# What's New with Screen Recording and Live Broadcast

Session 606

Johnny Trenh, Software Engineer
Alexander Subbotin, Software Engineer

# Screen Recording

Record app visuals and audio

Record microphone input

Share recordings

# Live Broadcast

Broadcast app visuals and audio

Xcode Templates

# Details

HD quality

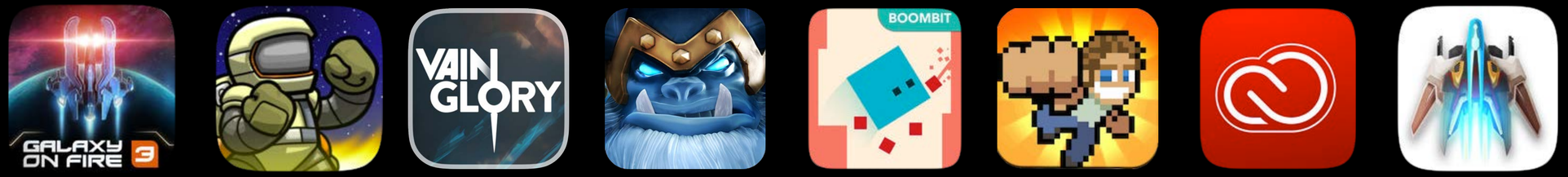
Low-performance impact

Minimal power usage

Privacy safeguards

User consent prompt

# Adoption



# Adoption

# ReplayKit 2

# ReplayKit 2

NEW

In-App Screen Capture

# ReplayKit 2

NEW

In-App Screen Capture

iOS Screen Record and Broadcast

# ReplayKit 2

NEW

In-App Screen Capture

iOS Screen Record and Broadcast

Broadcast Pairing

# ReplayKit 2

NEW

In-App Screen Capture

iOS Screen Record and Broadcast

Broadcast Pairing

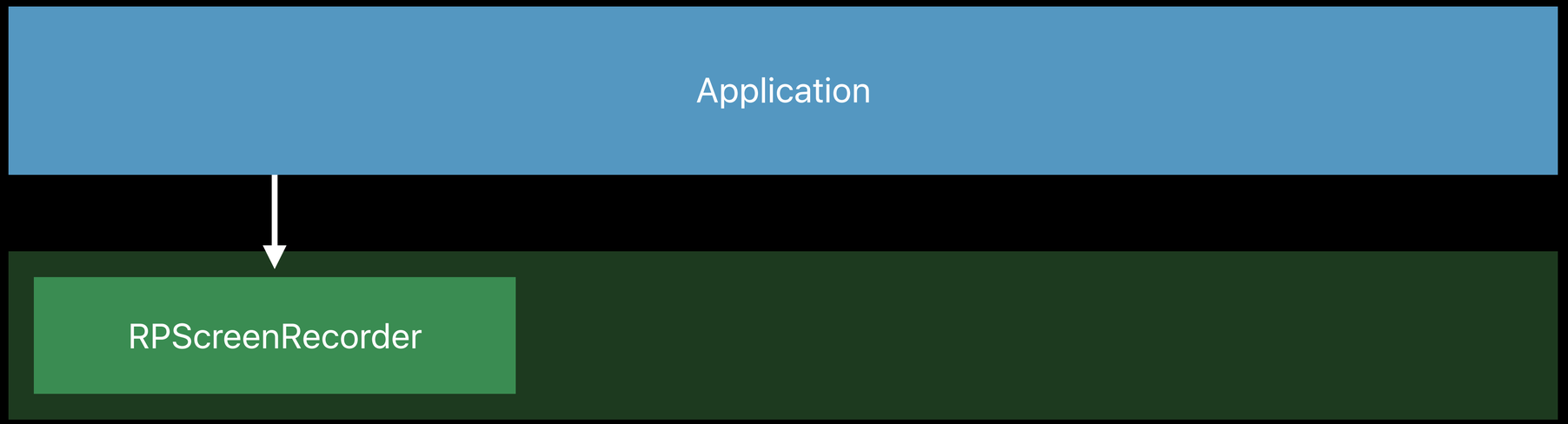
Fast Camera Switching

# In-App Screen Capture

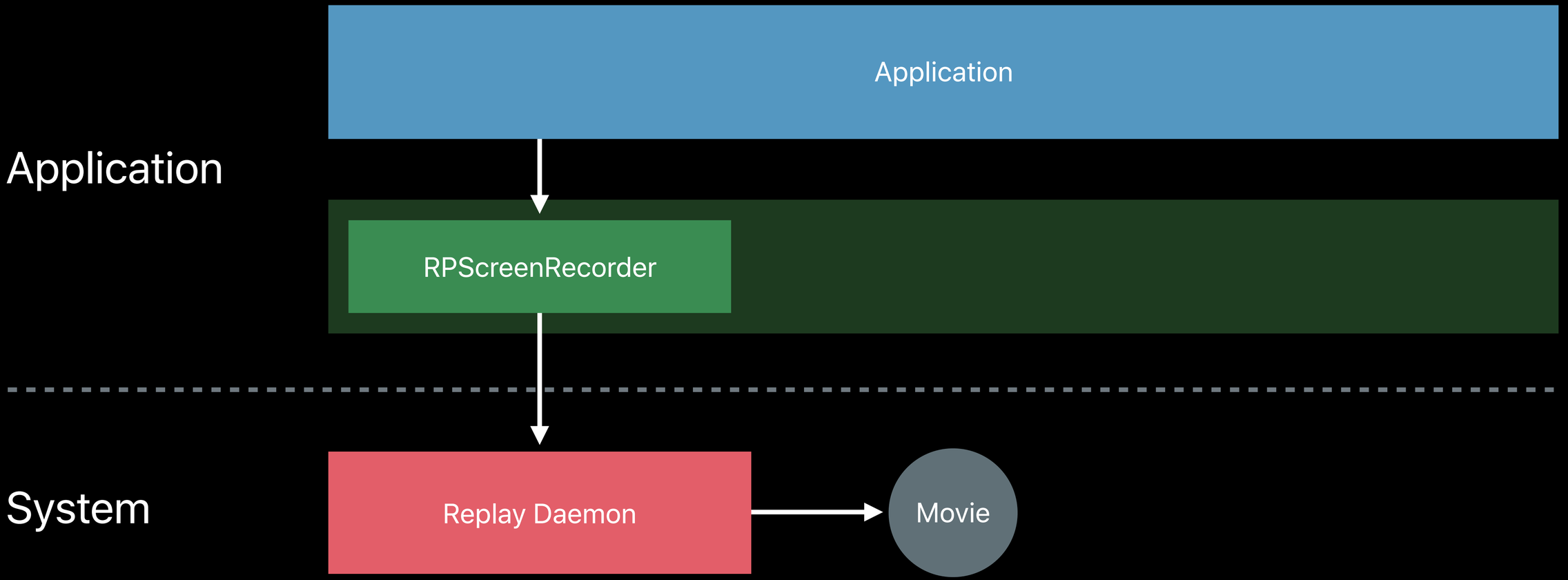
Johnny Trenh, Software Engineer

# Screen Recording

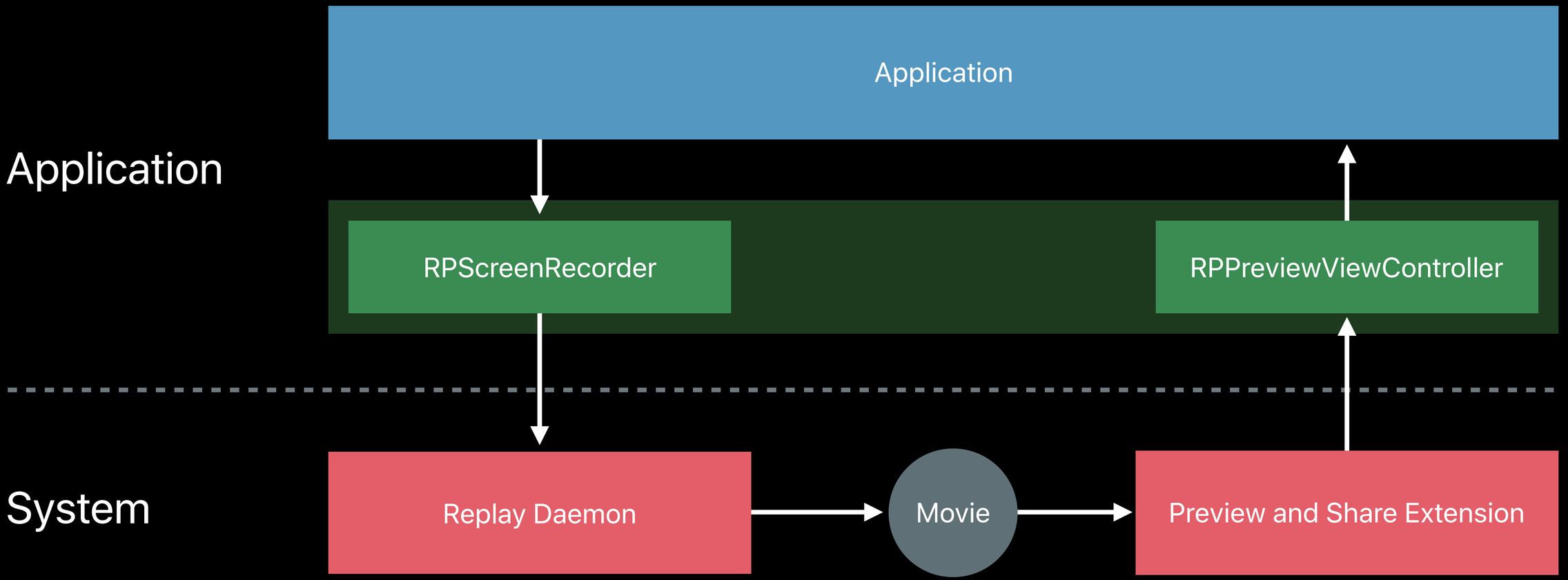
Application



# Screen Recording

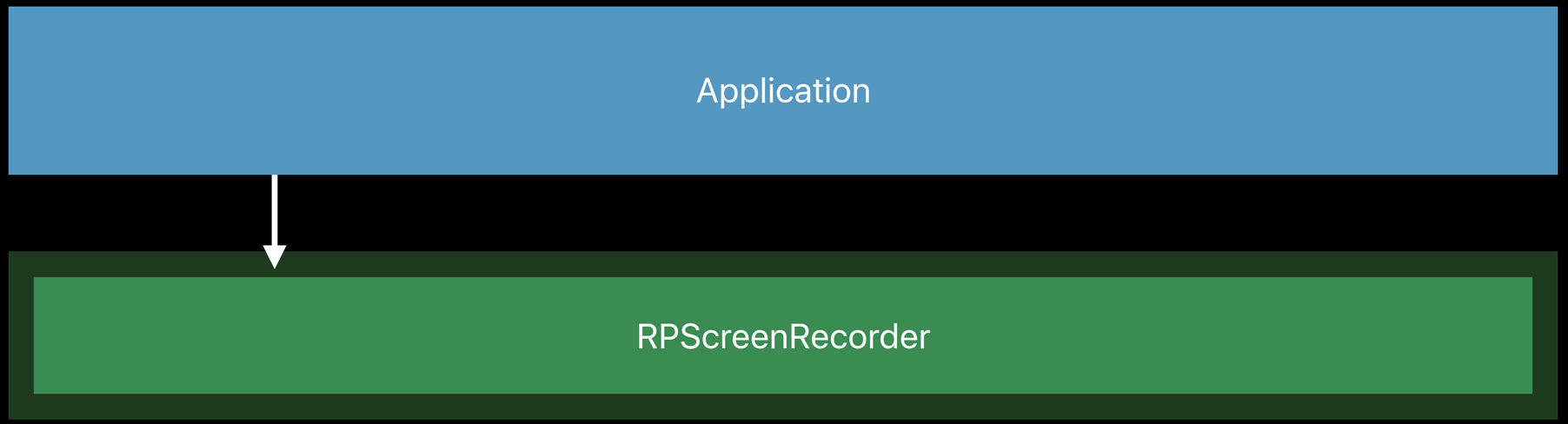


# Screen Recording

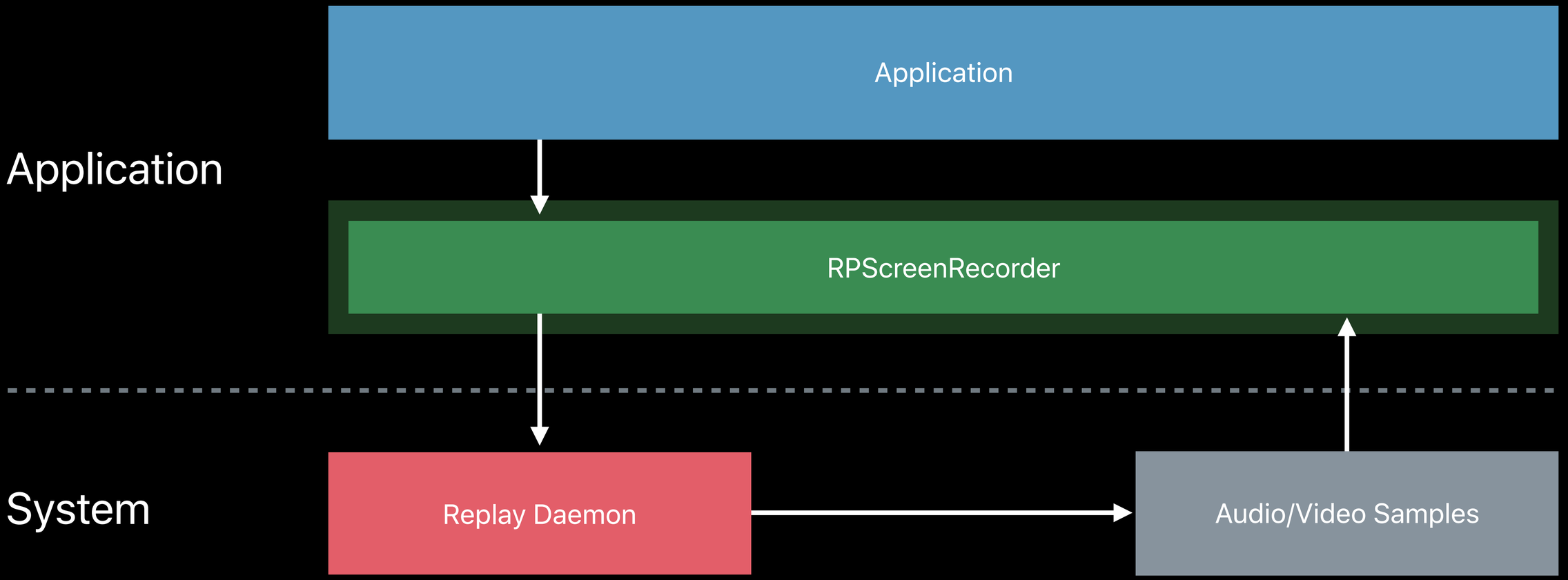


# In-App Screen Capture

Application

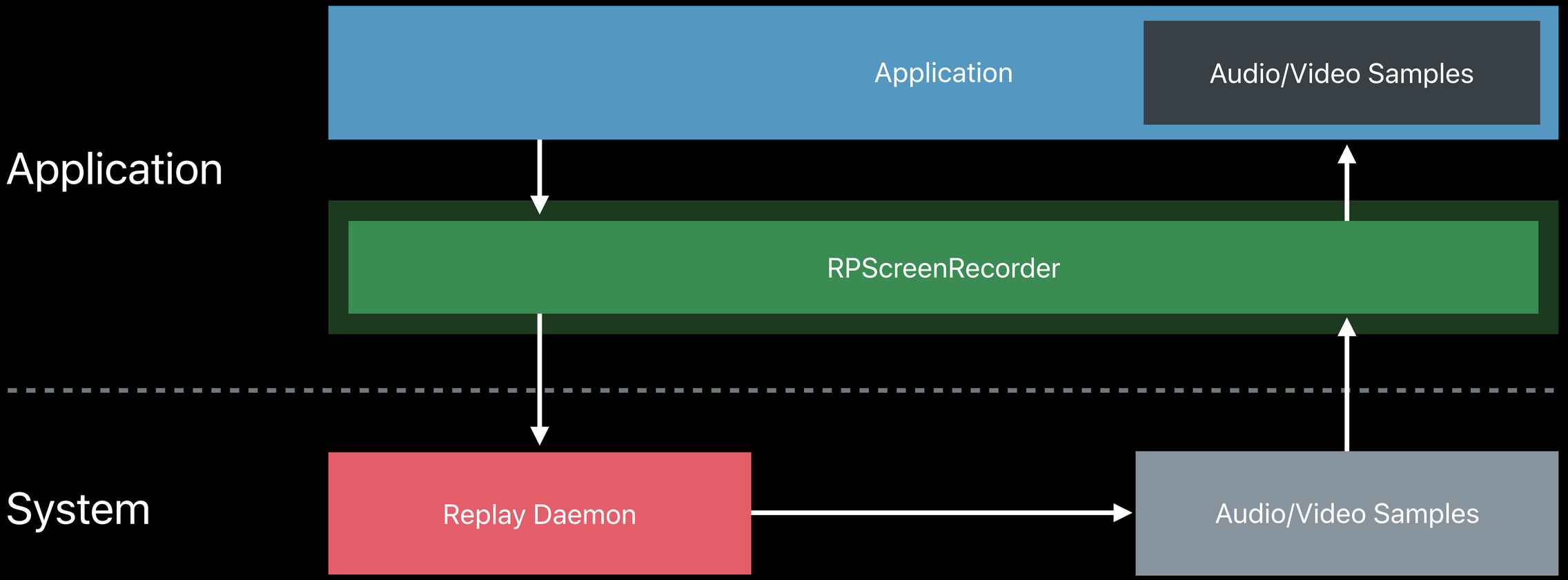


# In-App Screen Capture



# In-App Screen Capture

NEW



# In-App Screen Capture

Direct access to audio and video samples

More control and flexibility

HD quality

Minimal power usage

Privacy safeguards

Simple API

```
// Capture API

func startCapture(handler captureHandler: ((CMSampleBuffer, RPSampleBufferType, Error?) ->
Void)?, completionHandler: ((Error?) -> Void)? = nil)

func stopCapture(handler: ((Error?) -> Void)? = nil)
```

```
// Initiating Capture

func didPressCaptureButton() {
    let sharedRecorder = RPScreenRecorder.shared()
    sharedRecorder.startCapture(handler: { (cmSampleBuffer, rpSampleType, error) in
        // Handle Samples Passed Back from ReplayKit
    }) { (error) in
        // Update UI
    }
}
```

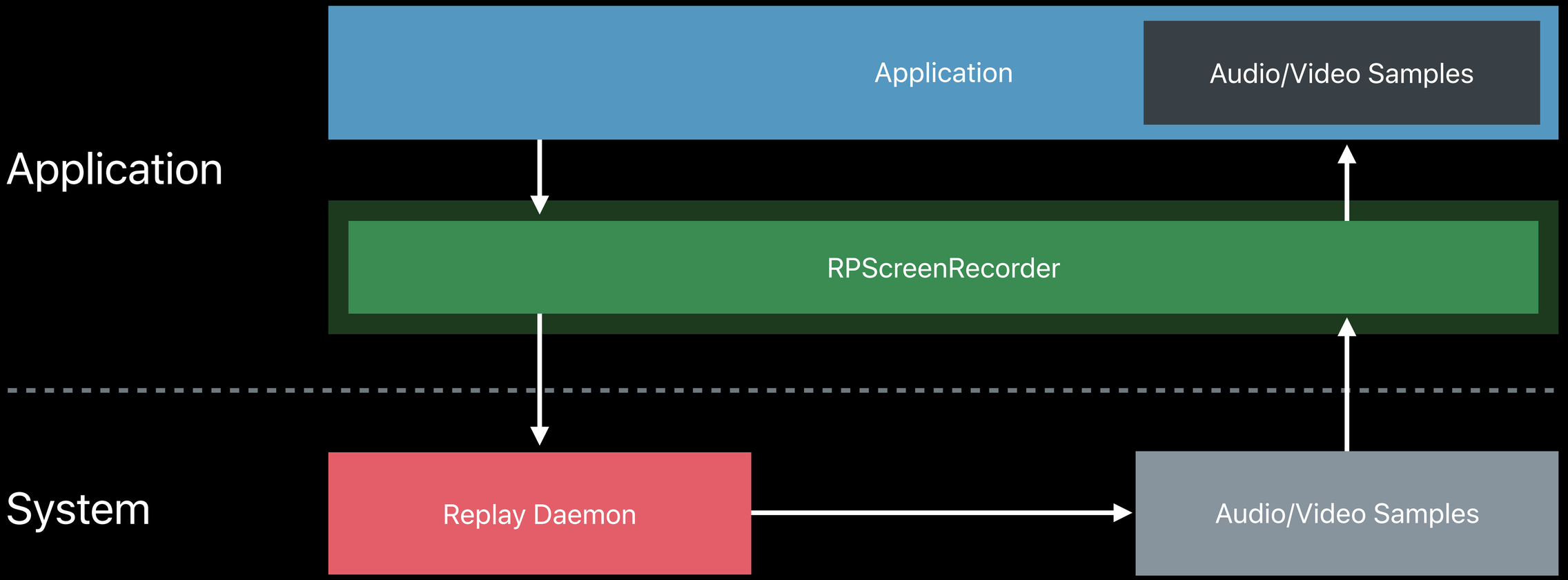
```
// Capture Handler Block

func didPressCaptureButton() {
    let sharedRecorder = RPScreenRecorder.shared()
    sharedRecorder.startCapture(handler: { (cmSampleBuffer, rpSampleType, error) in
        switch rpSampleType {
        case RPSampleBufferTypeVideo:
            self.videoInput.appendSampleBuffer(samples)
