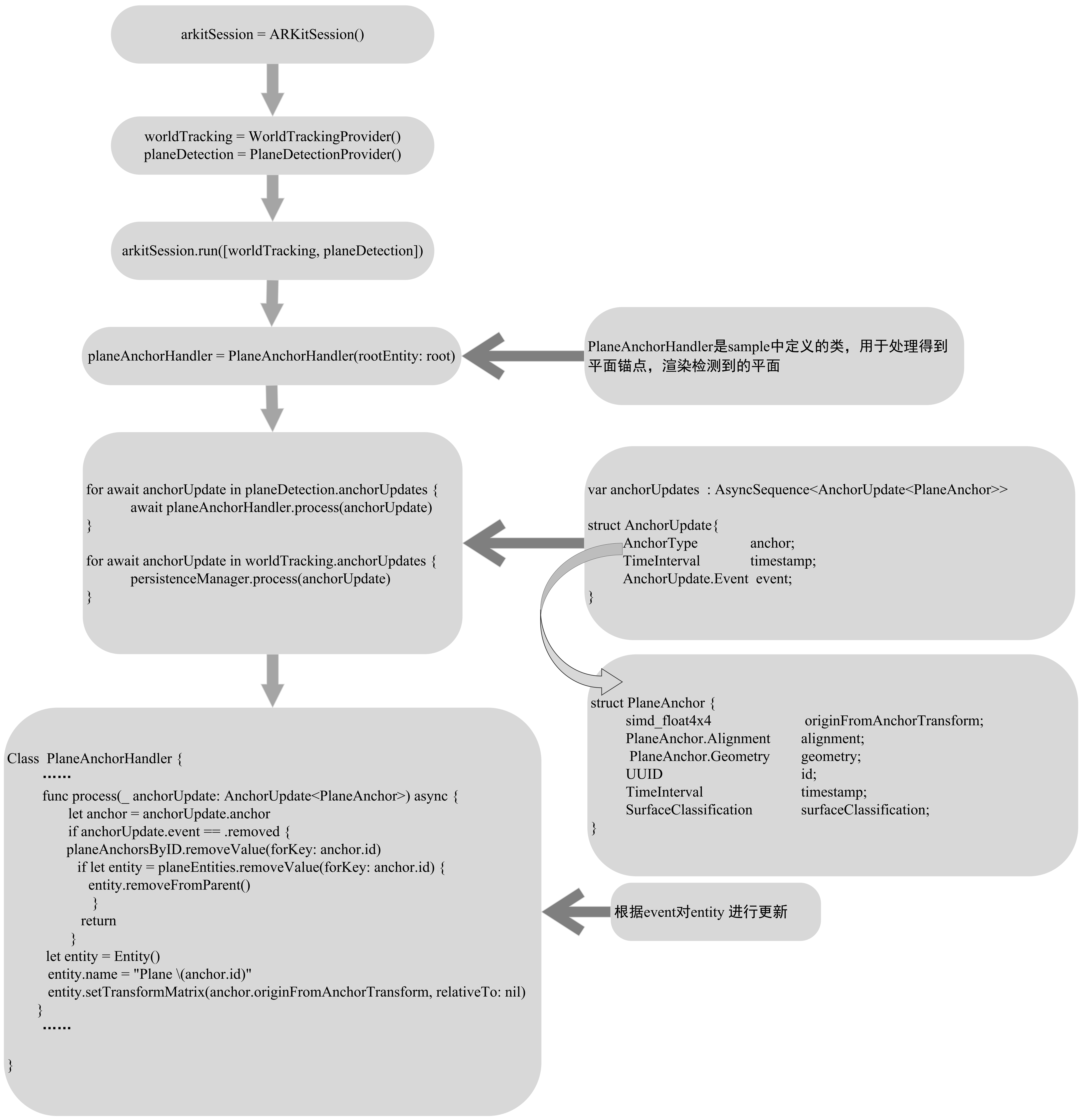


```
arkitSession = ARKitSession()
```

```
worldTracking = WorldTrackingProvider()
planeDetection = PlaneDetectionProvider()
```

```
arkitSession.run([worldTracking, planeDetection])
```

```
planeAnchorHandler = PlaneAnchorHandler(rootEntity: root)
```

PlaneAnchorHandler是sample中定义的类，用于处理得到平面锚点，渲染检测到的平面

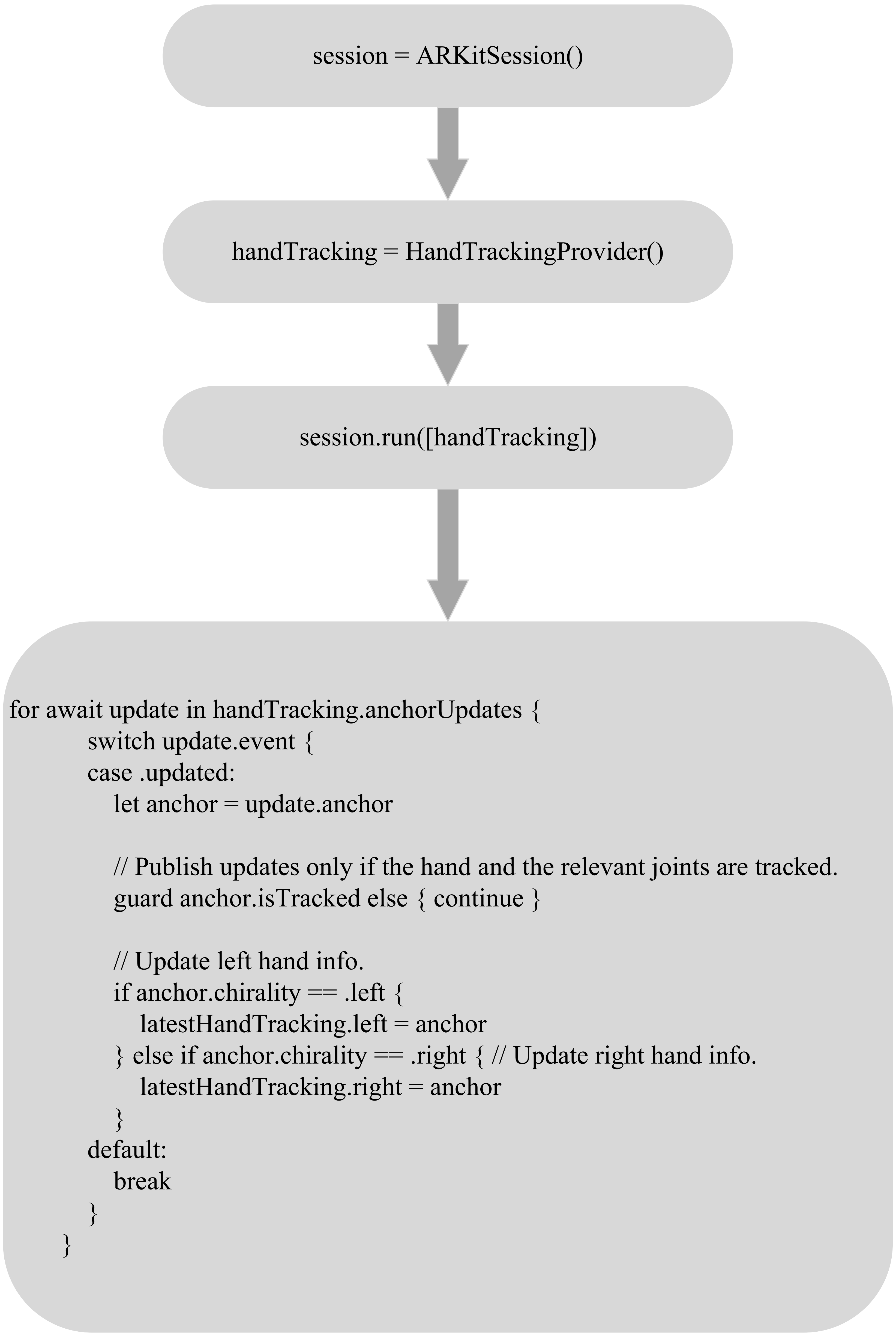
```
for await anchorUpdate in planeDetection.anchorUpdates {
          await planeAnchorHandler.process(anchorUpdate)
}

for await anchorUpdate in worldTracking.anchorUpdates {
          persistenceManager.process(anchorUpdate)
}
```

```
var anchorUpdates  : AsyncSequence<AnchorUpdate<PlaneAnchor>>

struct AnchorUpdate{
       AnchorType             anchor;
       TimeInterval           timestamp;
       AnchorUpdate.Event  event;
}
```

```
struct PlaneAnchor {
