#WWDC19

Building Collaborative AR Experiences

Kuen-han Lin, ARKit Engineering
David Paschich, Tools Foundation

Collaborative Session

ARAnchor Best Practice for Multi-User AR

SwiftStrike — A New Multiplayer AR Experience

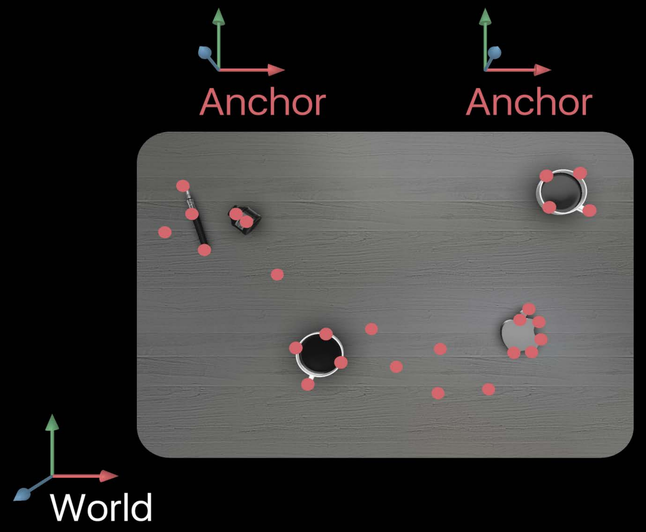
Collaborative Session

Saving and Loading Map in ARKit 2

Recap

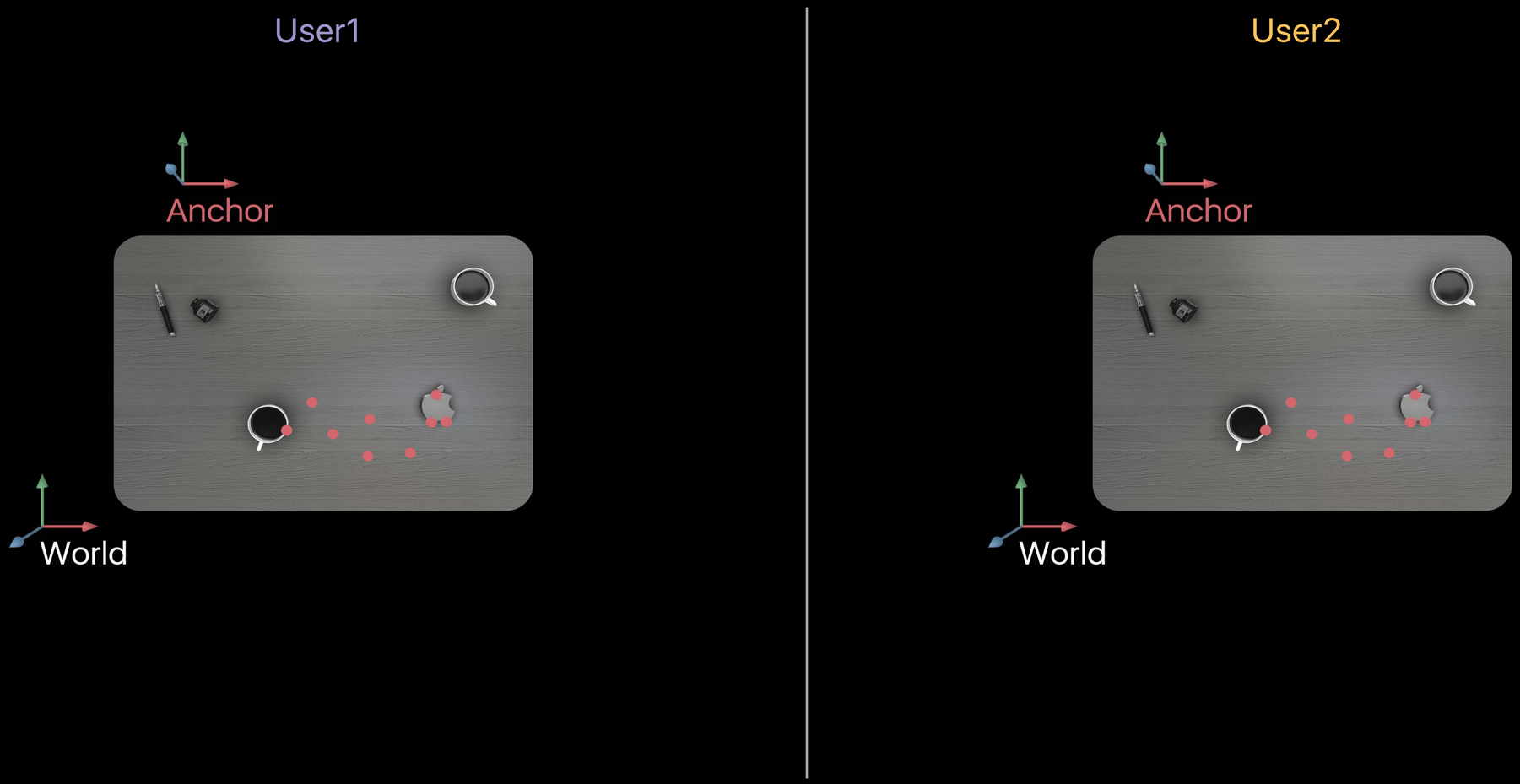
ARWorldMap data

- Map of sparse 3D landmarks
- List of named ARAnchors



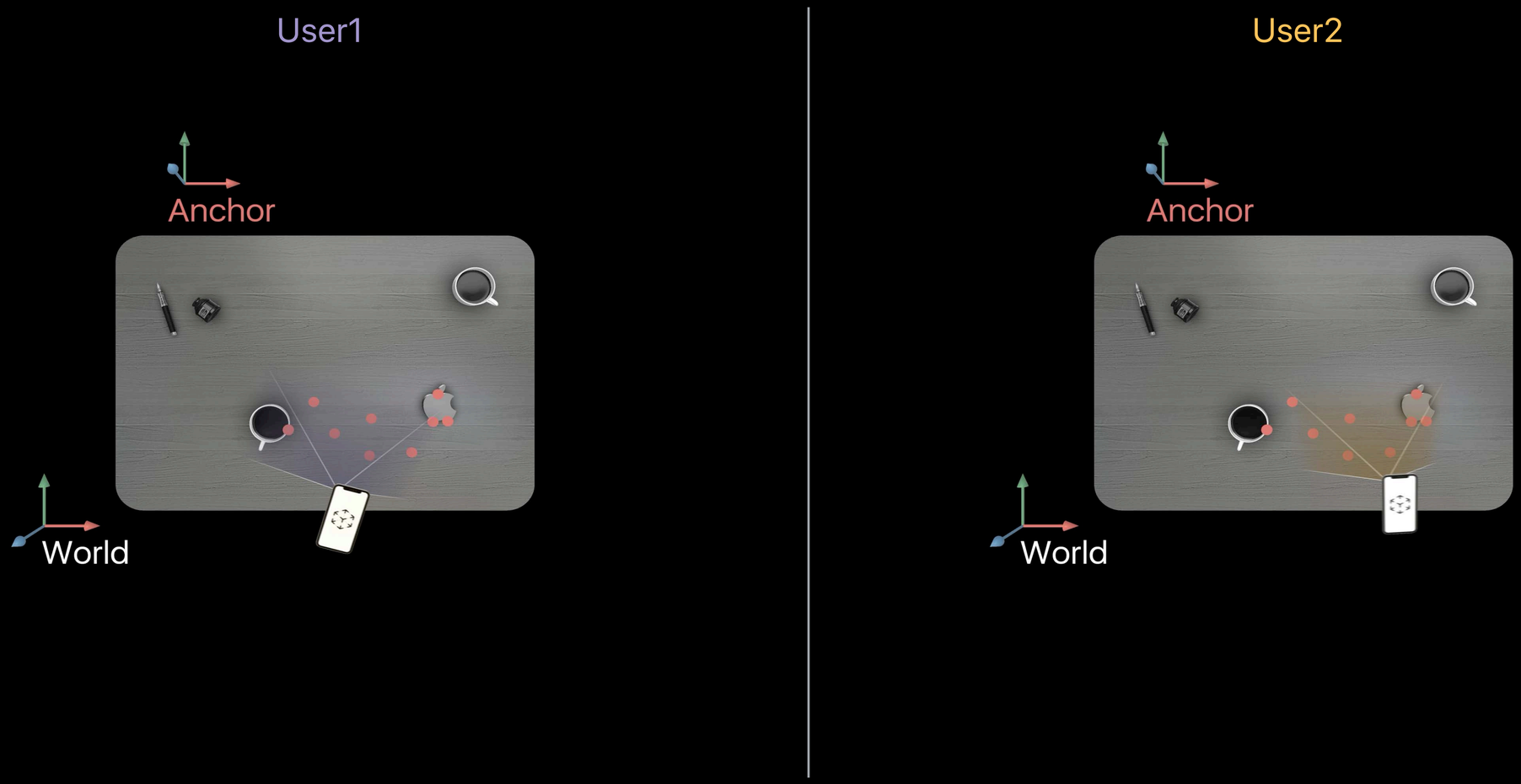
Multi-User Experiences in ARKit 2

Recap



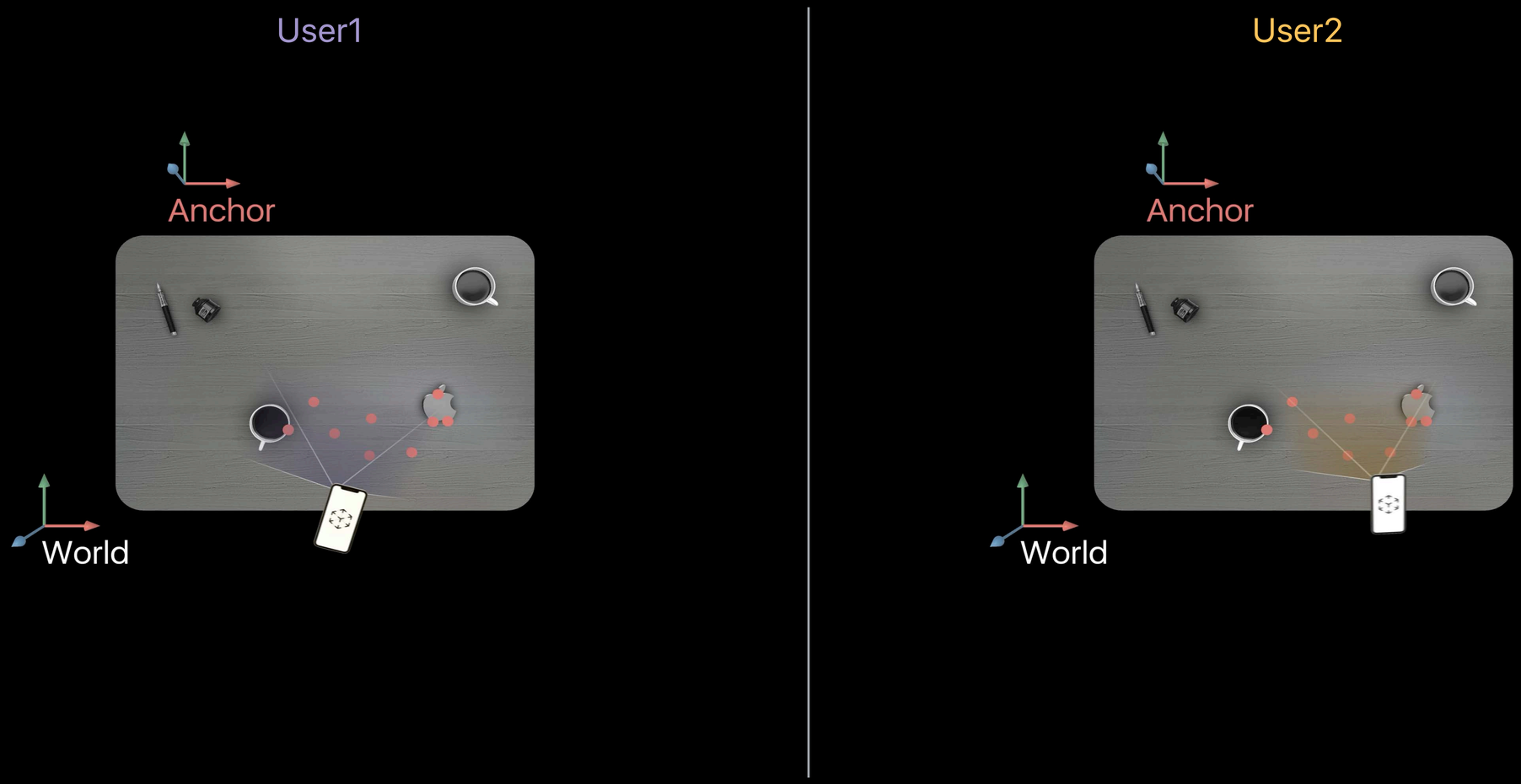
Multi-User Experiences in ARKit 2

Recap



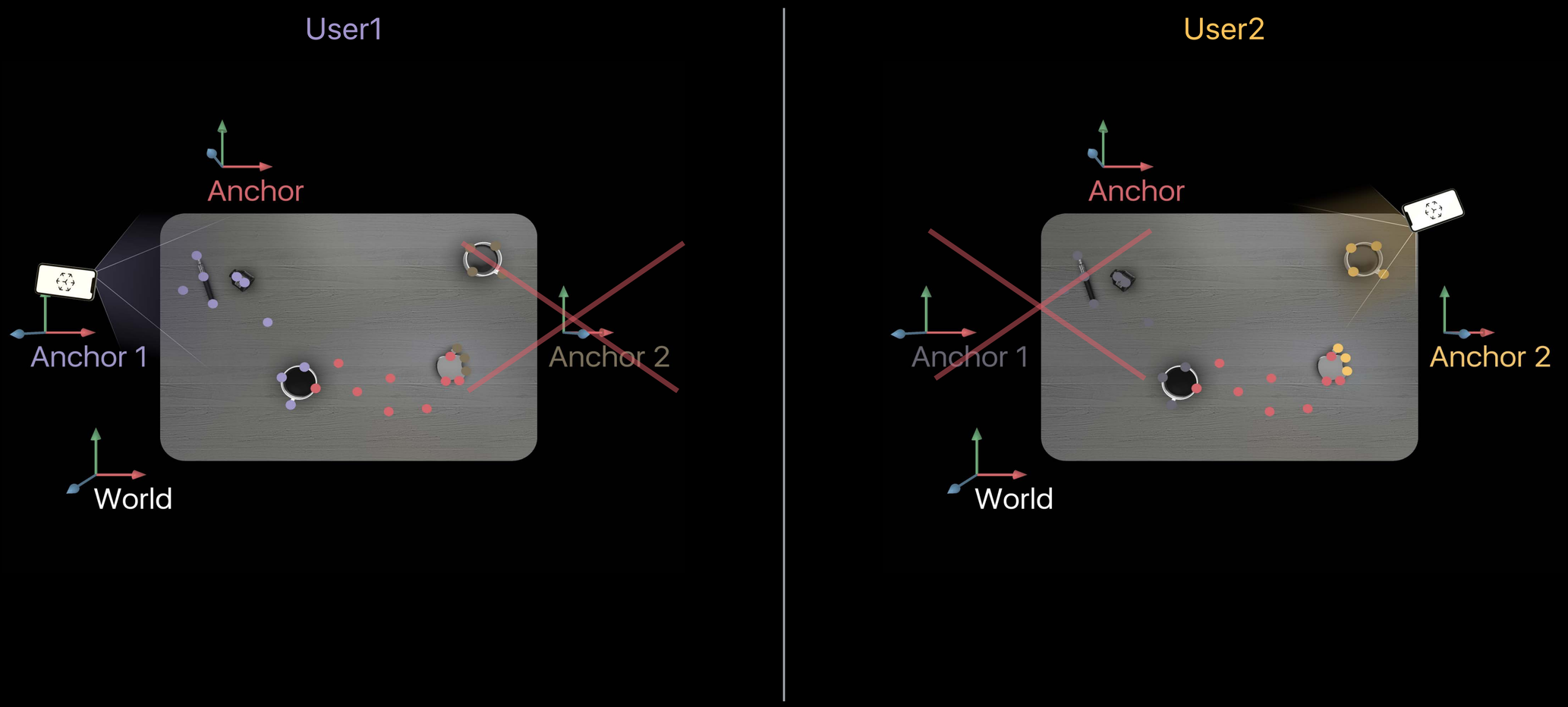
Multi-User Experiences in ARKit 2

