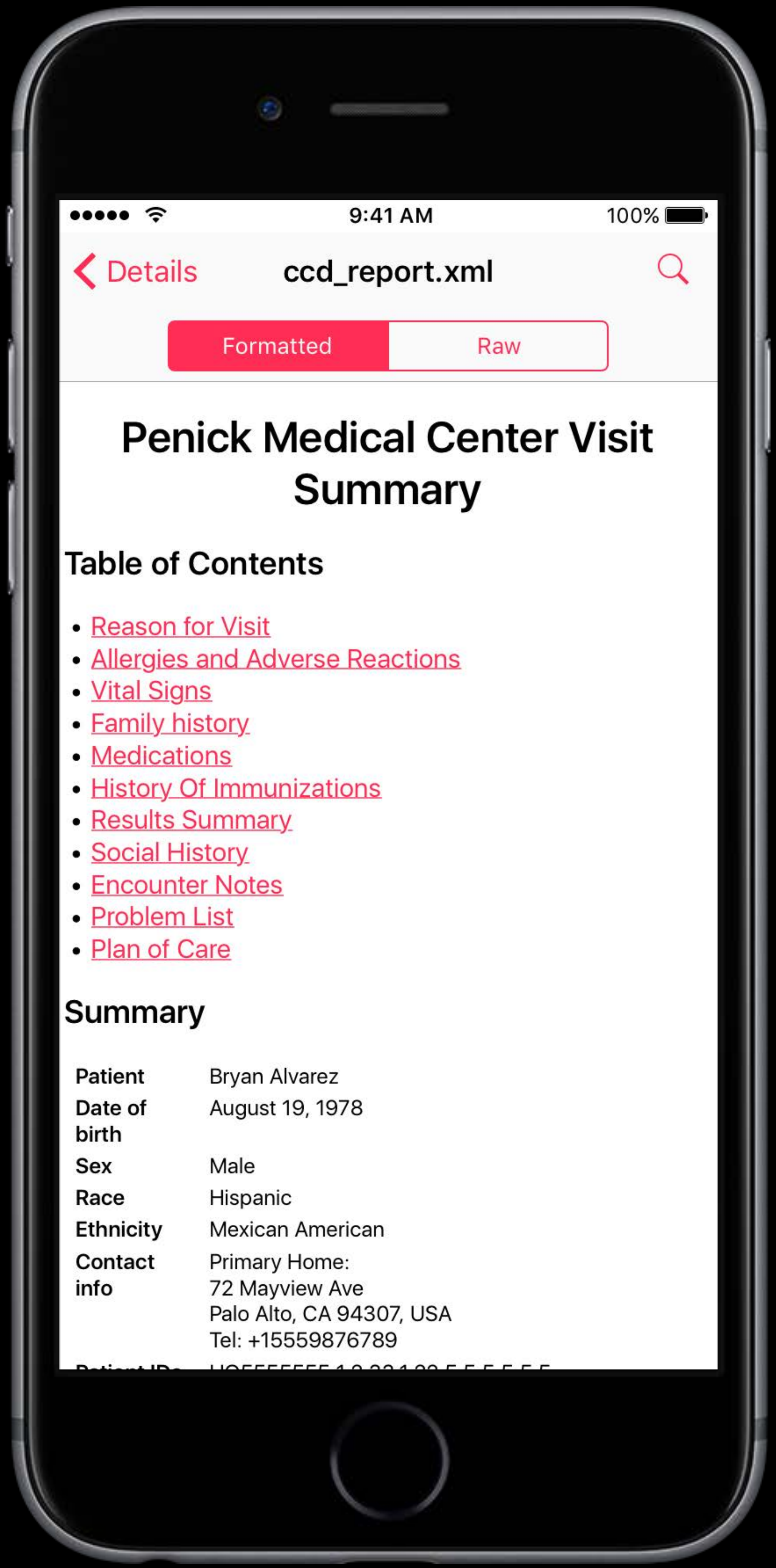
Health Records

Overview

NEW

Represent different types of patient visits

Support international HL-7 CDA standard



Penick Medical Center Visit Summary

Table of Contents

- Reason for Visit
- Allergies and Adverse Reactions
- Vital Signs
- Family history
- Medications
- History Of Immunizations
- Results Summary
- Social History
- Encounter Notes
- Problem List
- Plan of Care

Summary

Patient	Bryan Alvarez
Date of birth	August 19, 1978
Sex	Male
Race	Hispanic
Ethnicity	Mexican American
Contact info	Primary Home: 72 Mayview Ave Palo Alto, CA 94307, USA Tel: +15559876789

Health Records

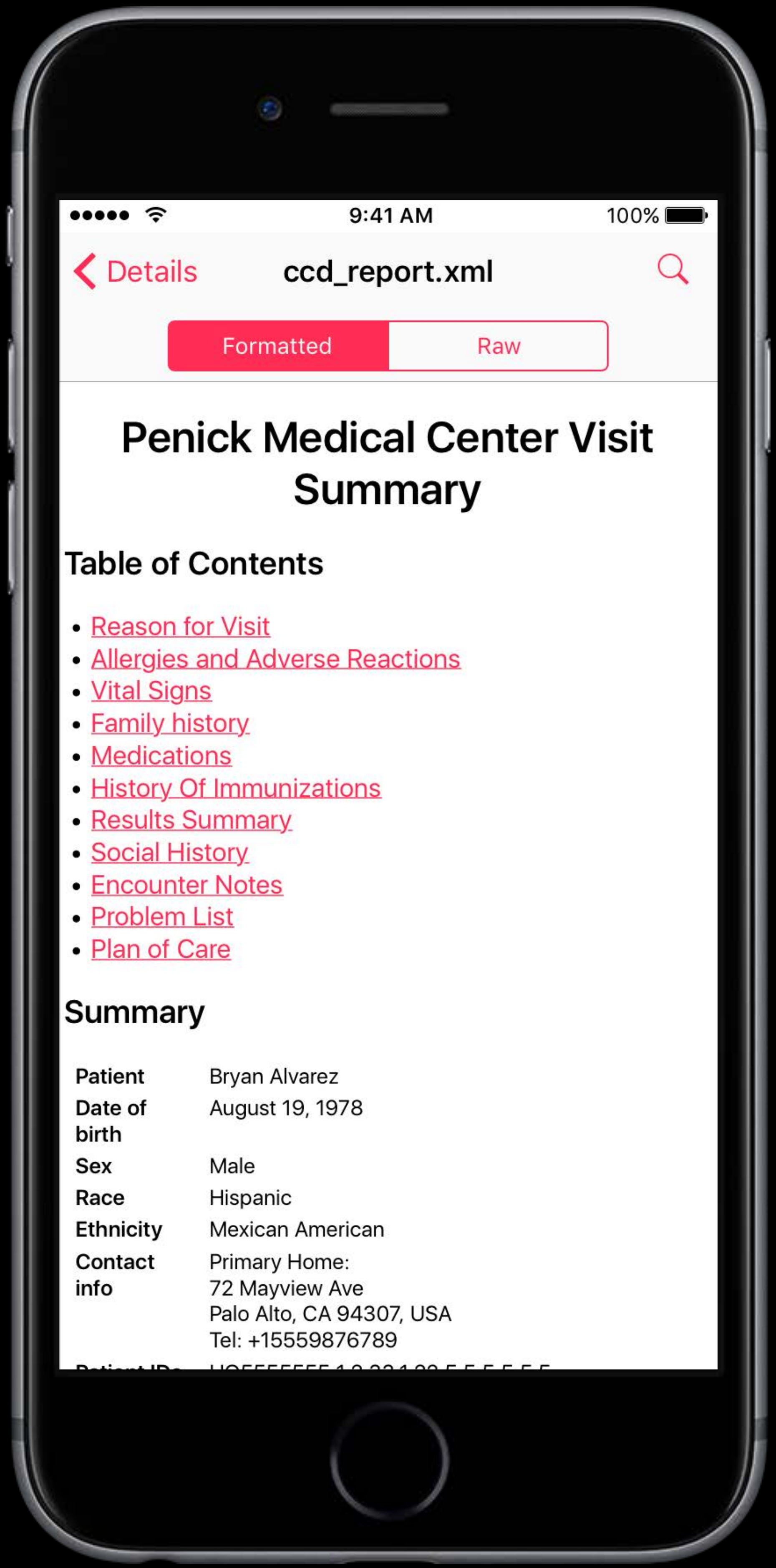
Overview

NEW

Represent different types of patient visits

Support international HL-7 CDA standard

Available through patient health care portals



Penick Medical Center Visit Summary

Table of Contents

- Reason for Visit
- Allergies and Adverse Reactions
- Vital Signs
- Family history
- Medications
- History Of Immunizations
- Results Summary
- Social History
- Encounter Notes
- Problem List
- Plan of Care

Summary

Patient	Bryan Alvarez
Date of birth	August 19, 1978
Sex	Male
Race	Hispanic
Ethnicity	Mexican American
Contact info	Primary Home: 72 Mayview Ave Palo Alto, CA 94307, USA Tel: +15559876789

Health Records

Overview

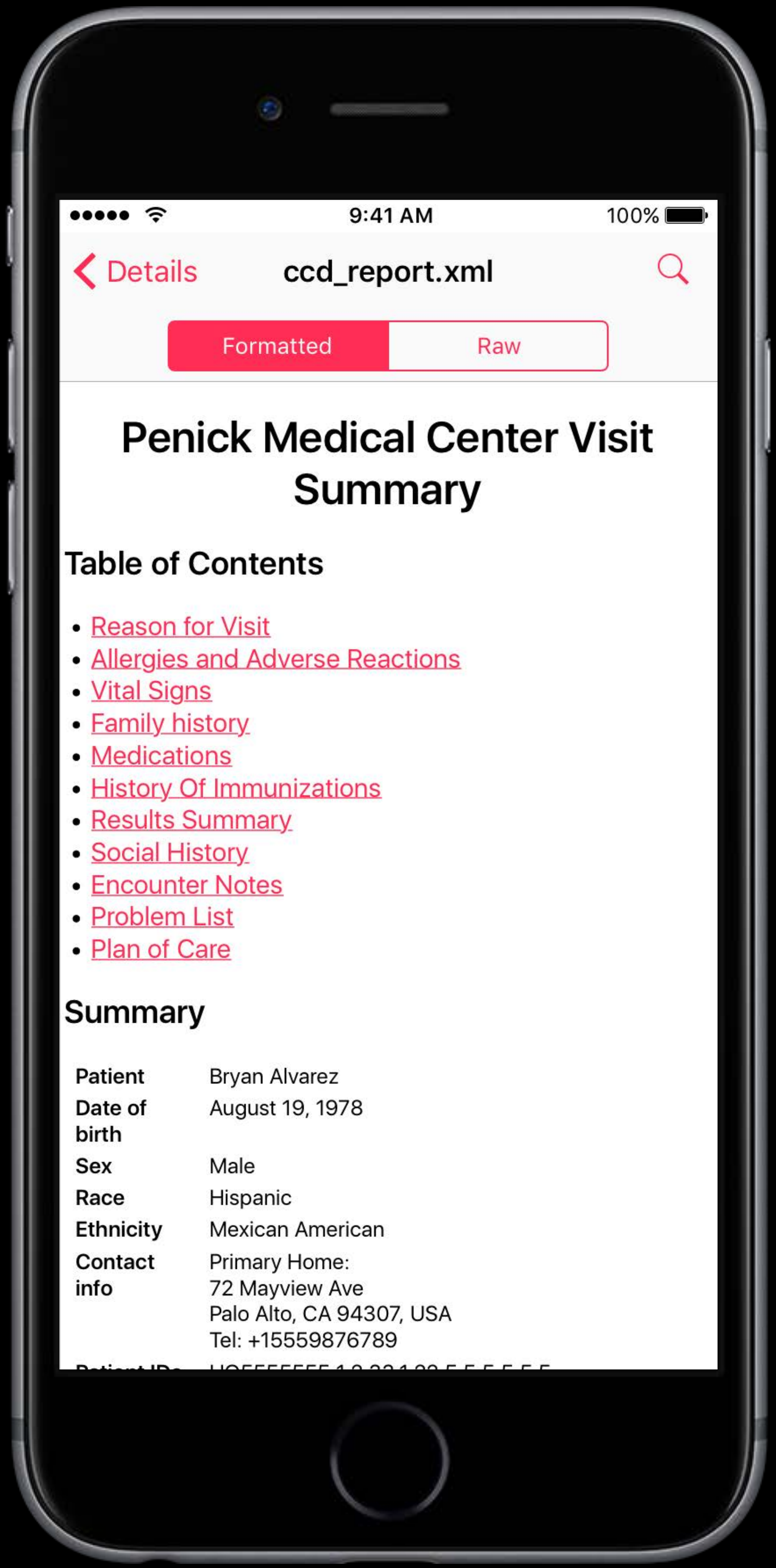
NEW

Represent different types of patient visits

Support international HL-7 CDA standard

Available through patient health care portals

Imported via Safari, Mail, and your apps



Penick Medical Center Visit Summary

Table of Contents

Summary

Patient	Bryan Alvarez
Date of birth	August 19, 1978
Sex	Male
Race	Hispanic
Ethnicity	Mexican American
Contact info	Primary Home: 72 Mayview Ave Palo Alto, CA 94307, USA Tel: +15559876789

Health Records

Overview

NEW

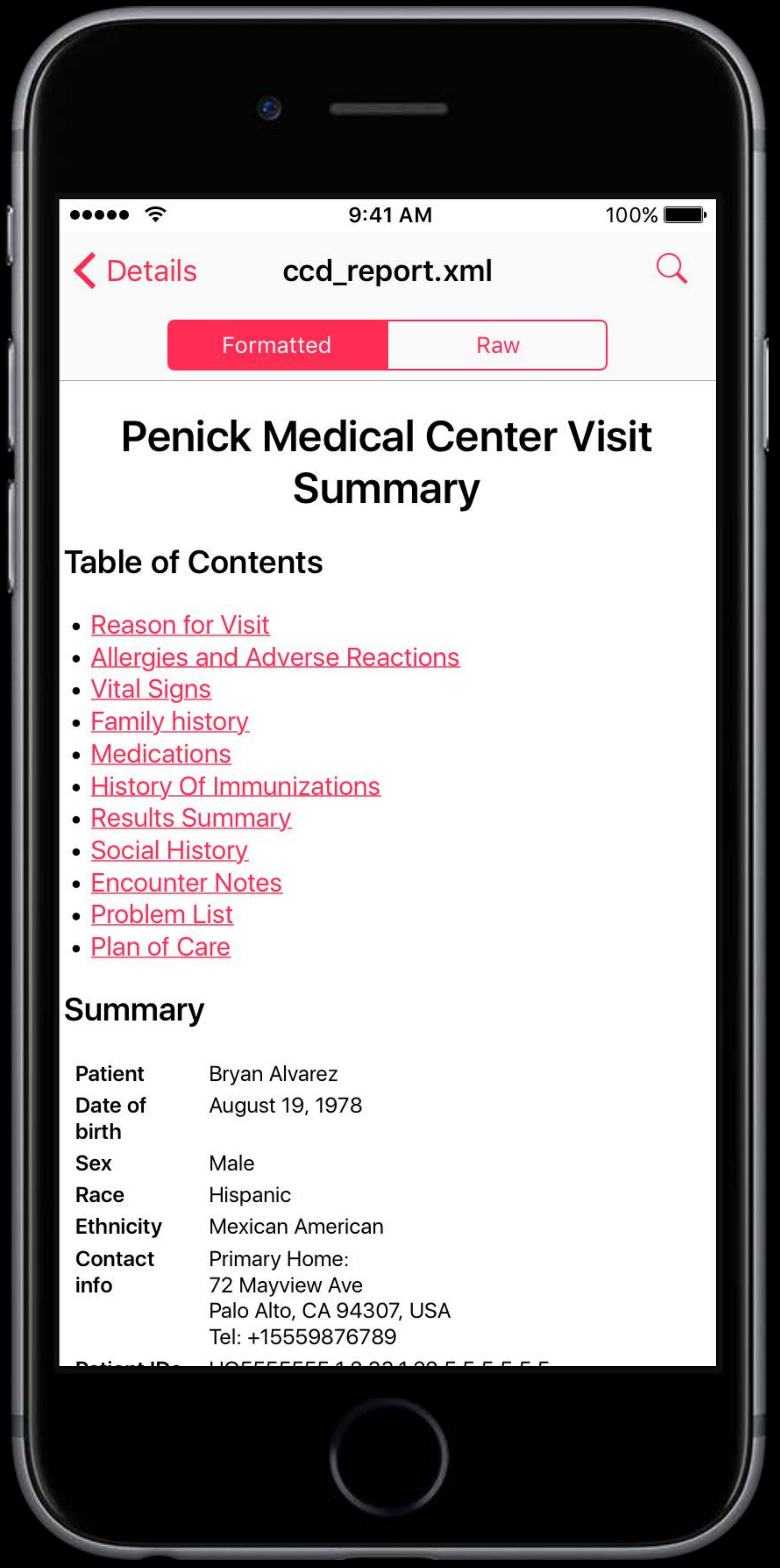
Represent different types of patient visits

Support international HL-7 CDA standard

Available through patient health care portals

Imported via Safari, Mail, and your apps

Stored securely



Penick Medical Center Visit Summary

Table of Contents

- Reason for Visit
- Allergies and Adverse Reactions
- Vital Signs
- Family history
- Medications
- History Of Immunizations
- Results Summary
- Social History
- Encounter Notes
- Problem List
- Plan of Care

Summary

Patient	Bryan Alvarez
Date of birth	August 19, 1978
Sex	Male
Race	Hispanic
Ethnicity	Mexican American
Contact info	Primary Home: 72 Mayview Ave Palo Alto, CA 94307, USA Tel: +15559876789

Health Records

Permissions

Health Records

Permissions

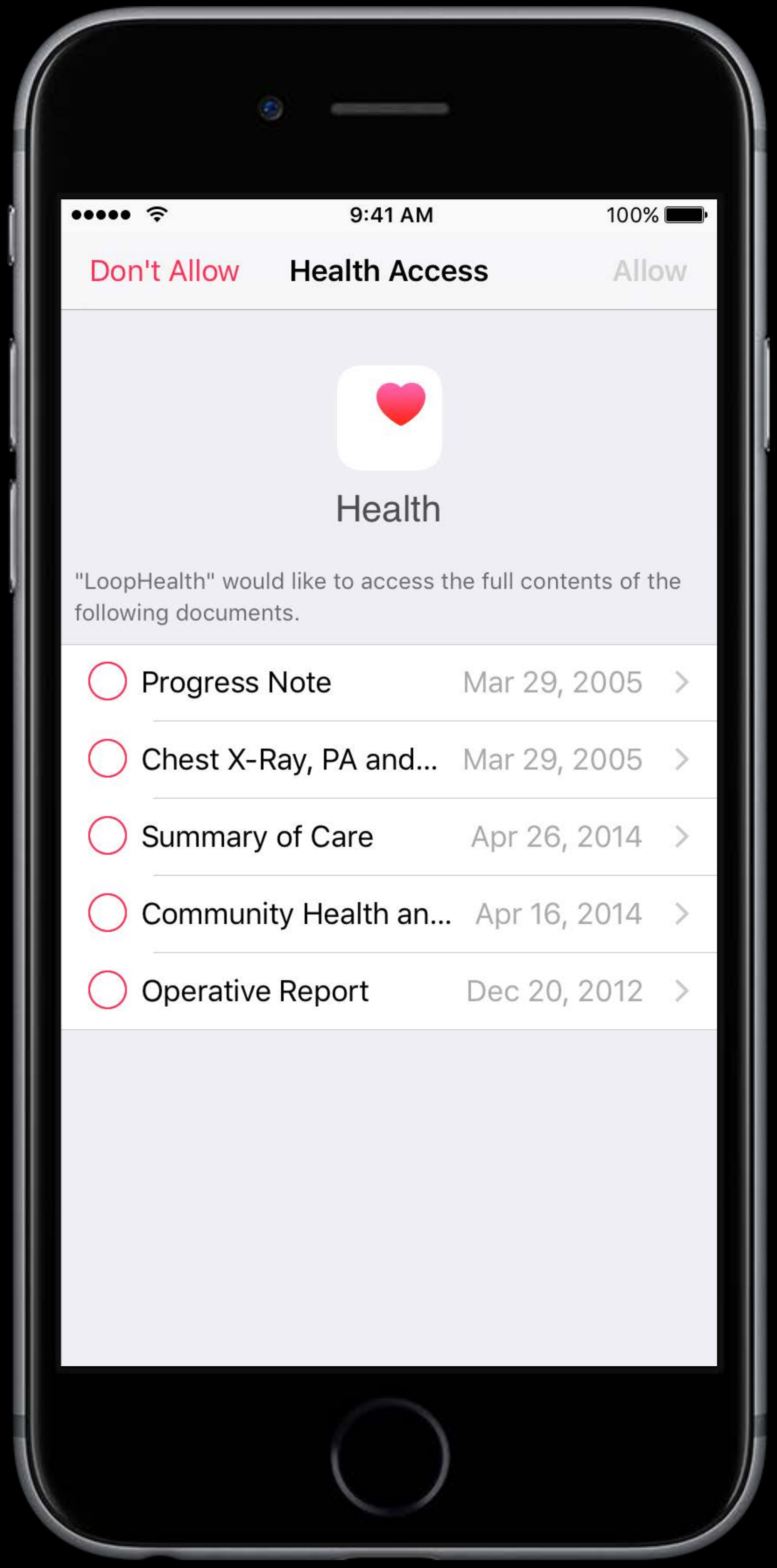
Access granted on a per document basis

Health Records

Permissions

Access granted on a per document basis

UI presented to grant access



"LoopHealth" would like to access the full contents of the following documents.