        case RPSampleBufferTypeAudio:
            self.audioInput.appendSampleBuffer(samples)
       case RPSampleBufferTypeMic:
            self.micInput.appendSampleBuffer(samples)
       default:
            println("sample has no matching type")
        }
    }) { (error) in
        updateCaptureButton(didCompleteError:error)
    }
}
```

```
// Capture Handler Block

func didPressCaptureButton() {
    let sharedRecorder = RPScreenRecorder.shared()
    sharedRecorder.startCapture(handler: { (cmSampleBuffer, rpSampleType, error) in
        switch rpSampleType {
        case RPSampleBufferTypeVideo:
            self.videoInput.appendSampleBuffer(samples)
        case RPSampleBufferTypeAudio:
            self.audioInput.appendSampleBuffer(samples)
       case RPSampleBufferTypeMic:
            self.micInput.appendSampleBuffer(samples)
       default:
            println("sample has no matching type")
        }
    }) { (error) in
        updateCaptureButton(didCompleteError:error)
    }
}
```

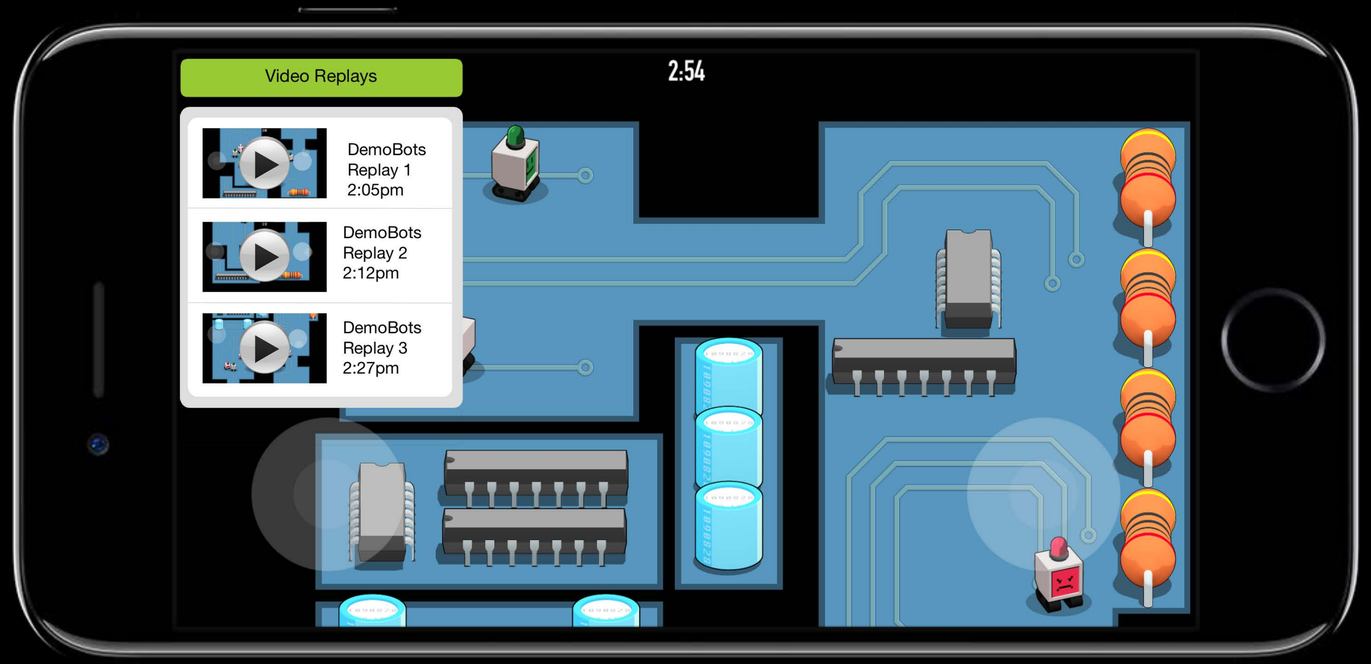
```
// Completion Handler Block

func didPressCaptureButton() {
    let sharedRecorder = RPScreenRecorder.shared()
    sharedRecorder.startCapture(handler: { (cmSampleBuffer, rpSampleType, error) in
        switch rpSampleType {
        case RPSampleBufferTypeVideo:
            self.videoInput.appendSampleBuffer(samples)
        case RPSampleBufferTypeAudio:
            self.audioInput.appendSampleBuffer(samples)
       case RPSampleBufferTypeMic:
            self.micInput.appendSampleBuffer(samples)
       default:
            println("sample has no matching type")
        }
    }) { (error) in
        updateCaptureButton(didCompleteError:error)
    }
}
```

```
// Completion Handler Block

func didPressCaptureButton() {
    let sharedRecorder = RPScreenRecorder.shared()
    sharedRecorder.startCapture(handler: { (cmSampleBuffer, rpSampleType, error) in
        switch rpSampleType {
        case RPSampleBufferTypeVideo:
            self.videoInput.appendSampleBuffer(samples)
        case RPSampleBufferTypeAudio:
            self.audioInput.appendSampleBuffer(samples)
       case RPSampleBufferTypeMic:
            self.micInput.appendSampleBuffer(samples)
       default:
            println("sample has no matching type")
        }
    }) { (error) in
        updateCaptureButton(didCompleteError:error)
    }
}
```