Recap



Multi-User Experiences in ARKit 2

Recap



Collaborative Session

Live multi-user AR experience

Continuously share ARAnchors and map data

No host user


5:17
Exploring

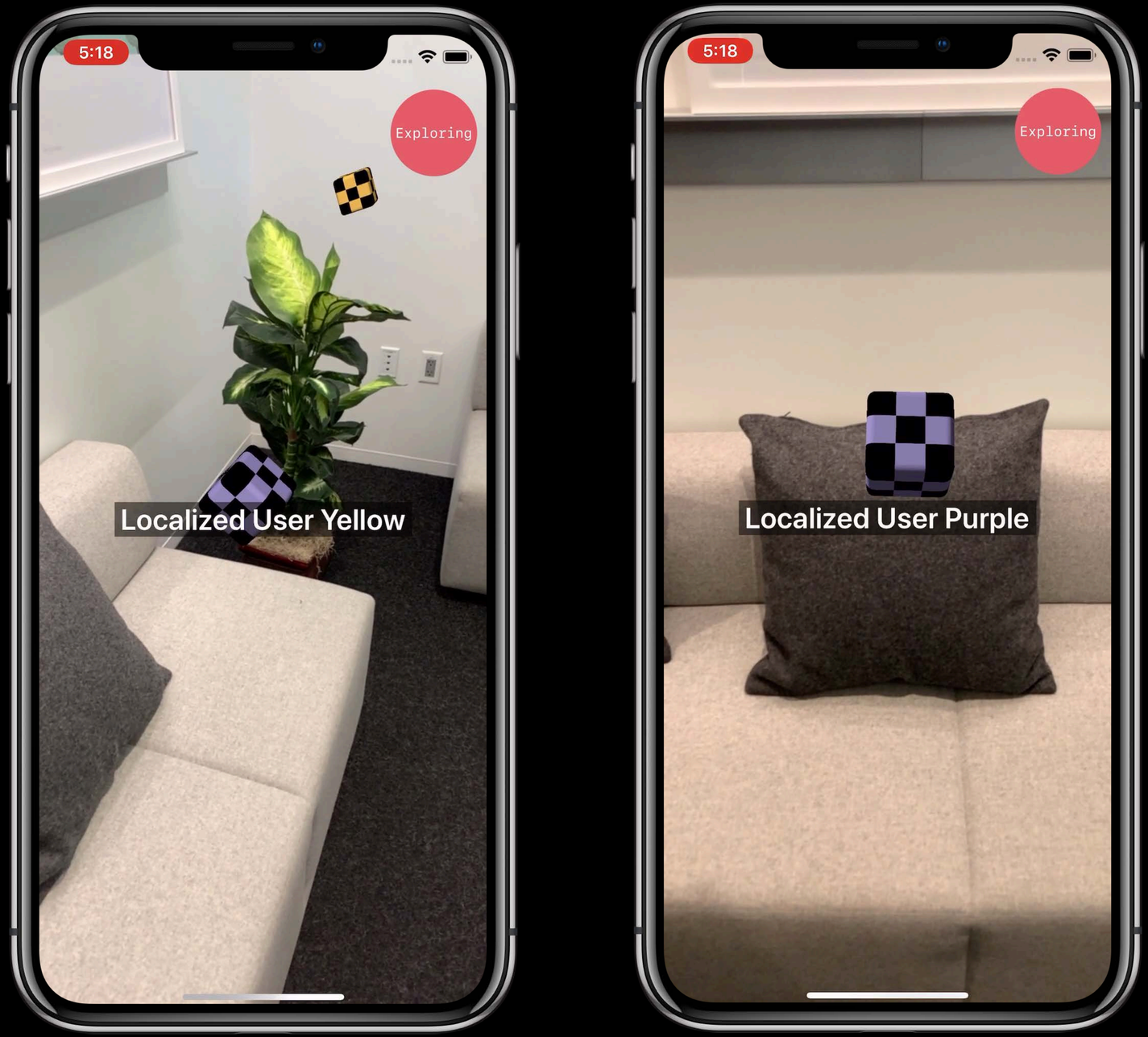

5:17
Exploring

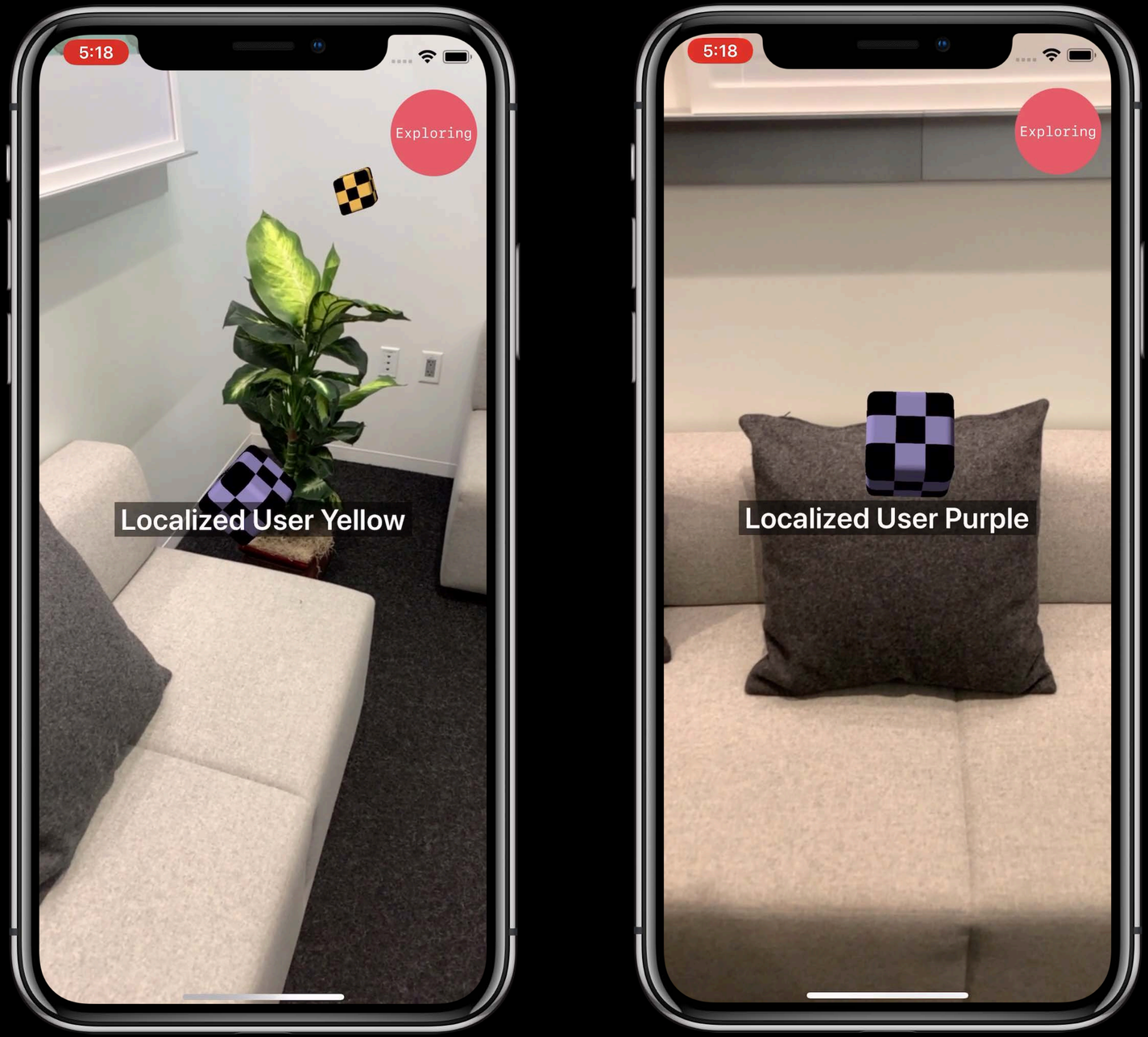

5:17
Exploring


5:17
Exploring

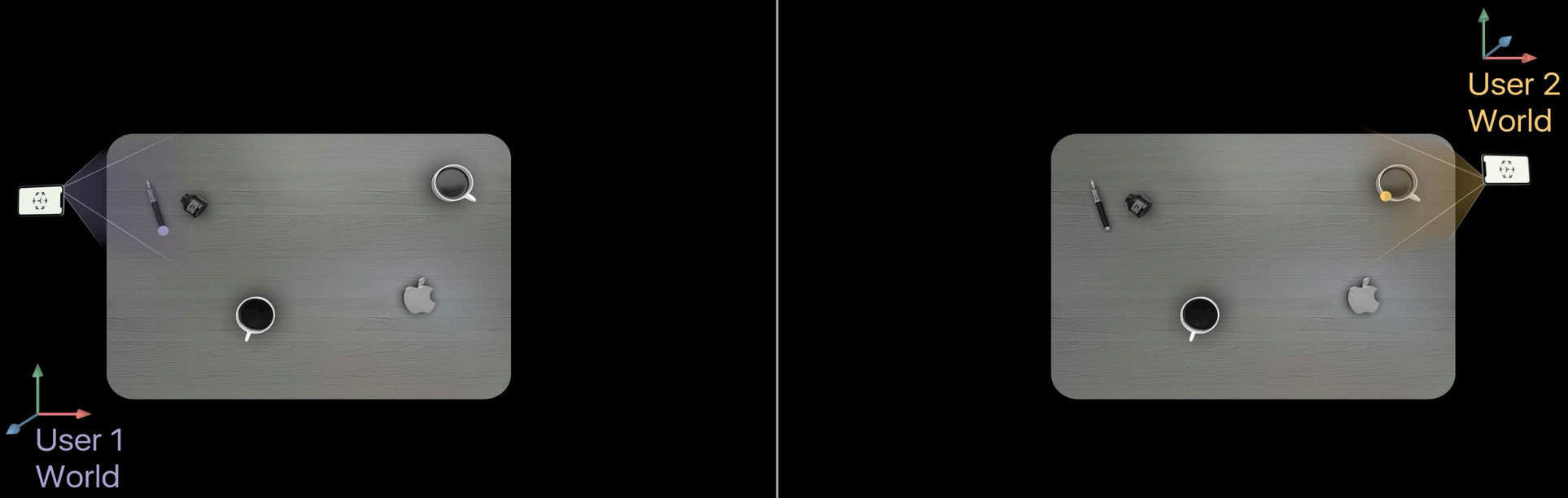

5:17
Mapped
5:17
Mapped


5:17
Mapped
5:17
Mapped

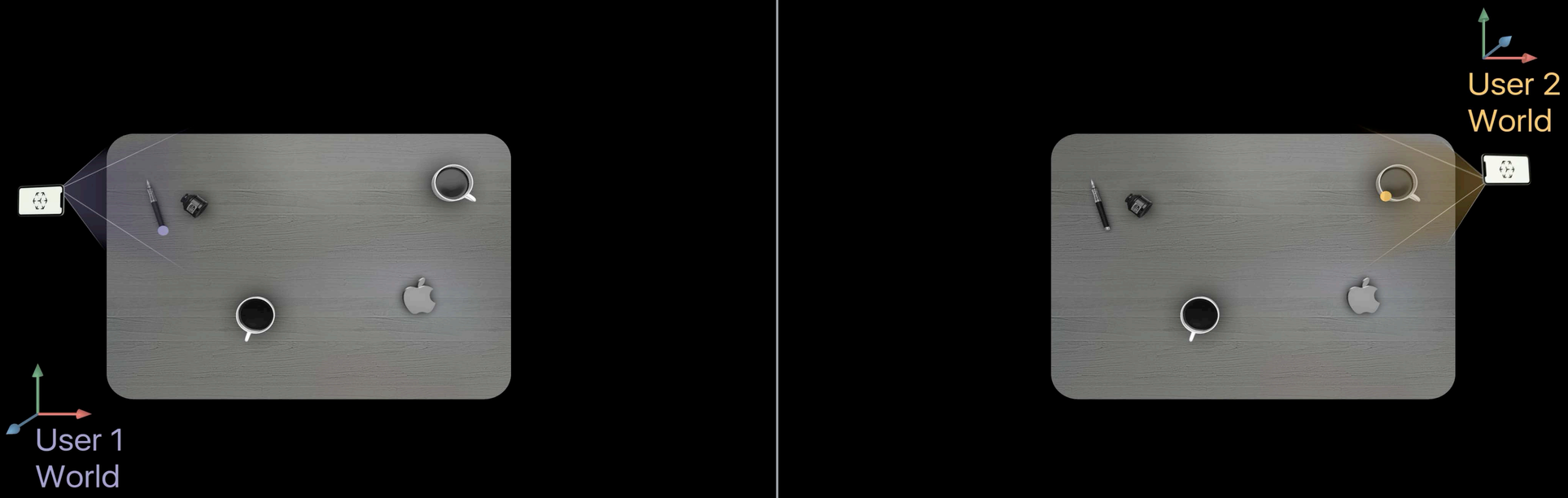

5:18
Exploring
Localized User Yellow
5:18
Exploring
Localized User Purple


5:18
Exploring
Localized User Yellow
5:18
Exploring
Localized User Purple

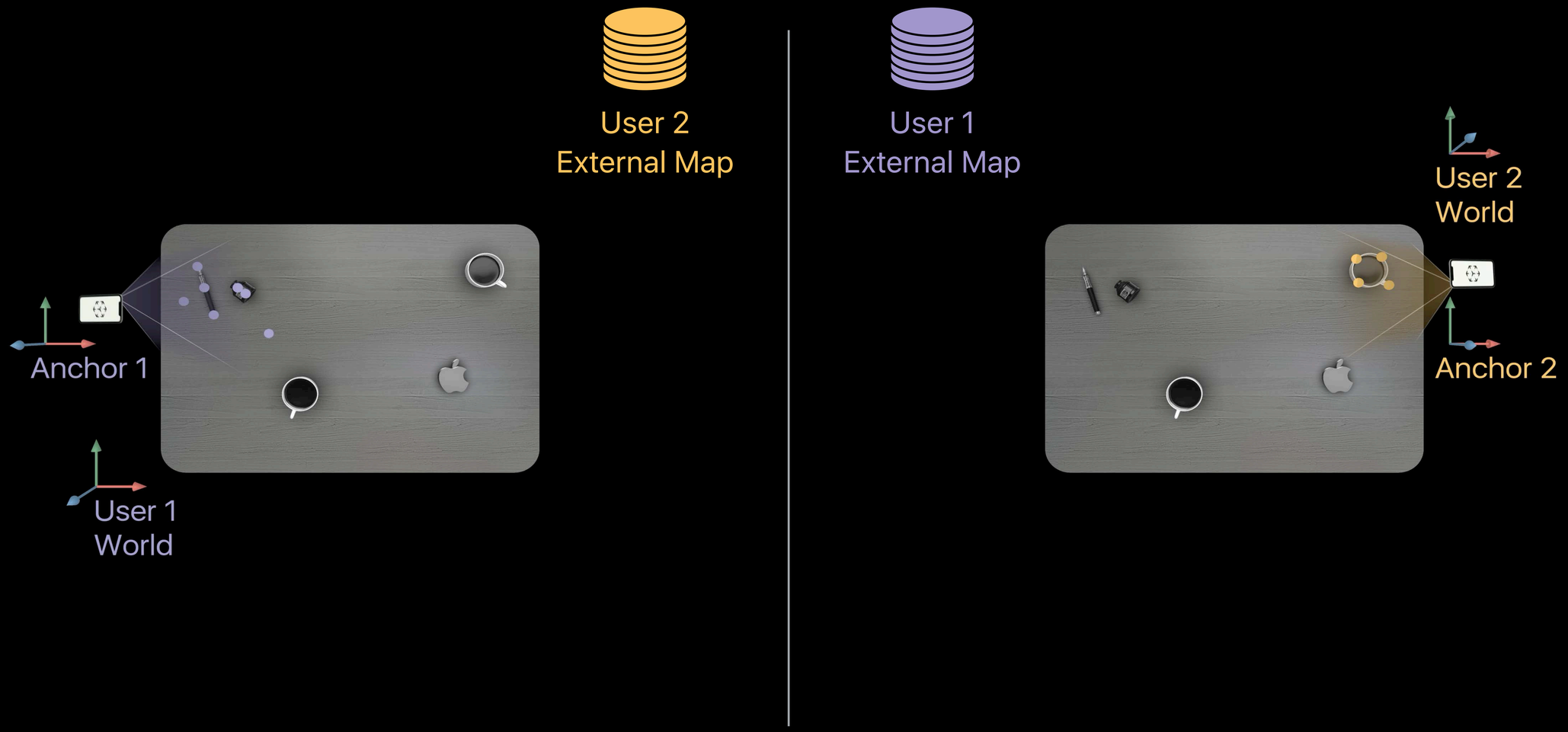
Different Coordinates in Collaborative Session