○ Progress Note	Mar 29, 2005
○ Chest X-Ray, PA and...	Mar 29, 2005
○ Summary of Care	Apr 26, 2014
○ Community Health an...	Apr 16, 2014
○ Operative Report	Dec 20, 2012

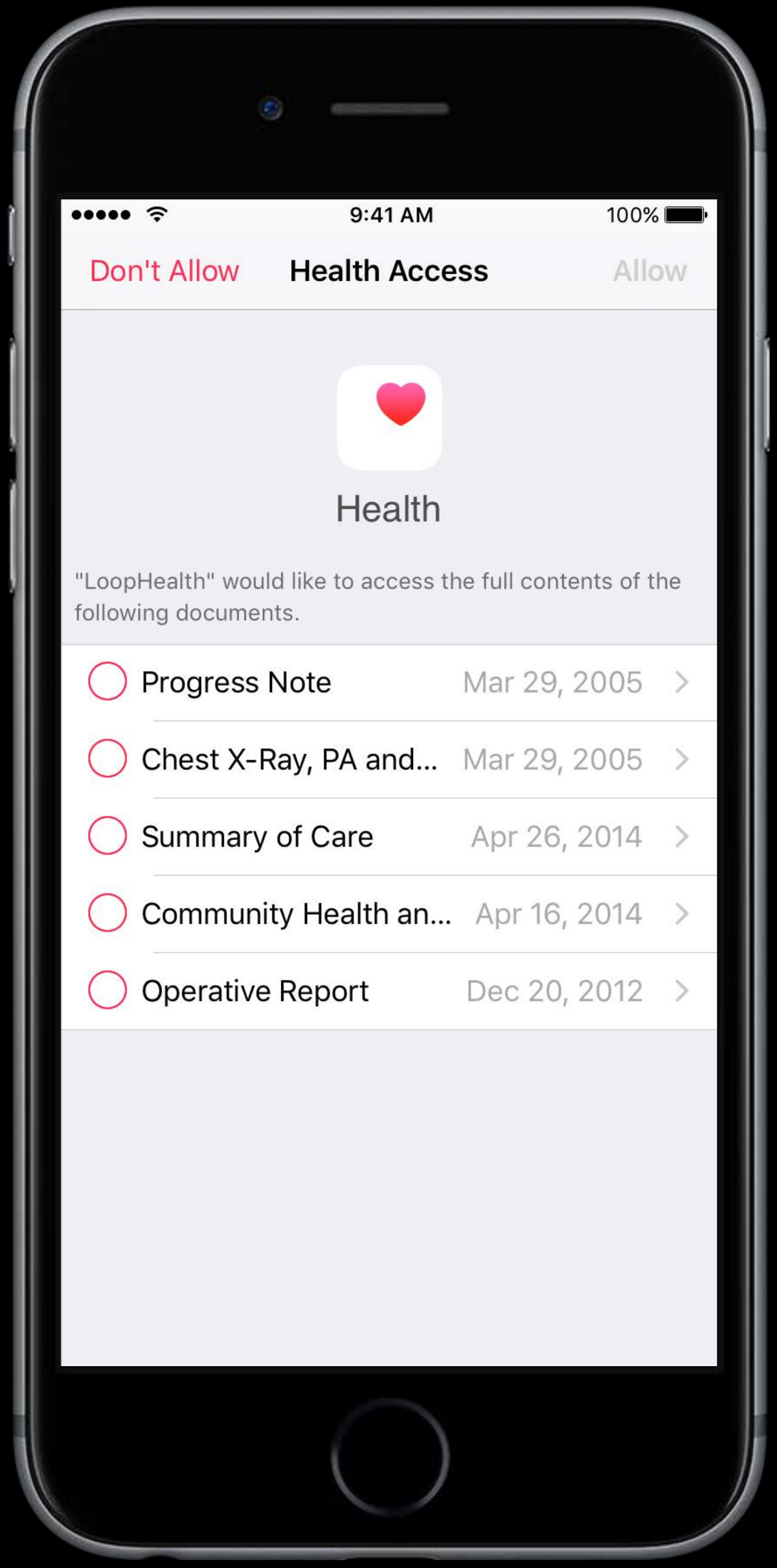
Health Records

Permissions

Access granted on a per document basis

UI presented to grant access

Queries will…



"LoopHealth" would like to access the full contents of the following documents.

Progress Note	Mar 29, 2005
Chest X-Ray, PA and...	Mar 29, 2005
Summary of Care	Apr 26, 2014
Community Health an...	Apr 16, 2014
Operative Report	Dec 20, 2012

Health Records

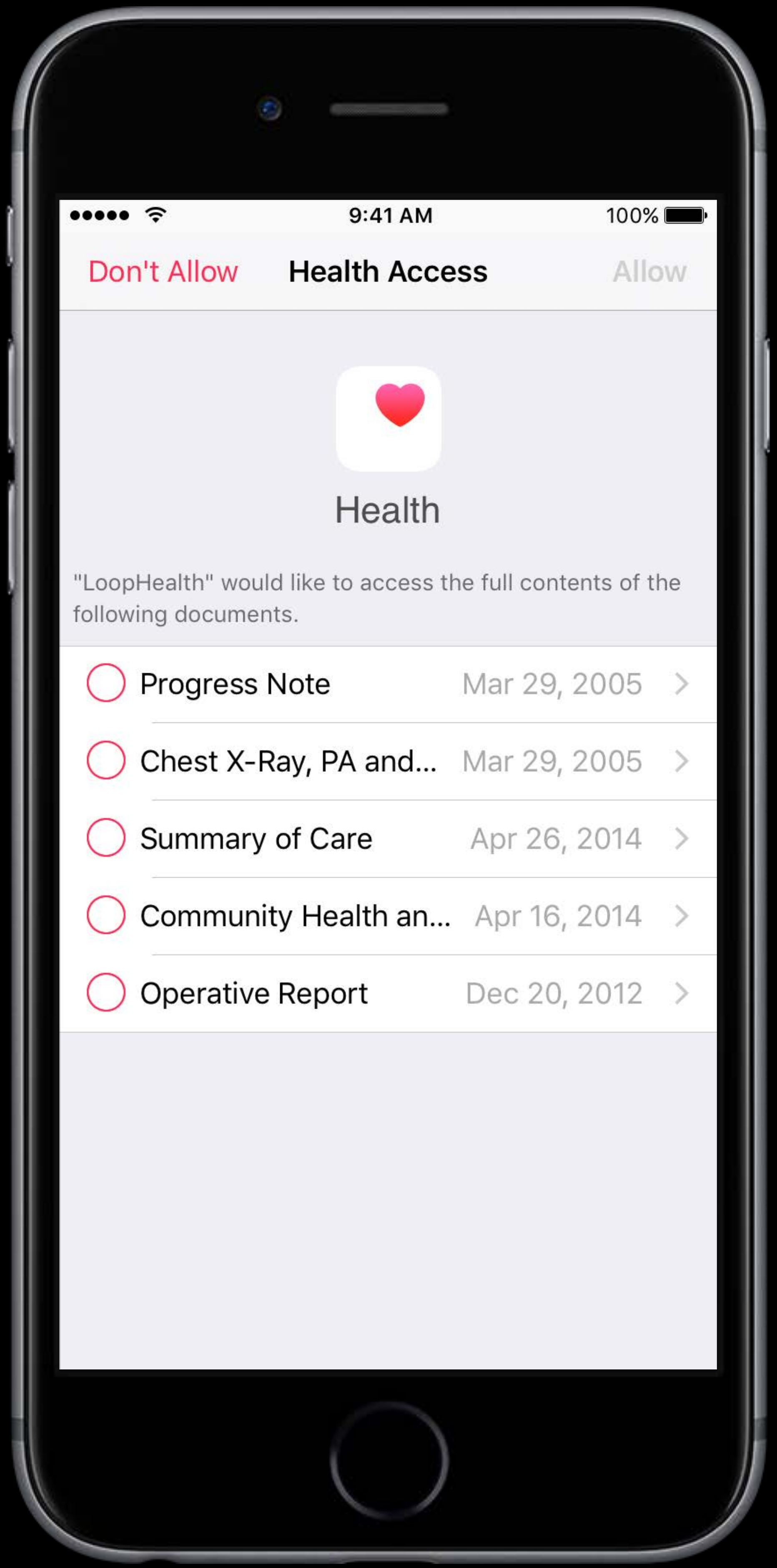
Permissions

Access granted on a per document basis

UI presented to grant access

Queries will…

- Prompt UI if new documents available



"LoopHealth" would like to access the full contents of the following documents.

Progress Note	Mar 29, 2005
Chest X-Ray, PA and...	Mar 29, 2005
Summary of Care	Apr 26, 2014
Community Health an...	Apr 16, 2014
Operative Report	Dec 20, 2012

Health Records

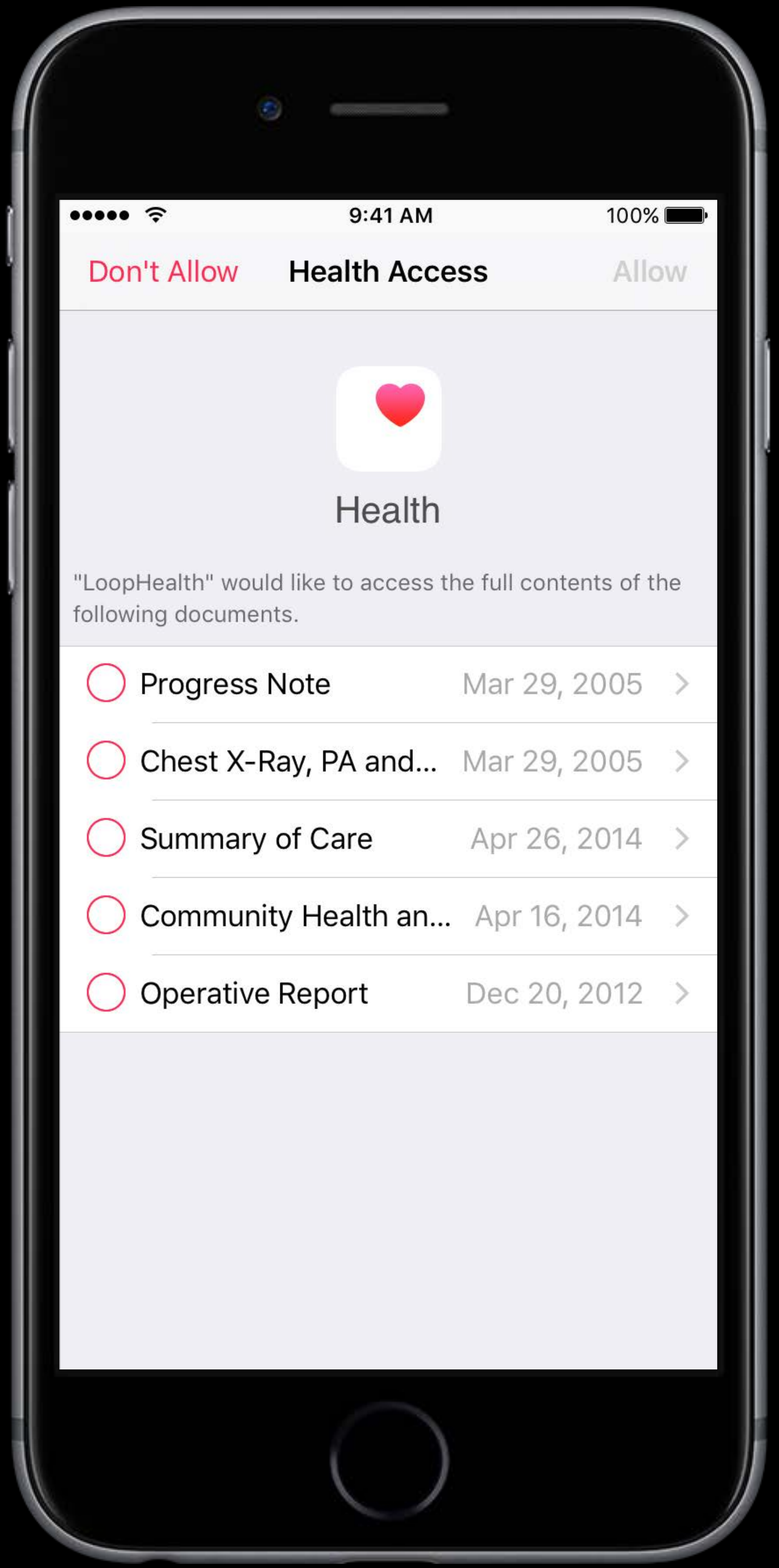
Permissions

Access granted on a per document basis

UI presented to grant access

Queries will…

- Prompt UI if new documents available
- Return immediately if no new documents



"LoopHealth" would like to access the full contents of the following documents.

Progress Note	Mar 29, 2005
Chest X-Ray, PA and...	Mar 29, 2005
Summary of Care	Apr 26, 2014
Community Health an...	Apr 16, 2014
Operative Report	Dec 20, 2012

Health Records

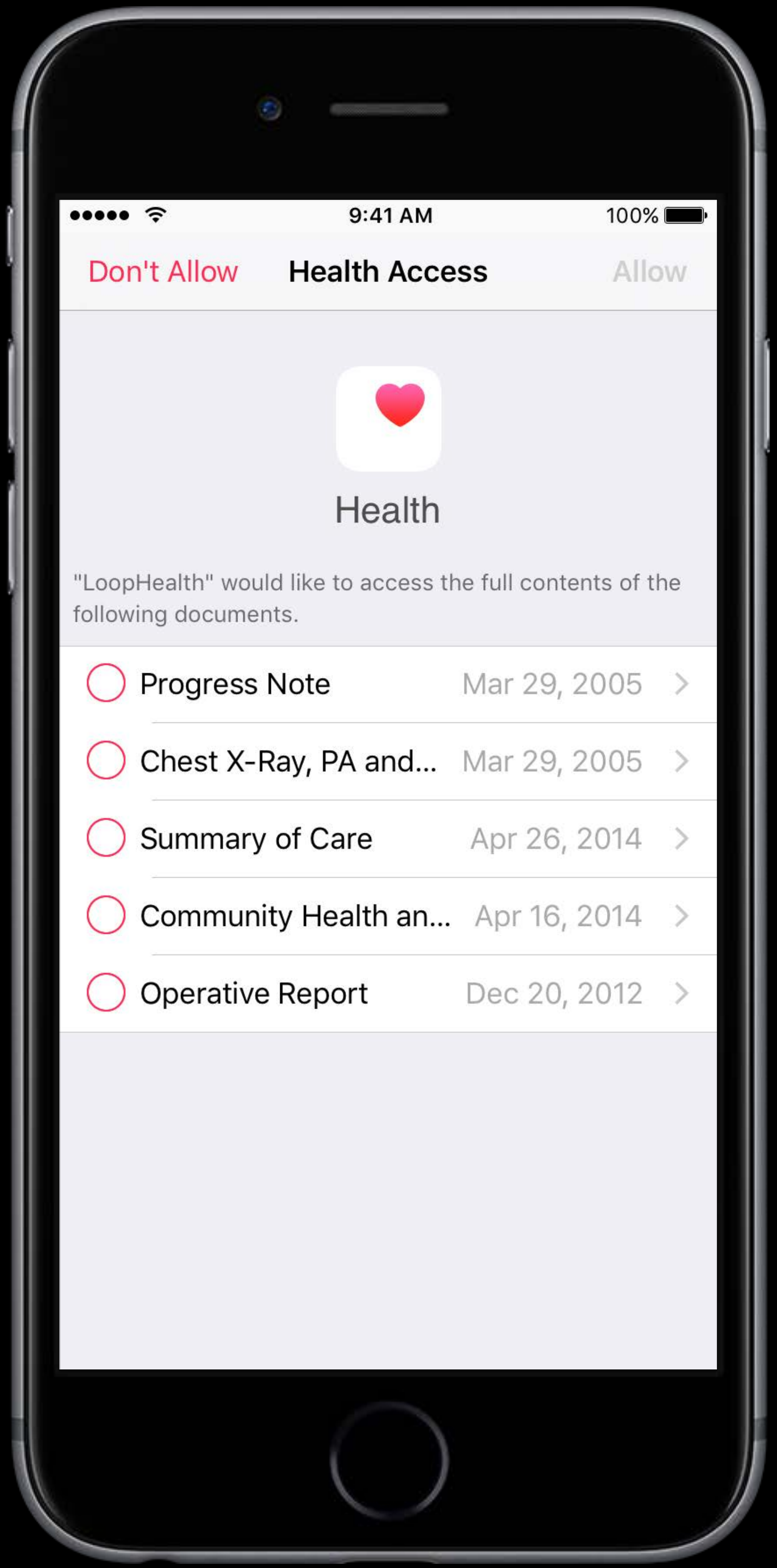
Permissions

Access granted on a per document basis

UI presented to grant access

Queries will…

- Prompt UI if new documents available
- Return immediately if no new documents
- **Never** prompt in background



"LoopHealth" would like to access the full contents of the following documents.