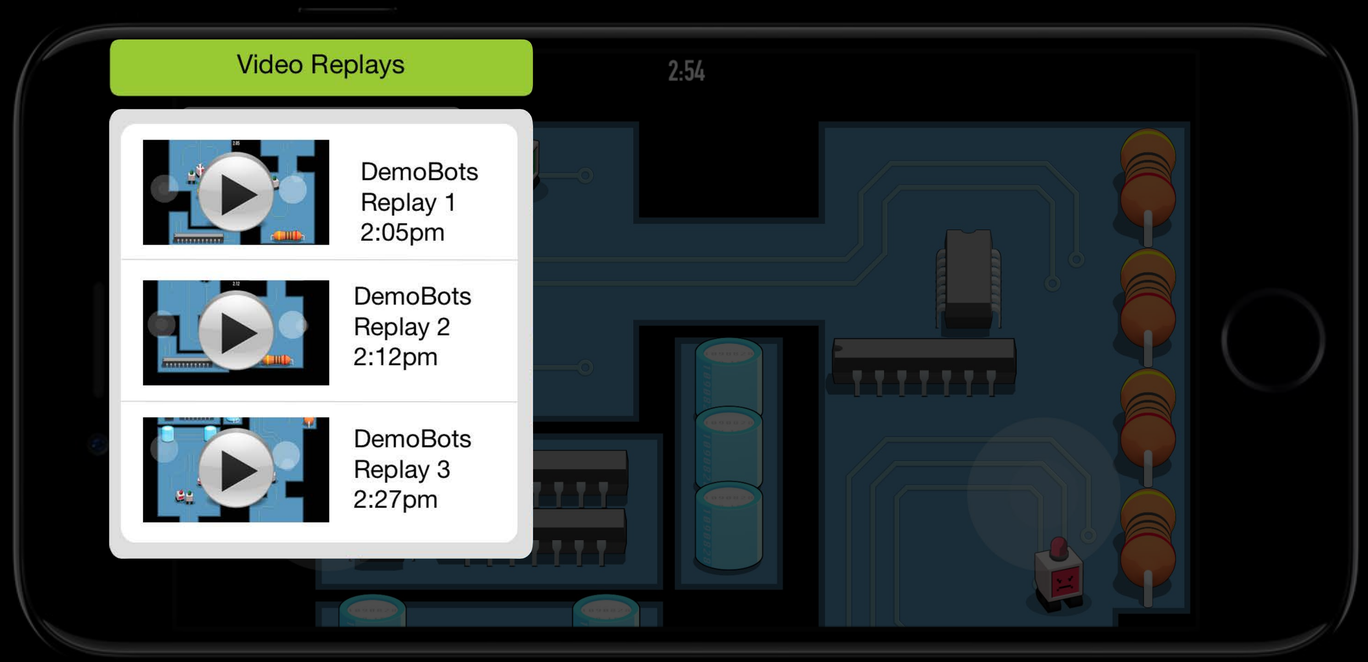
# Screen Capture Architecture



# Create and Manage Video

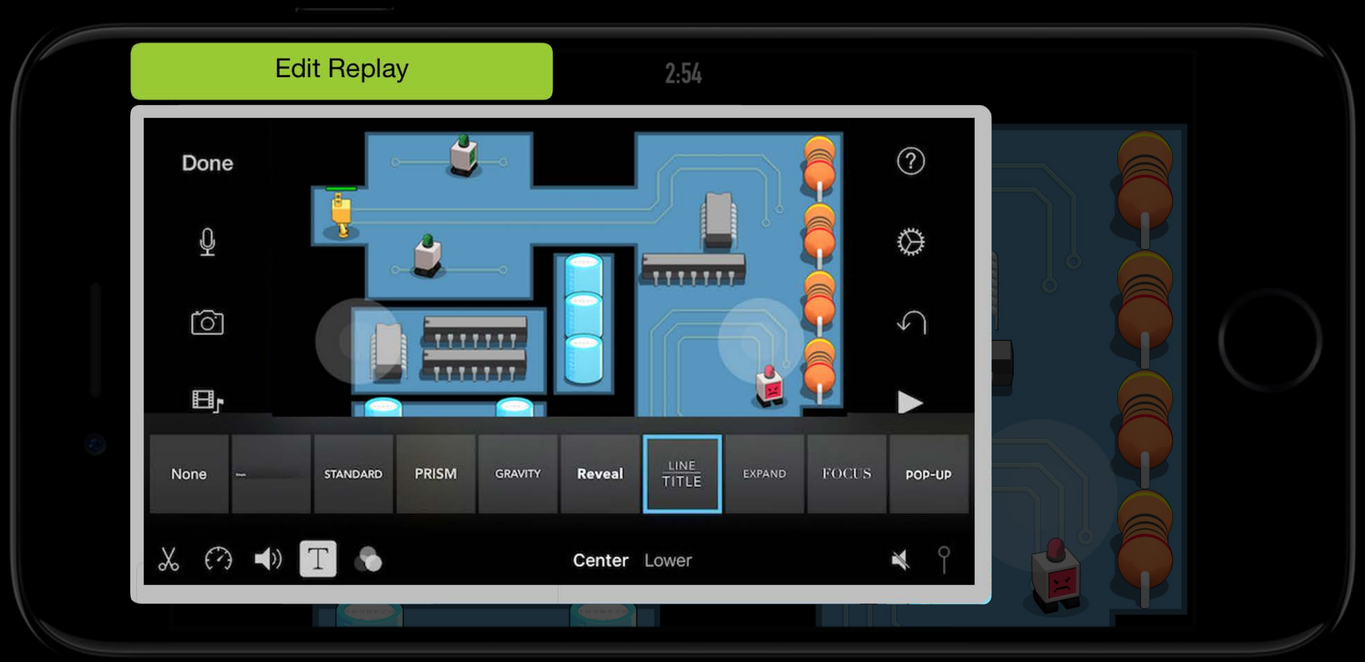


# Create and Manage Video

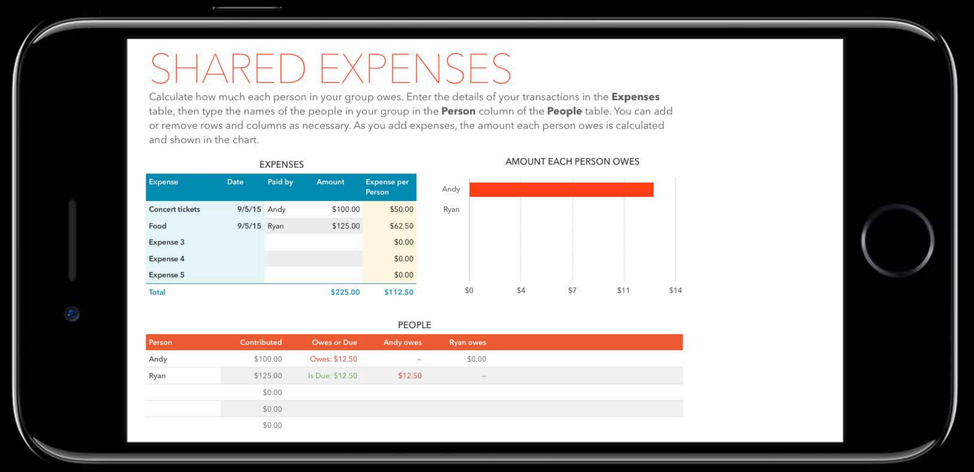


# Custom Video Editor

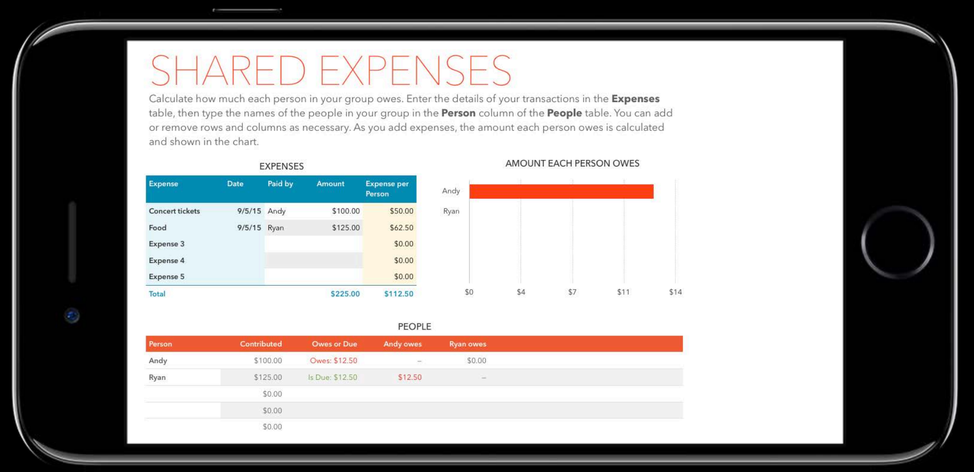
# Custom Video Editor



# Share Application Screen

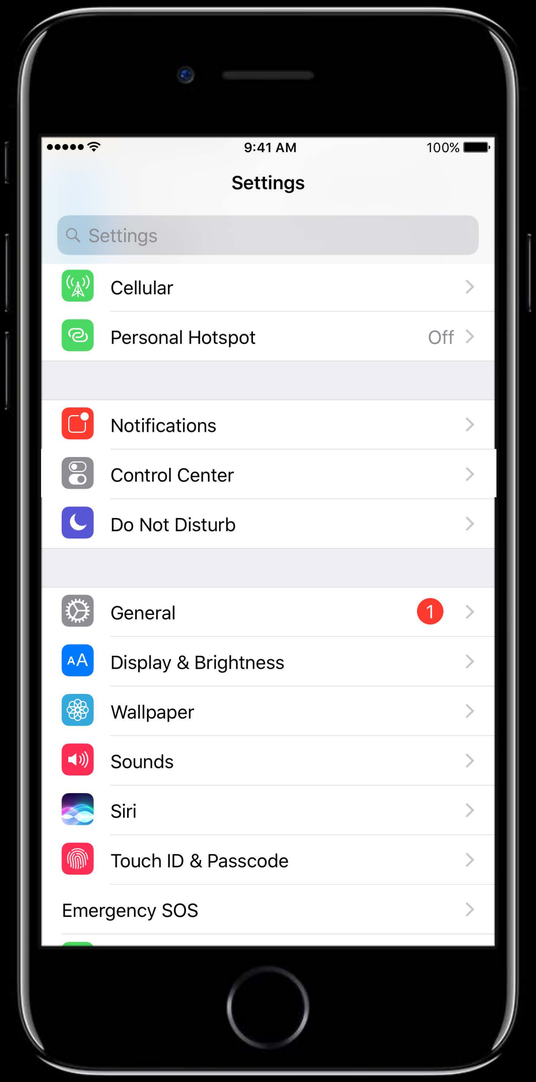



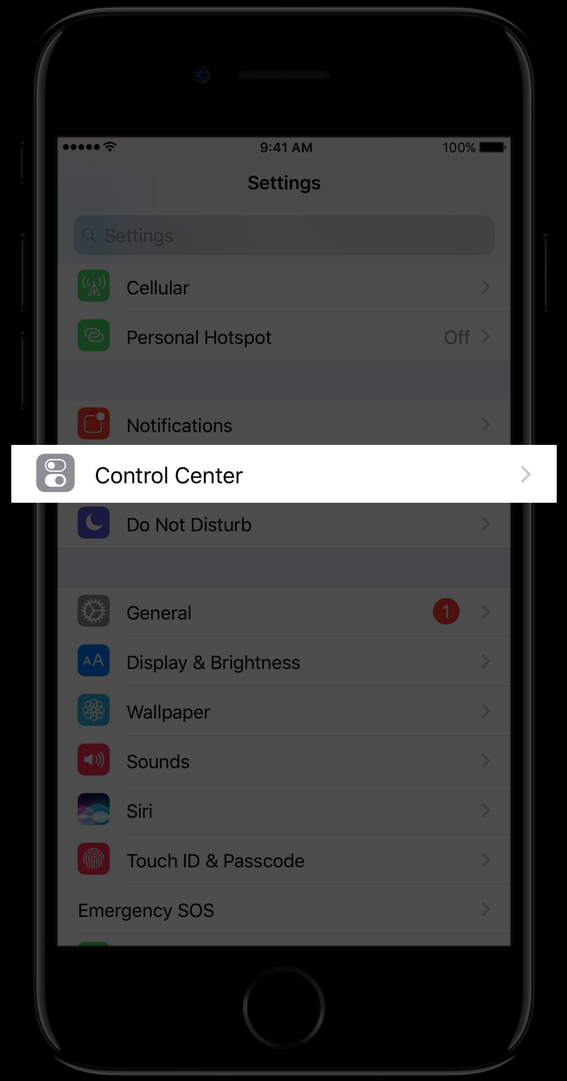
# Share Application Screen

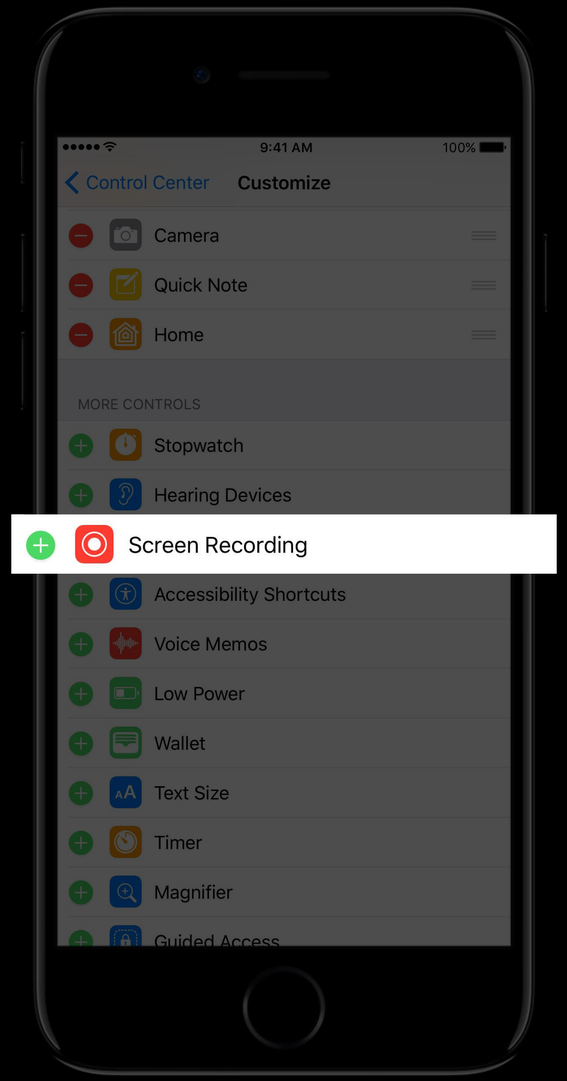
# Share Application Screen

# iOS Screen Recording and Broadcast


9:41 AM
100%
Settings
Settings
Cellular
Personal Hotspot
Off
Notifications
Control Center
Do Not Disturb
General
1
Display & Brightness
Wallpaper
Sounds
Siri
Touch ID & Passcode
Emergency SOS