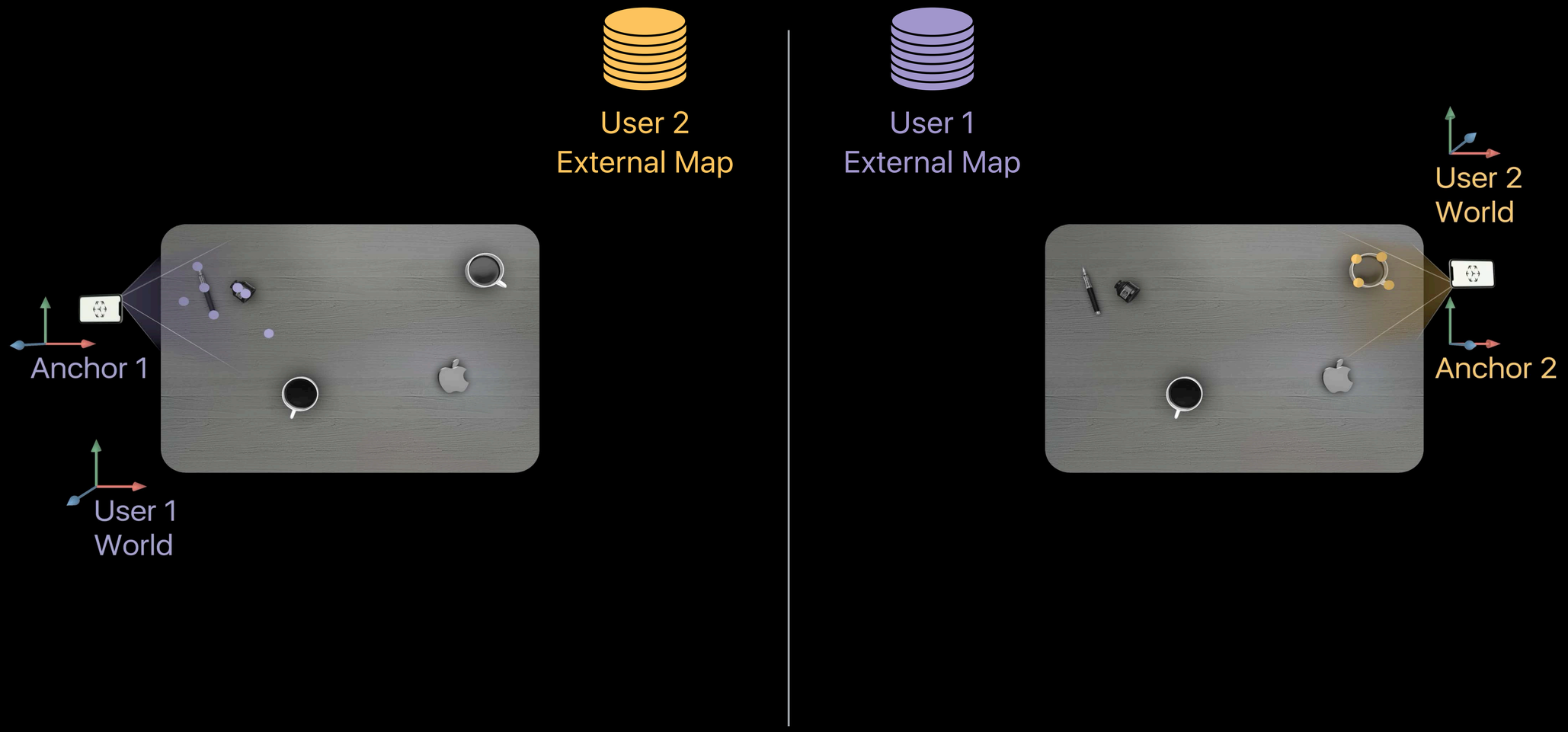
Different Coordinates in Collaborative Session



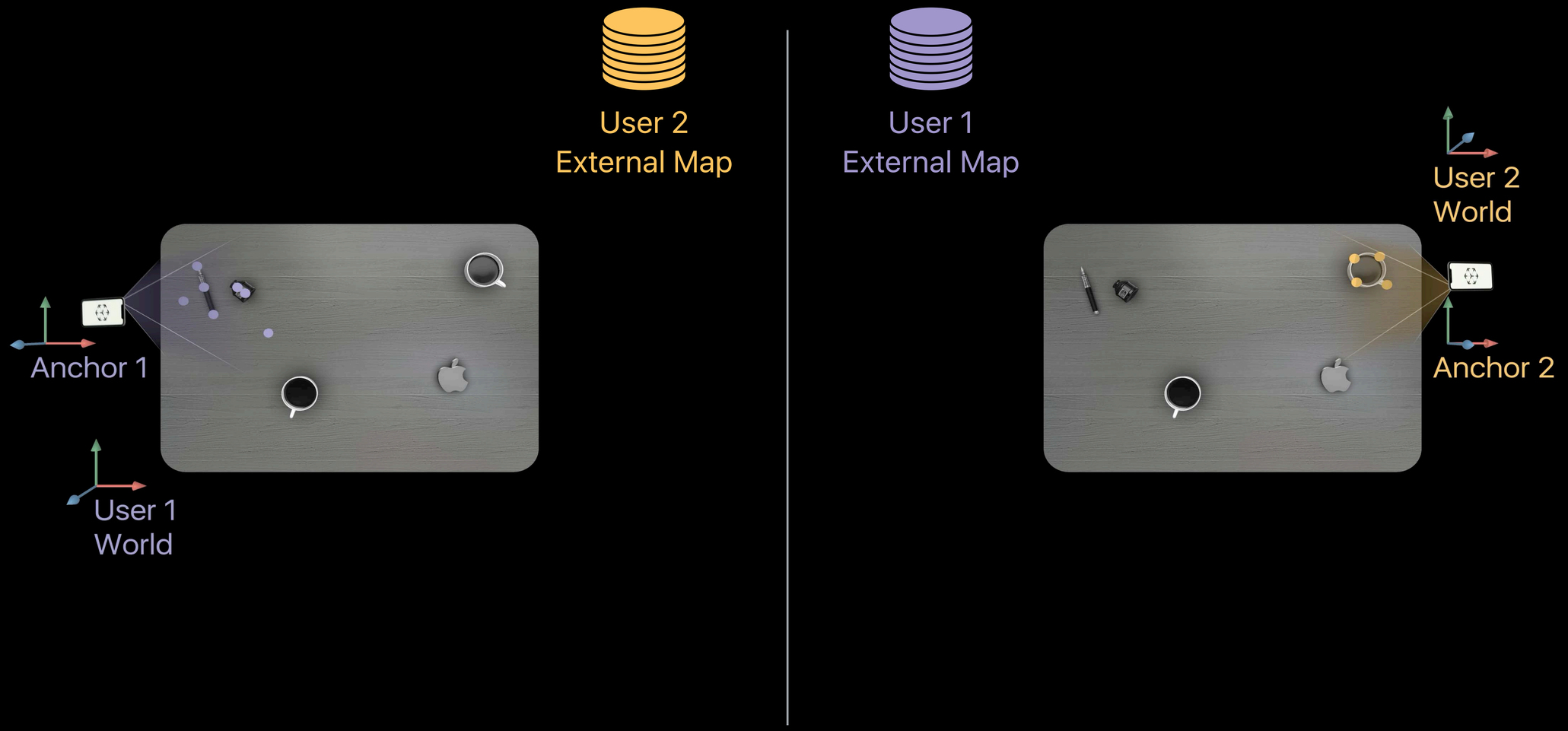
Different Coordinates in Collaborative Session



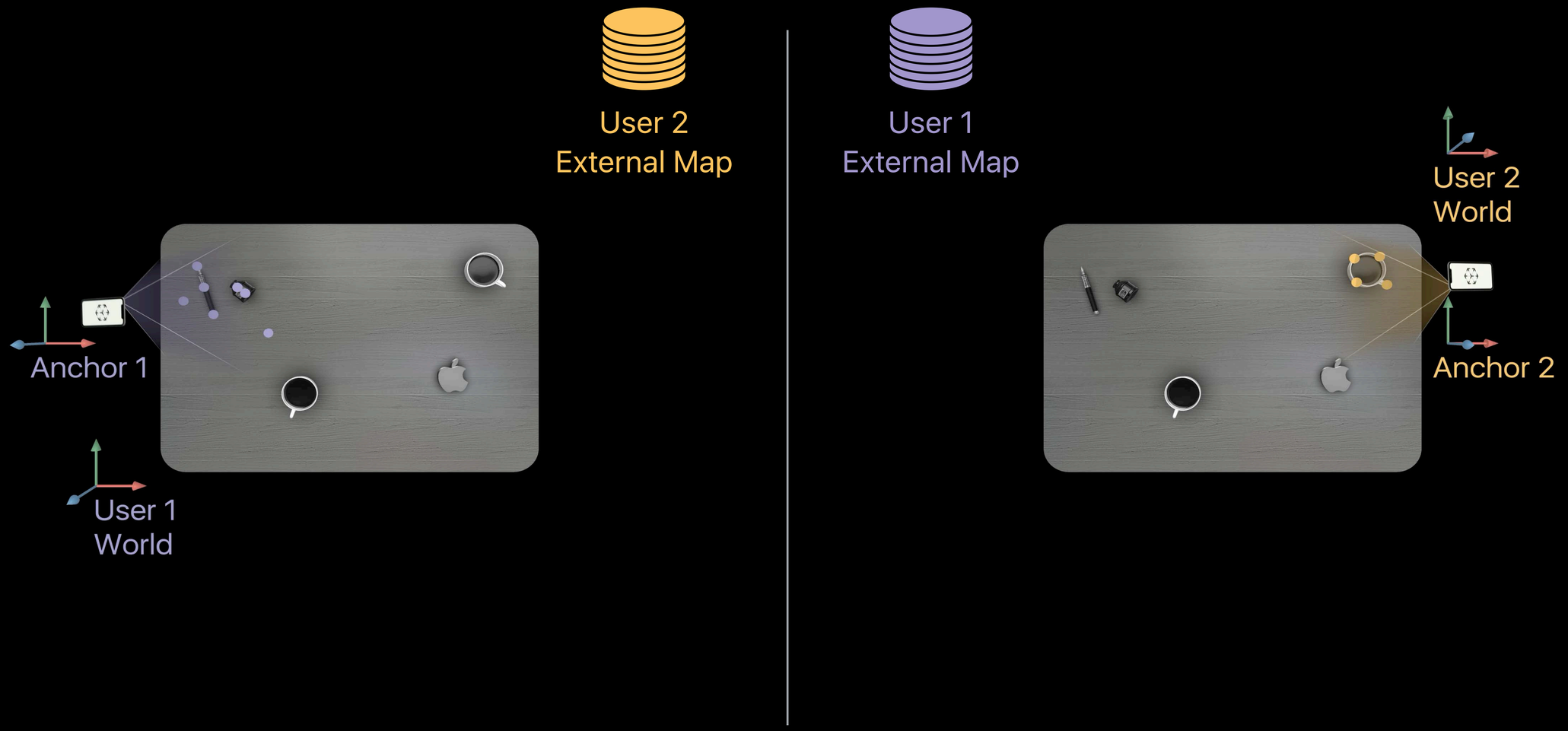
Different Coordinates in Collaborative Session



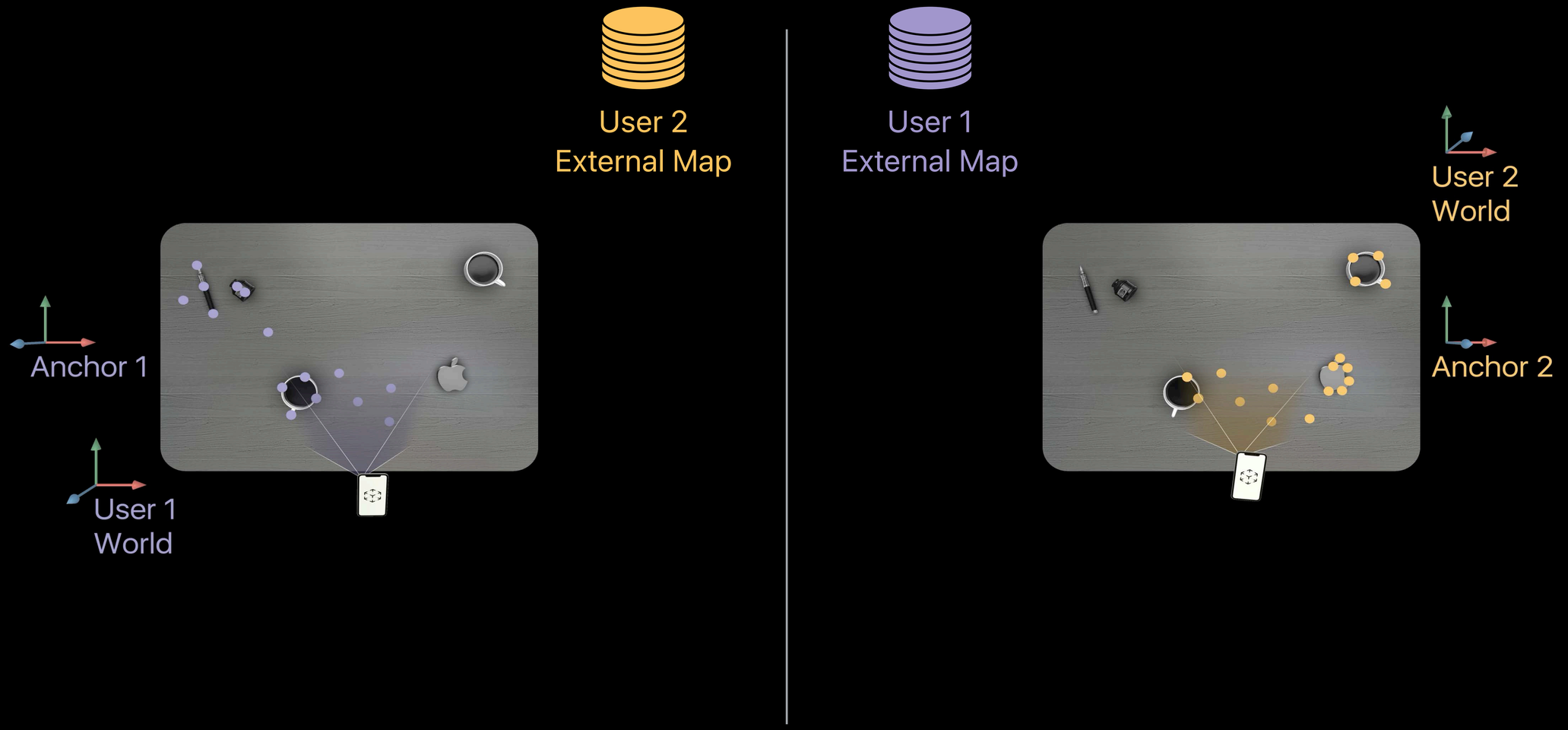
Different Coordinates in Collaborative Session