Progress Note	Mar 29, 2005
Chest X-Ray, PA and...	Mar 29, 2005
Summary of Care	Apr 26, 2014
Community Health an...	Apr 16, 2014
Operative Report	Dec 20, 2012

Health Records

Creating

Health Records

Creating

Save raw XML document as `Data` into `HKCDADocumentSample` type

Health Records

Creating

Save raw XML document as `Data` into `HKCDADocumentSample` type

Validated on sample creation

Health Records

Creating

Save raw XML document as `Data` into `HKCDADocumentSample` type

Validated on sample creation

Title, patient, custodian, and author names extracted automatically

```
// Creating a Health Document Using HKCDADocumentSample

let today = Date()

let documentData: Data = ... // XML from health organization server

do {
    let cdaSample = try HKCDADocumentSample.init(data: documentData, start: today, end:
        today, metadata: nil)

    healthStore.save(cdaSample) { success, error in
        // Handle saving error here...
    }
} catch {
    // Handle validation error creating sample here...
}
```

```swift
// Creating a Health Document Using HKCDADocumentSample

let today = Date()

let documentData: Data = ... // XML from health organization server

do {
    let cdaSample = try HKCDADocumentSample.init(data: documentData, start: today, end:
       today, metadata: nil)

    healthStore.save(cdaSample) { success, error in
       // Handle saving error here...
    }
} catch {
    // Handle validation error creating sample here...
}
```

```swift
// Creating a Health Document Using HKCDADocumentSample

let today = Date()

let documentData: Data = ... // XML from health organization server

do {
    let cdaSample = try HKCDADocumentSample.init(data: documentData, start: today, end:
       today, metadata: nil)

    healthStore.save(cdaSample) { success, error in
       // Handle saving error here...
    }
} catch {
    // Handle validation error creating sample here...
}
```

```swift
// Creating a Health Document Using HKCDADocumentSample

let today = Date()

let documentData: Data = ... // XML from health organization server

do {
    let cdaSample = try HKCDADocumentSample.init(data: documentData, start: today, end:
       today, metadata: nil)

    healthStore.save(cdaSample) { success, error in
       // Handle saving error here...
    }
} catch {
    // Handle validation error creating sample here...
}
```

Health Records

Querying

Health Records

Querying

Existing query objects continue to work

Health Records

Querying

Existing query objects continue to work

Need to use `HKDocumentQuery` to fetch raw XML

Health Records

Querying

Existing query objects continue to work

Need to use `HKDocumentQuery` to fetch raw XML

Predicate support for searching based on extracted fields

Health Records

Querying

Existing query objects continue to work

Need to use `HKDocumentQuery` to fetch raw XML

Predicate support for searching based on extracted fields

Document updates are considered new documents

```swift
// Querying for Health Documents Using HKDocumentQuery

guard let documentType = HKObjectType.documentType(forIdentifier: .CDA) else { return }

let cdaQuery = HKDocumentQuery(documentType: documentType, predicate: nil, limit:
   HKObjectQueryNoLimit, sortDescriptors: nil, includeDocumentData: false) { query, samples,
   done, error in
      // Handle HKCDADocumentSamples here...
}

healthStore.execute(cdaQuery)
```

```swift
// Querying for Health Documents Using HKDocumentQuery

guard let documentType = HKObjectType.documentType(forIdentifier: .CDA) else { return }

let cdaQuery = HKDocumentQuery(documentType: documentType, predicate: nil, limit:
   HKObjectQueryNoLimit, sortDescriptors: nil, includeDocumentData: false) { query, samples,
   done, error in
      // Handle HKCDADocumentSamples here...
}

healthStore.execute(cdaQuery)
```

```swift
// Querying for Health Documents Using HKDocumentQuery

guard let documentType = HKObjectType.documentType(forIdentifier: .CDA) else { return }

let cdaQuery = HKDocumentQuery(documentType: documentType, predicate: nil, limit:
   HKObjectQueryNoLimit, sortDescriptors: nil, includeDocumentData: false) { query, samples,
   done, error in
      // Handle HKCDADocumentSamples here...
}

healthStore.execute(cdaQuery)
```

```swift
// Querying for Health Documents Using HKDocumentQuery

guard let documentType = HKObjectType.documentType(forIdentifier: .CDA) else { return }

let cdaQuery = HKDocumentQuery(documentType: documentType, predicate: nil, limit:
   HKObjectQueryNoLimit, sortDescriptors: nil, includeDocumentData: false) { query, samples,
   done, error in
      // Handle HKCDADocumentSamples here...
}

healthStore.execute(cdaQuery)
```

```swift
// Querying for Health Documents Using HKDocumentQuery

guard let documentType = HKObjectType.documentType(forIdentifier: .CDA) else { return }

let cdaQuery = HKDocumentQuery(documentType: documentType, predicate: nil, limit:
   HKObjectQueryNoLimit, sortDescriptors: nil, includeDocumentData: false) { query, samples,
   done, error in
      // Handle HKCDADocumentSamples here...
}

healthStore.execute(cdaQuery)
```