        simd_float4x4                     originFromAnchorTransform;
        PlaneAnchor.Alignment       alignment;
         PlaneAnchor.Geometry       geometry;
        UUID                              id;
        TimeInterval                    timestamp;
        SurfaceClassification         surfaceClassification;
}
```

```
Class  PlaneAnchorHandler {
        ……
        func process(_ anchorUpdate: AnchorUpdate<PlaneAnchor>) async {
             let anchor = anchorUpdate.anchor
             if anchorUpdate.event == .removed {
             planeAnchorsByID.removeValue(forKey: anchor.id)
               if let entity = planeEntities.removeValue(forKey: anchor.id) {
                 entity.removeFromParent()
                  }
               return
             }
         let entity = Entity()
          entity.name = "Plane \(anchor.id)"
          entity.setTransformMatrix(anchor.originFromAnchorTransform, relativeTo: nil)
       }
        ……


}
```

根据event对entity 进行更新

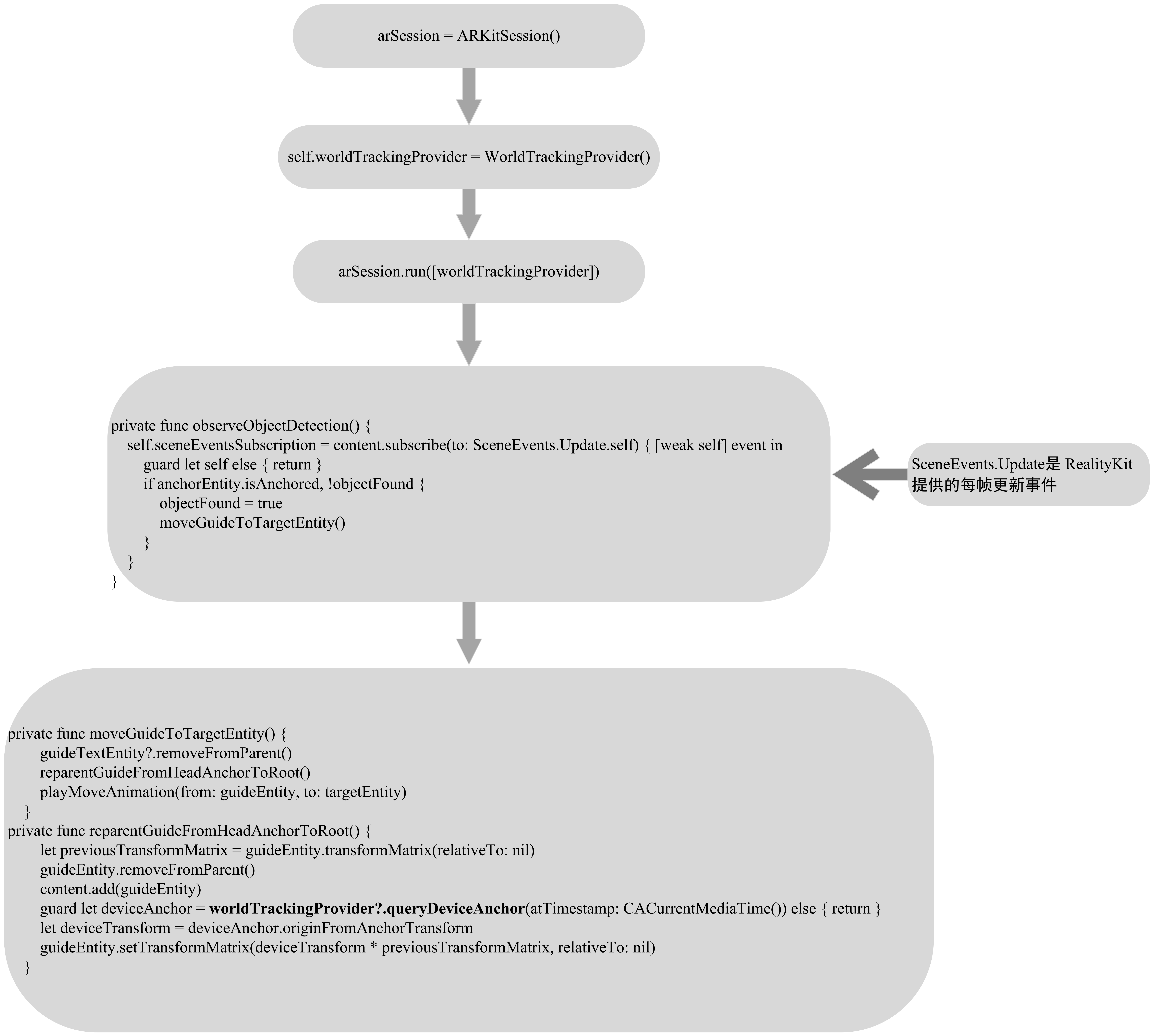


```
session = ARKitSession()
```

```
handTracking = HandTrackingProvider()
```

```
session.run([handTracking])
```

```
for await update in handTracking.anchorUpdates {
            switch update.event {
            case .updated:
                let anchor = update.anchor

                // Publish updates only if the hand and the relevant joints are tracked.
                guard anchor.isTracked else { continue }

                // Update left hand info.
                if anchor.chirality == .left {
                    latestHandTracking.left = anchor
                } else if anchor.chirality == .right { // Update right hand info.
                    latestHandTracking.right = anchor