9:41 AM
100%
Settings
Settings
Cellular
Personal Hotspot
Off
Notifications
Control Center
Do Not Disturb
General
1
Display & Brightness
Wallpaper
Sounds
Siri
Touch ID & Passcode
Emergency SOS


9:41 AM
100%
Control Center
Customize
Camera
Quick Note
Home
MORE CONTROLS
Stopwatch
Hearing Devices
Screen Recording
Accessibility Shortcuts
Voice Memos
Low Power
Wallet
Text Size
Timer
Magnifier
Guided Access


9:41 AM
100%
Control Center
Customize
Camera
Quick Note
Home
MORE CONTROLS
Stopwatch
Hearing Devices
Screen Recording
Accessibility Shortcuts
Voice Memos
Low Power
Wallet
Text Size
Timer
Magnifier
Guided Access

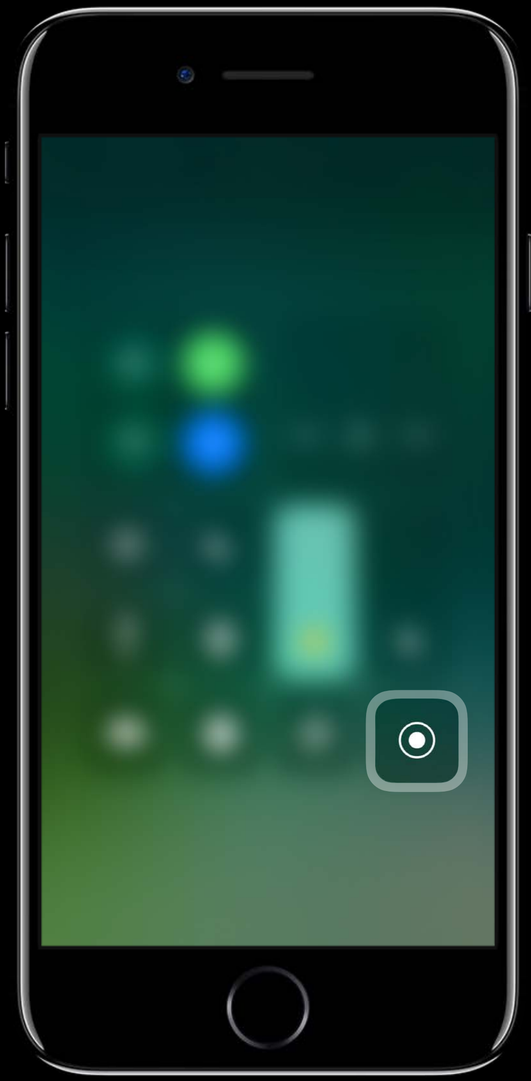

9:41 AM
100%


9:41 AM
100%


9:41 AM
100%
Recording 0:04


SCREEN RECORDING
now
Screen Recording video saved to Photos

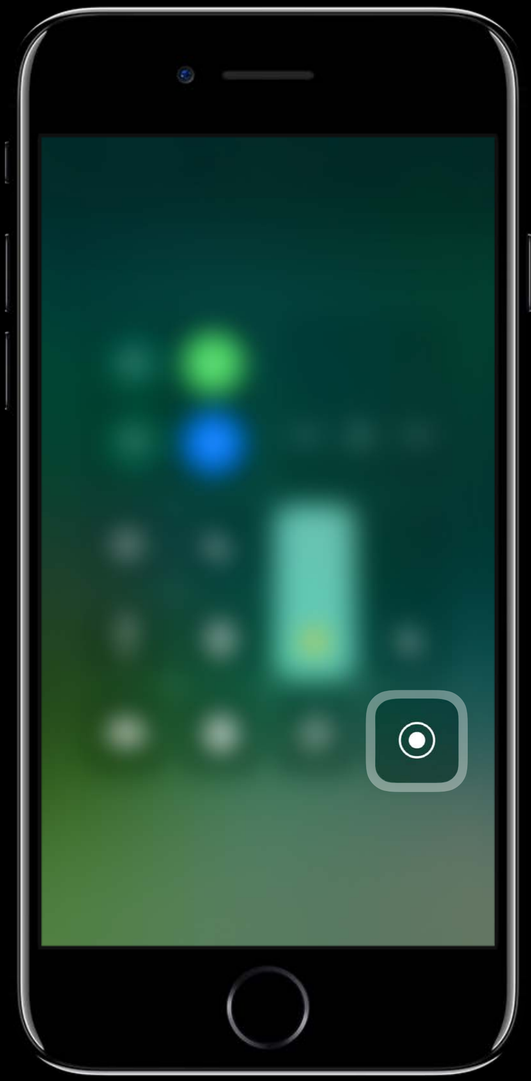

9:41 AM
100%




9:41 AM
100%
Screen Recording
Start Recording
Microphone Audio
Off


9:41 AM
100%
Recording 0:06
Screen Recording
Stop Recording
Microphone Audio
Off


9:41 AM
100%




9:41 AM
100%
Screen Recording
Everything on your screen, including notifications, will be recorded.
Camera Roll
Mobcrush
Start Broadcast
Microphone Audio
On


9:41 AM
100%
Mobcrush is Broadcasting Your Screen
Screen Recording
Everything on your screen, including notifications, will be recorded.
Camera Roll
Mobcrush
0:06
Stop Broadcast
Microphone Audio
On