Health Records

Best practices

Health Records

Best practices

Check validation errors

Health Records

Best practices

Check validation errors

Verify with Health app

Health Records

Best practices

Check validation errors

Verify with Health app

Request raw XML document only when needed

HL-7 CDA Standard

http://www.hl7.org/implement/standards/product_brief.cfm?product_id=7

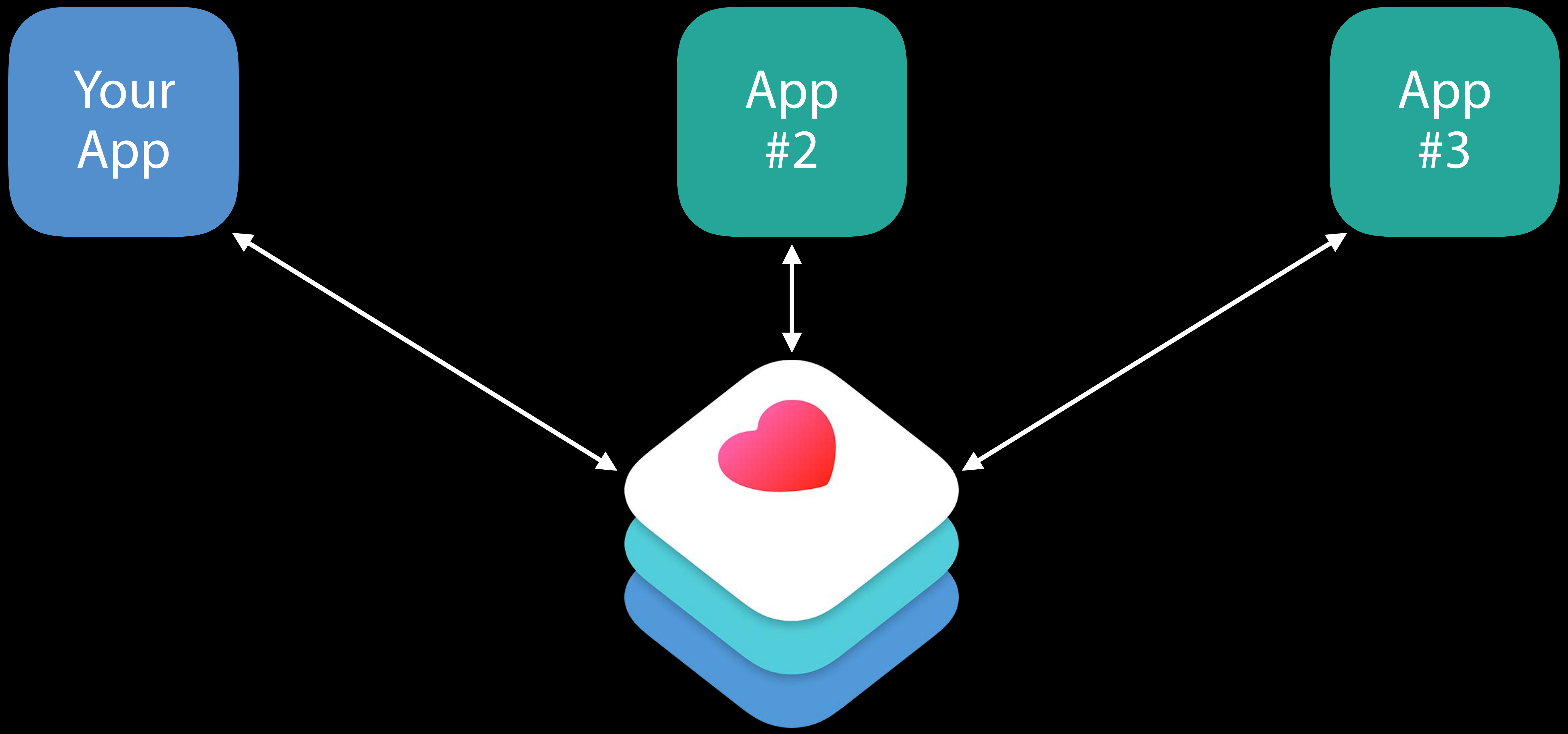
Handling Data

Handling Data

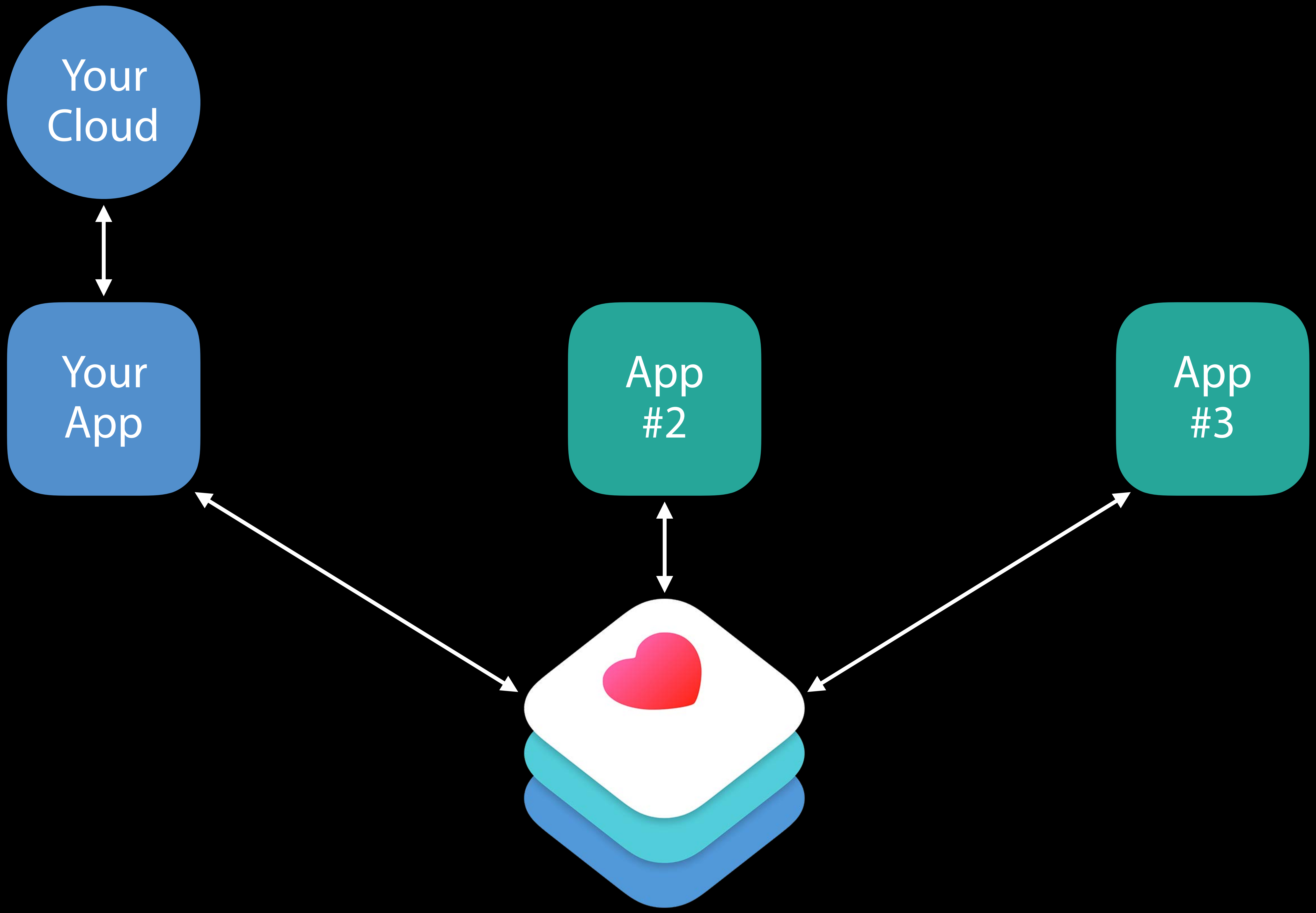
Handling Data

Your App
App #2
App #3

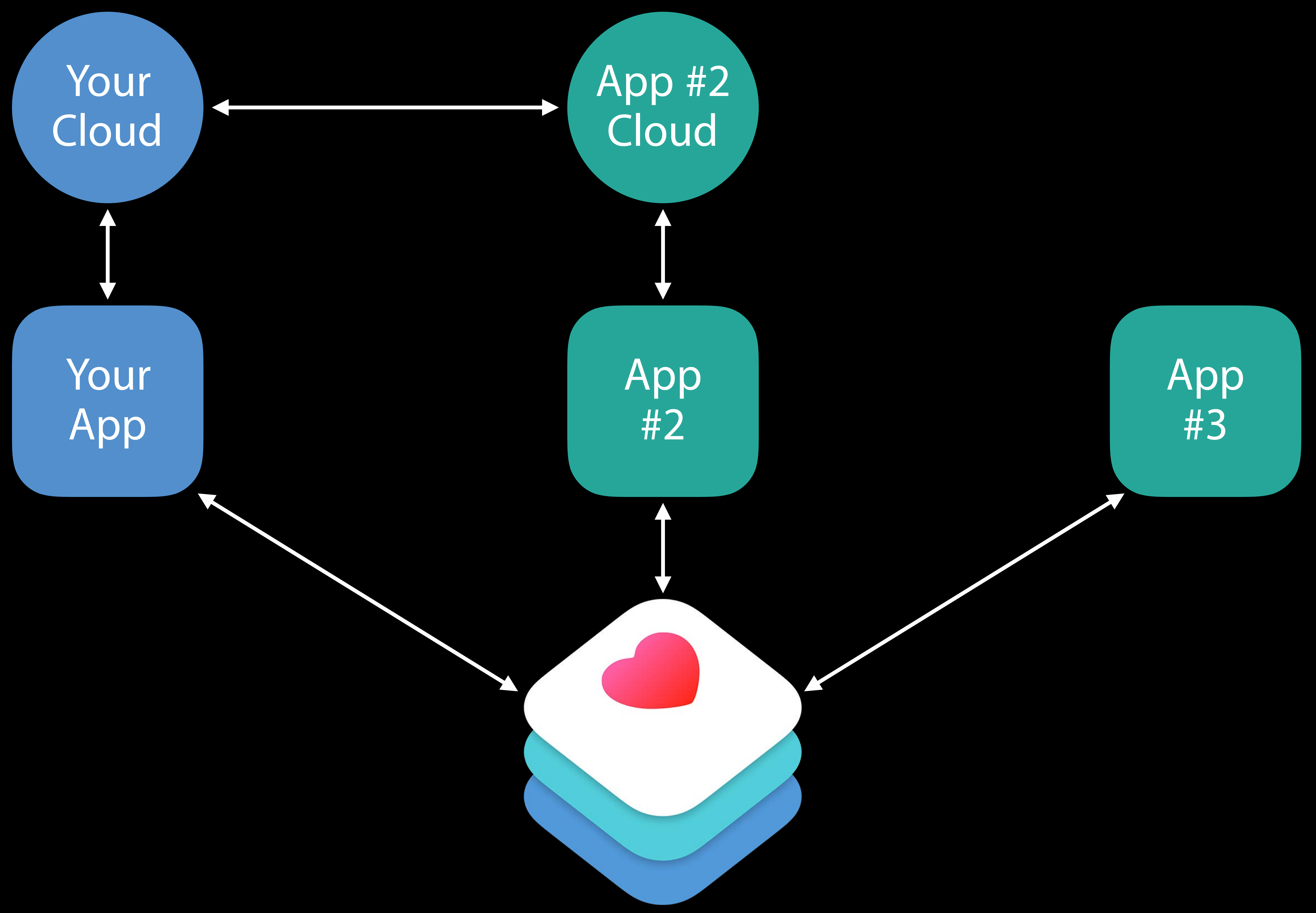
Handling Data



Handling Data



Handling Data



Handling Data

Handling Data

Syncing data

Handling Data

Syncing data

Tracking changed data

Handling Data

Syncing data

Tracking changed data

Migrating data

Handling Data

Syncing data

Handling Data

Syncing data

Use `HKAnchoredObjectQuery`

Handling Data

Syncing data

Use `HKAnchoredObjectQuery`

Anchors let you pick up where you last left off

Handling Data

Syncing data

Use `HKAnchoredObjectQuery`

Anchors let you pick up where you last left off

One query per sample type

Handling Data

Syncing data

Use `HKAnchoredObjectQuery`

Anchors let you pick up where you last left off

One query per sample type

Use an update handler

Handling Data

Background updates

Handling Data

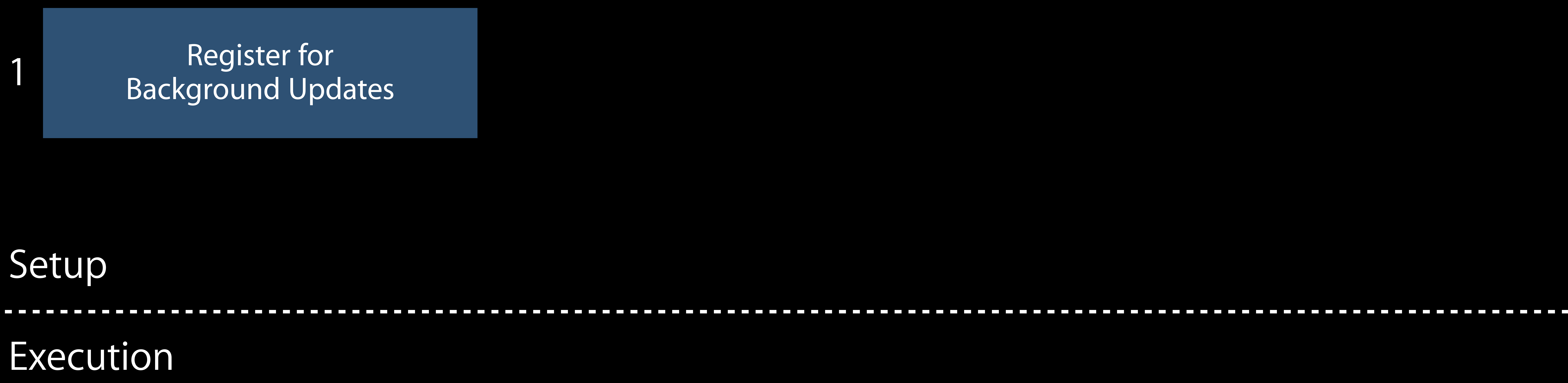
Background updates

Setup

Execution

Handling Data

Background updates



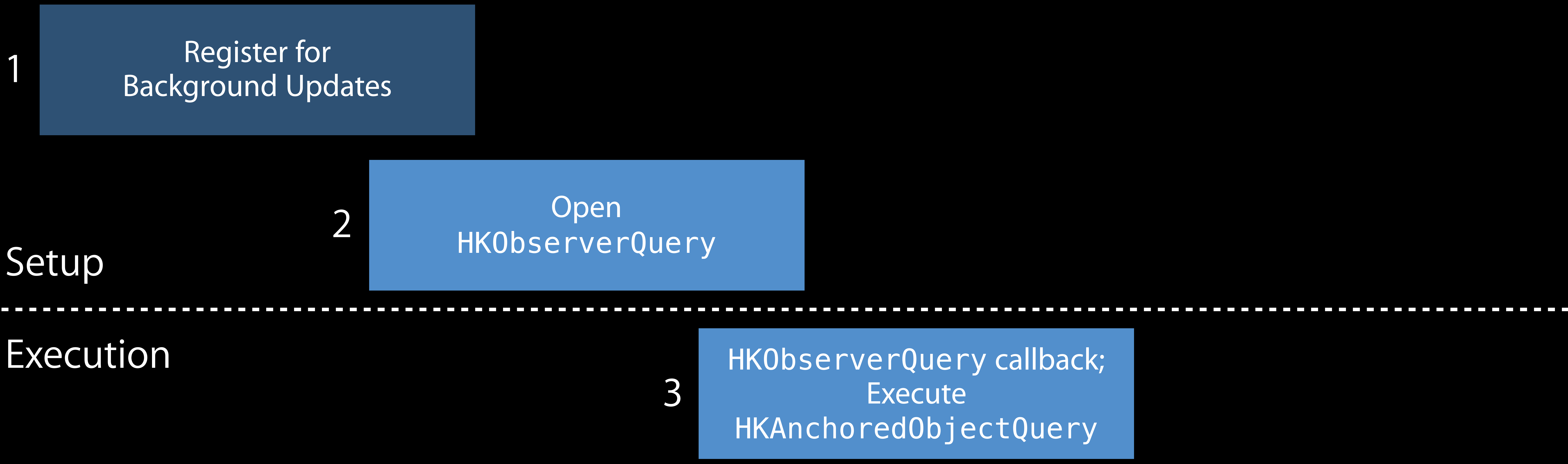
Handling Data

Background updates

1
Register for Background Updates
2
Open HKObserverQuery
Setup
Execution

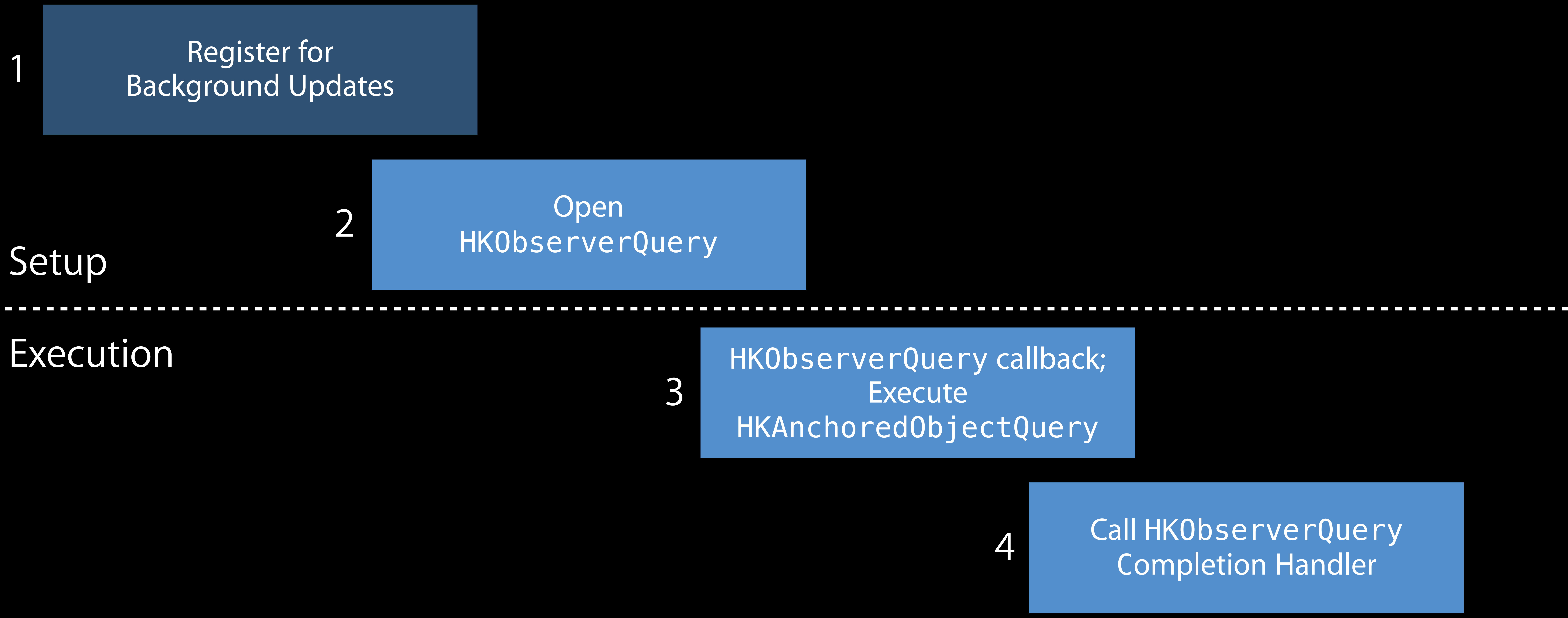
Handling Data

Background updates


1
Register for Background Updates
2
Open HKObserverQuery
Setup
Execution
3
HKObserverQuery callback; Execute HKAnchoredObjectQuery

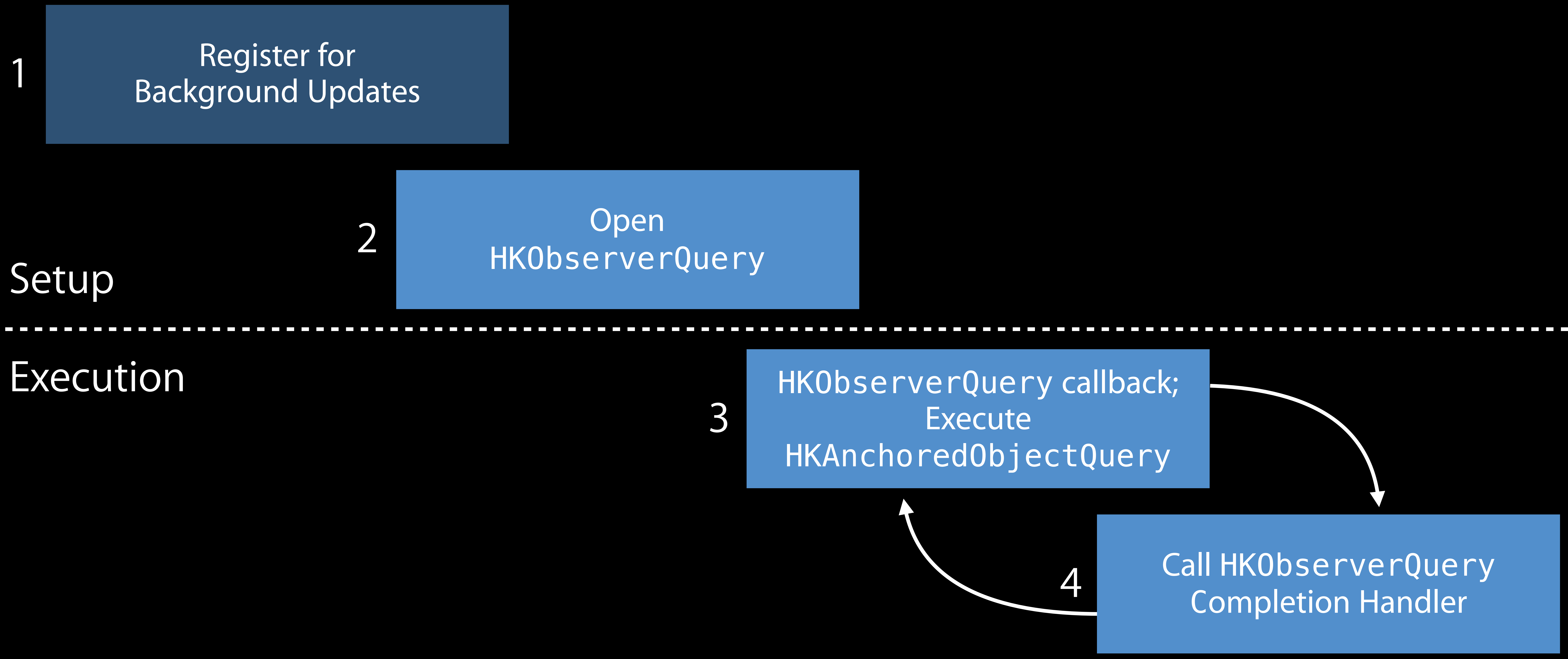
Handling Data

Background updates


1
Register for Background Updates
2
Open HKObserverQuery
Setup
Execution
3
HKObserverQuery callback; Execute HKAnchoredObjectQuery
4
Call HKObserverQuery Completion Handler

Handling Data

Background updates


1
Register for Background Updates
2
Open HKObserverQuery
Setup
Execution
3
HKObserverQuery callback; Execute HKAnchoredObjectQuery
4
Call HKObserverQuery Completion Handler

```swift
// Syncing Data
// 1. Register for background updates

func application(_ application: UIApplication, didFinishLaunchingWithOptions launchOptions:
    [NSObject: AnyObject]?) -> Bool {

    guard let stepsType = HKObjectType.quantityType(forIdentifier: .stepCount) else {
        return true
    }

    healthStore.enableBackgroundDelivery(for: stepsType, frequency: .daily) {
        success, error in
        // Handle enabling background delivery error here...
    }

    return true
}
```

```swift
// Syncing Data
// 1. Register for background updates

func application(_ application: UIApplication, didFinishLaunchingWithOptions launchOptions:
    [NSObject: AnyObject]?) -> Bool {

    guard let stepsType = HKObjectType.quantityType(forIdentifier: .stepCount) else {
        return true
    }

    healthStore.enableBackgroundDelivery(for: stepsType, frequency: .daily) {
        success, error in
        // Handle enabling background delivery error here...
    }

    return true
}
```

```swift
// Syncing Data
// 1. Register for background updates

func application(_ application: UIApplication, didFinishLaunchingWithOptions launchOptions:
    [NSObject: AnyObject]?) -> Bool {

    guard let stepsType = HKObjectType.quantityType(forIdentifier: .stepCount) else {
        return true
    }

    healthStore.enableBackgroundDelivery(for: stepsType, frequency: .daily) {
        success, error in
        // Handle enabling background delivery error here...
    }

    return true
}
```

```swift
// Syncing Data
// 1. Register for background updates

func application(_ application: UIApplication, didFinishLaunchingWithOptions launchOptions:
    [NSObject: AnyObject]?) -> Bool {

    guard let stepsType = HKObjectType.quantityType(forIdentifier: .stepCount) else {
        return true
    }

    healthStore.enableBackgroundDelivery(for: stepsType, frequency: .daily) {
        success, error in
        // Handle enabling background delivery error here...
    }

    return true
}
```

```swift
// Syncing Data
// 2. Open observer query

guard let stepsType = HKObjectType.quantityType(forIdentifier: .stepCount) else { return }

let query = HKObserverQuery(sampleType: stepsType, predicate: nil) {
    query, completionHandler, error in

    self.updateSteps() {
        completionHandler()
    }
}

healthStore.execute(query)
```

```swift
// Syncing Data
// 2. Open observer query

guard let stepsType = HKObjectType.quantityType(forIdentifier: .stepCount) else { return }

let query = HKObserverQuery(sampleType: stepsType, predicate: nil) {
    query, completionHandler, error in

    self.updateSteps() {
        completionHandler()
    }
}

healthStore.execute(query)
```

```swift
// Syncing Data
// 2. Open observer query

guard let stepsType = HKObjectType.quantityType(forIdentifier: .stepCount) else { return }

let query = HKObserverQuery(sampleType: stepsType, predicate: nil) {
    query, completionHandler, error in

    self.updateSteps() {
        completionHandler()
    }
}

healthStore.execute(query)
```

```swift
// Syncing Data
// 2. Open observer query

guard let stepsType = HKObjectType.quantityType(forIdentifier: .stepCount) else { return }

let query = HKObserverQuery(sampleType: stepsType, predicate: nil) {
    query, completionHandler, error in

    self.updateSteps() {
        completionHandler()
    }
}

healthStore.execute(query)
```

```swift
// Syncing Data
// 3. Execute anchored objected query

func updateSteps(completionHandler: () -> Void) {
    guard let stepsType = HKObjectType.quantityType(forIdentifier: .stepCount) else {return}

    let anchoredQuery = HKAnchoredObjectQuery(type: stepsType, predicate: nil, anchor:
    self.anchor, limit: Int(HKObjectQueryNoLimit)) { [unowned self] query, newSamples,
        deletedSamples, newAnchor, error -> Void in

        self.handleSteps(new: newSamples, deleted: deletedSamples)

        self.update(anchor: newAnchor)

        completionHandler()
    }
    healthStore.execute(anchoredQuery)
}
```

```
// Syncing Data
// 3. Execute anchored objected query

func updateSteps(completionHandler: () -> Void) {
    guard let stepsType = HKObjectType.quantityType(forIdentifier: .stepCount) else {return}

    let anchoredQuery = HKAnchoredObjectQuery(type: stepsType, predicate: nil, anchor:
    self.anchor, limit: Int(HKObjectQueryNoLimit)) { [unowned self] query, newSamples,
        deletedSamples, newAnchor, error -> Void in

        self.handleSteps(new: newSamples, deleted: deletedSamples)

        self.update(anchor: newAnchor)

        completionHandler()
    }
    healthStore.execute(anchoredQuery)
}
```

```swift
// Syncing Data
// 3. Execute anchored objected query

func updateSteps(completionHandler: () -> Void) {
    guard let stepsType = HKObjectType.quantityType(forIdentifier: .stepCount) else {return}

    let anchoredQuery = HKAnchoredObjectQuery(type: stepsType, predicate: nil, anchor:
    self.anchor, limit: Int(HKObjectQueryNoLimit)) { [unowned self] query, newSamples,
        deletedSamples, newAnchor, error -> Void in

        self.handleSteps(new: newSamples, deleted: deletedSamples)

        self.update(anchor: newAnchor)

        completionHandler()
    }
    healthStore.execute(anchoredQuery)
}
```

```swift
// Syncing Data
// 3. Execute anchored objected query

func updateSteps(completionHandler: () -> Void) {
    guard let stepsType = HKObjectType.quantityType(forIdentifier: .stepCount) else {return}

    let anchoredQuery = HKAnchoredObjectQuery(type: stepsType, predicate: nil, anchor:
    self.anchor, limit: Int(HKObjectQueryNoLimit)) { [unowned self] query, newSamples,
        deletedSamples, newAnchor, error -> Void in

        self.handleSteps(new: newSamples, deleted: deletedSamples)

        self.update(anchor: newAnchor)

        completionHandler()
    }
    healthStore.execute(anchoredQuery)
}
```

```swift
// Syncing Data
// 3. Execute anchored objected query

func updateSteps(completionHandler: () -> Void) {
    guard let stepsType = HKObjectType.quantityType(forIdentifier: .stepCount) else {return}

    let anchoredQuery = HKAnchoredObjectQuery(type: stepsType, predicate: nil, anchor:
    self.anchor, limit: Int(HKObjectQueryNoLimit)) { [unowned self] query, newSamples,
        deletedSamples, newAnchor, error -> Void in

        self.handleSteps(new: newSamples, deleted: deletedSamples)

        self.update(anchor: newAnchor)

        completionHandler()
    }
    healthStore.execute(anchoredQuery)
}
```

```swift
// Syncing Data
// 3. Execute anchored objected query

func updateSteps(completionHandler: () -> Void) {
    guard let stepsType = HKObjectType.quantityType(forIdentifier: .stepCount) else {return}

    let anchoredQuery = HKAnchoredObjectQuery(type: stepsType, predicate: nil, anchor:
    self.anchor, limit: Int(HKObjectQueryNoLimit)) { [unowned self] query, newSamples,
        deletedSamples, newAnchor, error -> Void in

        self.handleSteps(new: newSamples, deleted: deletedSamples)

        self.update(anchor: newAnchor)

        completionHandler()
    }
    healthStore.execute(anchoredQuery)
}
```

```swift
// Syncing Data
// 4. Call observer query completion handler

guard let stepsType = HKObjectType.quantityType(forIdentifier: .stepCount) else { return }

let query = HKObserverQuery(sampleType: stepsType, predicate: nil) {
    query, completionHandler, error in

    self.updateSteps() {
        completionHandler()
    }
}

healthStore.execute(query)
```

```swift
// Syncing Data
// 4. Call observer query completion handler

guard let stepsType = HKObjectType.quantityType(forIdentifier: .stepCount) else { return }

let query = HKObserverQuery(sampleType: stepsType, predicate: nil) {
    query, completionHandler, error in

    self.updateSteps() {
        completionHandler()
    }
}

healthStore.execute(query)
```