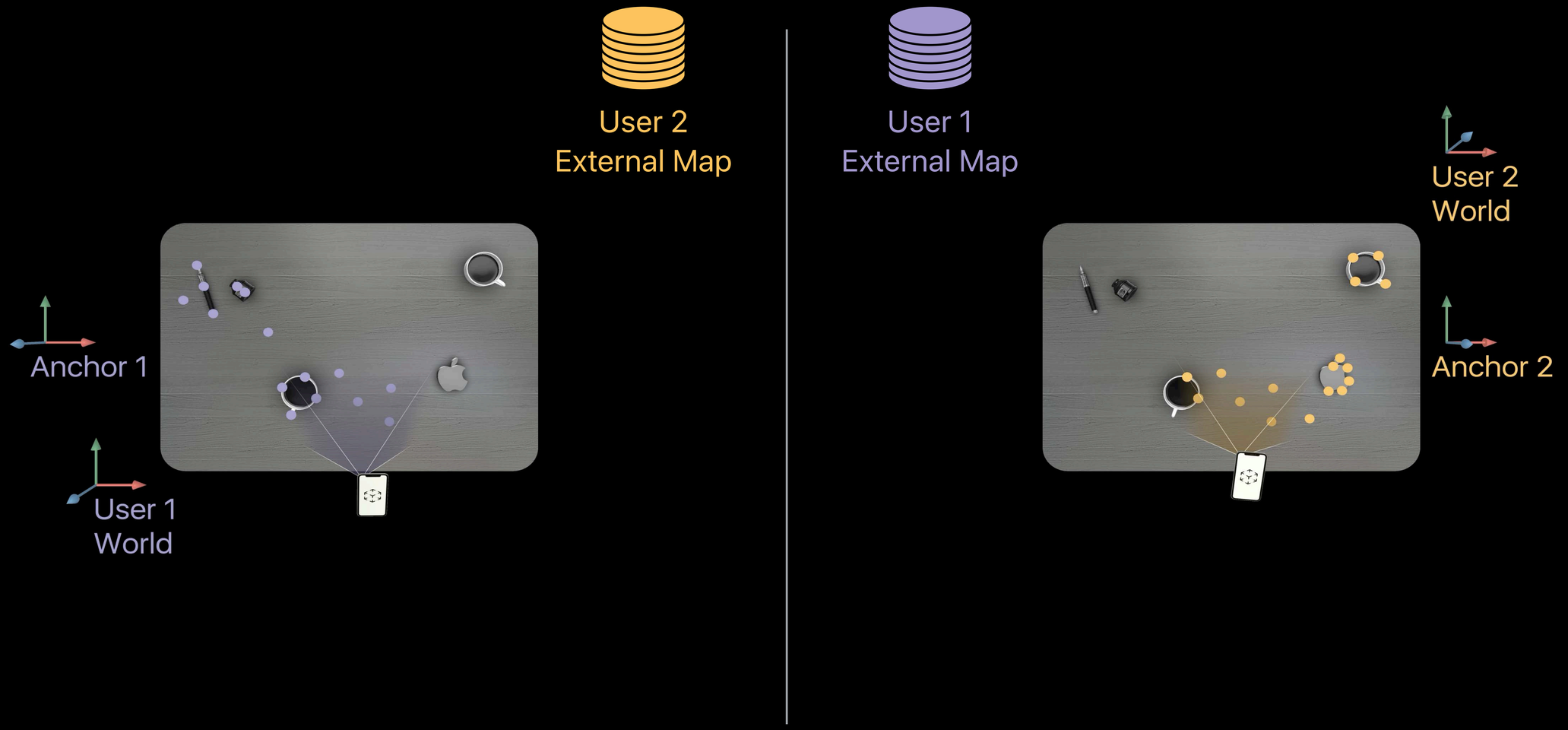
Different Coordinates in Collaborative Session



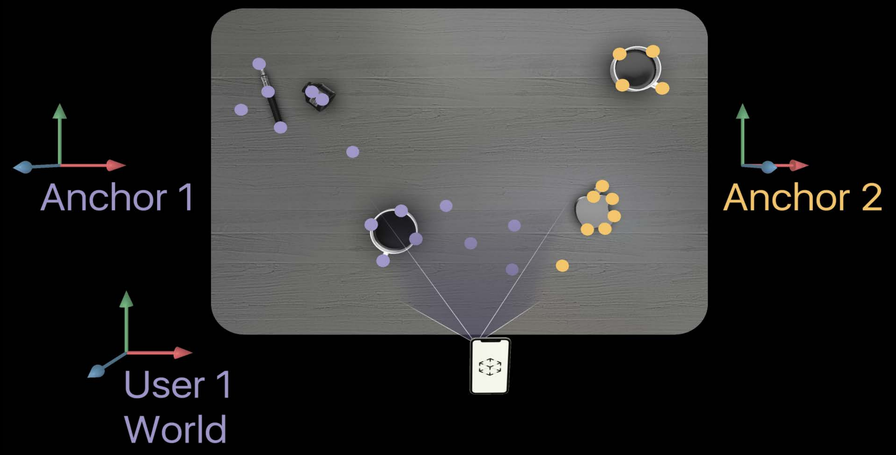
Different Coordinates in Collaborative Session

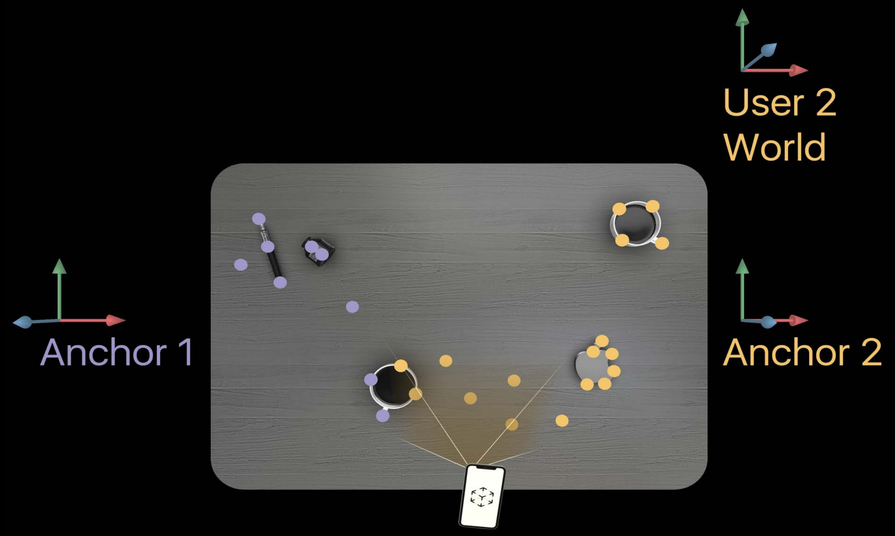


Different Coordinates in Collaborative Session



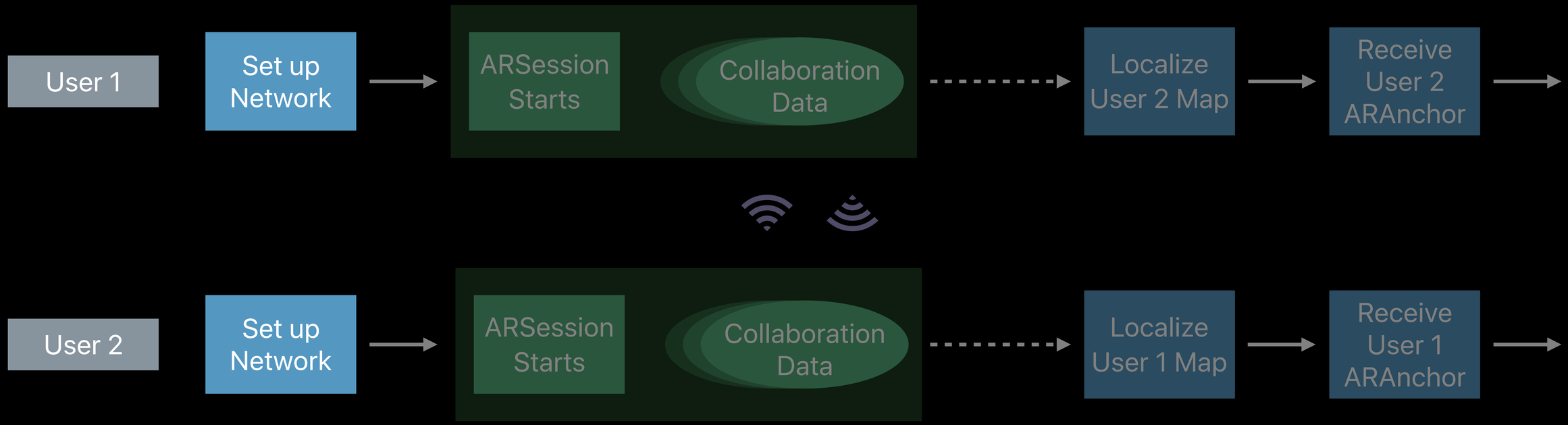
Different Coordinates in Collaborative Session





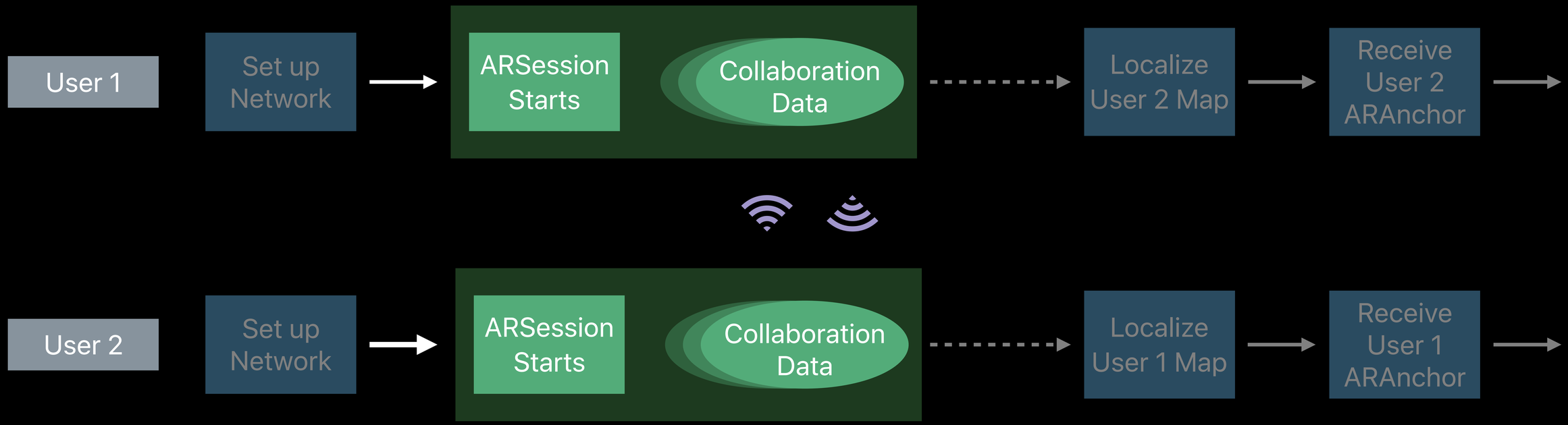
Collaborative Session Flow

Set up communication



Collaborative Session Flow

Transmit collaboration data



```
// Collaborative Session

// Create world tracking configuration
let config = ARWorldTrackingConfiguration()

// Enable collaborative session
config.isCollaborationEnabled = true

// Run the configuration
session.run(config)
```

```
// Collaborative Session

// Create world tracking configuration
let config = ARWorldTrackingConfiguration()

// Enable collaborative session
config.isCollaborationEnabled = true

// Run the configuration
session.run(config)
```

```
// Collaborative Session

// Create world tracking configuration
let config = ARWorldTrackingConfiguration()

// Enable collaborative session
config.isCollaborationEnabled = true

// Run the configuration
session.run(config)
```

```
// Collaborative Session

// Create world tracking configuration
let config = ARWorldTrackingConfiguration()

// Enable collaborative session
config.isCollaborationEnabled = true

// Run the configuration
session.run(config)
```

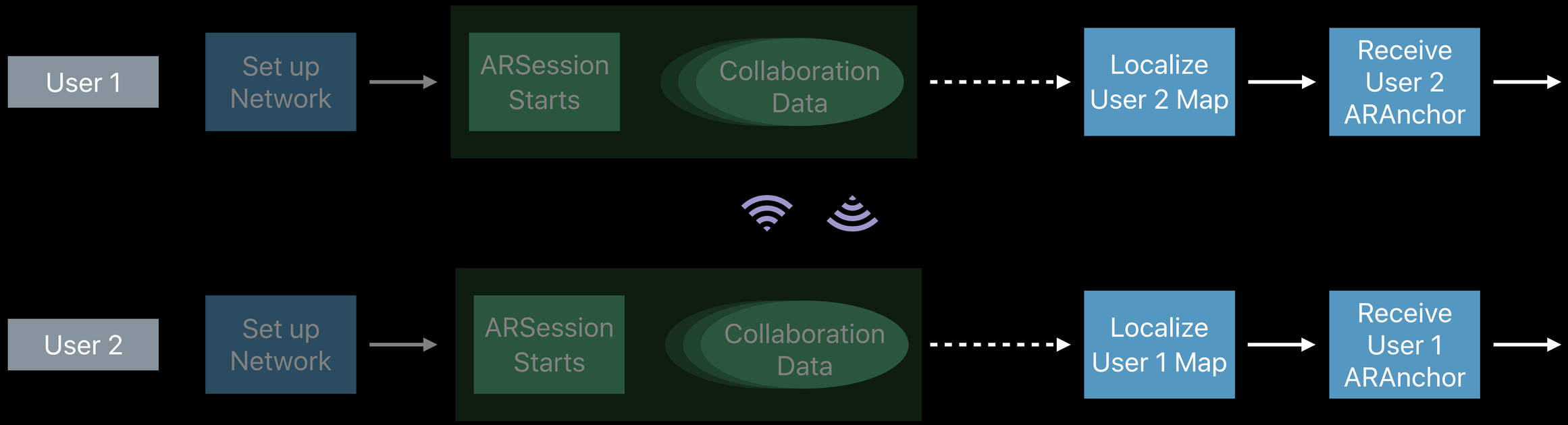
```swift
// ARSession delegate function to output ARCollaborationData
func session(_ session: ARSession,
             didOutputCollaborationData data: ARSession.CollaborationData) {
    // Transmit Data representation of the data to all other participants using MPC
    do {
        try self.mpcSession.send(data.data, toPeers: self.peerIds, with: .reliable)
    } catch {
        // Re-transmit the data if failed
    }
}

// MPC delegate function when receiving collaboration data from other users
func session(_ session: MCSession, didReceive data: Data, fromPeer peerID: MCPeerID) {
    // Pass the received data to ARSession
    self.arSession.update(data: ARSession.CollaborationData(data: data))
}
```

```swift
// ARSession delegate function to output ARCollaborationData
func session(_ session: ARSession,
             didOutputCollaborationData data: ARSession.CollaborationData) {
    // Transmit Data representation of the data to all other participants using MPC
    do {
        try self.mpcSession.send(data.data, toPeers: self.peerIds, with: .reliable)
    } catch {
        // Re-transmit the data if failed
    }
}

// MPC delegate function when receiving collaboration data from other users
func session(_ session: MCSession, didReceive data: Data, fromPeer peerID: MCPeerID) {
    // Pass the received data to ARSession
    self.arSession.update(data: ARSession.CollaborationData(data: data))
}
```

```swift
// ARSession delegate function to output ARCollaborationData
func session(_ session: ARSession,
             didOutputCollaborationData data: ARSession.CollaborationData) {
    // Transmit Data representation of the data to all other participants using MPC
    do {
        try self.mpcSession.send(data.data, toPeers: self.peerIds, with: .reliable)
    } catch {
        // Re-transmit the data if failed
    }
}

// MPC delegate function when receiving collaboration data from other users
func session(_ session: MCSession, didReceive data: Data, fromPeer peerID: MCPeerID) {
    // Pass the received data to ARSession
    self.arSession.update(data: ARSession.CollaborationData(data: data))
}
```