# Handling Interruptions

In-App Recording can be interrupted by iOS Screen Record and Broadcast

Recordings will be discarded if interrupted by iOS Screen Record and Broadcast

Application will get delegate call with appropriate RPError

# Live Broadcast

Alexander Subbotin, Software Engineer

# Live Broadcast

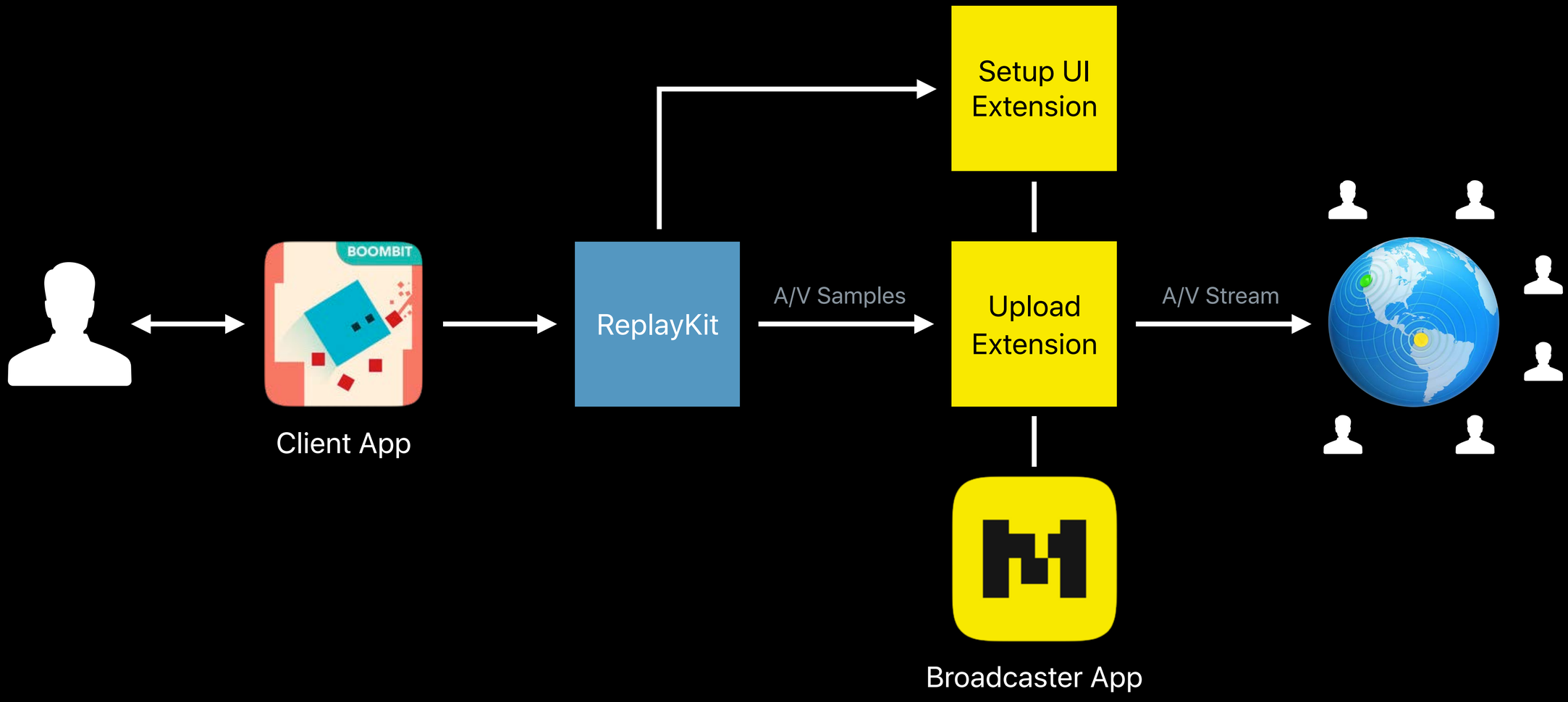
Broadcast app visuals and audio

iOS and tvOS

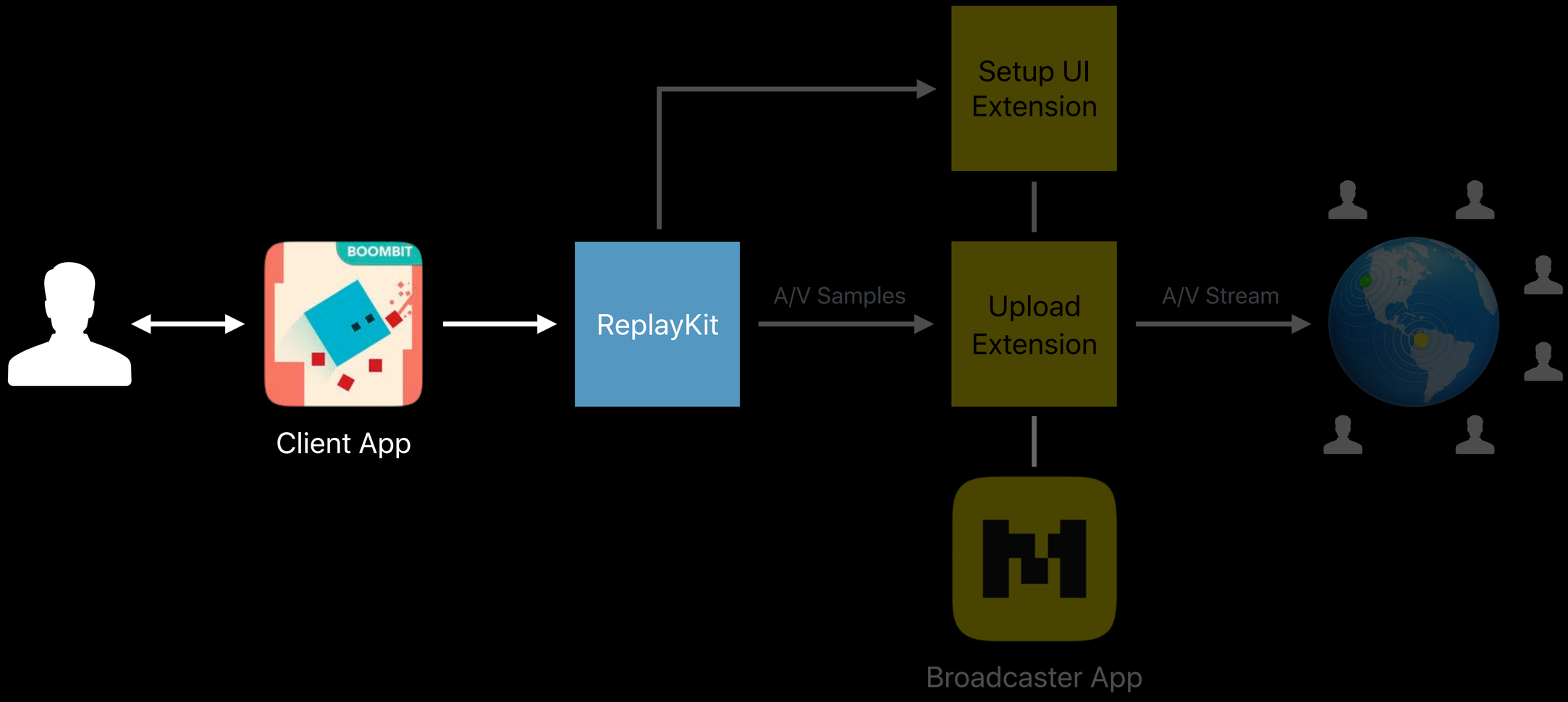
Can include microphone, camera feed

Content is secure

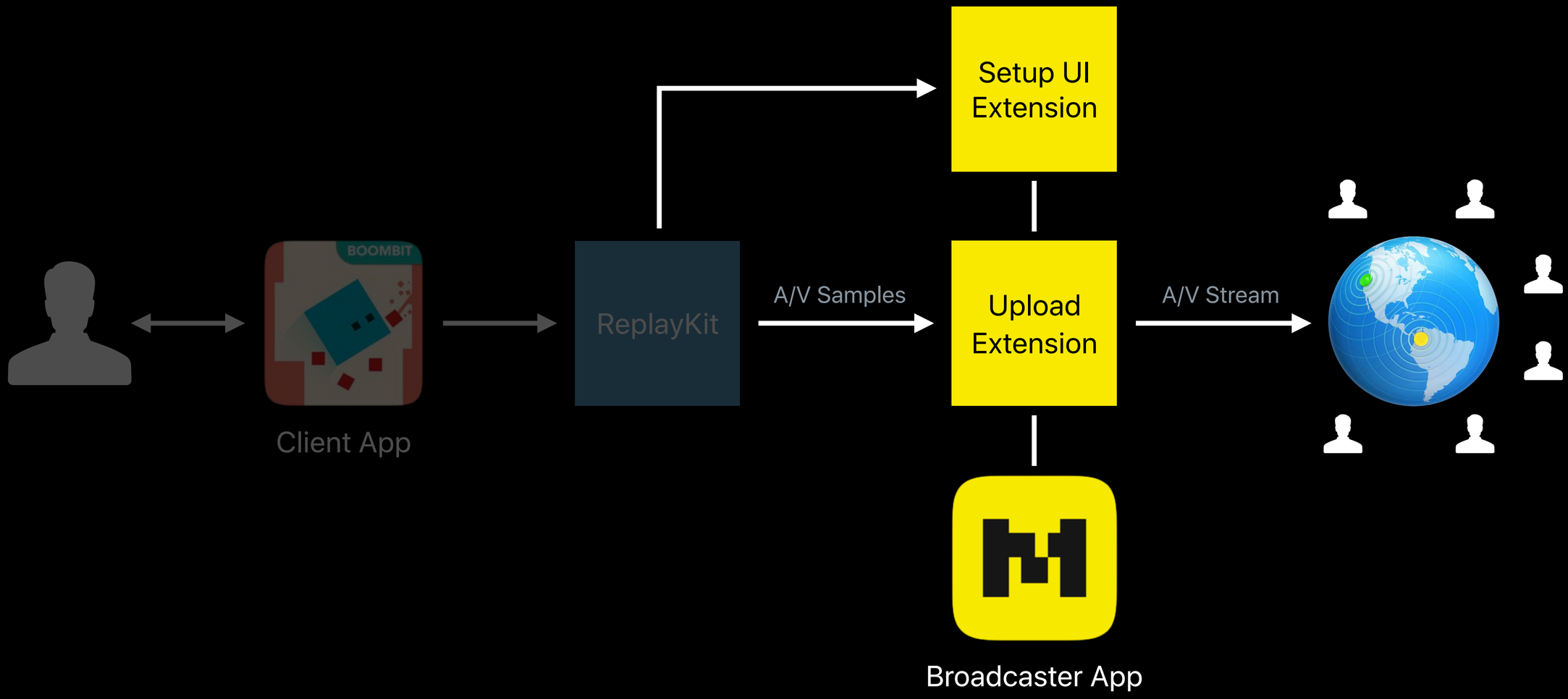
# Live Broadcast



# Live Broadcast



# Live Broadcast



# Broadcast API

## App Client

RPBroadcastActivityViewController

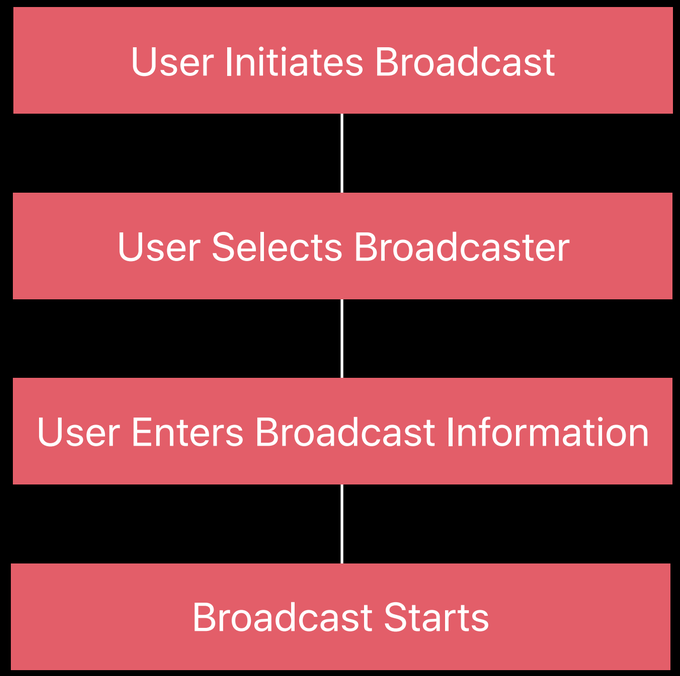
- Allow user to select broadcaster

RPBroadcastController

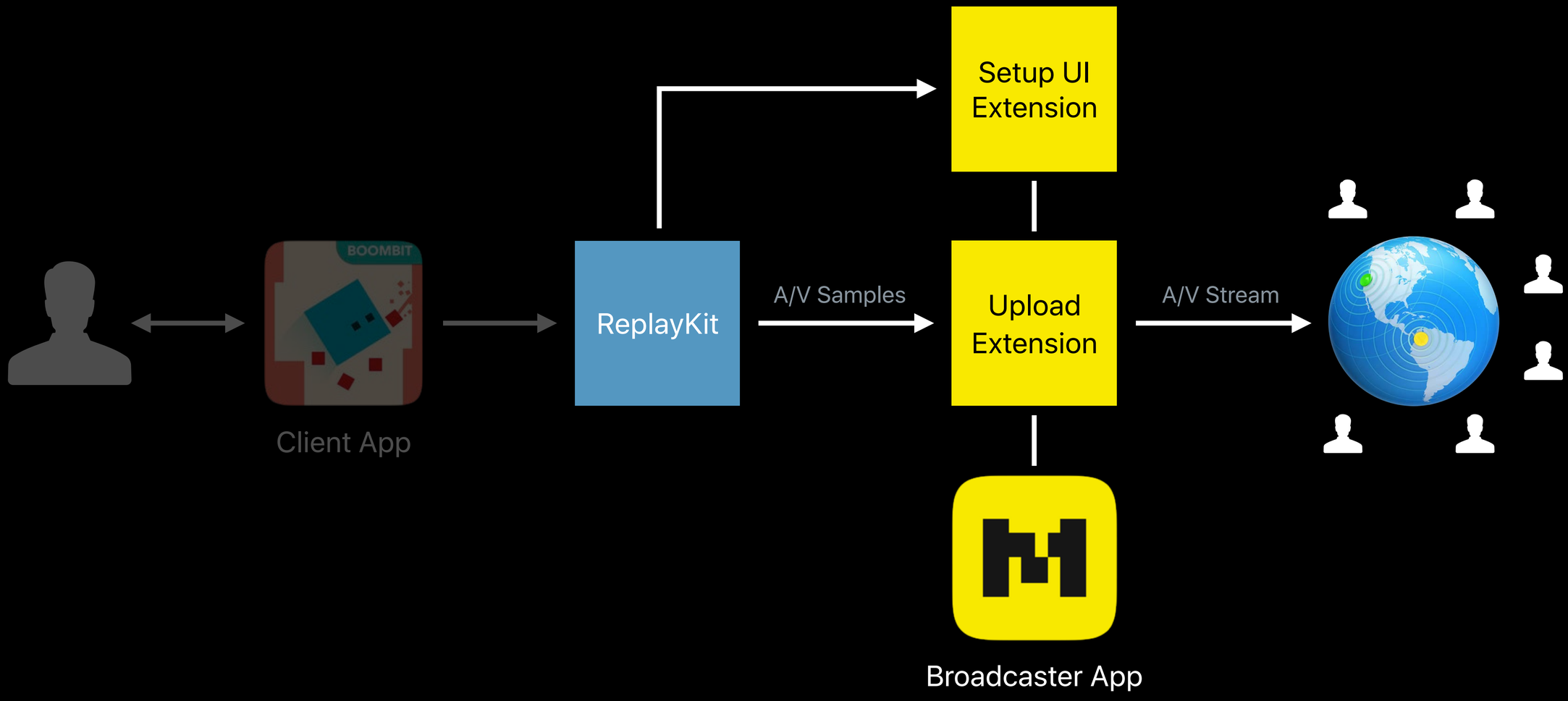
- Manage broadcast (start, stop, pause, resume)