Tracking Changed Data

Tracking Changed Data

Use UUIDs to keep track of unique HKObjects

Tracking Changed Data

Use UUIDs to keep track of unique HKObjects

Record UUIDs of HKObjects in your own data store

Tracking Changed Data

Use UUIDs to keep track of unique HKObjects

Record UUIDs of HKObjects in your own data store

When samples are deleted, remove data corresponding to those UUIDs

Tracking Changed Data

Use UUIDs to keep track of unique HKObjects

Record UUIDs of HKObjects in your own data store

When samples are deleted, remove data corresponding to those UUIDs

Ensure sync doesn’t re-add already deleted samples

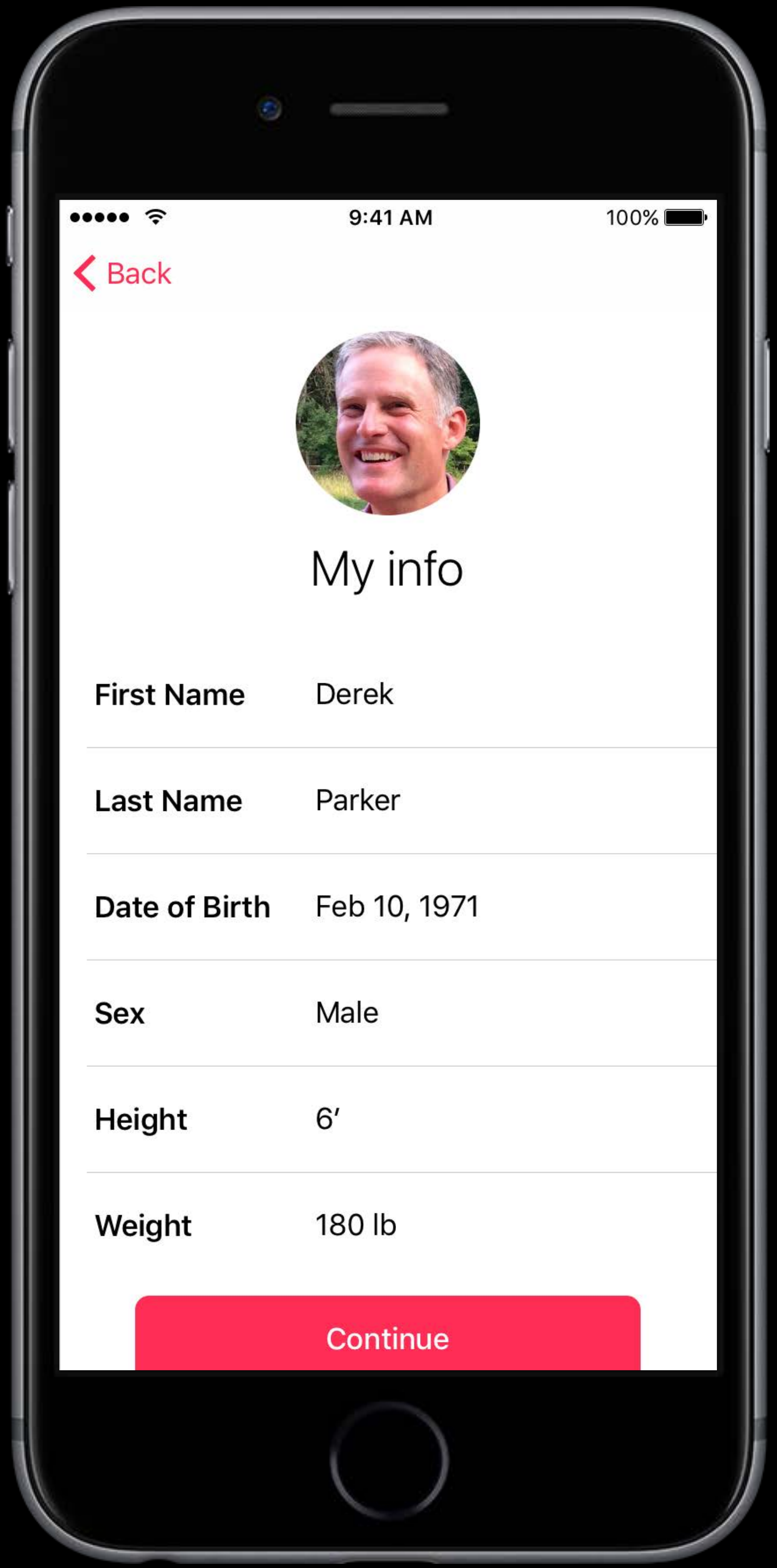
Duplication

Potential problems

Duplication

Potential problems

Pre-populating data during on-boarding saves time



My info

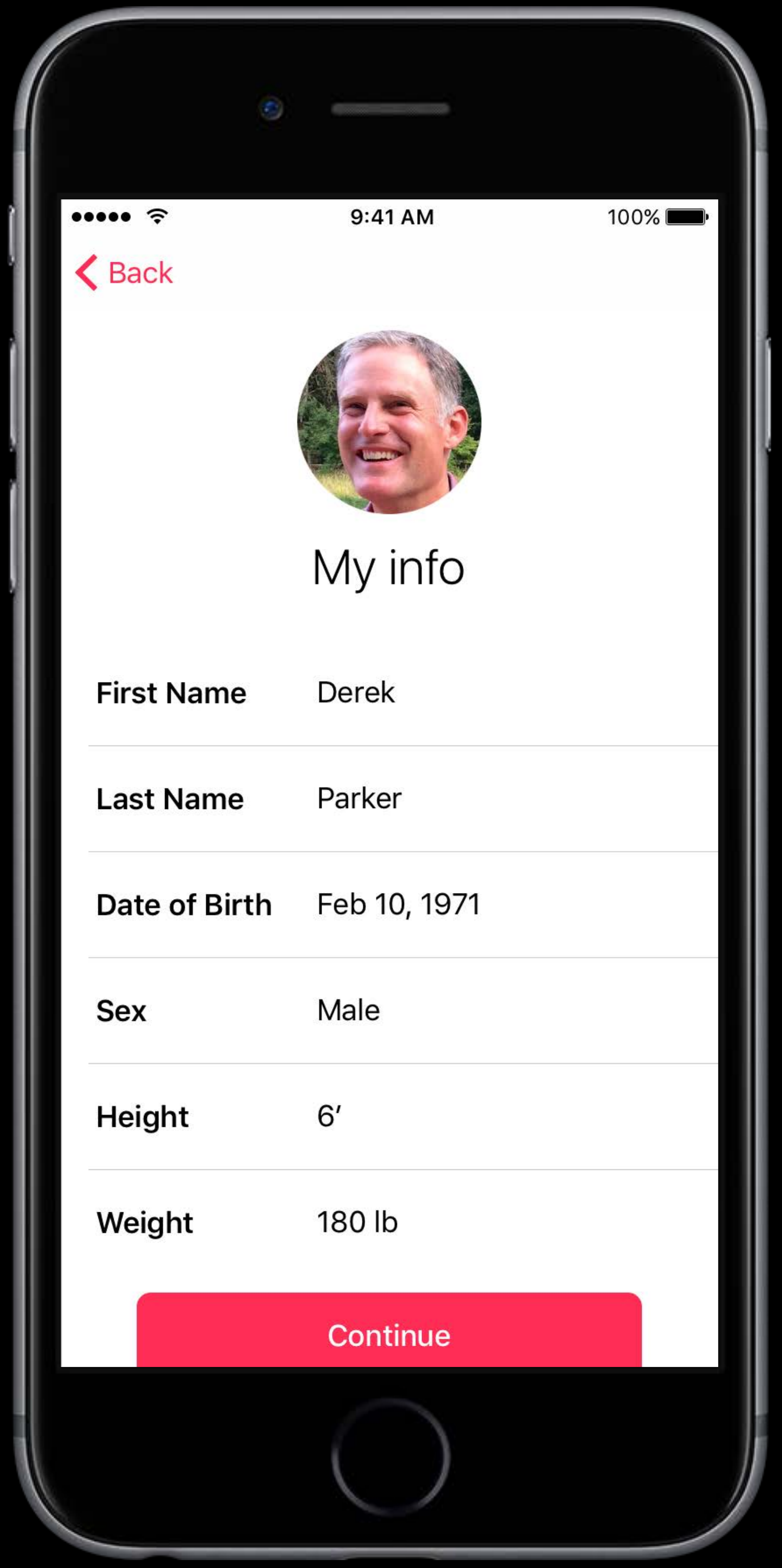
First Name	Derek
Last Name	Parker
Date of Birth	Feb 10, 1971
Sex	Male
Height	6′
Weight	180 lb

Duplication

Potential problems

Pre-populating data during on-boarding saves time

Users can verify/change data



First Name	Derek
Last Name	Parker
Date of Birth	Feb 10, 1971
Sex	Male
Height	6′
Weight	180 lb

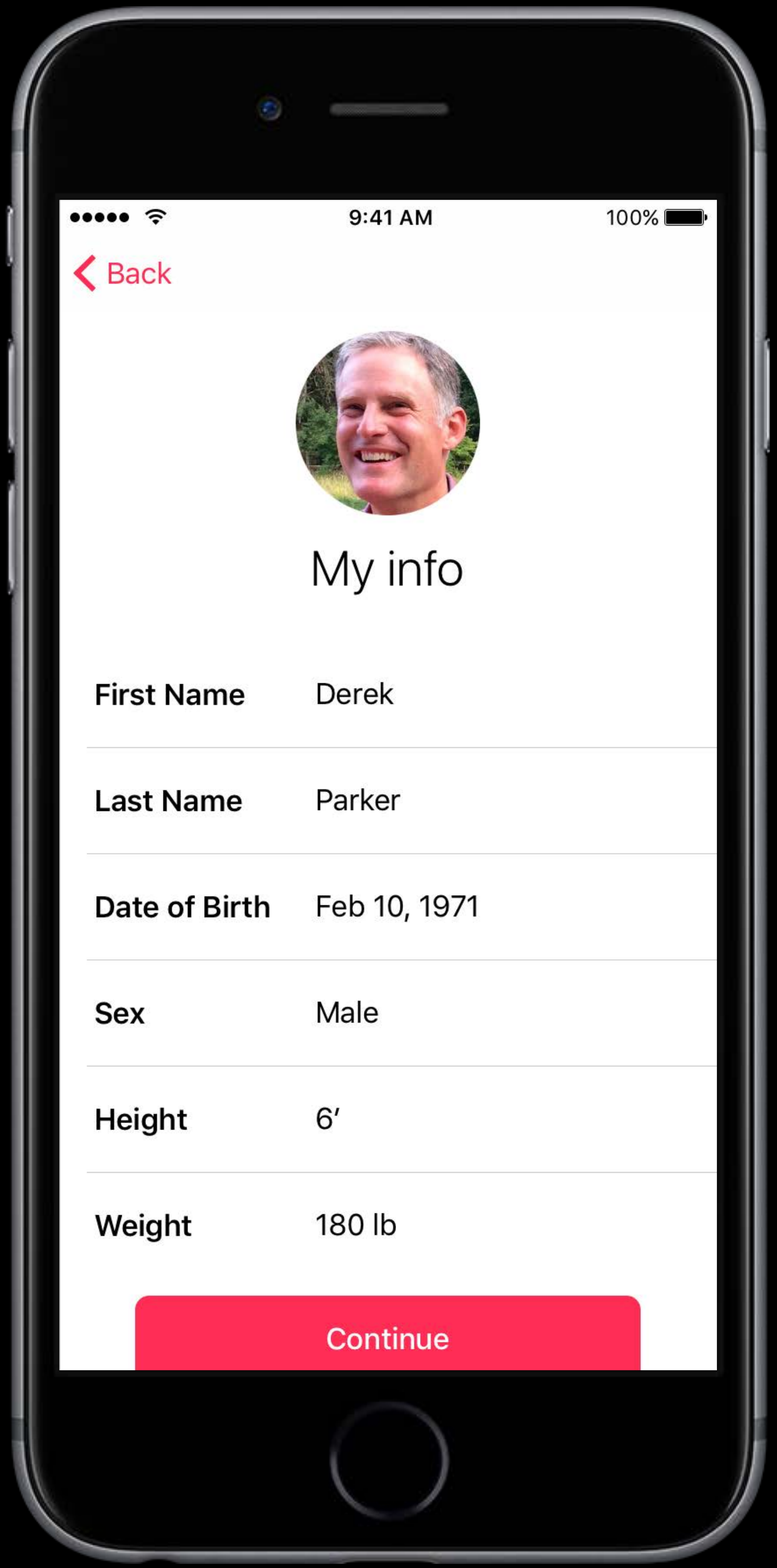
Duplication

Potential problems

Pre-populating data during on-boarding saves time

Users can verify/change data

Problem: Saving unchanged values



My info

First Name	Derek
Last Name	Parker
Date of Birth	Feb 10, 1971
Sex	Male
Height	6′
Weight	180 lb

Duplication

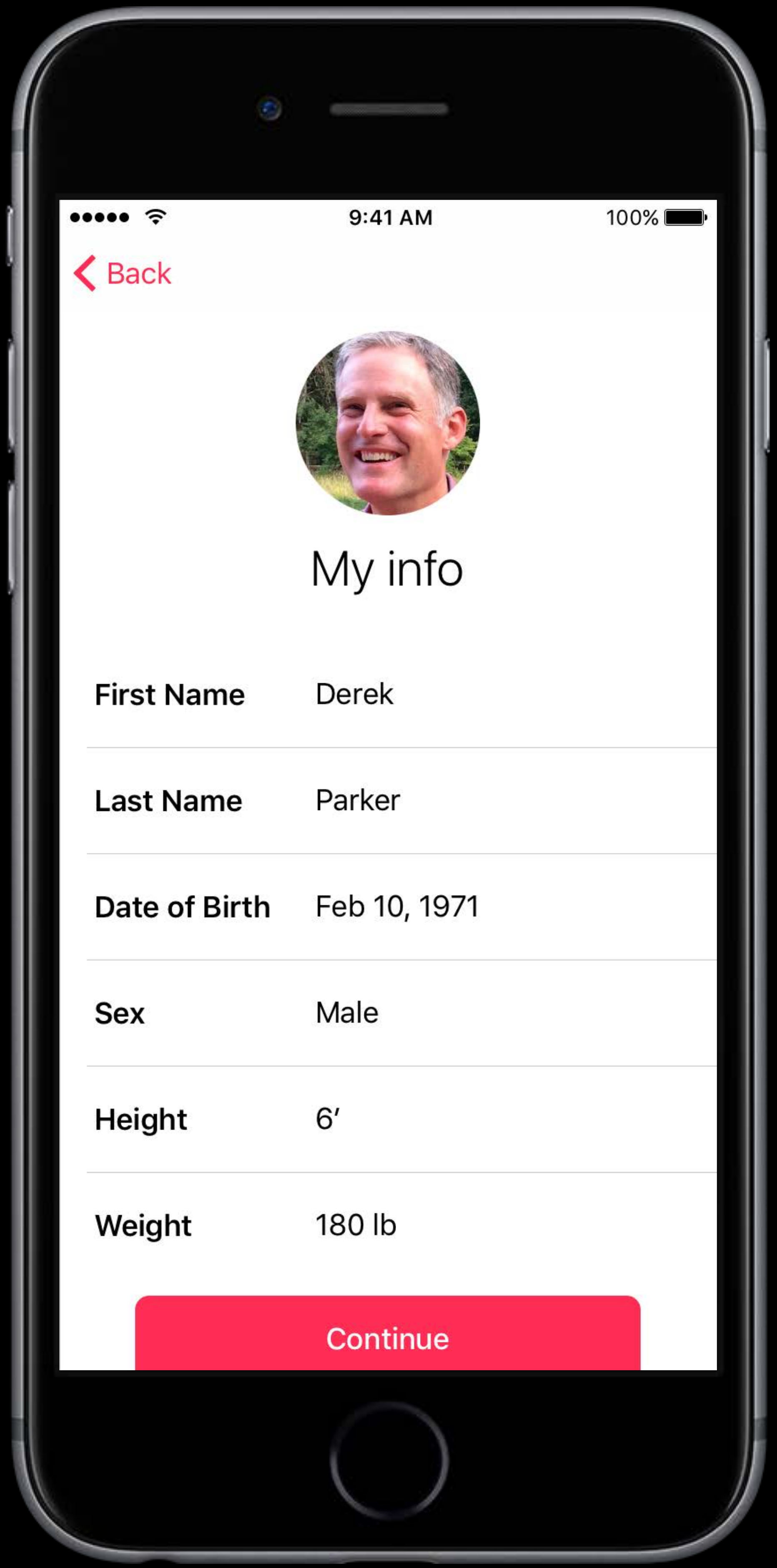
Potential problems

Pre-populating data during on-boarding saves time

Users can verify/change data

Problem: Saving unchanged values

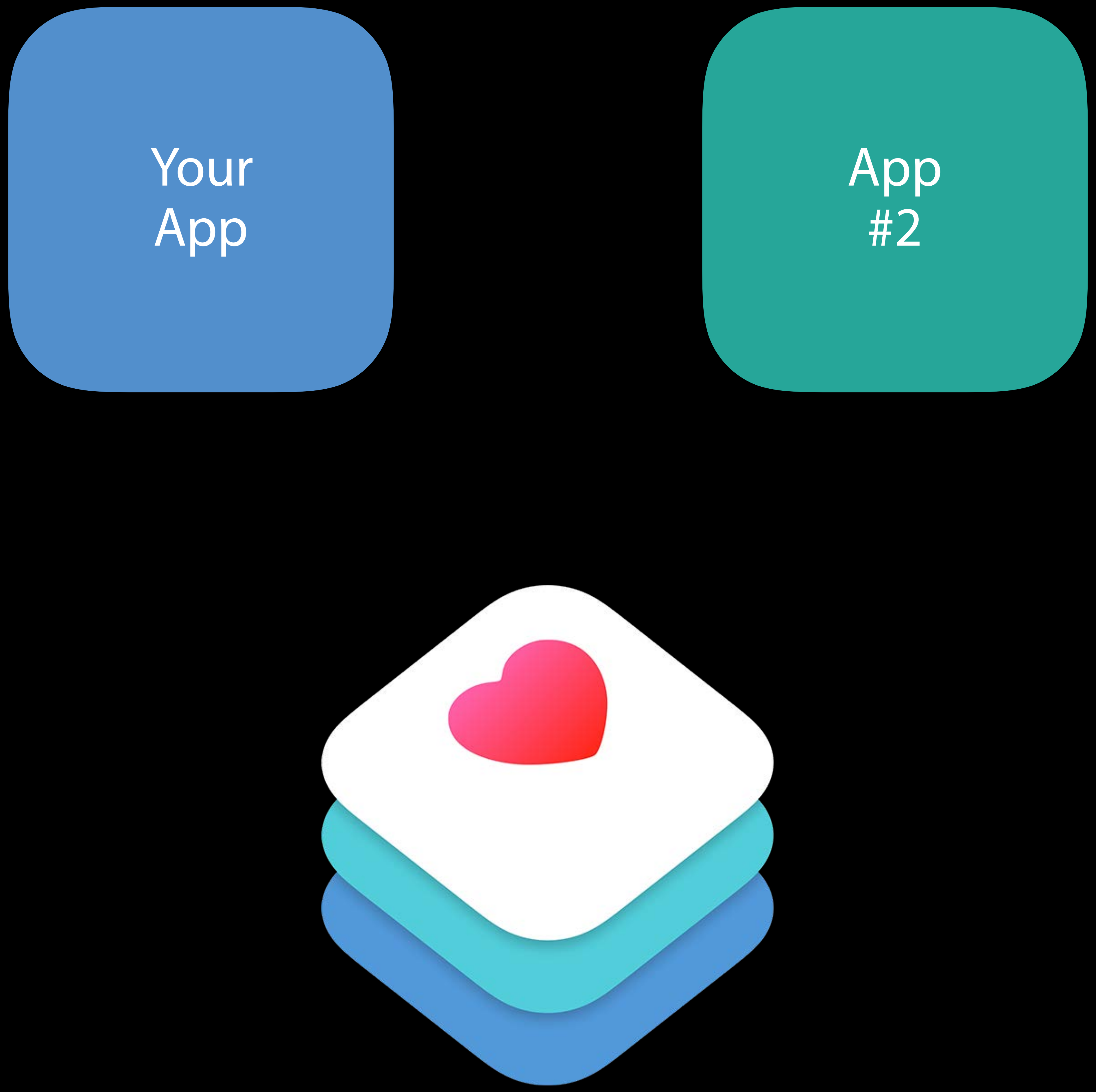
- Only save data again if this is the user's intent



First Name	Derek
Last Name	Parker
Date of Birth	Feb 10, 1971
Sex	Male
Height	6′
Weight	180 lb