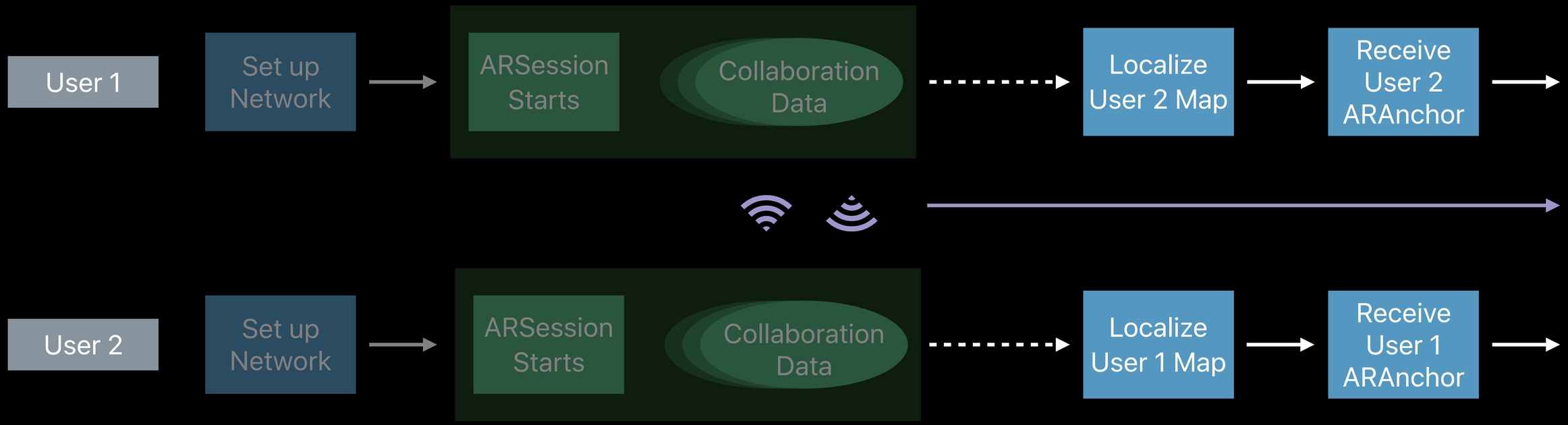
Collaborative Session Flow

Start shared AR experience



Collaborative Session Flow

Start shared AR experience



ARAnchor in Collaborative Session

ARAnchor

- Synchronized lifetime
- Session identifier
- Subclass ARAnchors are not shared

```
// ARAnchor in collaborative session

// ARSession delegate function when an ARAnchor is added.
func session(_ session: ARSession, didAdd anchors: [ARAnchor]) {
    for anchor in anchors {
        // Use session identifier to determine creator of the ARAnchor
        if anchor.sessionIdentifier == session.identifier {
            // Self-placed ARanchor
        } else {
            // ARAnchor from another participant
        }
    }
}
```

```swift
// ARAnchor in collaborative session

// ARSession delegate function when an ARAnchor is added.
func session(_ session: ARSession, didAdd anchors: [ARAnchor]) {
    for anchor in anchors {
        // Use session identifier to determine creator of the ARAnchor
        if anchor.sessionIdentifier == session.identifier {
            // Self-placed ARanchor
        } else {
            // ARAnchor from another participant
        }
    }
}
```

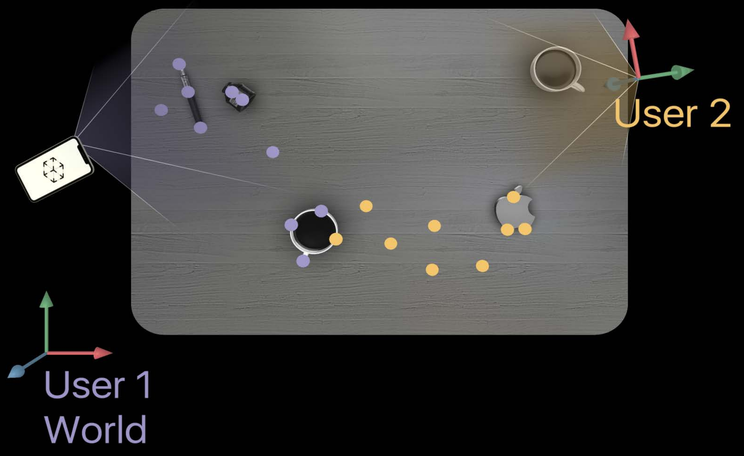
```swift
// ARSession delegate function when an ARAnchor is removed.
func session(_ session: ARSession, didRemove anchors: [ARAnchor]) {
    for anchor in anchors {
        // Use session identifier to determine creator of the ARAnchor
        if anchor.sessionIdentifier == session.identifier {
            // Self-placed ARanchor
        } else {
            // ARAnchor from another participant
        }
    }
}
```

```
// ARSession delegate function when an ARAnchor is removed.
func session(_ session: ARSession, didRemove anchors: [ARAnchor]) {
    for anchor in anchors {
        // Use session identifier to determine creator of the ARAnchor
        if anchor.sessionIdentifier == session.identifier {
            // Self-placed ARanchor
        } else {
            // ARAnchor from another participant
        }
    }
}
```

ARParticipantAnchor

ARParticipantAnchor

- Position of other user
- Update at high frame rate
- Created after localizing other user’s map



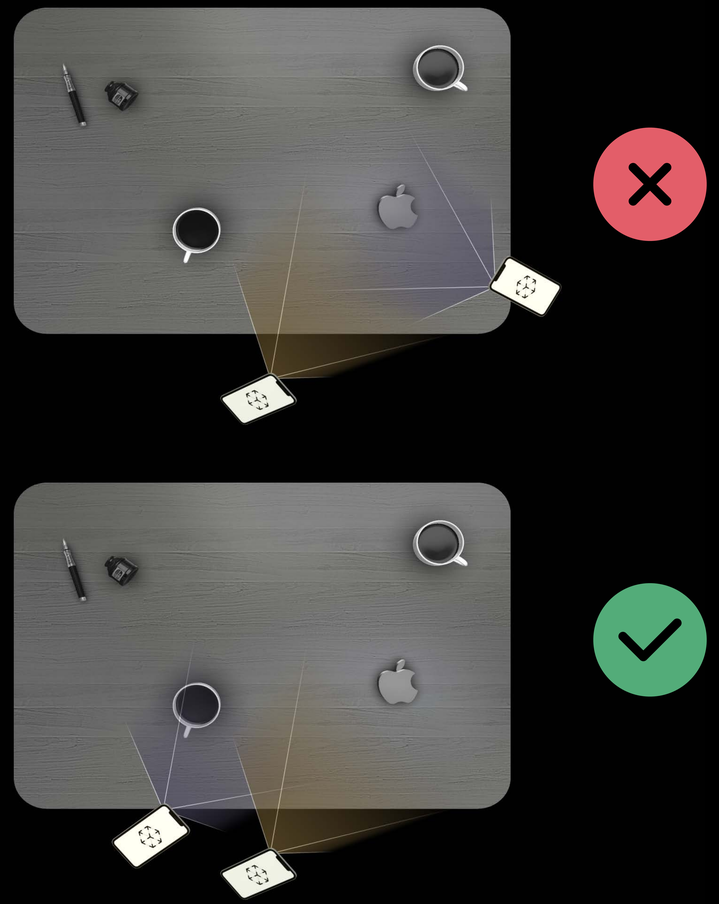
Practical Advice for Localizing Other Users

Shared AR experience starts after seeing
same area

Practical Advice for Localizing Other Users

Shared AR experience starts after seeing same area

Approach other user to have same camera perspective



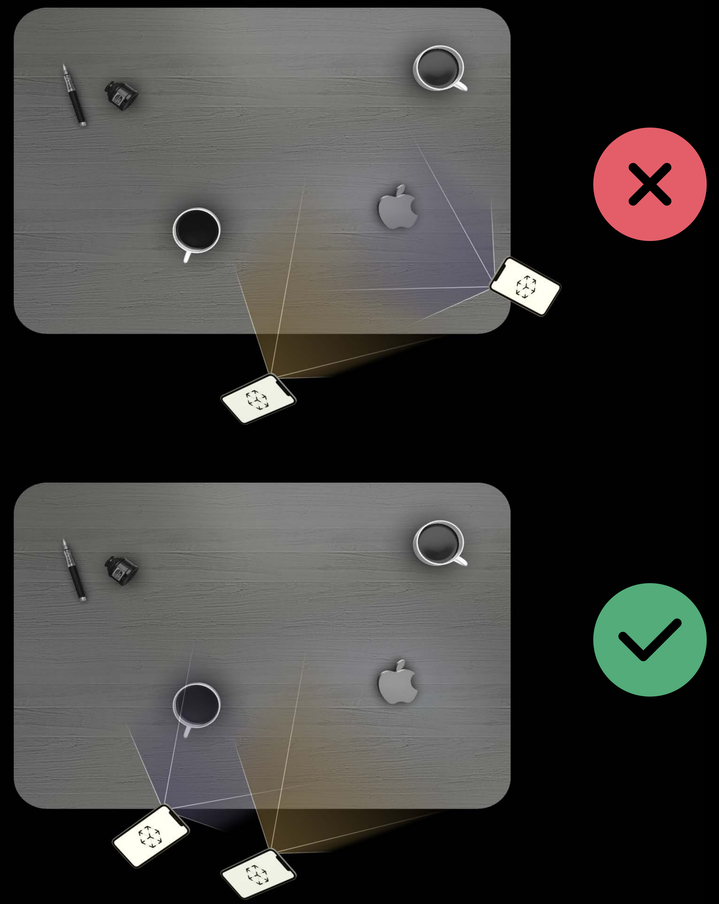
Practical Advice for Localizing Other Users

Shared AR experience starts after seeing same area

Approach other user to have same camera perspective

Stay in map-tracking status

```
ARFrame.WorldMappingStatus.mapped
```




4:59
Exploring


4:59
Exploring


4:59
Mapped


4:59
Exploring


4:59
Mapped

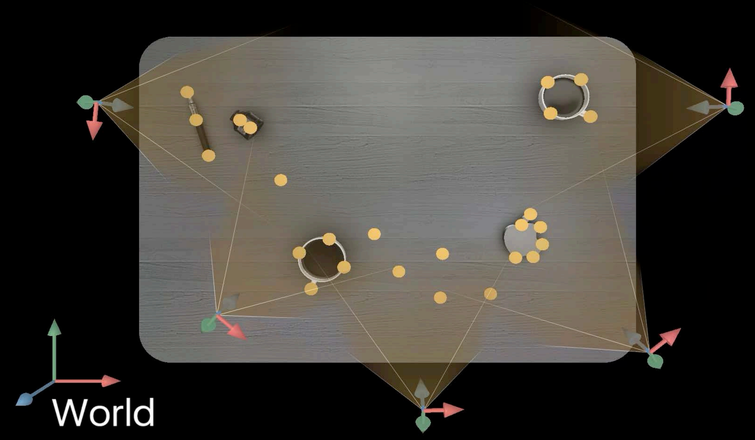

4:59
Exploring

ARAnchor Best Practice for Multi-User AR

Behind the Scenes — ARWorldMap

Internal tracking data

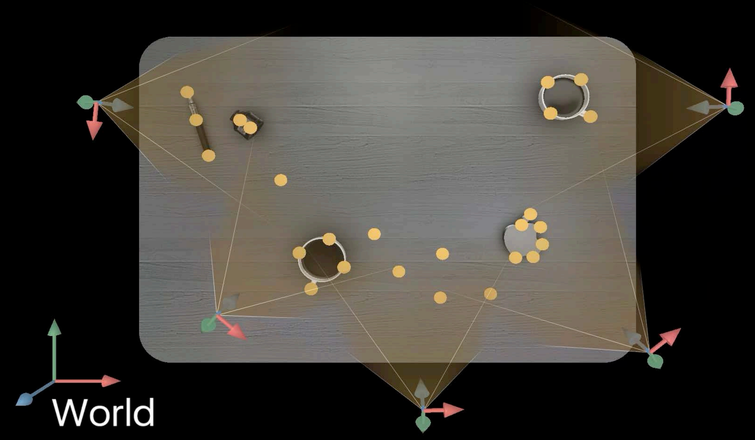
- Map of sparse 3D landmarks
- Camera poses



Behind the Scenes — ARWorldMap

Internal tracking data

- Map of sparse 3D landmarks
- Camera poses

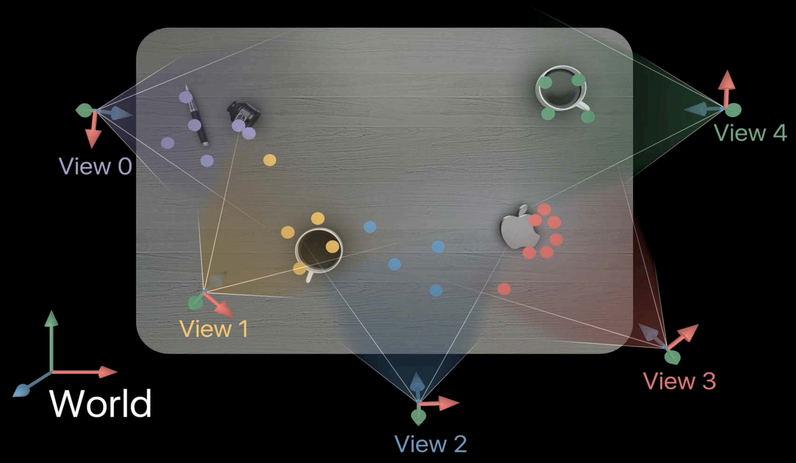


Behind the Scenes — ARWorldMap

Internal tracking data

- Map of sparse 3D landmarks
- Camera poses

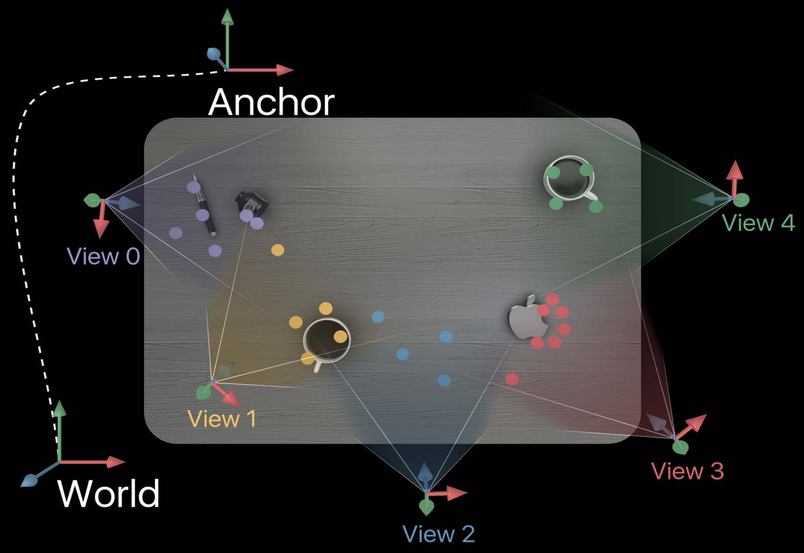
Landmarks are grouped based on camera view