                }
            default:
                break
            }
        }
```



```
arSession = ARKitSession()
```

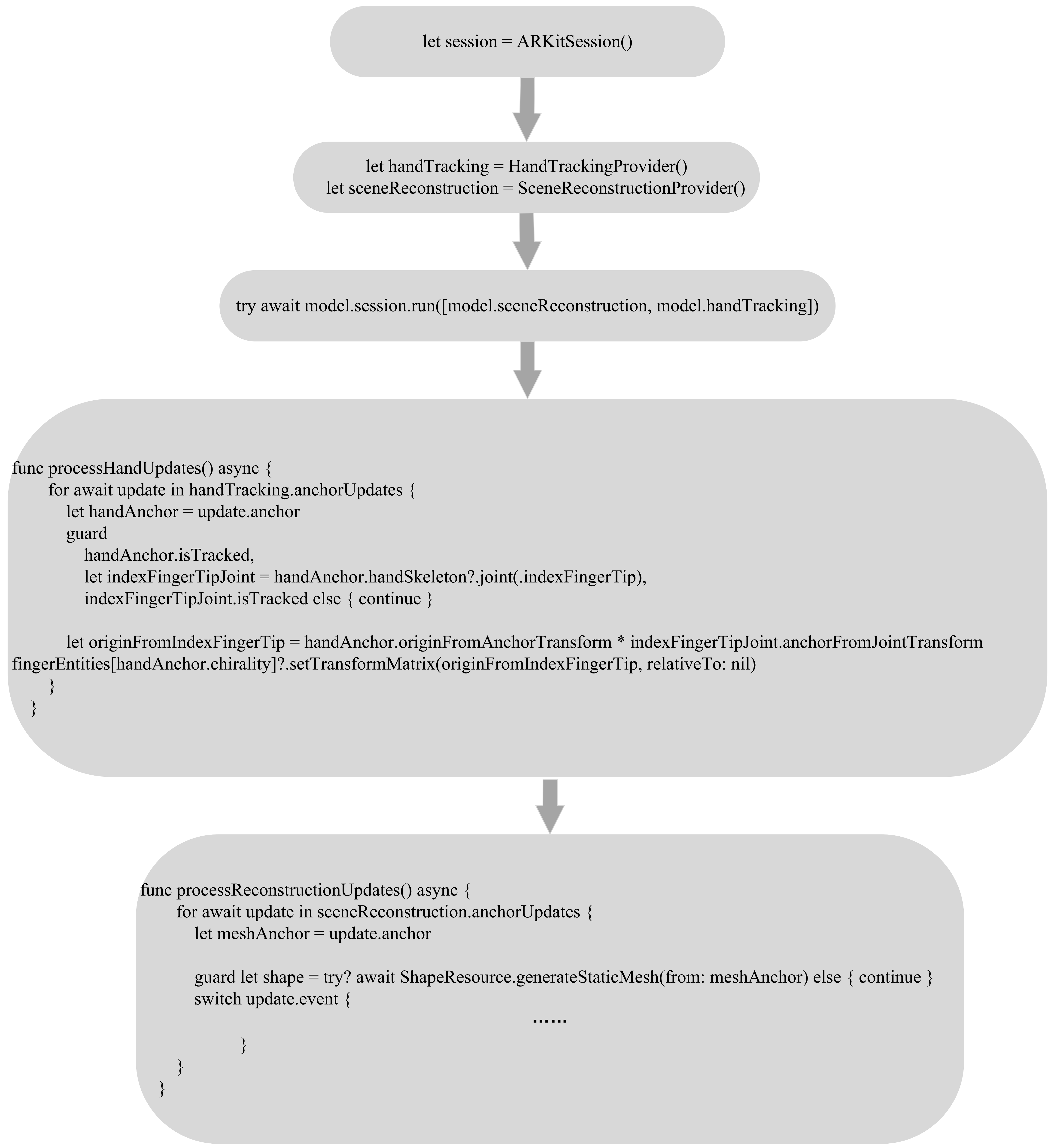
```
self.worldTrackingProvider = WorldTrackingProvider()
```

```
arSession.run([worldTrackingProvider])
```

```
private func observeObjectDetection() {
    self.sceneEventsSubscription = content.subscribe(to: SceneEvents.Update.self) { [weak self] event in
        guard let self else { return }
        if anchorEntity.isAnchored, !objectFound {
            objectFound = true
            moveGuideToTargetEntity()
        }
    }
}
```

SceneEvents.Update是 RealityKit提供的每帧更新事件

```
private func moveGuideToTargetEntity() {
        guideTextEntity?.removeFromParent()
        reparentGuideFromHeadAnchorToRoot()
        playMoveAnimation(from: guideEntity, to: targetEntity)
    }