Duplication

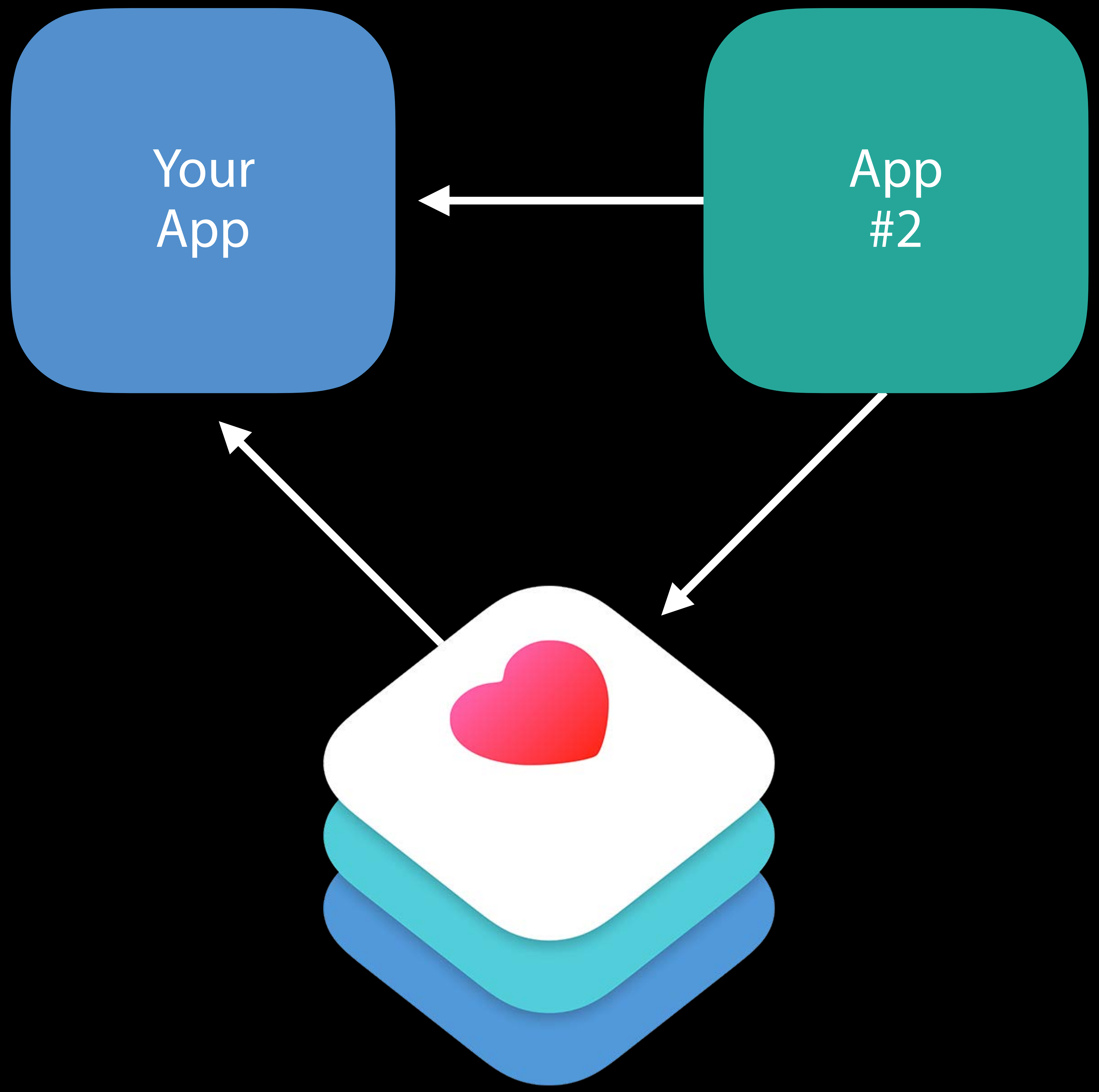
Potential problems



Duplication

Potential problems

Problem: Ingesting information from another app and HealthKit

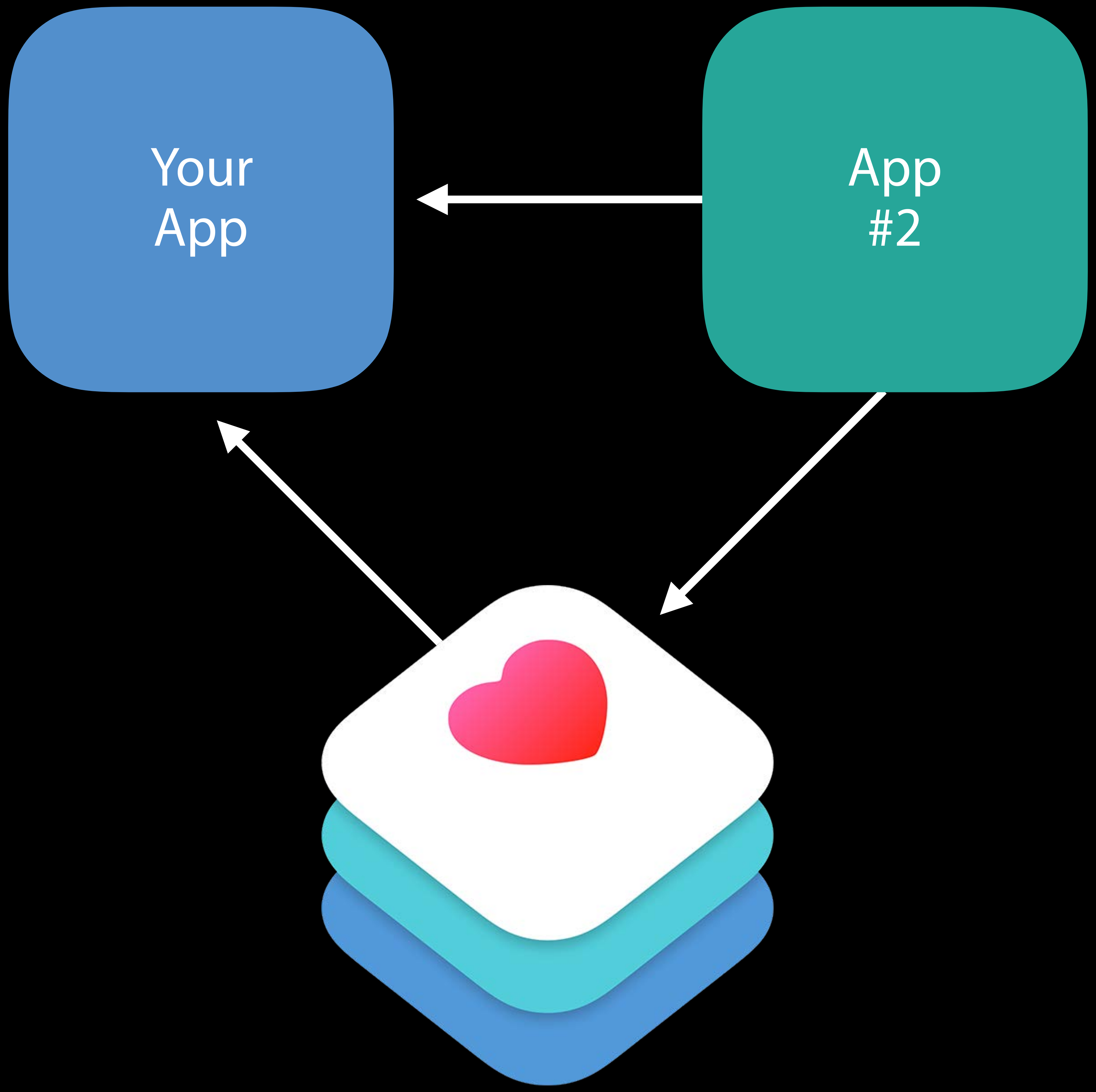


Duplication

Potential problems

Problem: Ingesting information from another app and HealthKit

- Pick only **one** source most appropriate to your app

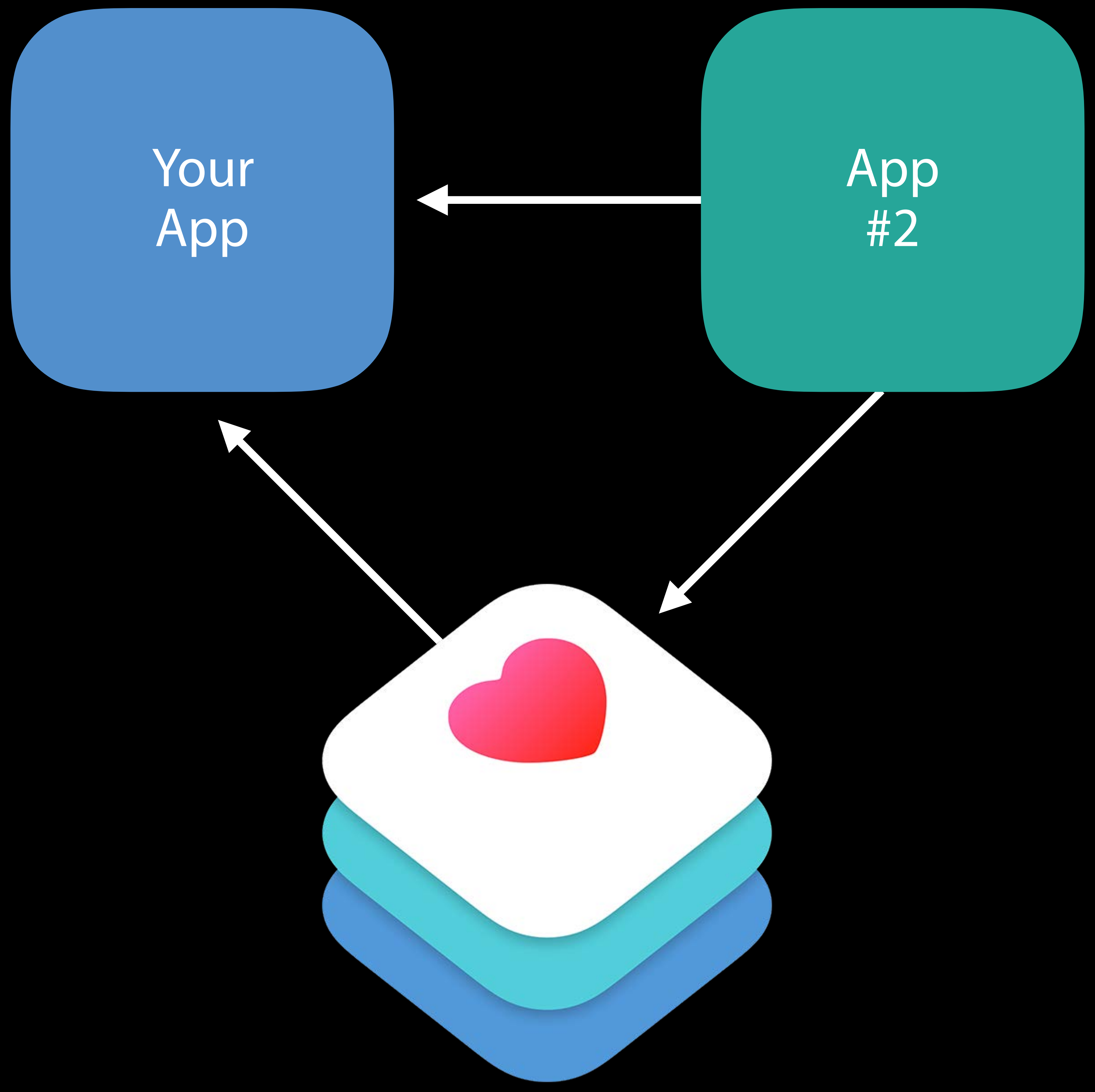


Duplication

Potential problems

Problem: Ingesting information from another app and HealthKit

- Pick only **one** source most appropriate to your app
- Do **not** save data on another app's behalf

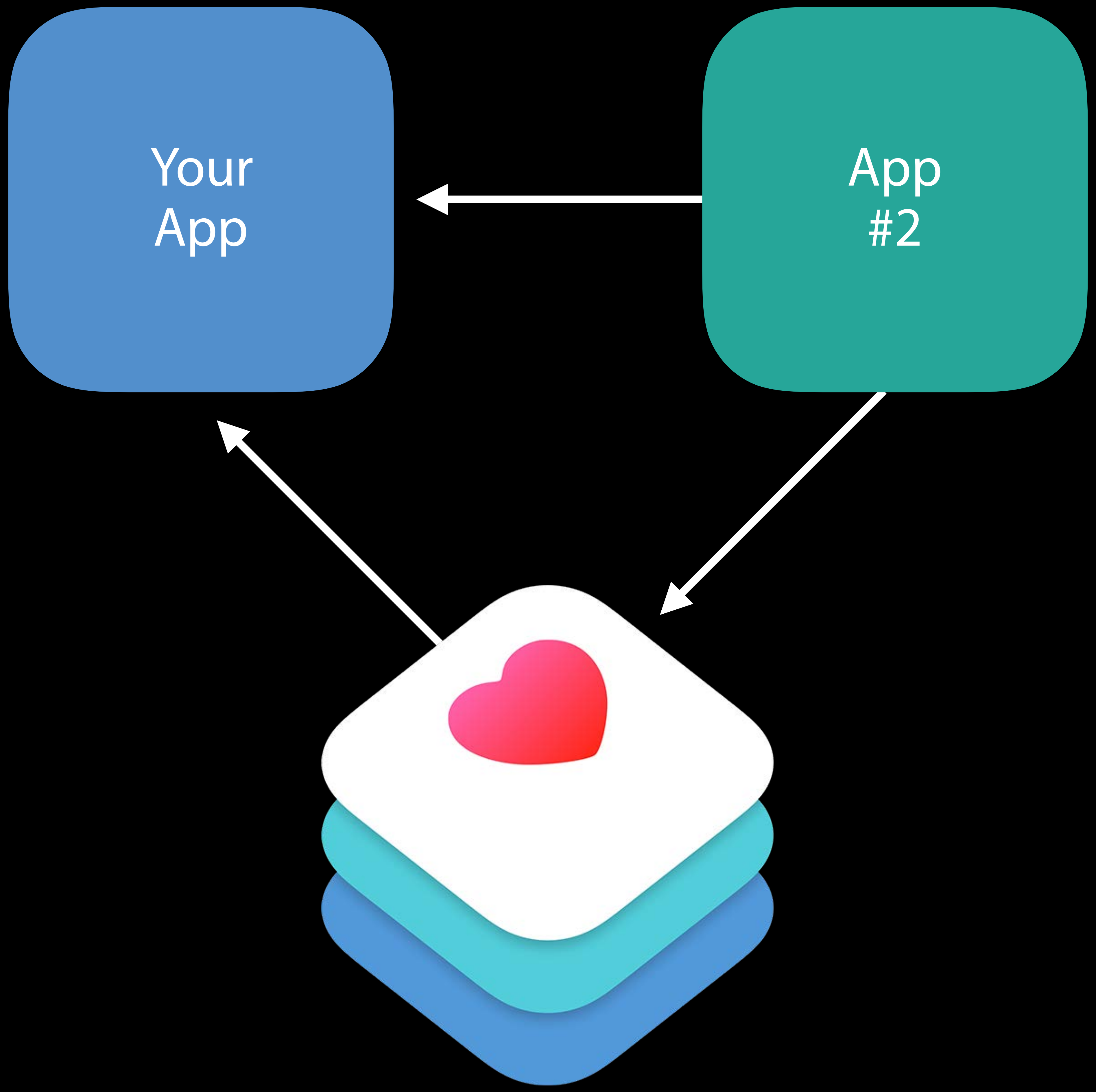


Duplication

Potential problems

Problem: Ingesting information from another app and HealthKit

- Pick only **one** source most appropriate to your app
- Do **not** save data on another app's behalf
- Write only your **own** data **once**



Duplication

Exceptions

Duplication

Exceptions

Sometimes duplication is intentional

Duplication

Exceptions

Sometimes duplication is intentional

- Data from multiple sources

Duplication

Exceptions

Sometimes duplication is intentional

- Data from multiple sources

`HKStatisticsQuery` and `HKStatisticsCollectionQuery` automatically de-duplicate data

Duplication

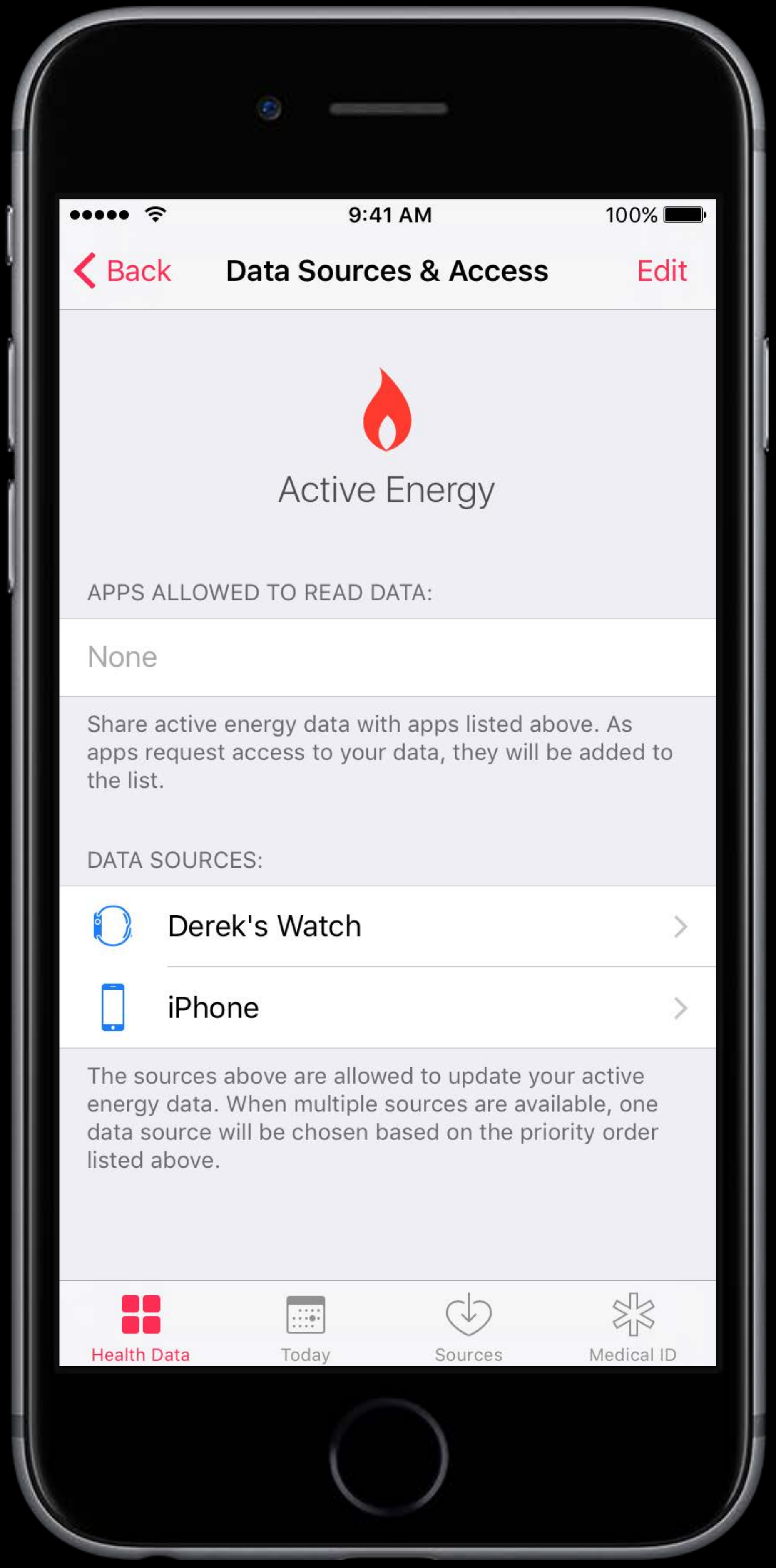
Exceptions

Sometimes duplication is intentional

- Data from multiple sources

`HKStatisticsQuery` and `HKStatisticsCollectionQuery` automatically de-duplicate data

Order of preferred data sources can change in Health app



Data Sources & Access

Active Energy

APPS ALLOWED TO READ DATA:

None

Share active energy data with apps listed above. As apps request access to your data, they will be added to the list.

DATA SOURCES:

Derek's Watch

iPhone

The sources above are allowed to update your active energy data. When multiple sources are available, one data source will be chosen based on the priority order listed above.