Behind the Scenes — ARAnchor

ARAnchor

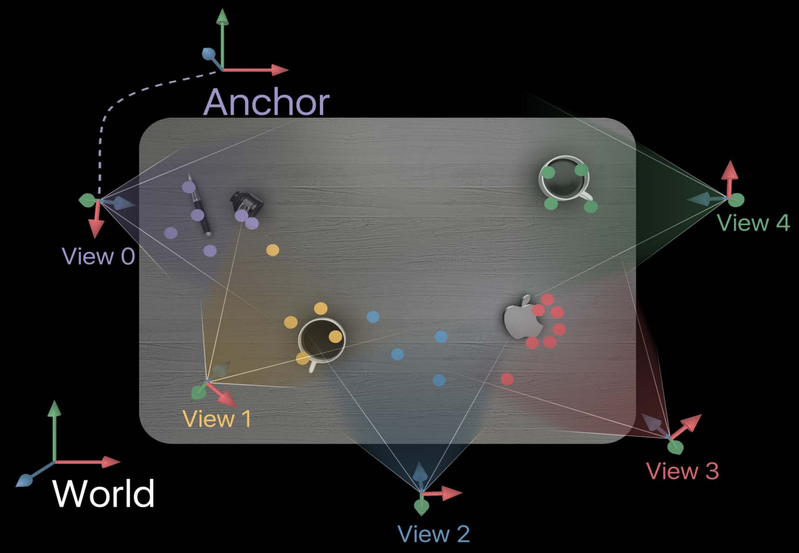
- Global position against world



Behind the Scenes — ARAnchor

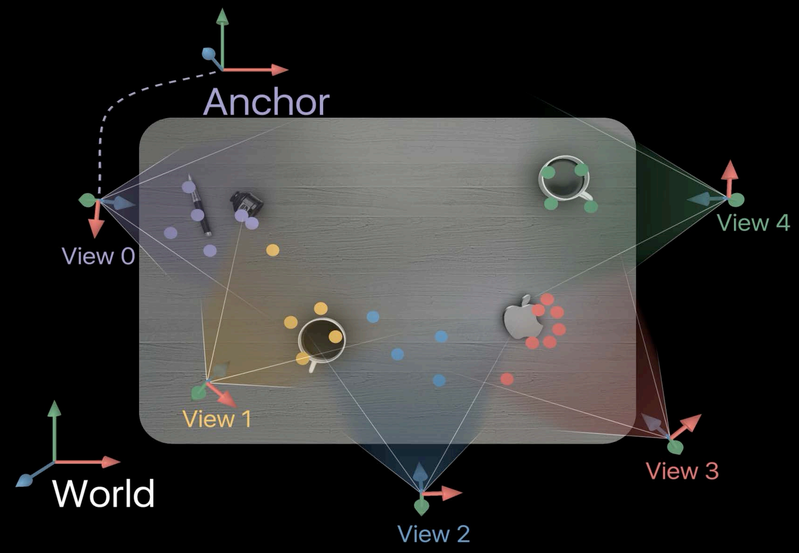
ARAnchor

- Global position against world
- Relative position against map



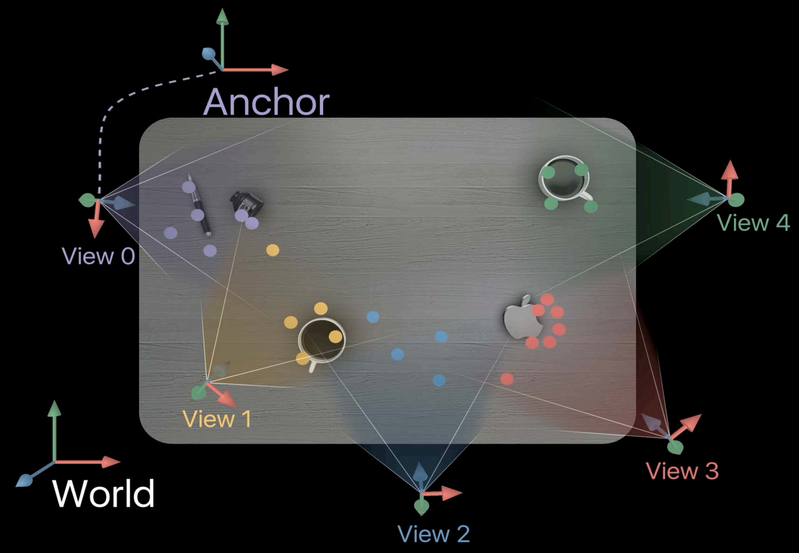
Best Practice for Using ARAnchor

Respond to anchor update delegate



Best Practice for Using ARAnchor

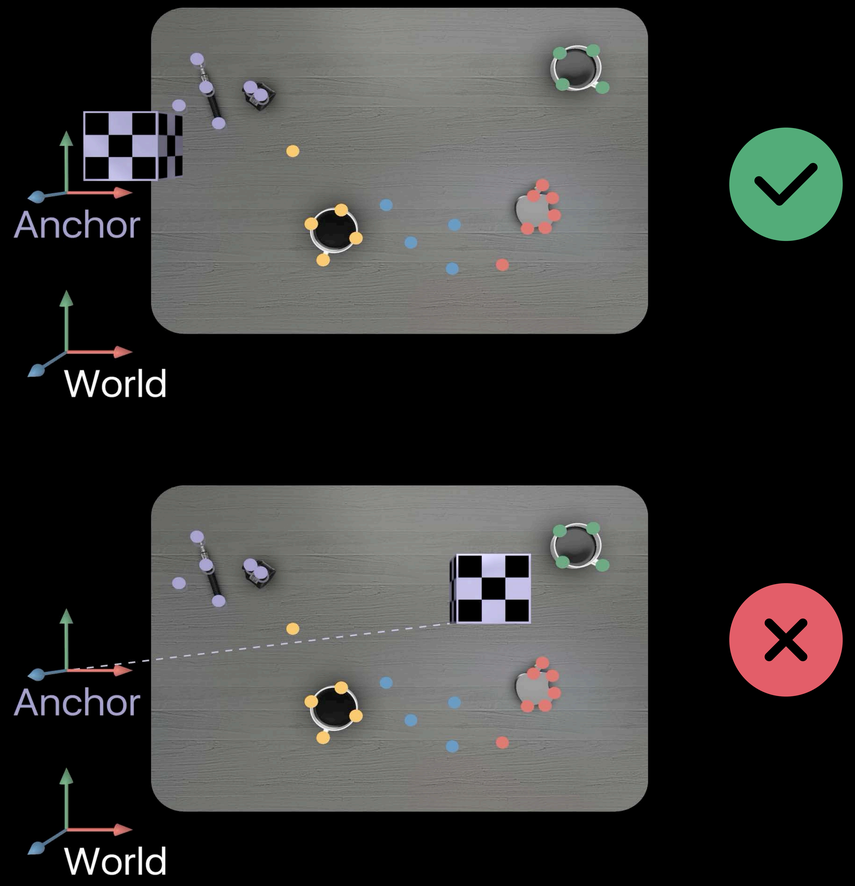
Respond to anchor update delegate



Best Practice for Using ARAnchor

Respond to anchor update delegate

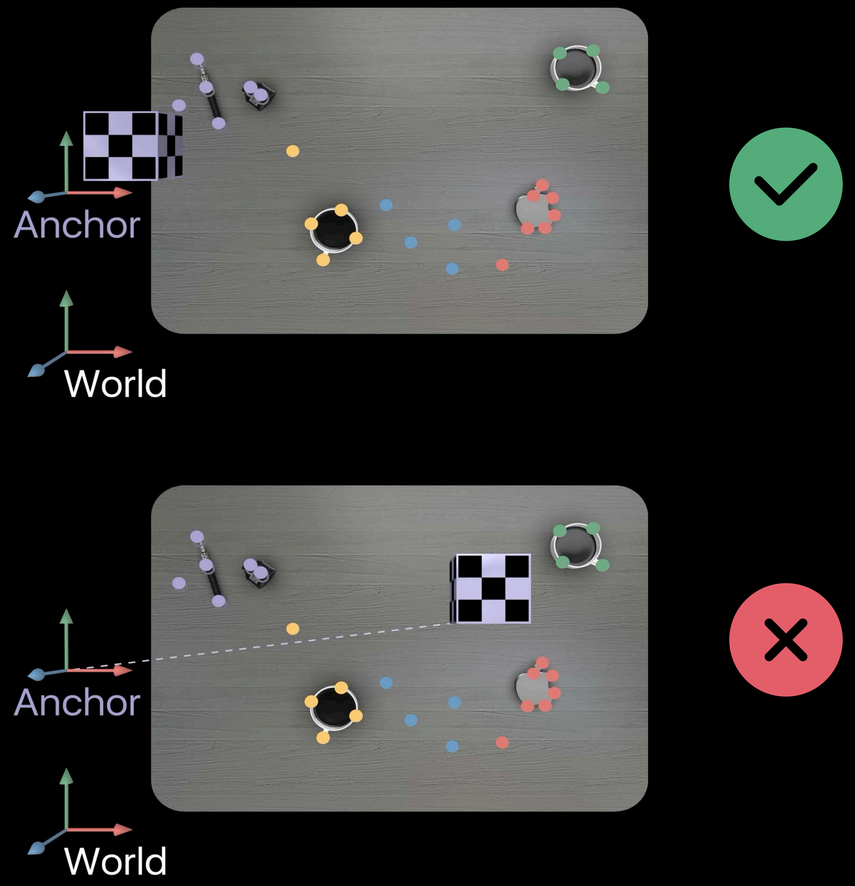
Place 3D content near the ARAnchor



Best Practice for Using ARAnchor

Respond to anchor update delegate

Place 3D content near the ARAnchor

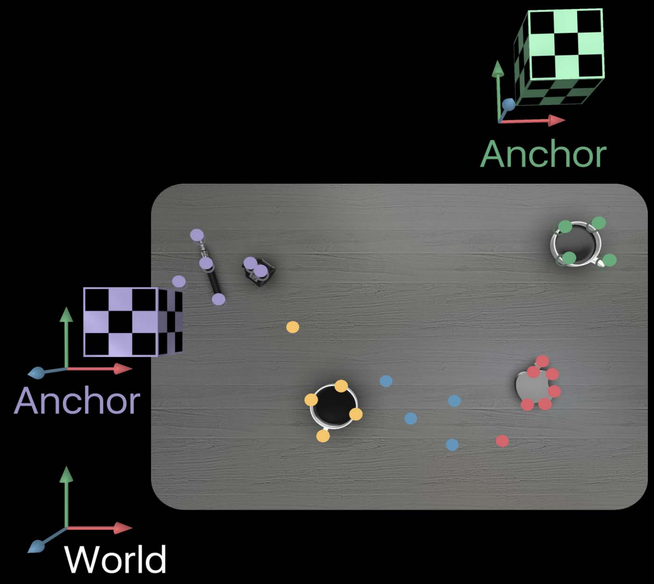


Best Practice for Using ARAnchor

Respond to anchor update delegate

Place 3D content near the ARAnchor

Multiple anchors for independent 3D content

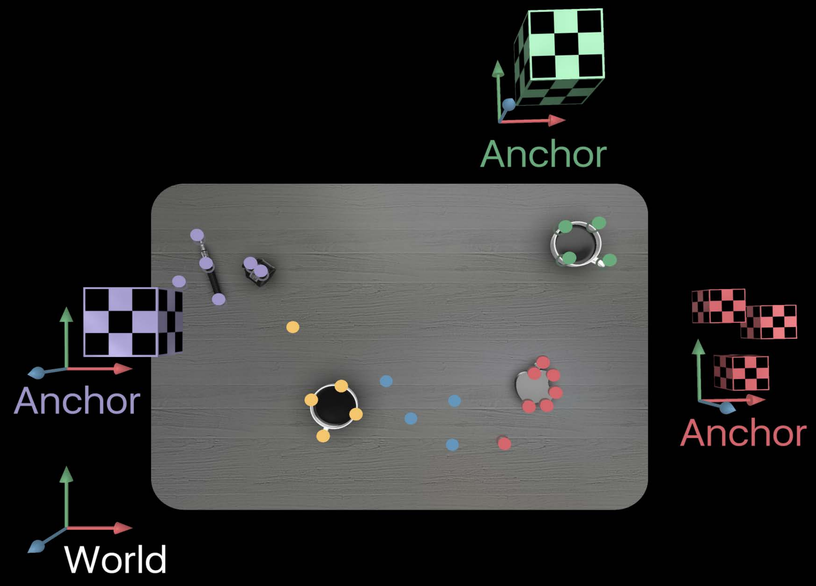


Best Practice for Using ARAnchor

Respond to anchor update delegate

Place 3D content near the ARAnchor

Multiple anchors for independent 3D content



SwiftStrike — A New Multiplayer AR Experience

David Paschich, Tools Foundation

SwiftStrike — A New Multiplayer AR Experience

Inspired by SwiftShot

All-new experience for 2019

Built with RealityKit and ARKit 3

SwiftStrike Tabletop available now

Inside SwiftShot: Creating an AR Game

WWDC 2018

RealityKit networking

Physics simulation

SwiftStrike game design

RealityKit Networking

Based on entity-component architecture

All state synchronized, including physics!