RPBroadcastControllerDelegate

- Handle broadcast events



# Live Broadcast



# Broadcast API

## App extensions

Broadcast Setup UI extension

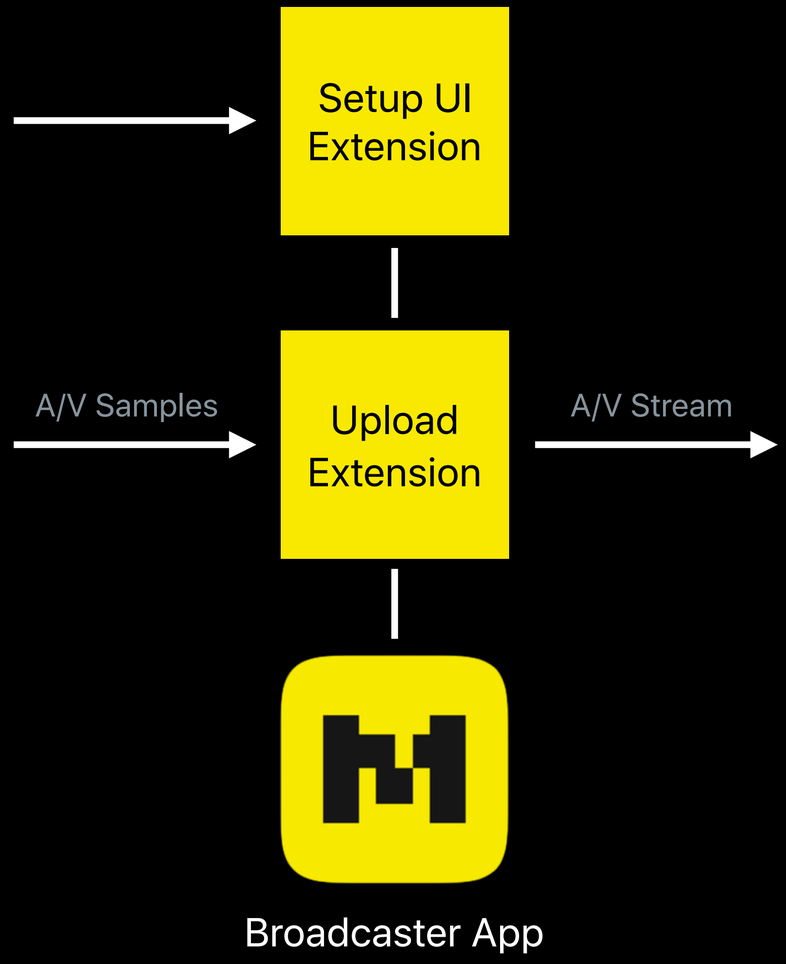
- Account sign-in, broadcast title

Broadcast Upload extension

- Encode samples, upload to service

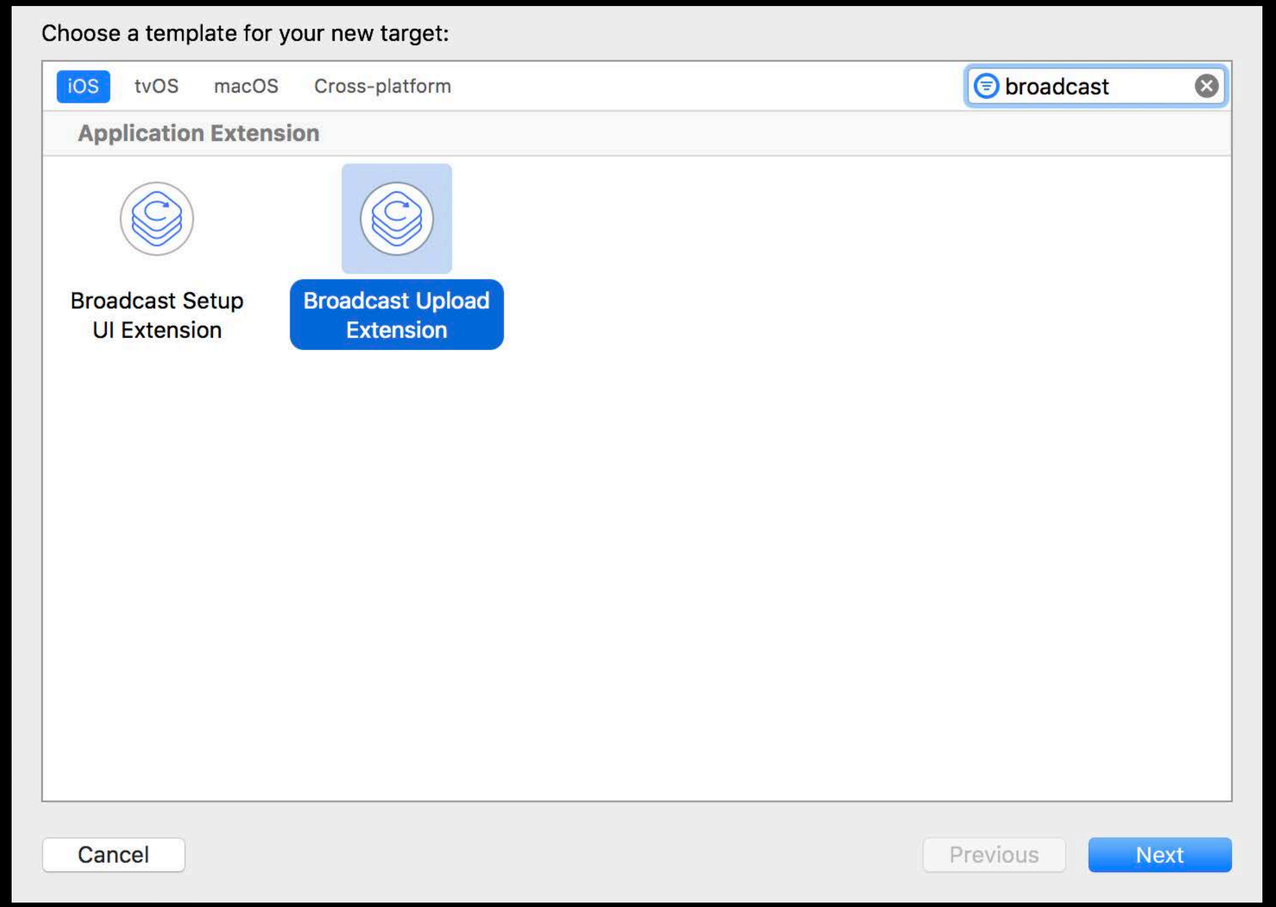
Each runs in its own process

Installed with broadcaster’s app



# Broadcast API

## Xcode templates

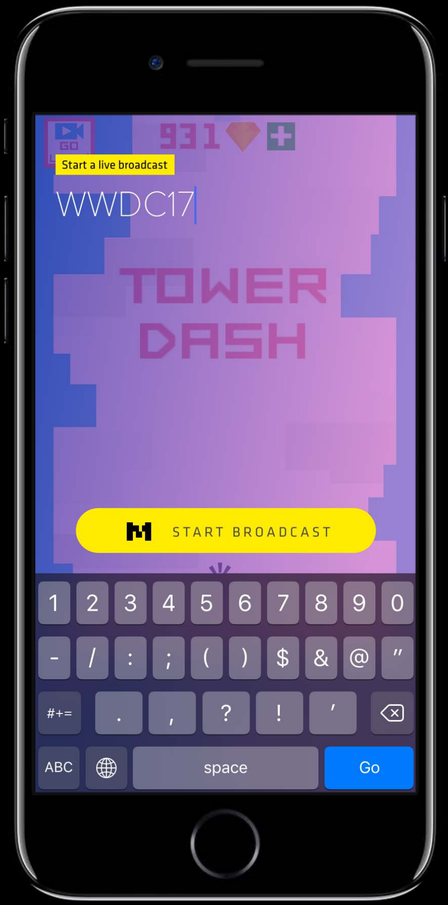


# Broadcast SetupUI Extension

UI to login, get broadcast name and other

Uploads name and icon of the app

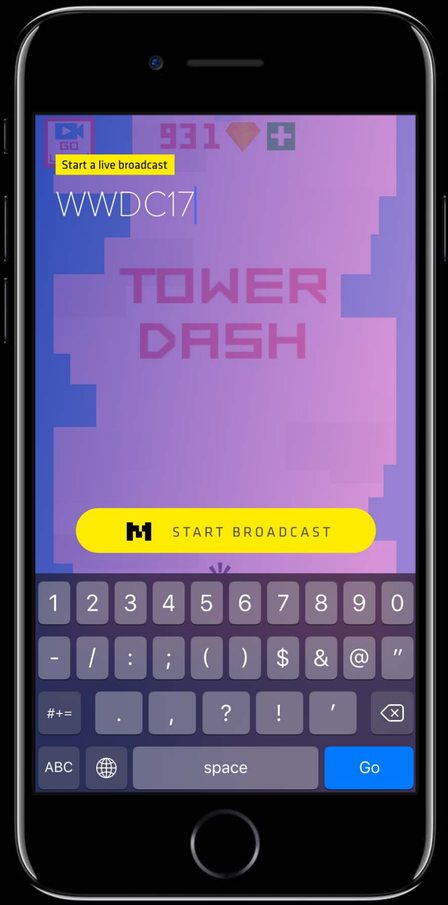
Provides broadcast URL back to client app



# Broadcast SetupUI Extension

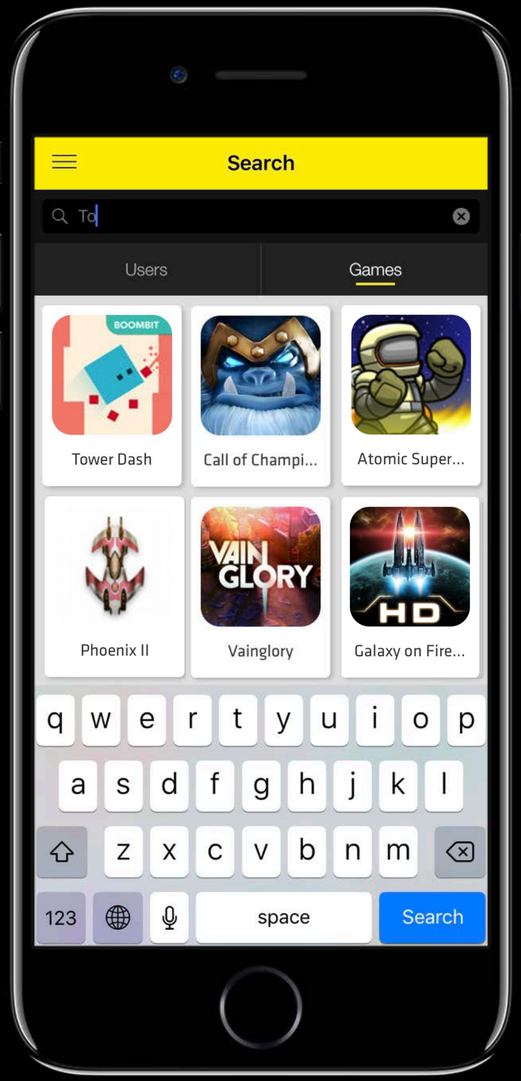
NSExtensionContext(RPBroadcastExtension)

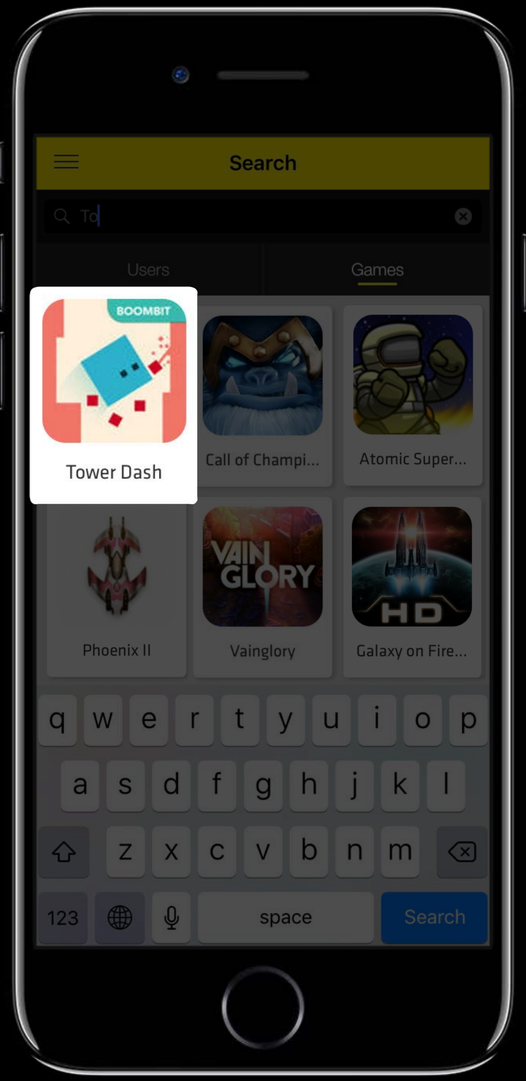
- loadBroadcastingApplicationInfo
- completeRequest(withBroadcast)



```swift
// Get name and icon for client application and provide to the broadcast service
class BroadcastSetupViewController: UIViewController {
    override func viewDidLoad() {
        super.viewDidLoad()
        extensionContext?.loadBroadcastingApplicationInfo(completion: {
           (bundleID, displayName, appIcon) in
            broadcastSession.setAppInfo(bundleID, displayName, appIcon)
        })
    }
}
```

```swift
// Get name and icon for client application and provide to the broadcast service
class BroadcastSetupViewController: UIViewController {
    override func viewDidLoad() {
        super.viewDidLoad()
        extensionContext?.loadBroadcastingApplicationInfo(completion: {
           (bundleID, displayName, appIcon) in
            broadcastSession.setAppInfo(bundleID, displayName, appIcon)
        })
    }
}
```


Search
To
Users
Games
Tower Dash
Call of Champi...
Atomic Super...
Phoenix II
Vainglory
Galaxy on Fire...
q w e r t y u i o p
a s d f g h j k l
z x c v b n m
123
space
Search


Search
To
Users
Games
BOOMBIT
Tower Dash
Call of Champi...
Atomic Super...
Phoenix II
VAIN GLORY
Vainglory
HD
Galaxy on Fire...
q w e r t y u i o p
a s d f g h j k l
z x c v b n m
123
space
Search

```swift
// Complete setup extension request with broadcastURL and setupInfo
class BroadcastSetupViewController: UIViewController {
    func done() {
        let broadcastURL = URL(string:”http://myCompany.com/broadcast/streamID")

        let setupInfo: [String : NSCoding & NSObjectProtocol] =
               ["broadcastName": "example" as NSCoding & NSObjectProtocol]

        extensionContext?.completeRequest(withBroadcast: broadcastURL!, setupInfo: setupInfo)
    }
}
```

```swift
// Complete setup extension request with broadcastURL and setupInfo
class BroadcastSetupViewController: UIViewController {
    func done() {
        let broadcastURL = URL(string:"http://myCompany.com/broadcast/streamID")

        let setupInfo: [String : NSCoding & NSObjectProtocol] =
               ["broadcastName": "example" as NSCoding & NSObjectProtocol]

        extensionContext?.completeRequest(withBroadcast: broadcastURL!, setupInfo: setupInfo)
    }
}
```

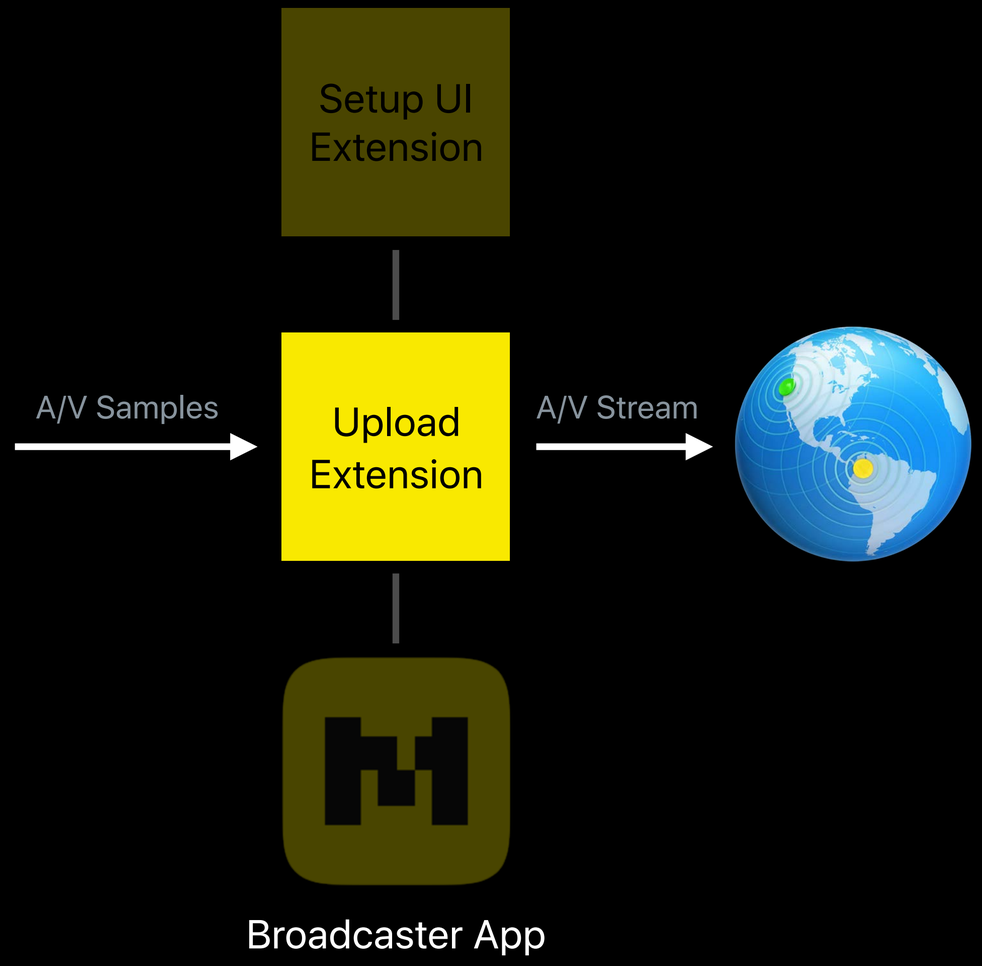
```swift
// You should always provide option to cancel broadcast
class BroadcastSetupViewController: UIViewController {
    func cancel() {
        let error = NSError(domain: "broadcast", code: -1, userInfo: nil)
        extensionContext?.cancelRequest(withError: error)
    }
}
```

```
// You should always provide option to cancel broadcast
class BroadcastSetupViewController: UIViewController {
    func cancel() {
        let error = NSError(domain: "broadcast", code: -1, userInfo: nil)
        extensionContext?.cancelRequest(withError: error)
    }
}
```