Migrating Data

Migrating Data

Migrating Data

98° C

Migrating Data

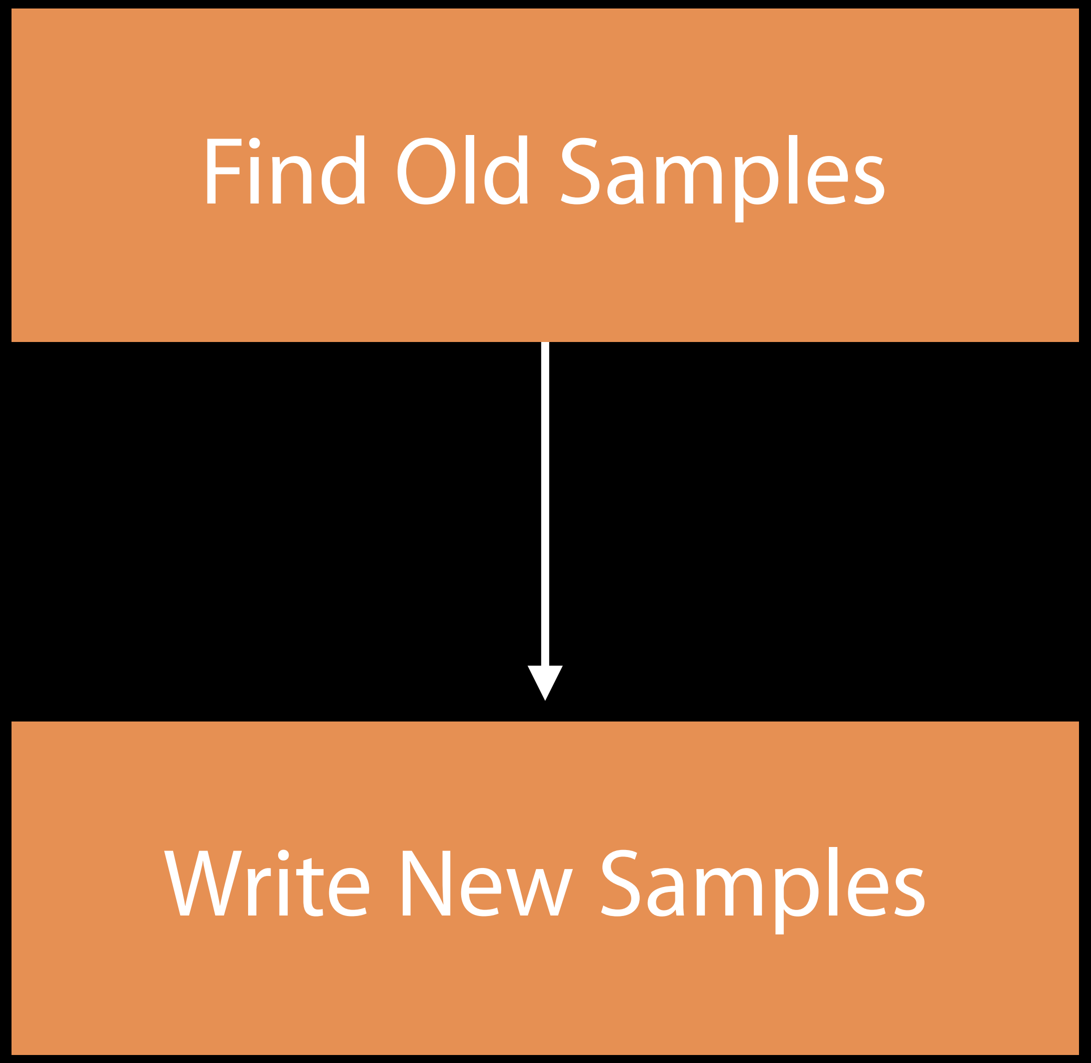
98° C

Find Old Samples

Migrating Data

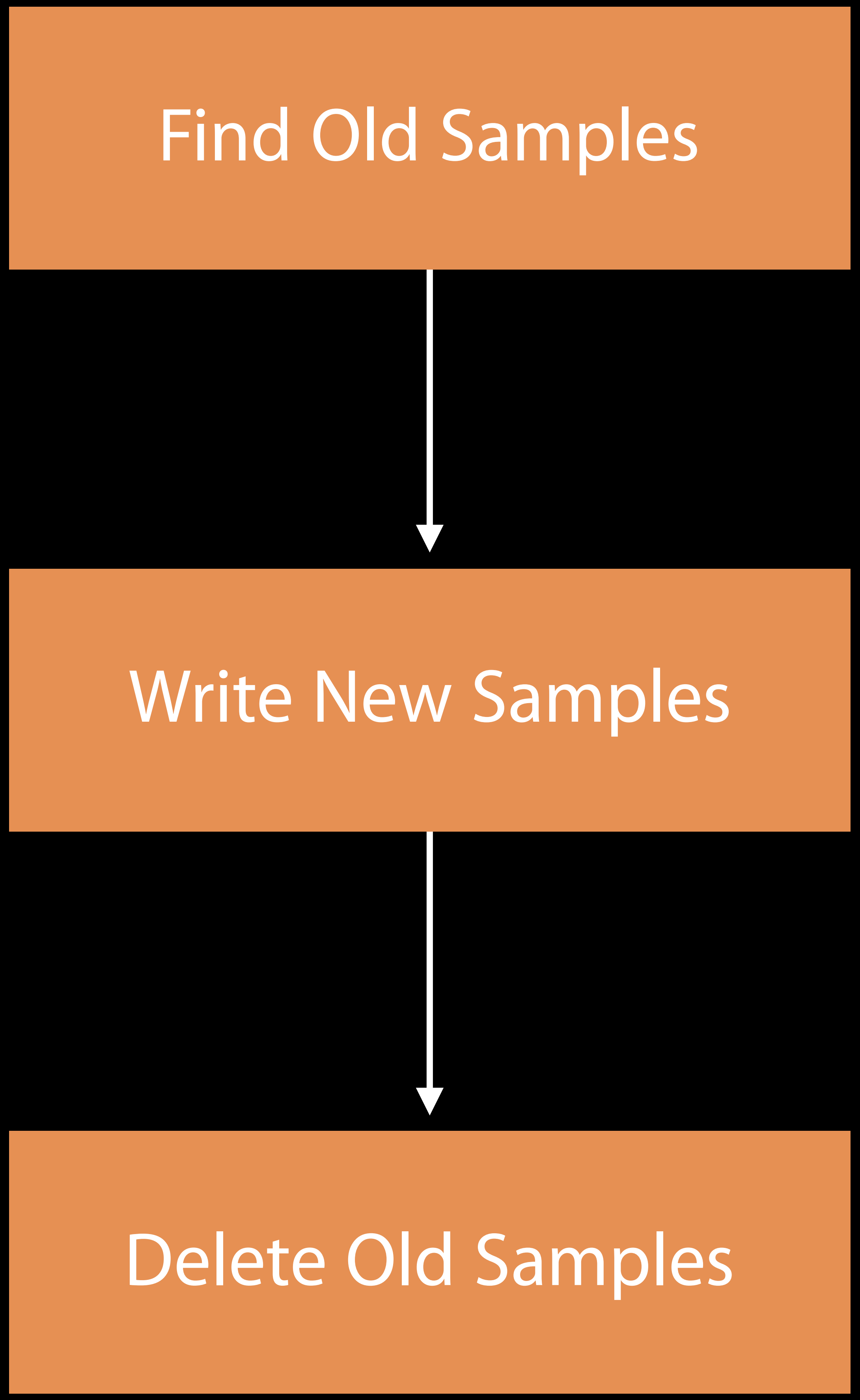
98° C

98° F



Migrating Data

98° F



Handling Data

Flow between iPhone and Apple Watch

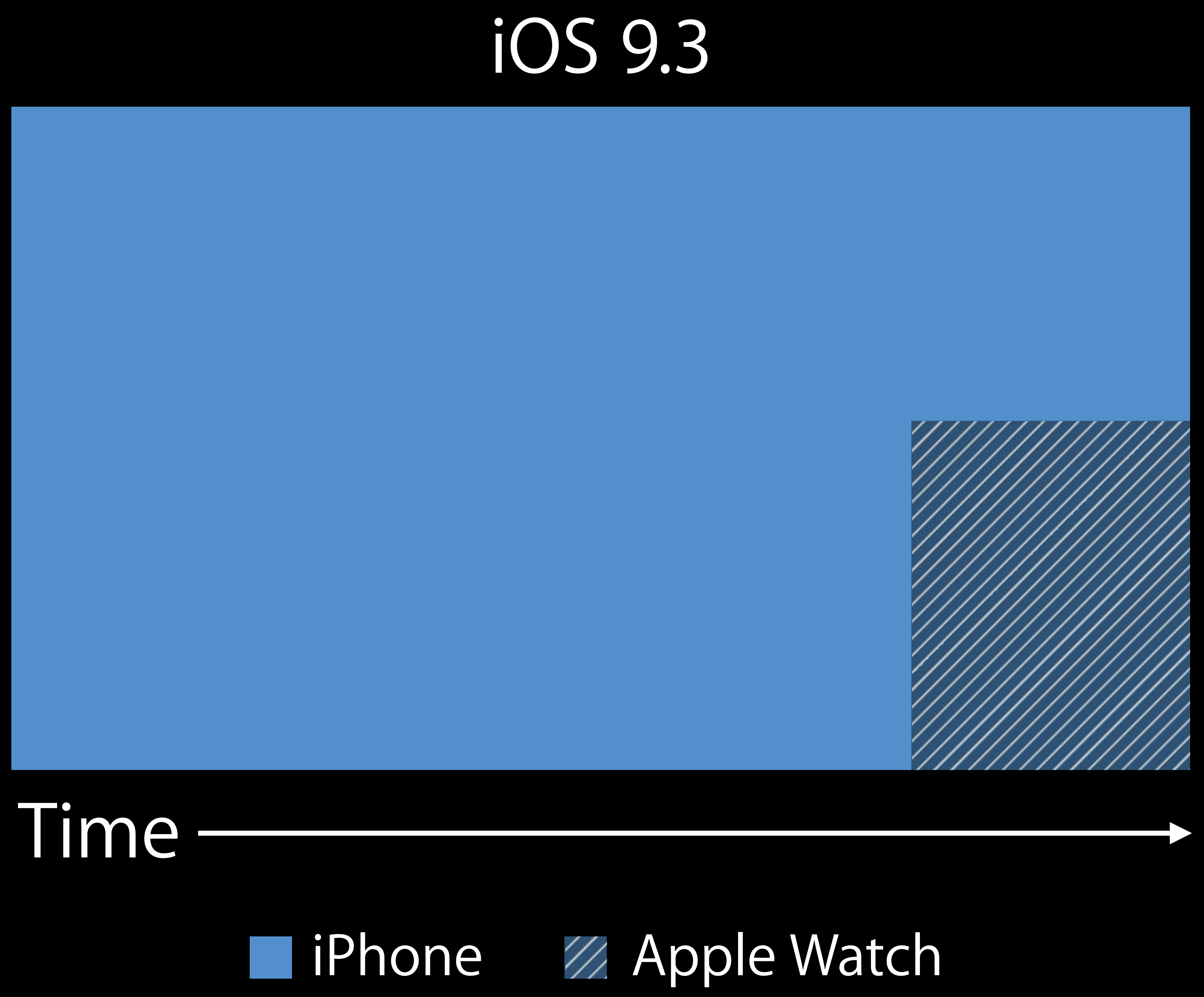
NEW

Handling Data

Flow between iPhone and Apple Watch

NEW

Recent samples from iPhone are periodically synced to Apple Watch



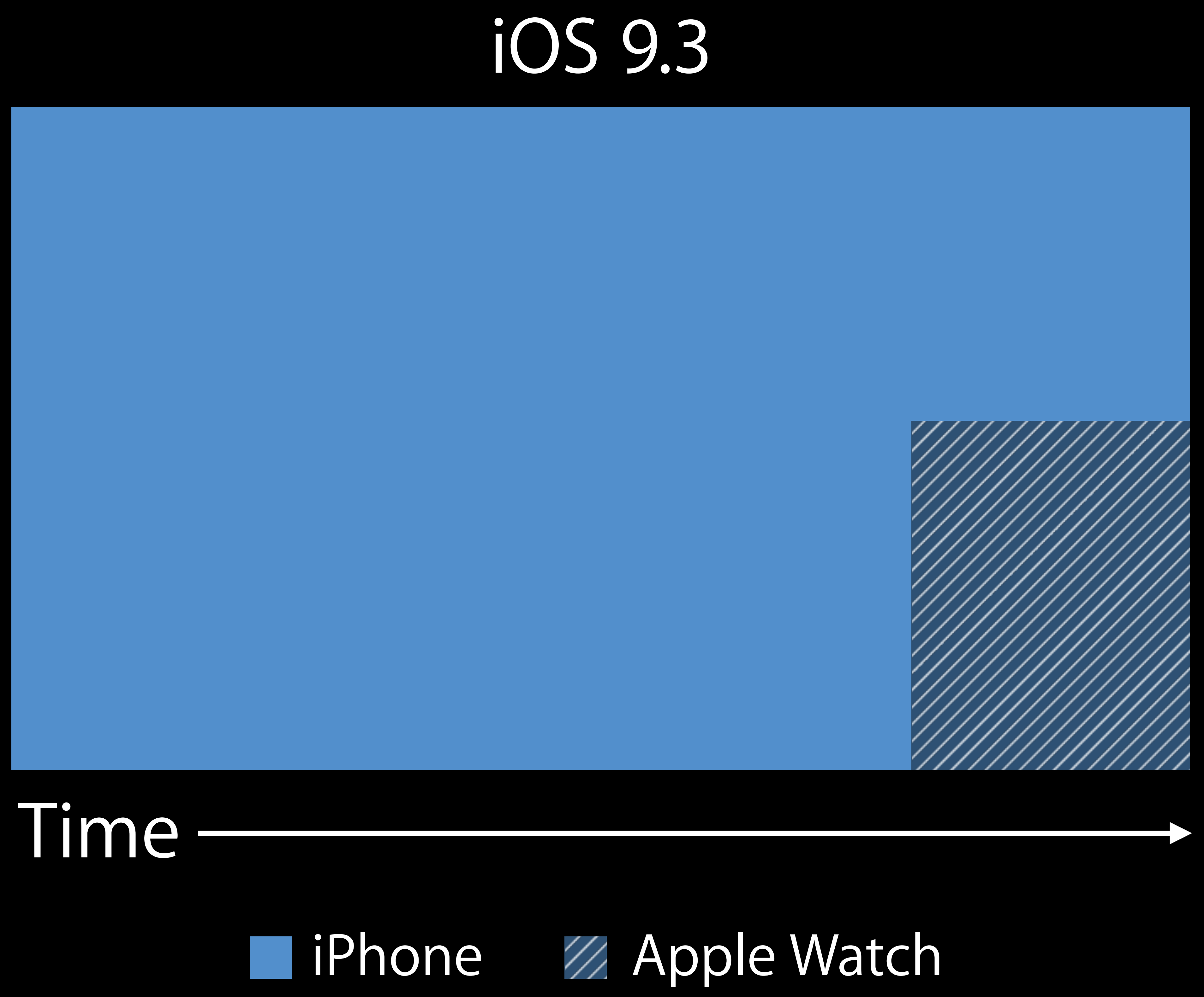
Handling Data

Flow between iPhone and Apple Watch

NEW

Recent samples from iPhone are periodically synced to Apple Watch

Samples on Apple Watch are pruned by end date



Handling Data

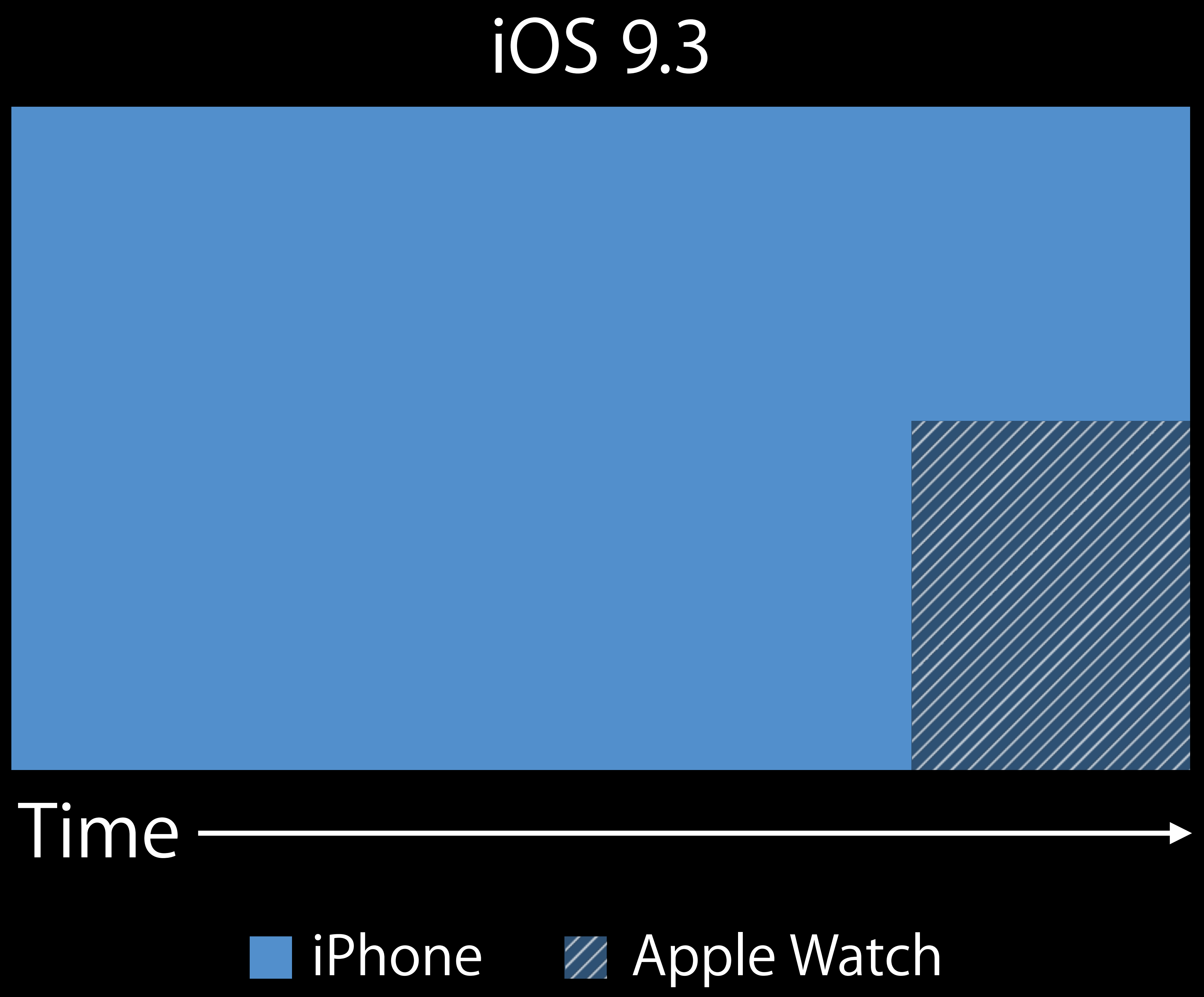
NEW

Flow between iPhone and Apple Watch

Recent samples from iPhone are periodically synced to Apple Watch

Samples on Apple Watch are pruned by end date

Save samples with endDate after `HKHealthStore`'s `earliestPermittedSampleDate`



Handling Data

Flow between iPhone and Apple Watch

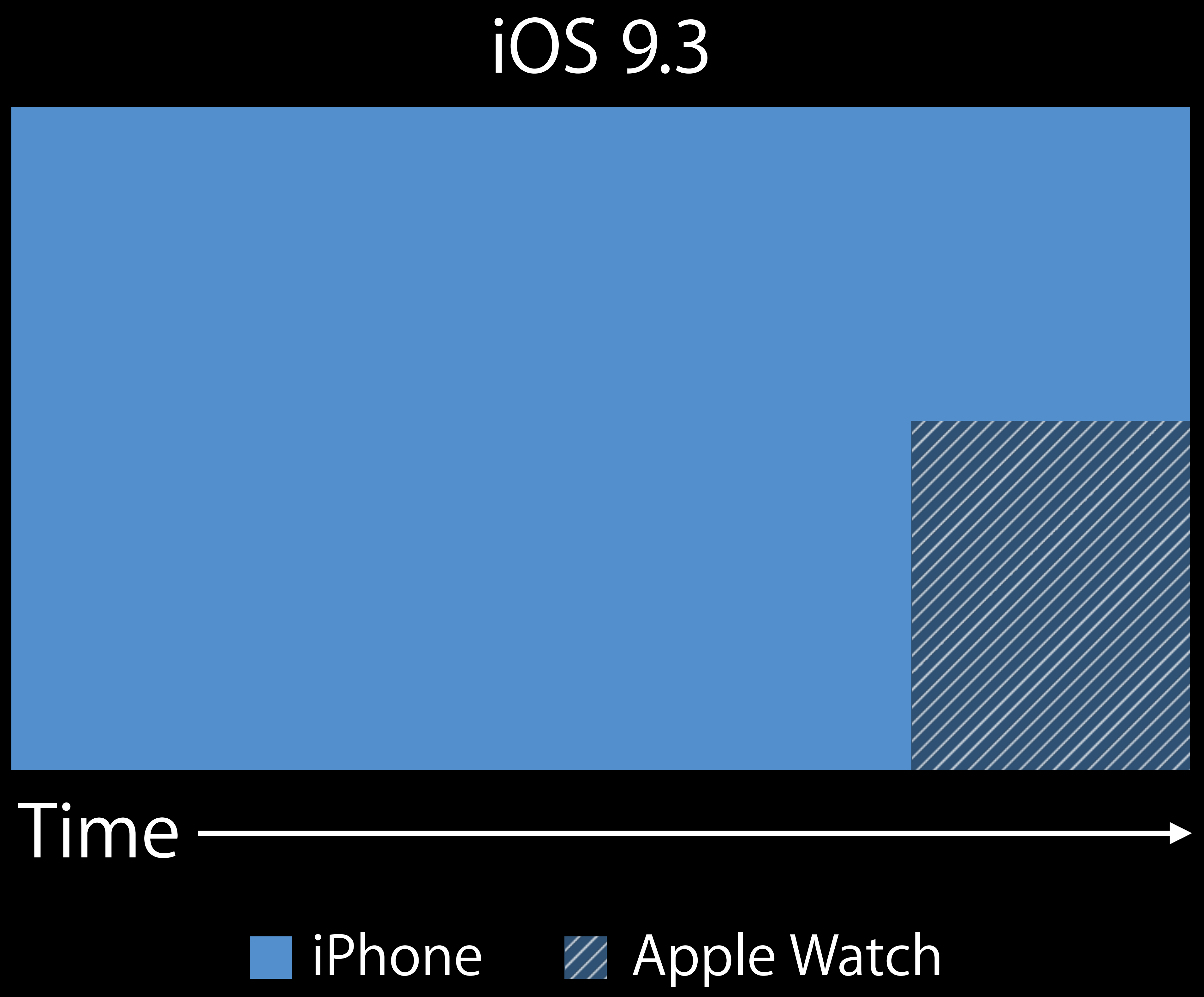
NEW

Recent samples from iPhone are periodically synced to Apple Watch

Samples on Apple Watch are pruned by end date

Save samples with endDate after `HKHealthStore`'s `earliestPermittedSampleDate`

Save samples on iPhone or Apple Watch, not both



Wheelchair Support

Wheelchair Support

NEW

Wheelchair Support

NEW

New characteristic data type

Wheelchair Support

NEW

New characteristic data type

New quantity types

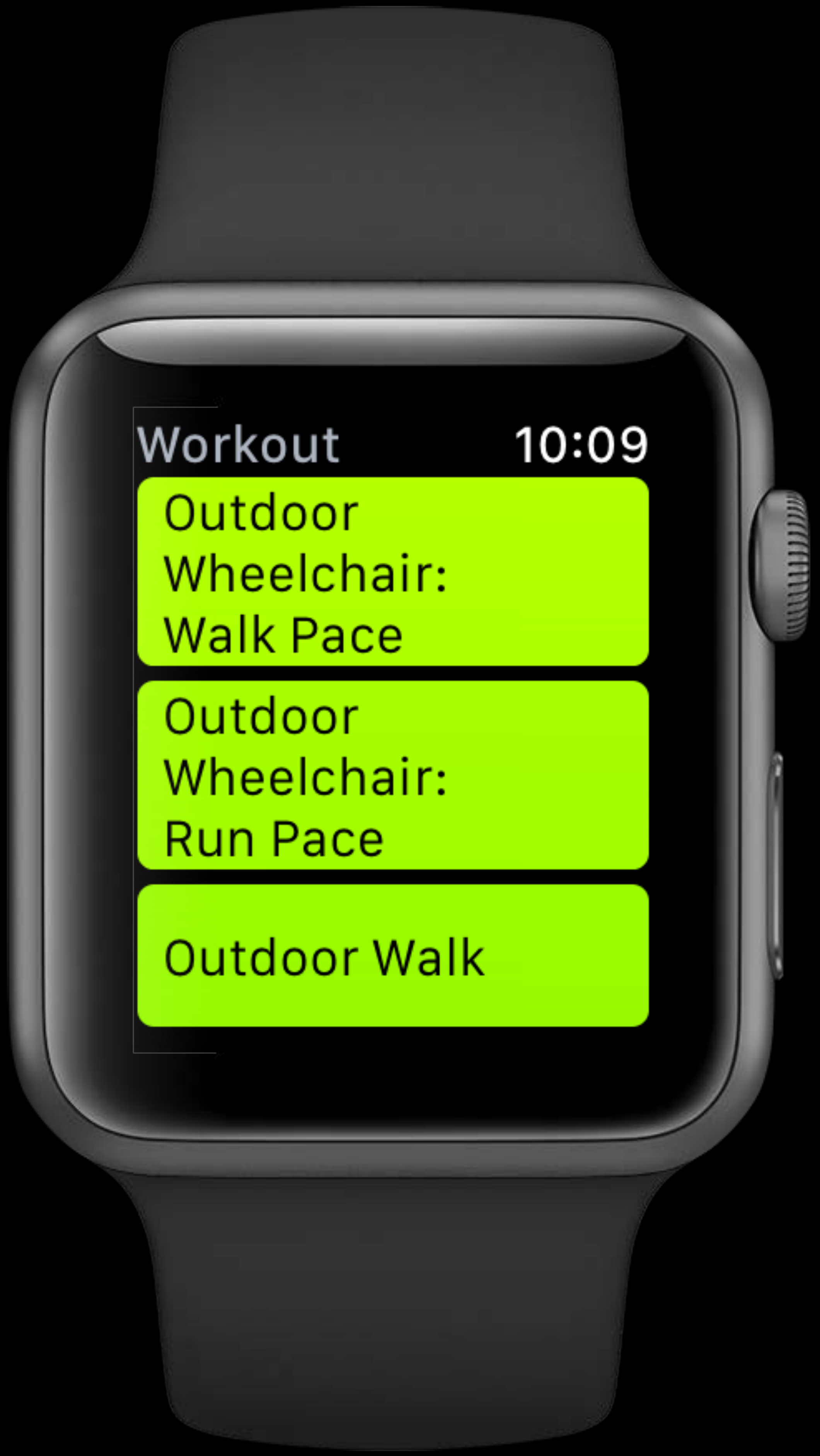
Wheelchair Support

NEW

New characteristic data type

New quantity types

New workout activity types



Wheelchair Support

Details

NEW

Wheelchair Support

NEW

Details

Steps → Push count

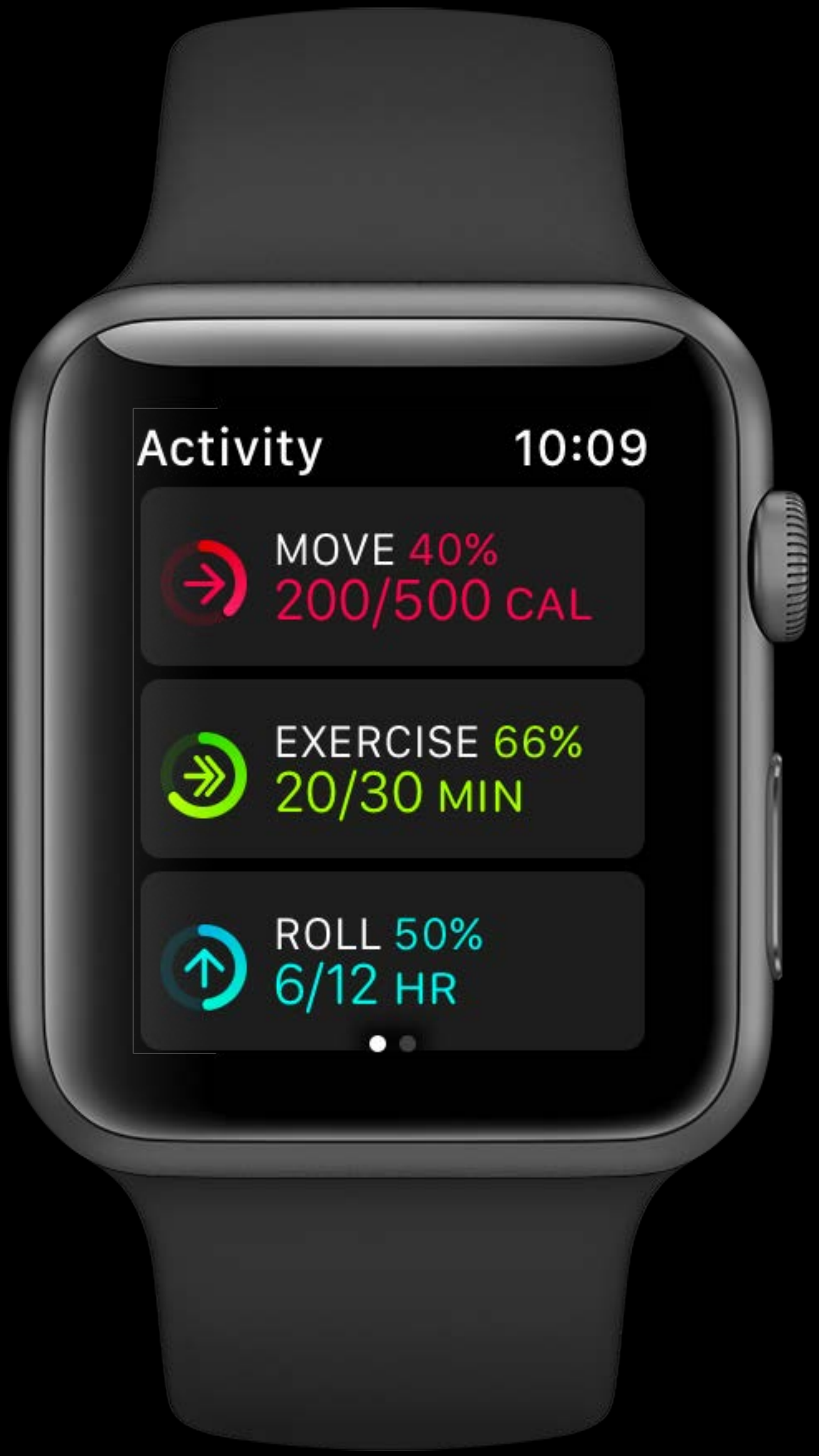