Custom components for app/game logic

Uses MultipeerConnectivity as network layer

Create network session, hand to ARView

Building Apps with RealityKit

Wednesday, 10:00

Different Roles

First device acts as “host”

Controls game state, physics simulation

Other devices participate

Custom Components in RealityKit

Define your own components to store application state

Register components before instantiating ARView

Implement `Codable` to enable synchronization

Use Case — Starting the Game

Match object tracks whether enough players are ready to start the game

State maintained on host, synced to clients

Component maintains full log


Swift
Strike


Swift
Strike

```
// Custom component for game start

struct MatchStateComponent: Component, Codable {
    struct Transition: Codable {
        var date: Date
        var state: MatchOutput
    }
    var transitions = [Transition]()
}

// Registering the component, in application(_:didFinishLaunchingWithOptions:)
MatchStateComponent.self.registerComponent()

// On client
class MatchObserver {
    var matchOutputEvents: AnyPublisher<MatchOutput, Never>
}
```

```
// Custom component for game start

struct MatchStateComponent: Component, Codable {
    struct Transition: Codable {
        var date: Date
        var state: MatchOutput
    }
    var transitions = [Transition]()
}

// Registering the component, in application(_:didFinishLaunchingWithOptions:)
MatchStateComponent.self.registerComponent()

// On client
class MatchObserver {
    var matchOutputEvents: AnyPublisher<MatchOutput, Never>
}
```

```
// Custom component for game start

struct MatchStateComponent: Component, Codable {
    struct Transition: Codable {
        var date: Date
        var state: MatchOutput
    }
    var transitions = [Transition]()
}

// Registering the component, in application(_:didFinishLaunchingWithOptions:)
MatchStateComponent.self.registerComponent()

// On client
class MatchObserver {
    var matchOutputEvents: AnyPublisher<MatchOutput, Never>
}
```

```
// Custom component for game start

struct MatchStateComponent: Component, Codable {
    struct Transition: Codable {
        var date: Date
        var state: MatchOutput
    }
    var transitions = [Transition]()
}

// Registering the component, in application(_:didFinishLaunchingWithOptions:)
MatchStateComponent.self.registerComponent()
```

```
// On client
class MatchObserver {
    var matchOutputEvents: AnyPublisher<MatchOutput, Never>
}
```

Ray
CameraInfo
GameVelocity
HitCatapult
SlingData
GrabInfo
LeverMove
PhysicsSyncData
CollisionSoundData
PhysicsSyncSceneData
PhysicsSyncSeneDataDelegate

GamePhysicsSmoothComponent
SlingshotComponent
GameBoardDescription
GameBoardLocation
LevelConfiguration
LevelInfo
LeverMove
BitStreamEncodable
BitStreamDecodable
WriteableBitStream
ReadableBitStream

Ray
CameraInfo
GameVelocity
HitCatapult
SlingData
GrabInfo
LeverMove
PhysicsSyncData
CollisionSoundData
PhysicsSyncSceneData
PhysicsSyncSeneDataDelegate

GamePhysicsSmoothComponent
SlingshotComponent
GameBoardDescription
GameBoardLocation
LevelConfiguration
LevelInfo
LeverMove
BitStreamEncodable
BitStreamDecodable
WriteableBitStream
ReadableBitStream

Ray
CameraInfo
GameVelocity
HitCatapult
SlingData
GrabInfo
LeverMove
PhysicsSyncData
CollisionSoundData
PhysicsSyncSceneData
PhysicsSyncSeneDataDelegate

GamePhysicsSmoothComponent
SlingshotComponent
GameBoardDescription
GameBoardLocation
LevelConfiguration
LevelInfo
LeverMove
BitStreamEncodable
BitStreamDecodable
WriteableBitStream
ReadableBitStream

Physics

Physics state synchronization handled by RealityKit

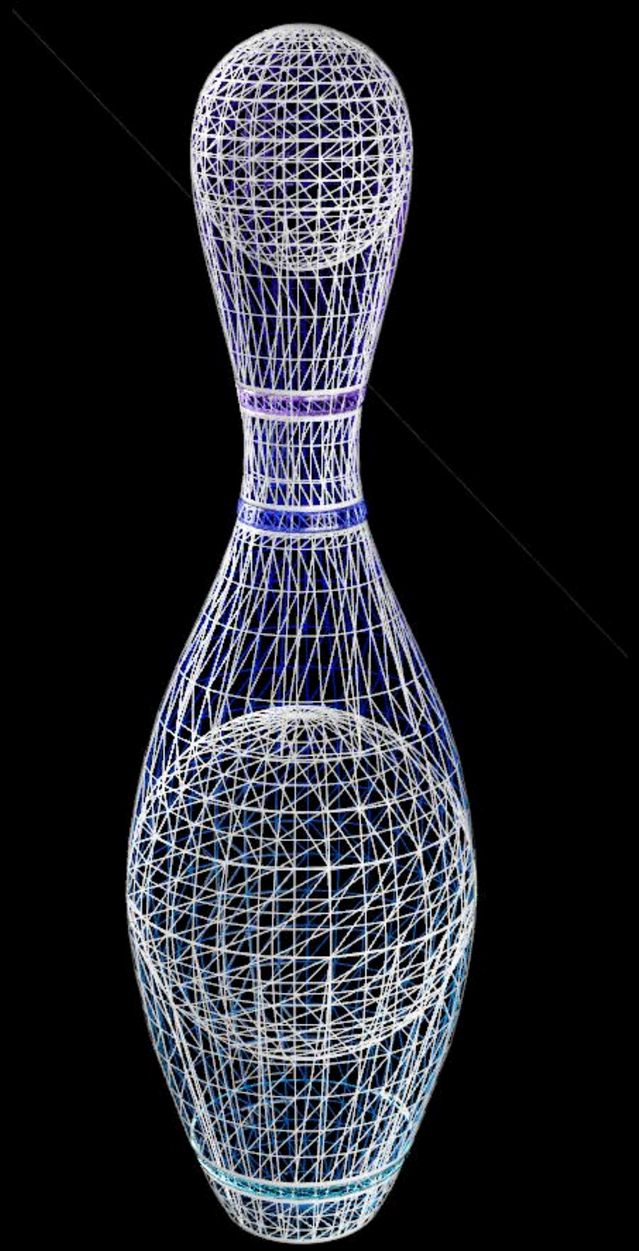
Configure physics properties via components

- Rigid bodies
- Collision masks
- Mass, friction, restitution (“bounciness”)

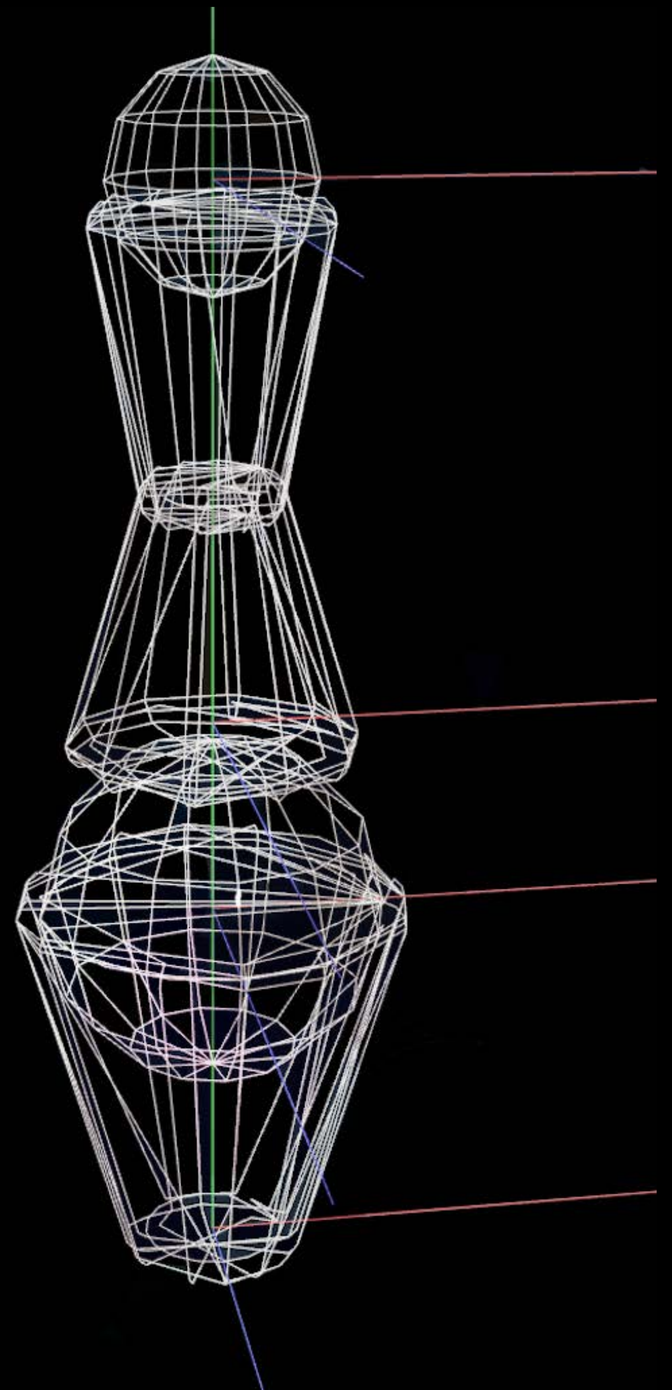
Host device owns simulation

Client devices interpolate between updates

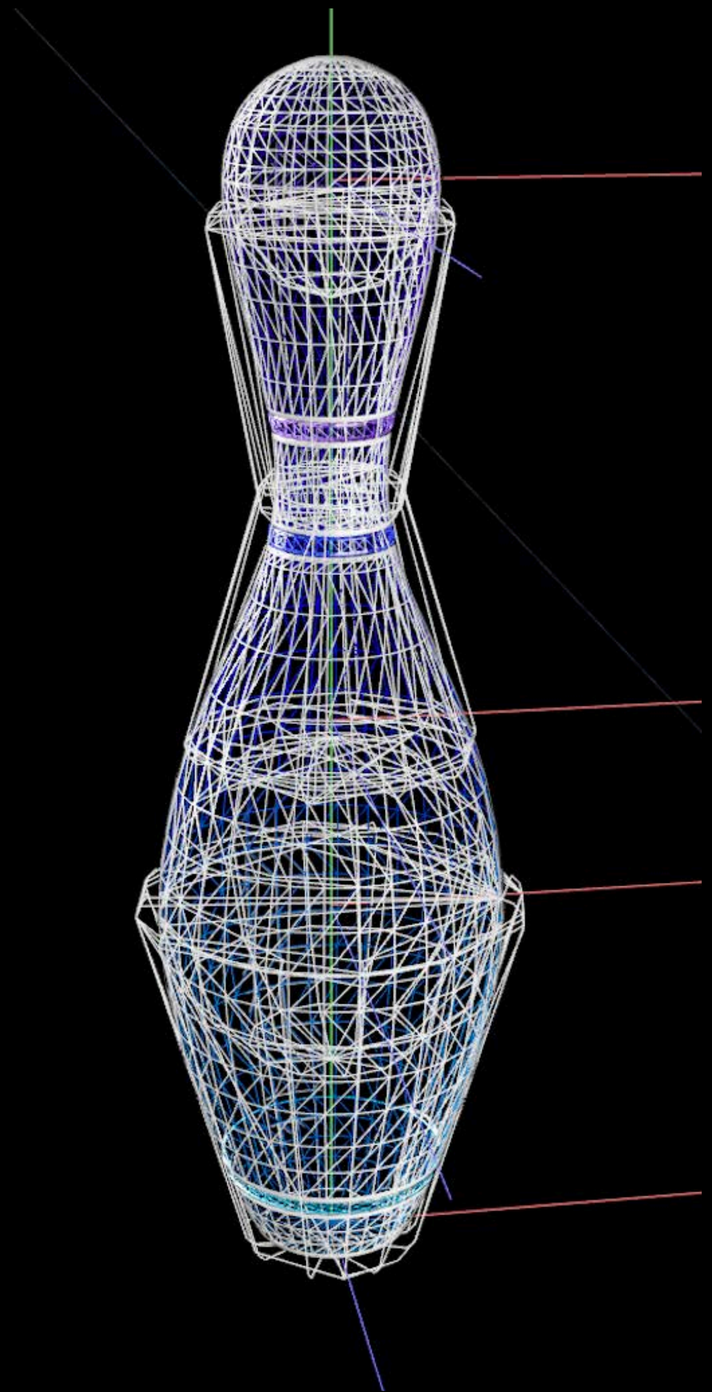
Bowling Pin



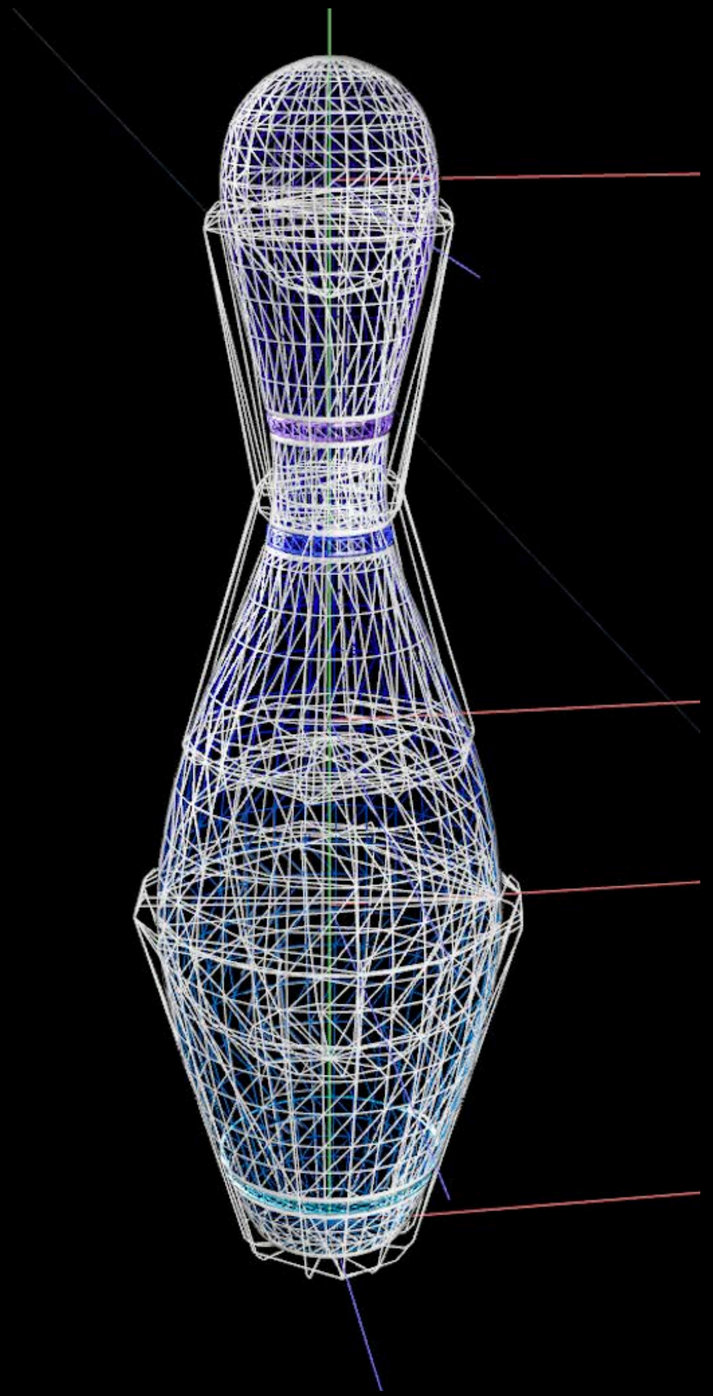
Bowling Pin



Bowling Pin



Bowling Pin



SwiftStrike Game Design

Designing for People Occlusion

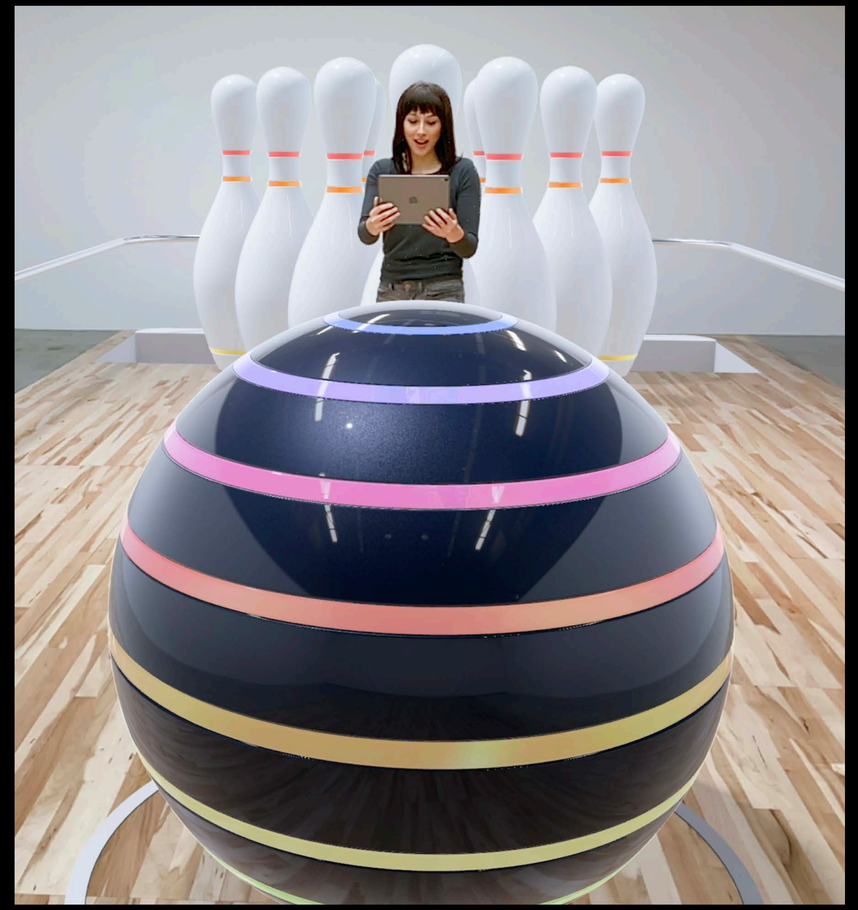
On-site experience

Control mechanism

People Occlusion

SwiftStrike designed with this in mind

Opens up new experiences



SwiftStrike On-Site Experience

Full scale experience

Floor features designed for quick localization

SwiftStrike Control Mechanism

Device is your controller

Faster movement means larger push

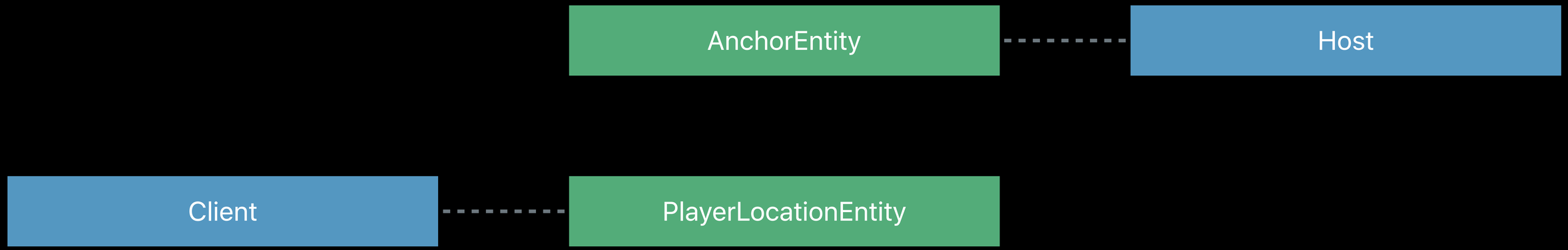
Physics body in the scene

- Ball bounces off people
- Pins and ball bounce off each other
- Pins don’t bounce off people