# Broadcast Upload Extension

Receives audio and video samples

Encodes and uploads stream to broadcaster



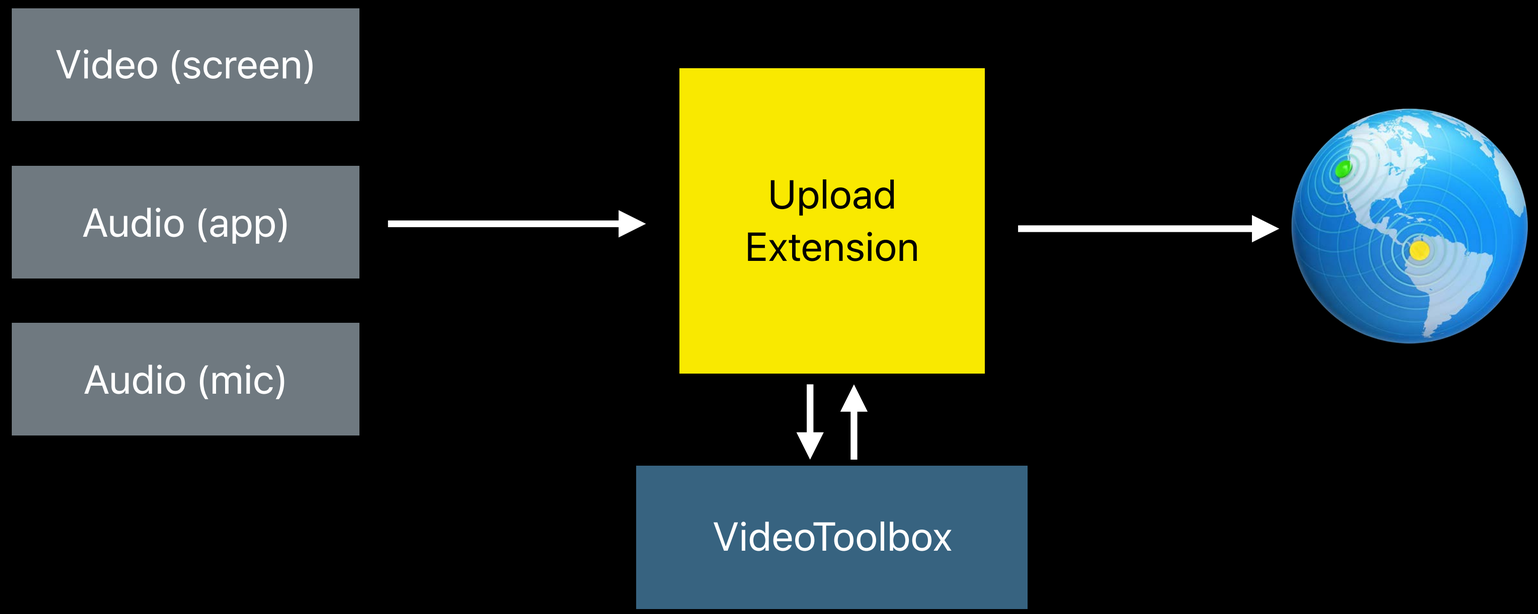
```swift
// SampleHandler created by Xcode templates for Upload Extension
class SampleHandler: RPBroadcastSampleHandler {

    override func broadcastStarted(withSetupInfo setupInfo: [String : NSObject]?) {
        // User has requested to start the broadcast
    }

    override func broadcastPaused() {
        // User has requested to pause the broadcast. Samples will stop being delivered.
    }

    override func broadcastResumed() {
        // User has requested to resume the broadcast. Samples delivery will resume.
    }

    override func broadcastFinished() {
        // User has requested to finish the broadcast
    }

    override func processSampleBuffer(_ sampleBuffer: CMSampleBuffer,
                                      with sampleBufferType: RPSampleBufferType) {
        // Handle the sample buffer here
    }
}
```

```swift
// Override broadcastStarted to prepare to receive media samples
override func broadcastStarted(withSetupInfo setupInfo: [String : NSObject]?) {
        if (setupInfo != nil) {
            session.broadcastDescription.name = setupInfo["name"]
        } else {
            session.broadcastDescription.iOSScreenBroadcast = true
        }
}
```

```
// Override broadcastStarted to prepare to receive media samples
override func broadcastStarted(withSetupInfo setupInfo: [String : NSObject]?) {
        if (setupInfo != nil) {
            session.broadcastDescription.name = setupInfo["name"]
        } else {
            session.broadcastDescription.iOSScreenBroadcast = true
        }
}
```

# Broadcast Upload Extension

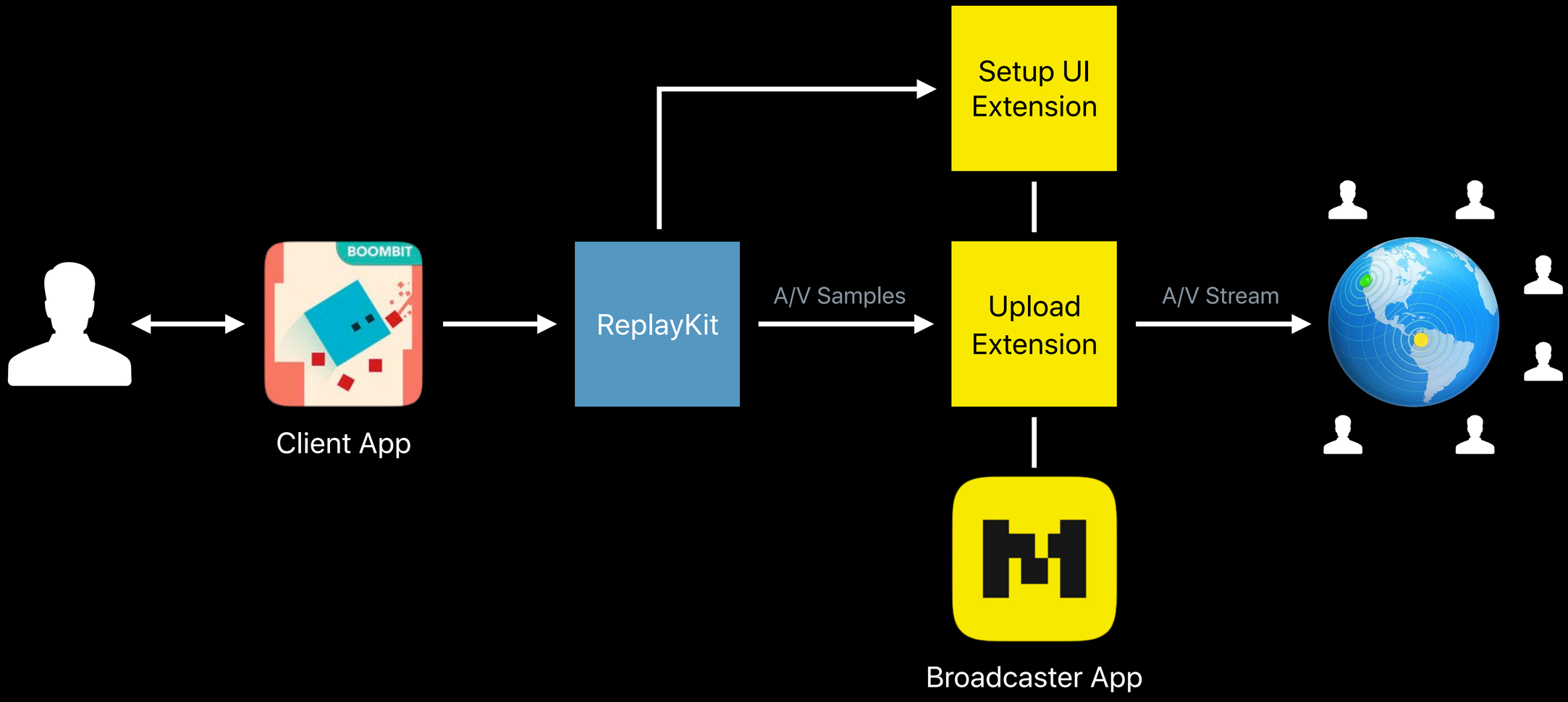
processSampleBuffer



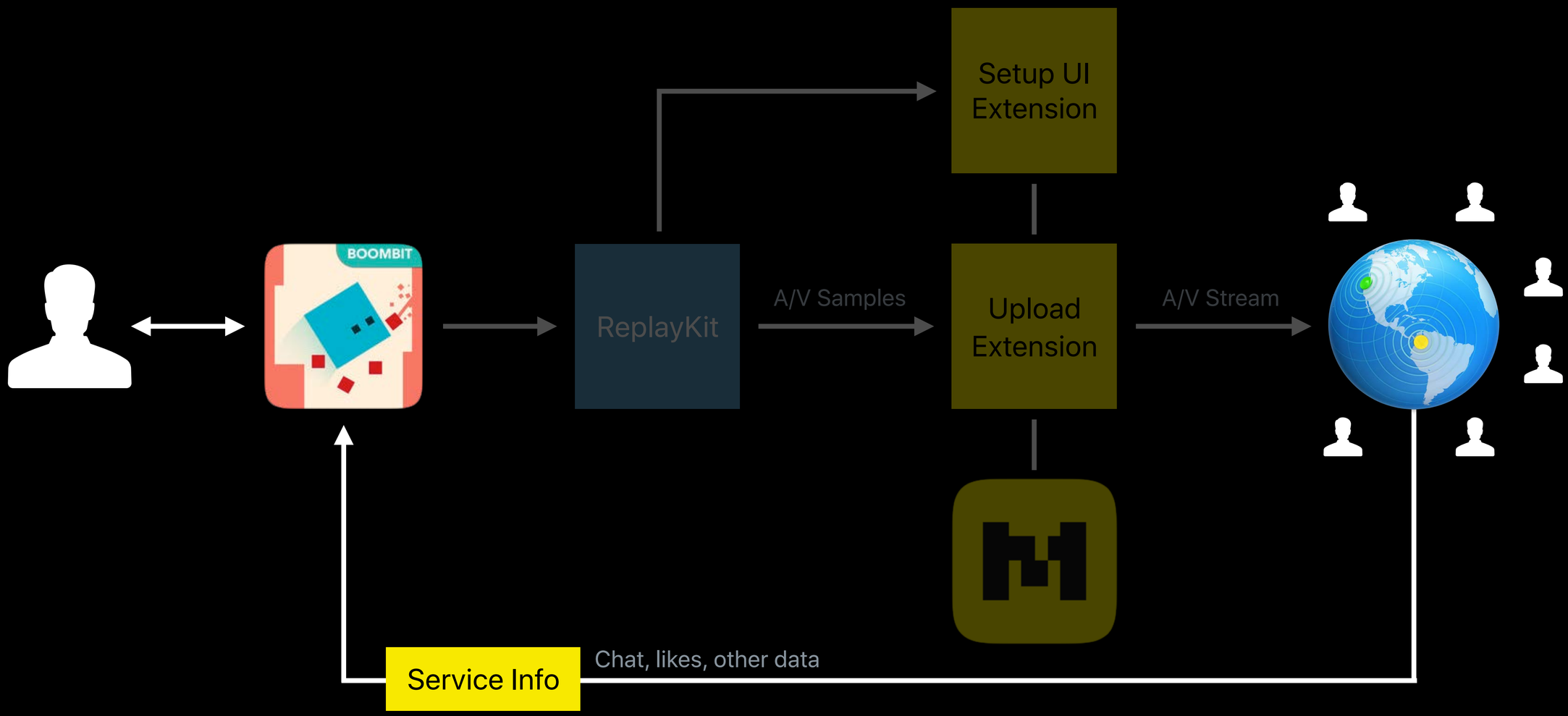
```swift
// Both audio and video samples are handled by processSampleBuffer routine
override func processSampleBuffer(_ sampleBuffer: CMSampleBuffer,
                                  with sampleBufferType: RPSampleBufferType) {
    switch sampleBufferType {
        case RPSampleBufferType.video:
            var imageBuffer:CVImageBuffer = CMSampleBufferGetImageBuffer(sampleBuffer)!
            var pts = CMSampleBufferGetPresentationTimeStamp(sampleBuffer) as CMTime
            VTCompressionSessionEncodeFrame(session, imageBuffer, pts,
                                            kCMTimeInvalid, nil, nil, nil)
            break
        case RPSampleBufferType.audioApp:
            // Handle audio sample buffer for app audio
            break
        case RPSampleBufferType.audioMic:
            // Handle audio sample buffer for mic audio
            break
    }
}
```

```swift
// Both audio and video samples are handled by processSampleBuffer routine
override func processSampleBuffer(_ sampleBuffer: CMSampleBuffer,
                                  with sampleBufferType: RPSampleBufferType) {
    switch sampleBufferType {
        case RPSampleBufferType.video:
            var imageBuffer:CVImageBuffer = CMSampleBufferGetImageBuffer(sampleBuffer)!
            var pts = CMSampleBufferGetPresentationTimeStamp(sampleBuffer) as CMTime
            VTCompressionSessionEncodeFrame(session, imageBuffer, pts,
                                            kCMTimeInvalid, nil, nil, nil)
            break
        case RPSampleBufferType.audioApp:
            // Handle audio sample buffer for app audio
            break
        case RPSampleBufferType.audioMic:
            // Handle audio sample buffer for mic audio
            break
    }
}
```

# Live Broadcast



# Service Information



# Service Information

Information provided by the service during broadcast

- Dictionary
- KVO observable

Can use to communicate service data back to user

- Likes, viewer count, and chat

```
class SampleHandler: RPBroadcastSampleHandler {
    func updateViewersCount(_ count: UInt) {
        updateServiceInfo(["count": (count)])
    }
}
```