Wheelchair Support

Details

NEW

Steps → Push count

Stand hours → Roll hours



Wheelchair Support

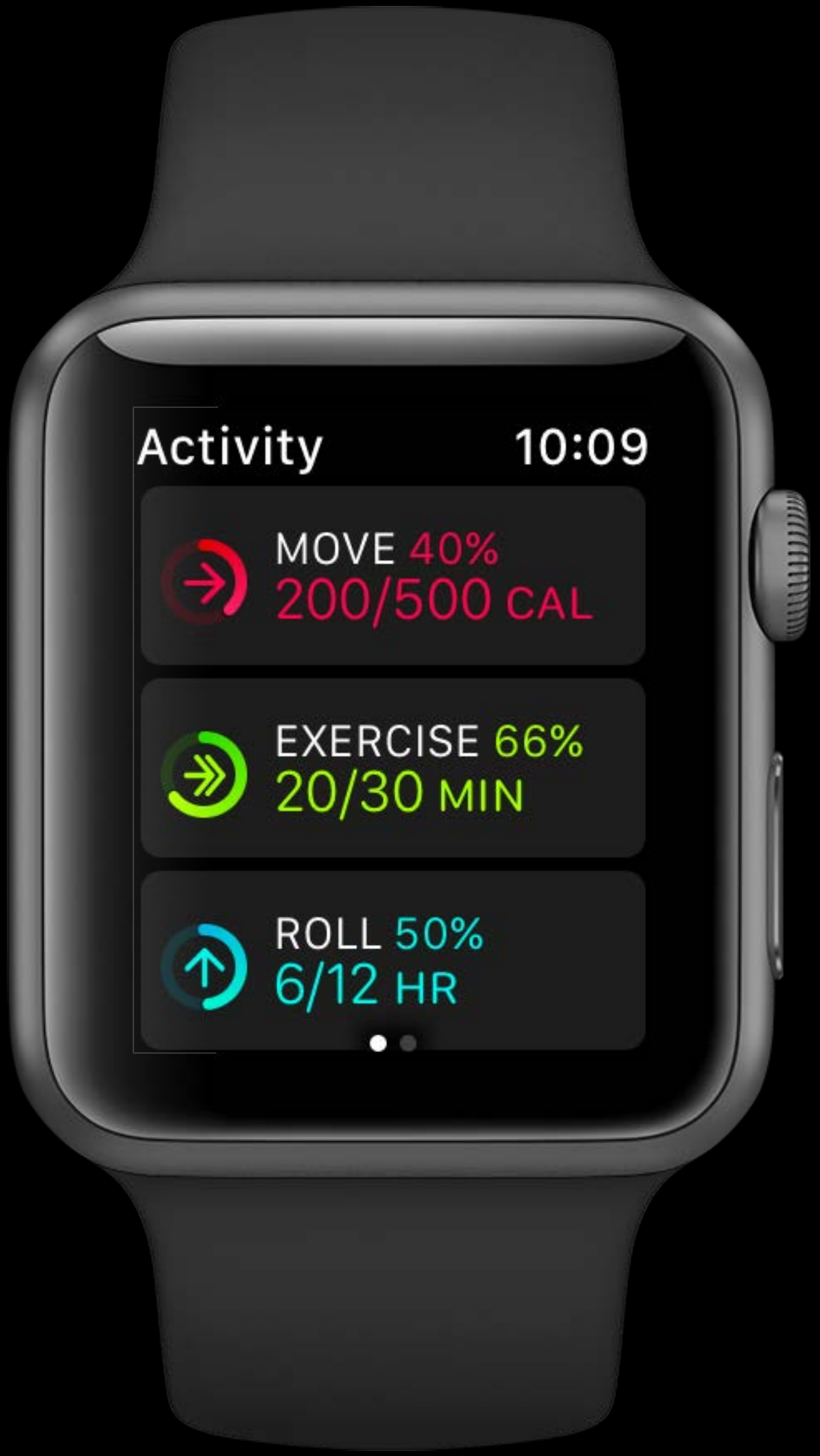
Details

NEW

Steps → Push count

Stand hours → Roll hours

Wheelchair distance only recorded during workouts



Wheelchair Support

Details

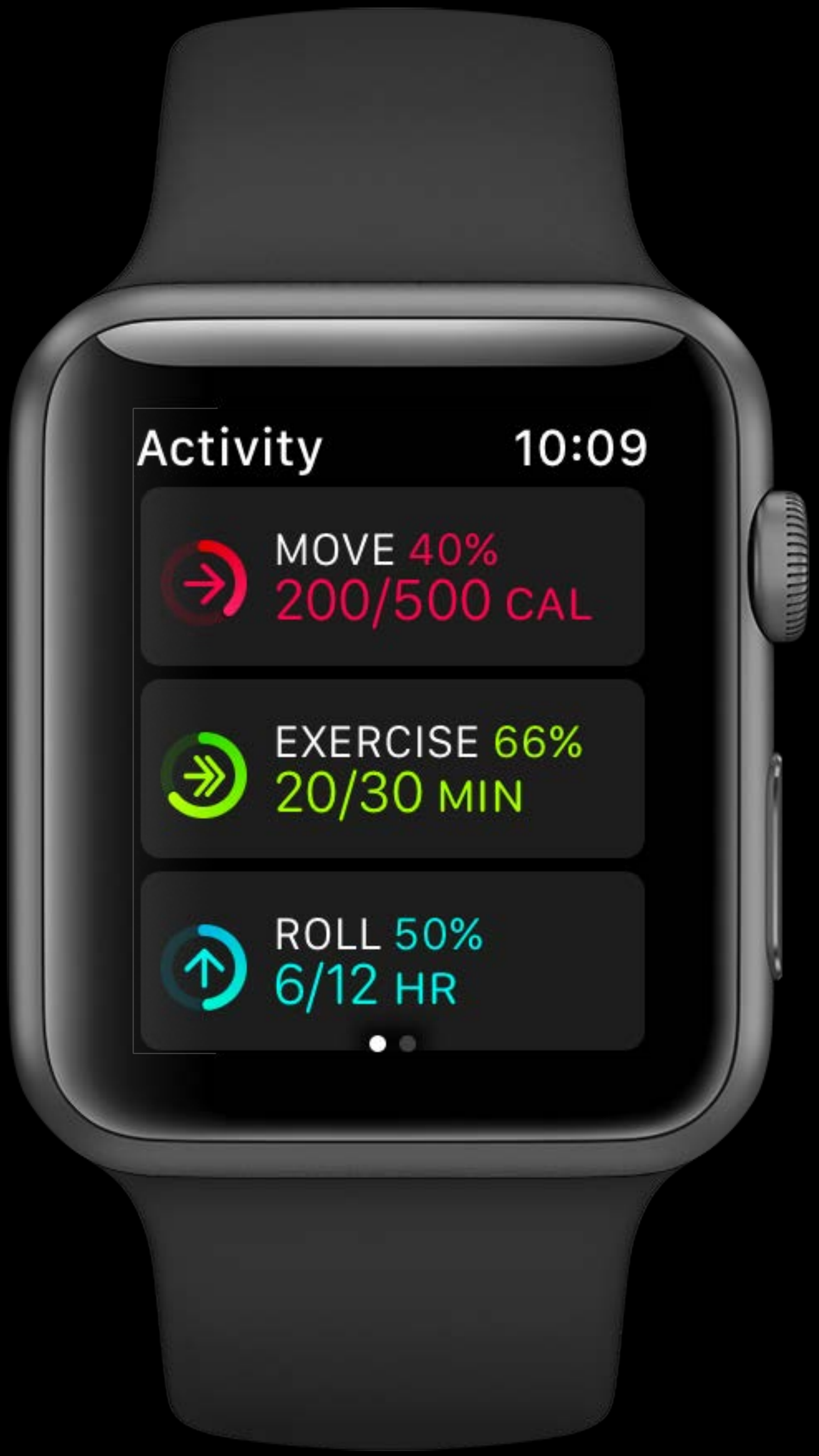
NEW

Steps ➞ Push count

Stand hours ➞ Roll hours

Wheelchair distance only recorded during workouts

Wheelchair use status can change over time



Summary

Summary

Pay attention to the authorization user experience

Summary

Pay attention to the authorization user experience

Incorporate Apple’s activity rings into your app

Summary

Pay attention to the authorization user experience

Incorporate Apple's activity rings into your app

Take care when handling and synchronizing HealthKit data

Summary

Pay attention to the authorization user experience

Incorporate Apple’s activity rings into your app

Take care when handling and synchronizing HealthKit data

Make your experience accessible to wheelchair users

More Information

https://developer.apple.com/wwdc16/209

Related Sessions

Health and Fitness with Core Motion	Nob Hill	Thursday 3:00PM
What's New in ResearchKit	Nob Hill	Friday 10:00AM
Building Great Workout Apps	Pacific Heights	Friday 11:00AM
Getting Started with CareKit	Pacific Heights	Friday 3:00PM
Introducing HealthKit		WWDC 2014
Designing Accessories for iOS and OS X		WWDC 2014
What's New in HealthKit		WWDC 2015

Labs

HealthKit Lab	Frameworks Lab A	Wednesday 10:00AM
HealthKit Lab	Frameworks Lab A	Thursday 9:00AM
ResearchKit and CareKit Lab	Fort Mason	Friday 10:30AM
Core Motion Lab	Frameworks Lab D	Friday 12:30PM
ResearchKit and CareKit Lab	Fort Mason	Friday 3:30PM

WWDC16