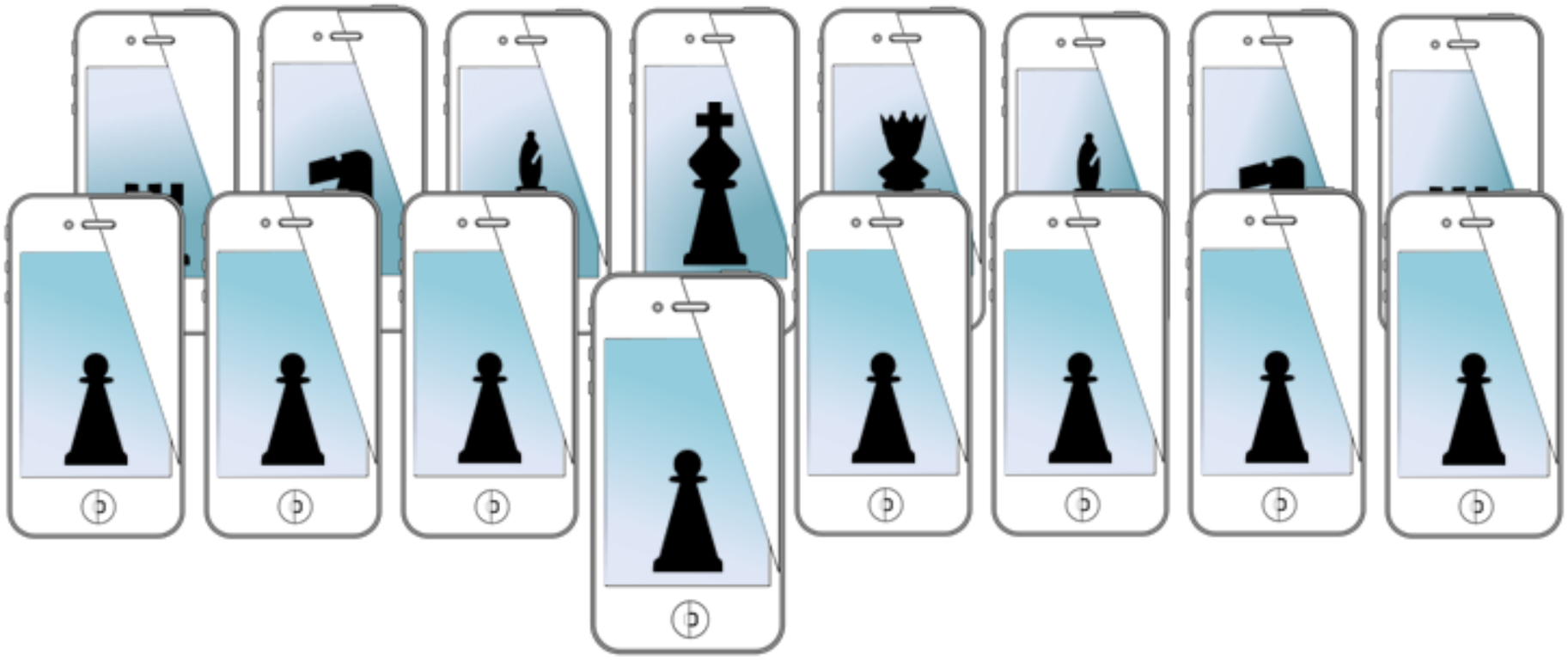
MOBILE SENSING LEARNING



CS5323 & 7323

Mobile Sensing and Learning

CoreML and ARKit

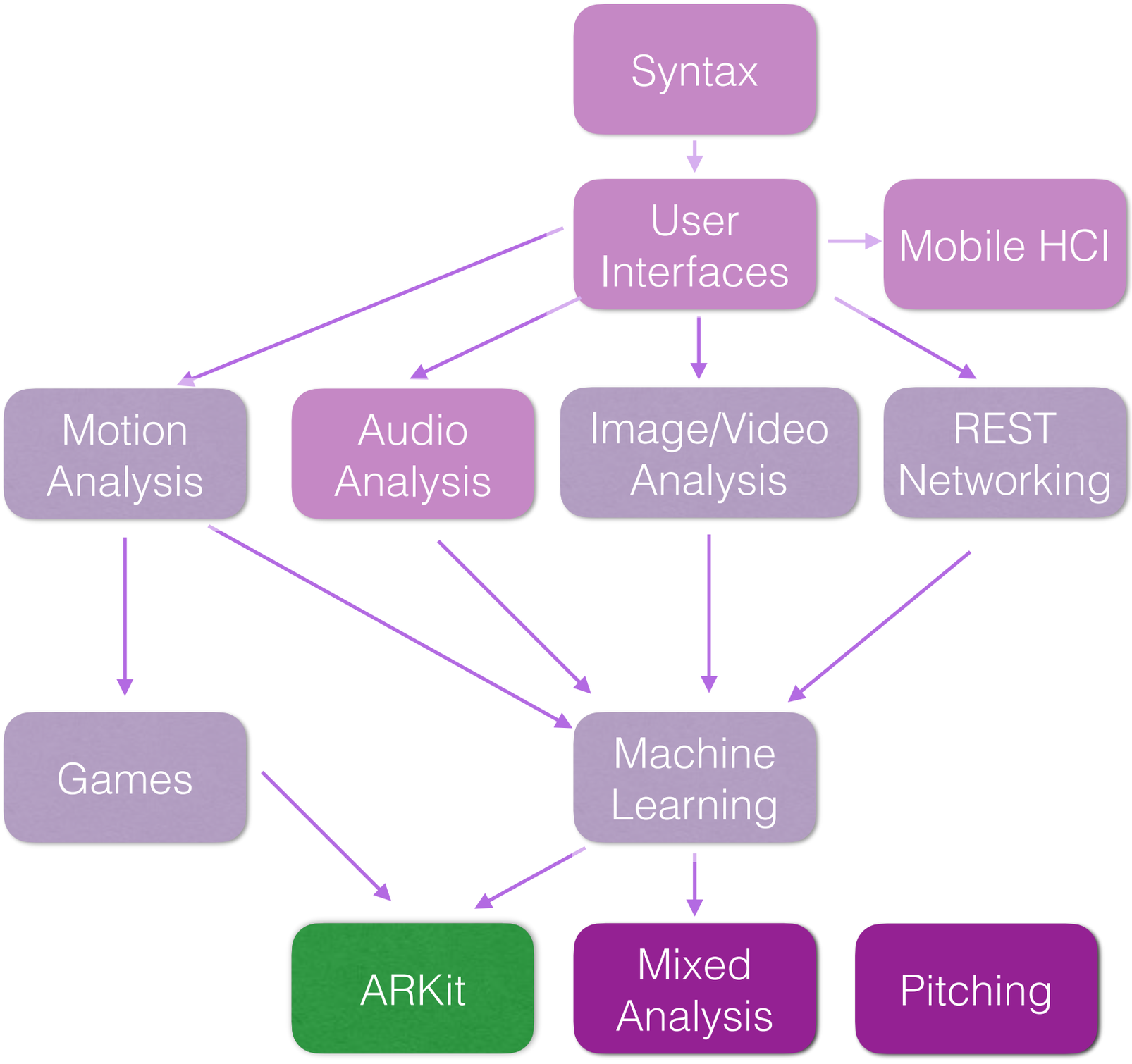
Eric C. Larson, Lyle School of Engineering,
Computer Science, Southern Methodist University