# Broadcast Pairing

Johnny Trenh, Software Engineer

# Broadcast Pairing Flow


Budget App

Broadcast Extension


Conference App

# Broadcast Pairing Flow

Application

RPBroadcastActivity
ViewController


Budget App

Broadcast
Extension

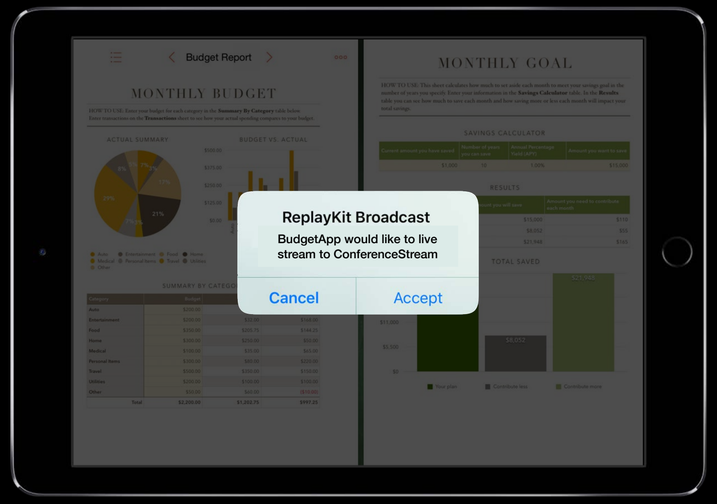

Conference App

# Broadcast Pairing Flow

NEW

Application

RPBroadcastActivity
ViewController




Budget App

Broadcast
Extension

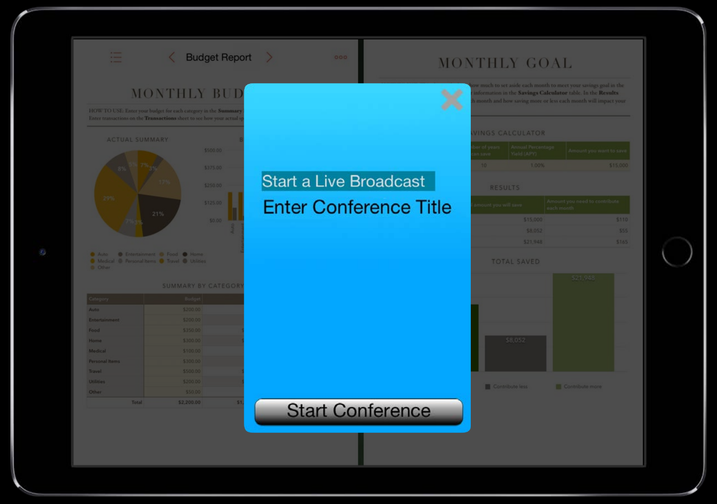

Conference App

# Broadcast Pairing Flow

Application

RPBroadcastActivity ViewController

User Enters Broadcast Information




Budget App

Broadcast Extension


Conference App

# Broadcast Pairing Flow

Application

RPBroadcastActivity ViewController

User Enters Broadcast Information

Broadcast Starts


Budget App

Broadcast Extension

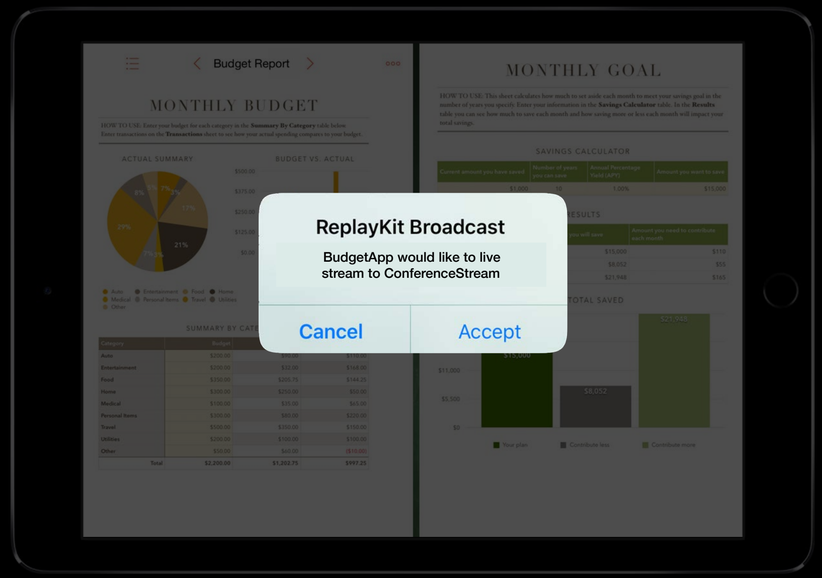

Conference App

```
// Broadcast Pairing API

class func load(withPreferredExtension preferredExtension: String?, handler: @escaping
(RPBroadcastActivityViewController?, Error?) -> Void)
```

# Initiating Broadcast Pairing

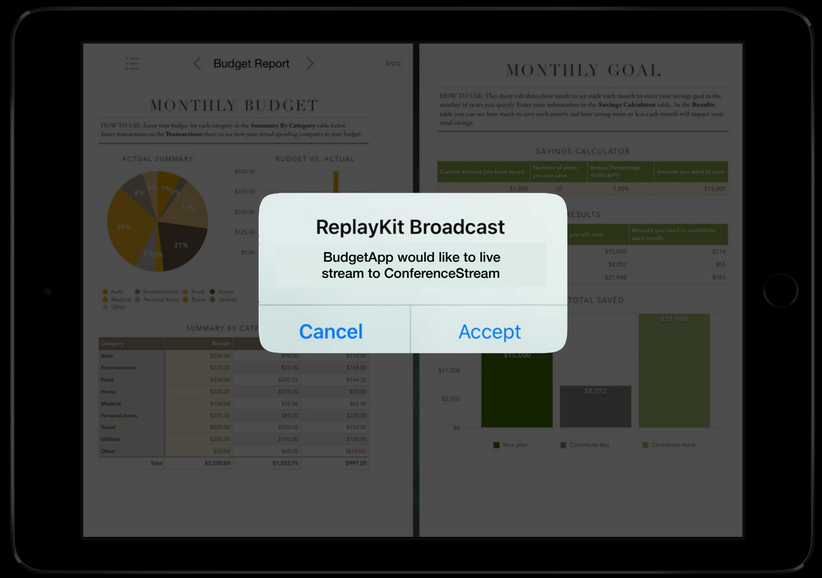

```
func didPressBroadcastPairButton () {

RPBroadcastActivityViewController.load(withPreferredExtension:"com.conferenceApp.broadcastExten
stion") { (broadcastAVC, error) in
            broadcastAVC?.delegate = self
            self.present(broadcastAVC, animated: true, completion: nil)
        }
}
```

# Initiating Broadcast Pairing




```
func didPressBroadcastPairButton () {

RPBroadcastActivityViewController.load(withPreferredExtension:"com.conferenceApp.broadcastExten
stion") { (broadcastAVC, error) in
            broadcastAVC?.delegate = self
            self.present(broadcastAVC, animated: true, completion: nil)
        }
}
```

# Initiating Broadcast Pairing




```
func didPressBroadcastPairButton () {

RPBroadcastActivityViewController.load(withPreferredExtension:"com.conferenceApp.broadcastExten
stion") { (broadcastAVC, error) in
            broadcastAVC?.delegate = self
            self.present(broadcastAVC, animated: true, completion: nil)
        }
}
```

# Broadcast Pairing

Application provides extension bundleID

User approves extension

# Fast Camera Switching

# Fast Camera Switching

Front Camera and Back Camera switching

Camera preview view available in RPScreenRecorder

Subclass of UIView

Developer is responsible for UI elements for fast switching

# Fast Camera Switching

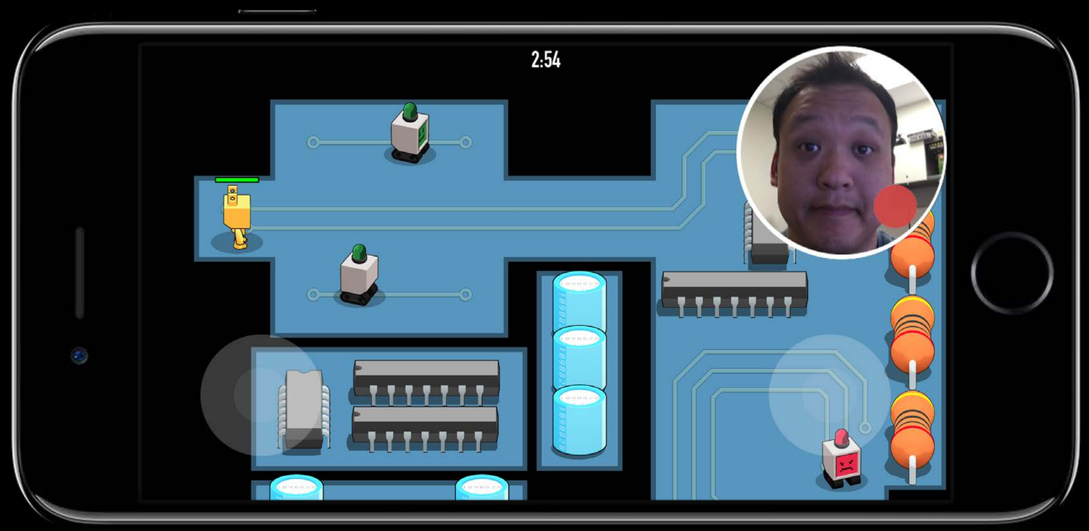
## RPScreenRecorder.cameraPosition

```
var cameraPosition: RPCameraPosition
```

## RPCameraPosition

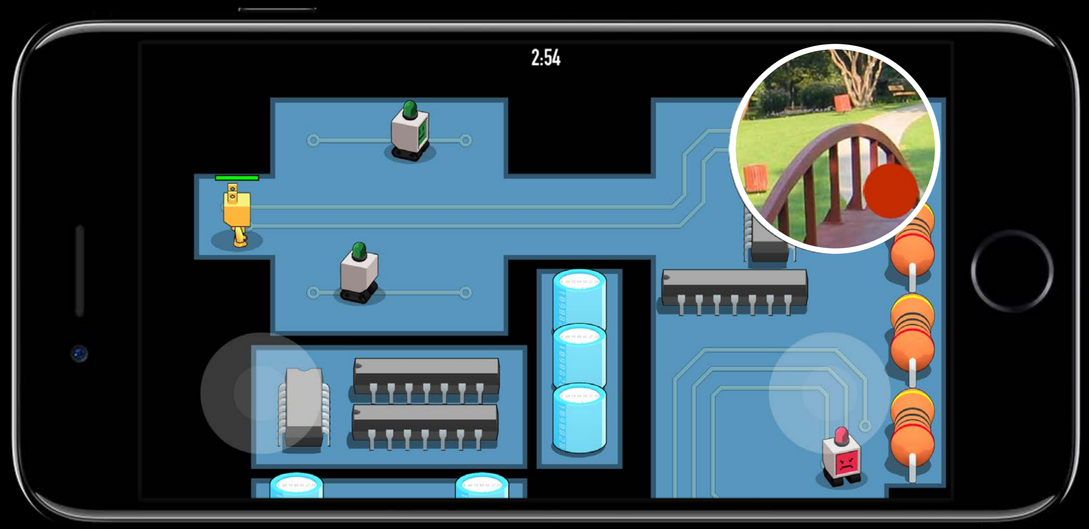
```
public enum RPCameraPosition: Int {
    case front

    case back
}
```

# Initiating Camera Preview



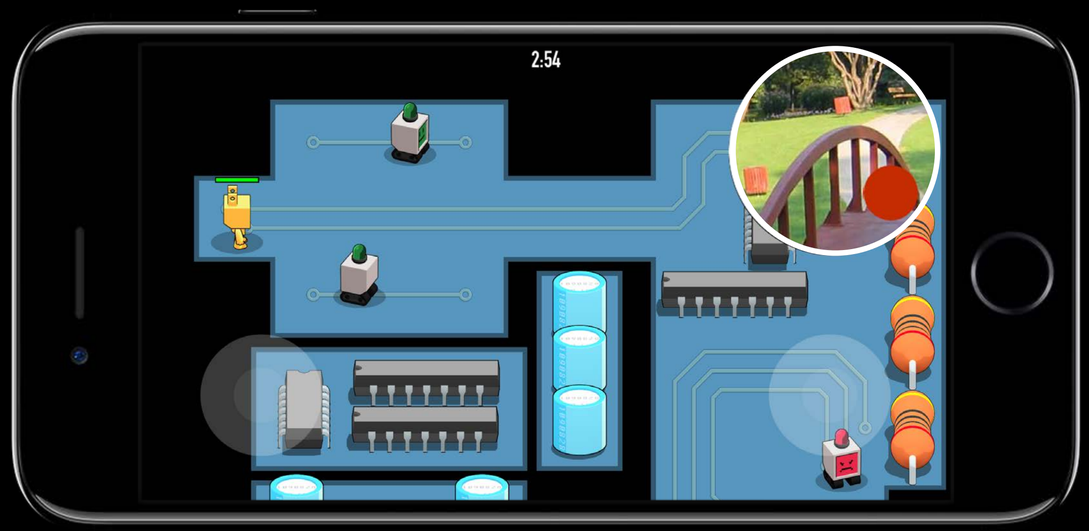

```
func showPreviewView() {
        let sharedRecorder = RPScreenRecorder.shared()
        let cameraView = sharedRecorder.cameraPreviewView
        self.view.addSubview(cameraView!)
}
```

# Initiating Camera Switching




```
func didPressCameraSwitch() {
        let sharedRecorder = RPScreenRecorder.shared()
        if (sharedRecorder.cameraPosition == RPCameraPosition.back) {
            sharedRecorder.cameraPosition = RPCameraPosition.front
        }
        else {
            sharedRecorder.cameraPosition = RPCameraPosition.back
        }
 }
```

# Initiating Camera Switching




```
func didPressCameraSwitch() {
        let sharedRecorder = RPScreenRecorder.shared()
        if (sharedRecorder.cameraPosition == RPCameraPosition.back) {
            sharedRecorder.cameraPosition = RPCameraPosition.front
        }
        else {
            sharedRecorder.cameraPosition = RPCameraPosition.back
        }
 }
```

# Summary

In-App Screen Capture

iOS Screen Record and Broadcast

Broadcast Pairing

Fast Camera Switching

# More Information

https://developer.apple.com/wwdc17/606

# Labs

| | | |
|---|---|---|
| ReplayKit 2 Lab | Technology Lab A | Fri 11:00AM–1:50PM |


WWDC17