private func reparentGuideFromHeadAnchorToRoot() {
        let previousTransformMatrix = guideEntity.transformMatrix(relativeTo: nil)
        guideEntity.removeFromParent()
        content.add(guideEntity)
        guard let deviceAnchor = worldTrackingProvider?.queryDeviceAnchor(atTimestamp: CACurrentMediaTime()) else { return }
        let deviceTransform = deviceAnchor.originFromAnchorTransform
        guideEntity.setTransformMatrix(deviceTransform * previousTransformMatrix, relativeTo: nil)
    }
```



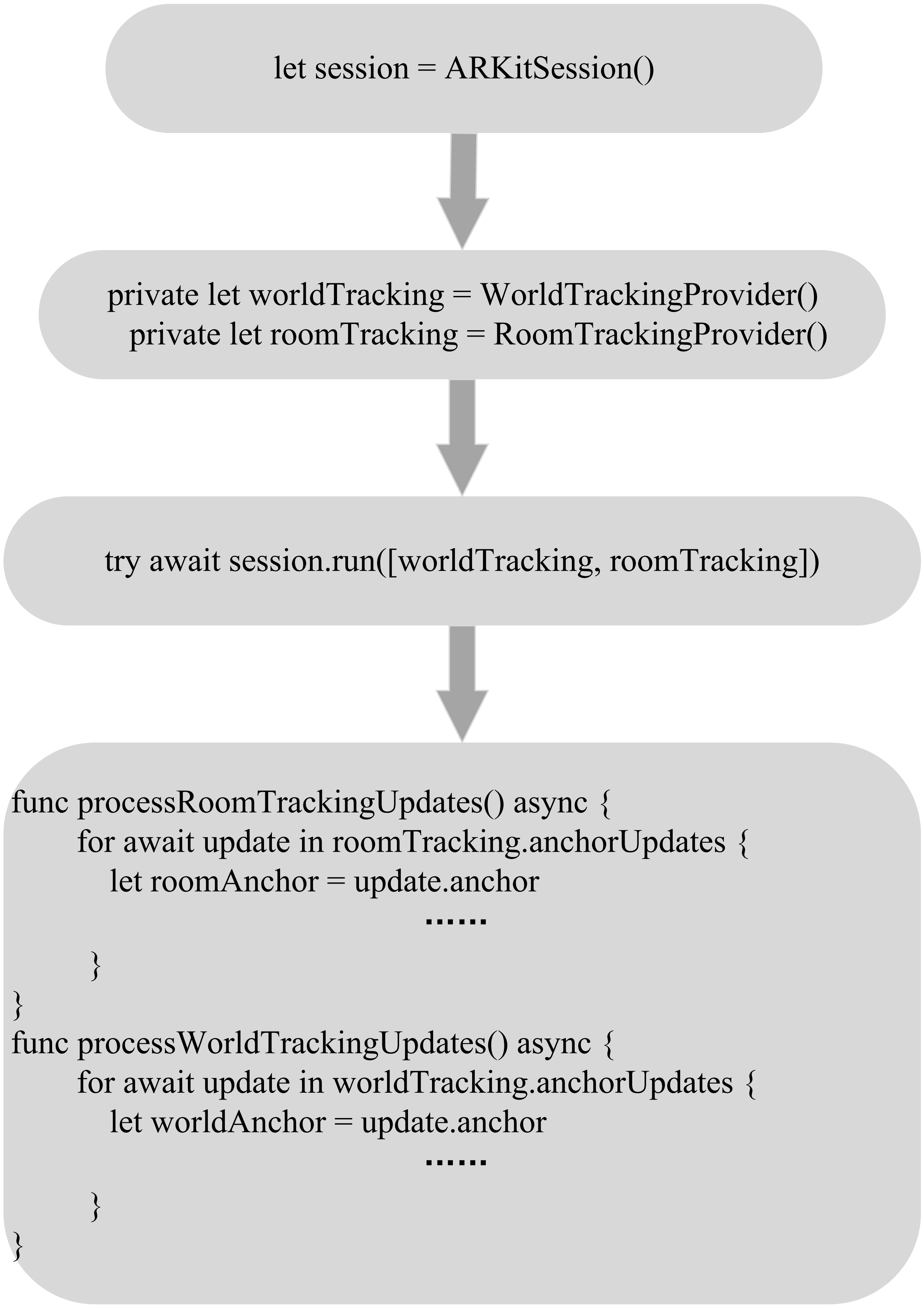
```
let session = ARKitSession()
```

```
let handTracking = HandTrackingProvider()
let sceneReconstruction = SceneReconstructionProvider()
```

```
try await model.session.run([model.sceneReconstruction, model.handTracking])
```

```
func processHandUpdates() async {
        for await update in handTracking.anchorUpdates {
            let handAnchor = update.anchor
            guard
                handAnchor.isTracked,
                let indexFingerTipJoint = handAnchor.handSkeleton?.joint(.indexFingerTip),
                indexFingerTipJoint.isTracked else { continue }

            let originFromIndexFingerTip = handAnchor.originFromAnchorTransform * indexFingerTipJoint.anchorFromJointTransform
fingerEntities[handAnchor.chirality]?.setTransformMatrix(originFromIndexFingerTip, relativeTo: nil)
        }
    }
```

```
func processReconstructionUpdates() async {
        for await update in sceneReconstruction.anchorUpdates {
            let meshAnchor = update.anchor

            guard let shape = try? await ShapeResource.generateStaticMesh(from: meshAnchor) else { continue }
            switch update.event {
                                                  ......