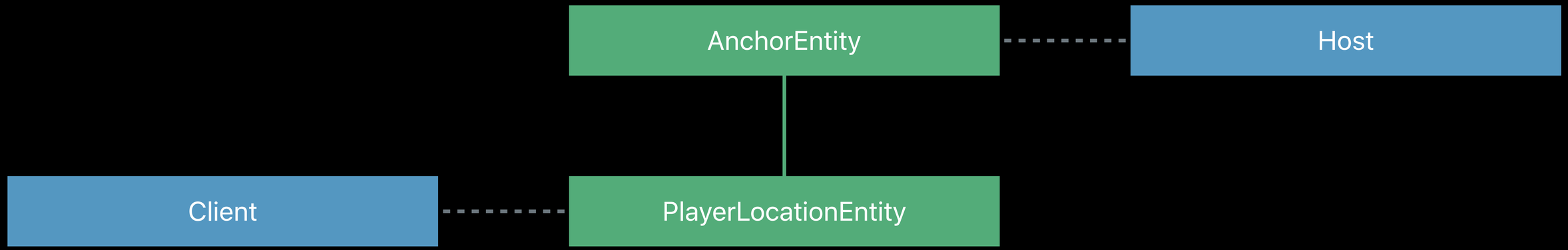
Ownership of Player Paddle

Ownership of Player Paddle

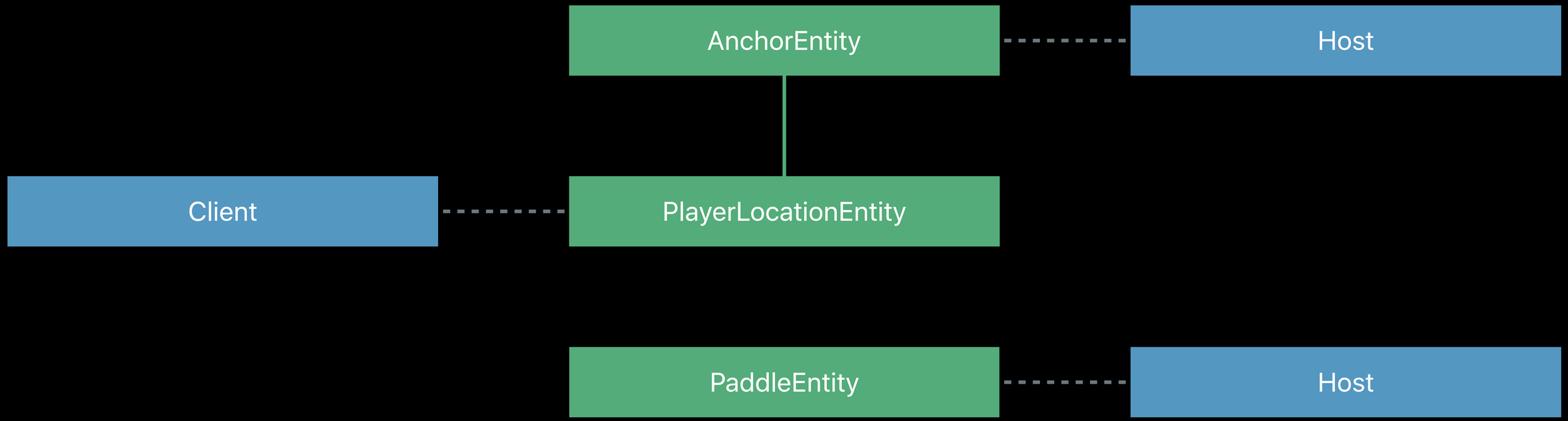
Ownership of Player Paddle



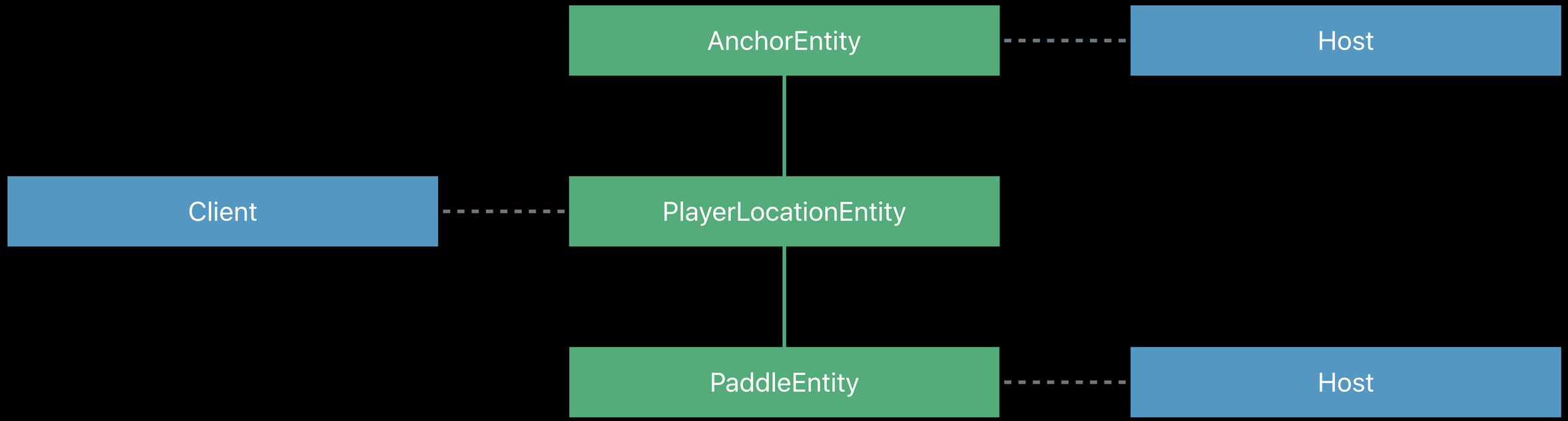
Ownership of Player Paddle

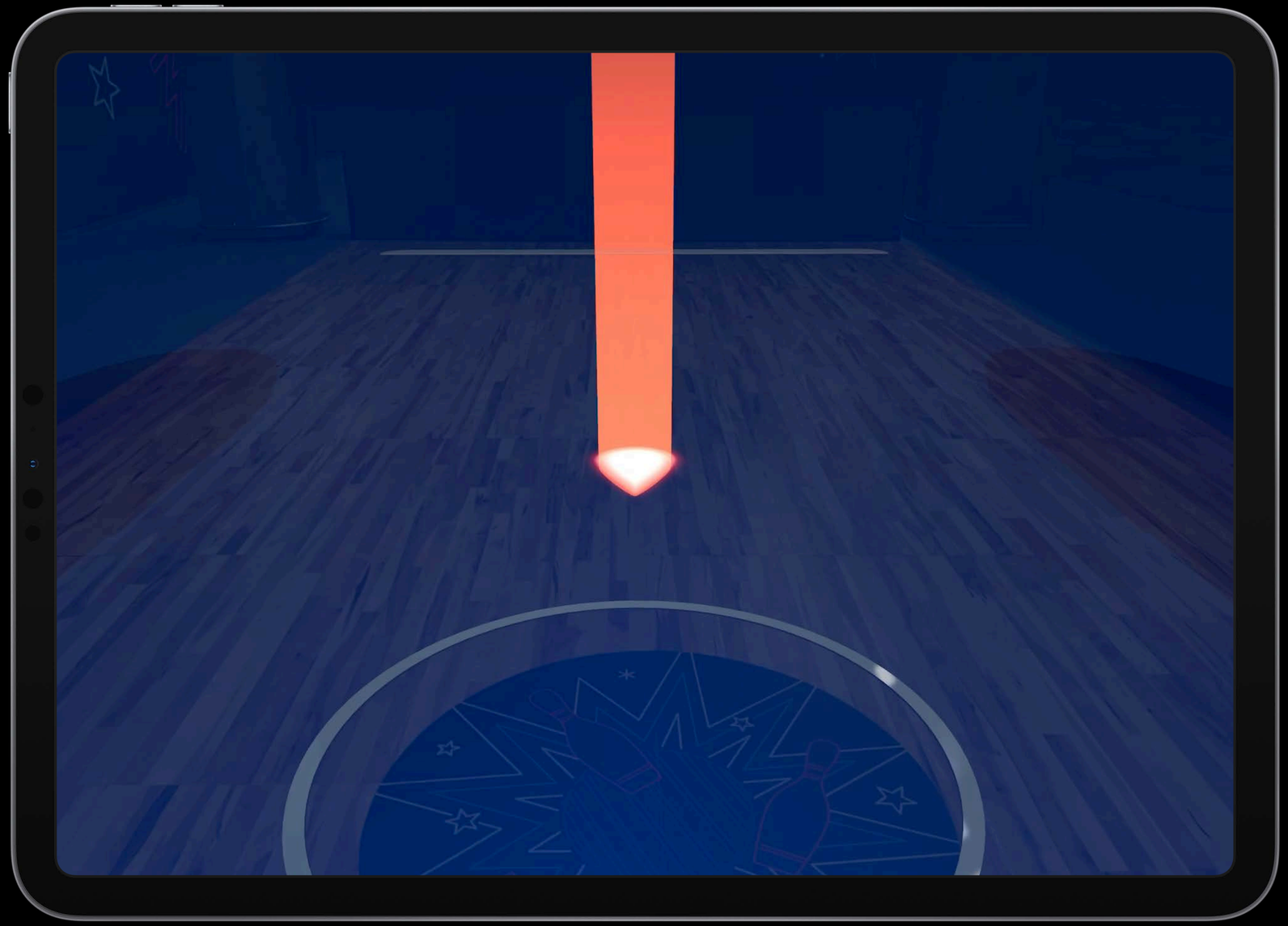


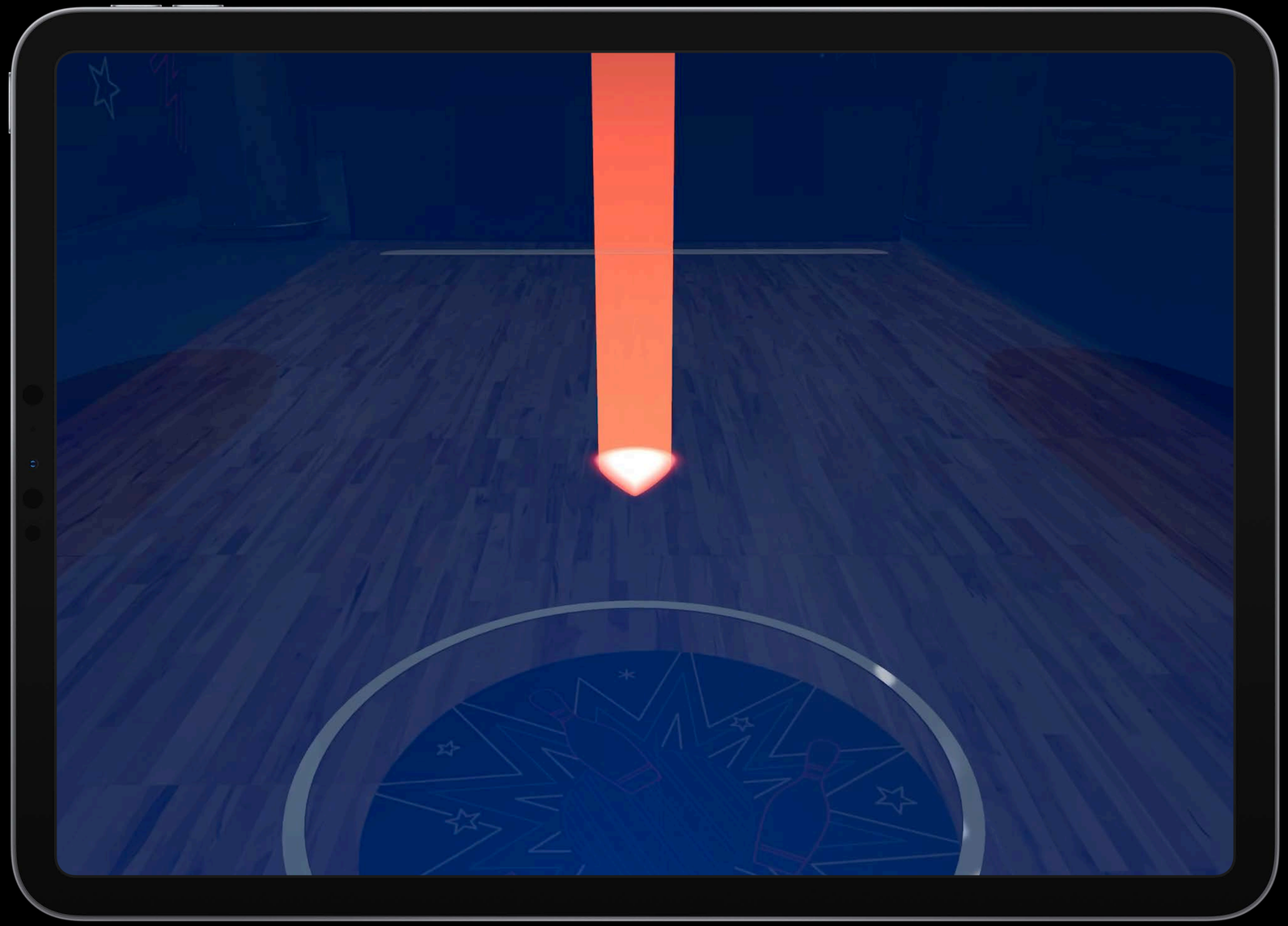
Ownership of Player Paddle



Ownership of Player Paddle







Summary

Collaborative Sessions enable new experiences

Use ARAnchors to attach content to the real world

SwiftStrike Tabletop available now!

More Information

developer.apple.com/wwdc19/610

ARKit Lab	Thursday, 3:00
RealityKit and Reality Composer Lab	Thursday, 3:00


WWDC19