                       }
        }
    }
```

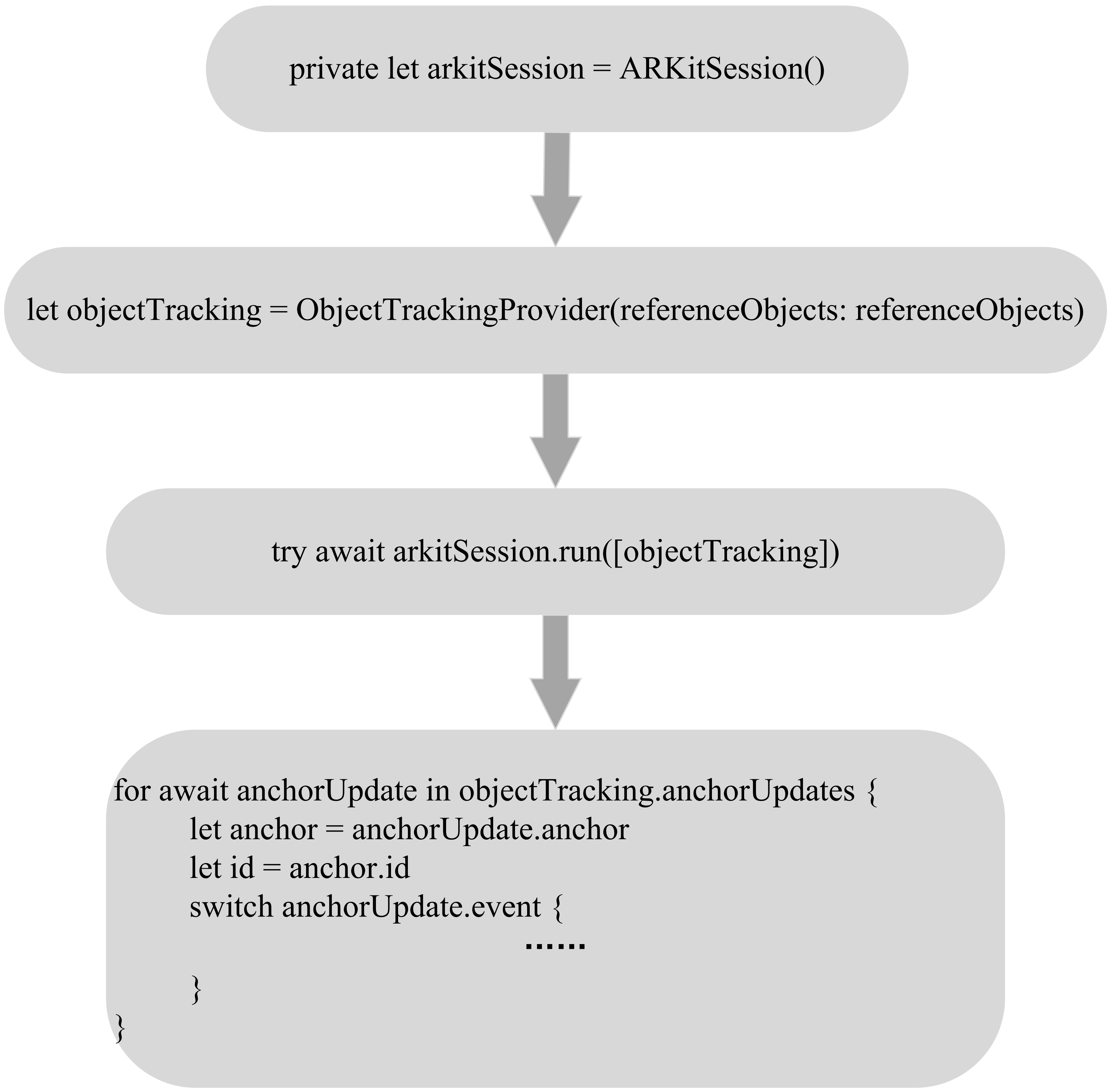


```
let session = ARKitSession()
```

```
private let worldTracking = WorldTrackingProvider()
  private let roomTracking = RoomTrackingProvider()
```

```
try await session.run([worldTracking, roomTracking])
```

```
func processRoomTrackingUpdates() async {
        for await update in roomTracking.anchorUpdates {
            let roomAnchor = update.anchor
                                          ……
         }
}
func processWorldTrackingUpdates() async {
        for await update in worldTracking.anchorUpdates {
            let worldAnchor = update.anchor
                                          ……
         }
}
```



```
private let arkitSession = ARKitSession()
```

```
let objectTracking = ObjectTrackingProvider(referenceObjects: referenceObjects)
```

```
try await arkitSession.run([objectTracking])
```

```
for await anchorUpdate in objectTracking.anchorUpdates {
        let anchor = anchorUpdate.anchor
        let id = anchor.id
        switch anchorUpdate.event {
                                ……
        }
}
```