course logistics and agenda

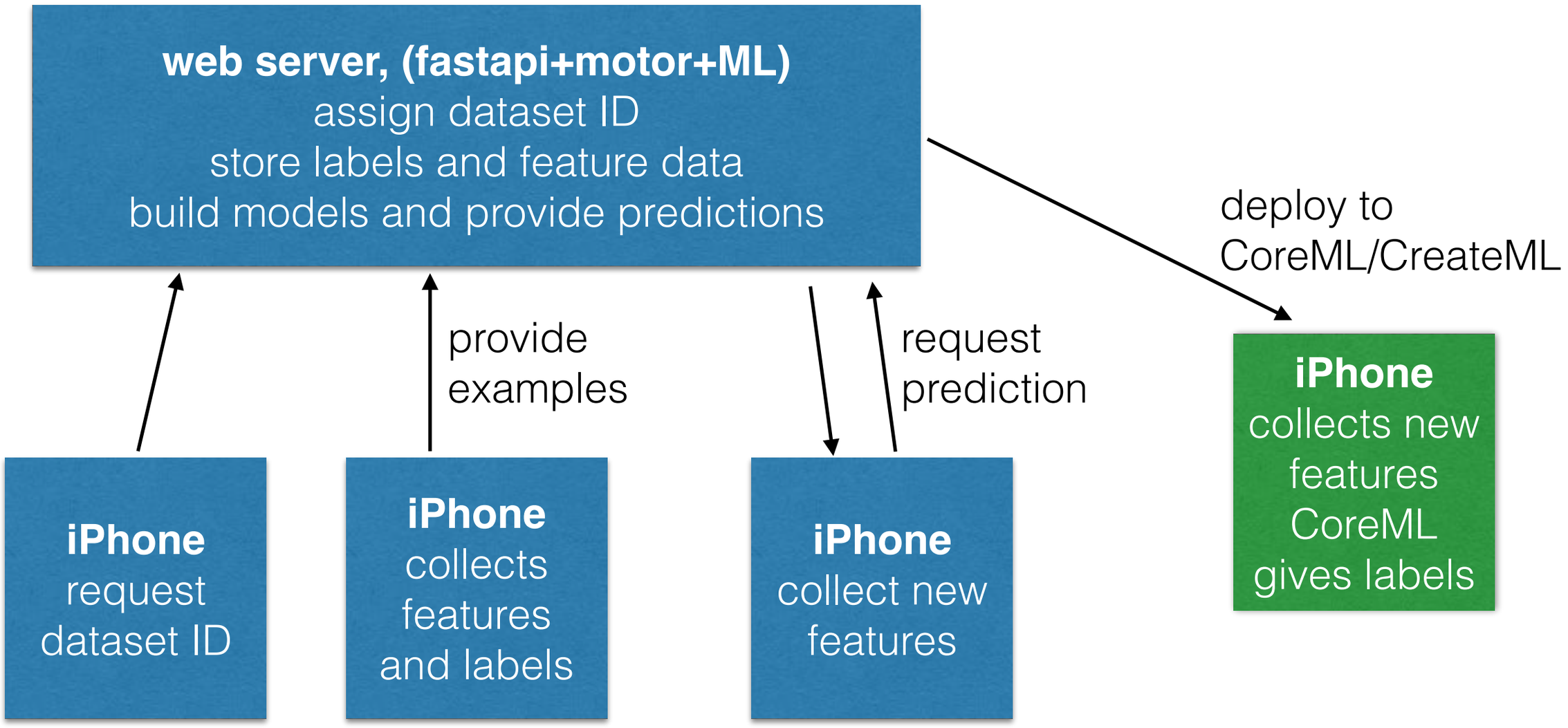
- agenda:
  - ARKit
  - ARKit demo
  - ARKit Object Recognition
  - If time: Using YOLO with ARKit integration

https://developer.apple.com/videos/play/wwdc2017/602/

class overview



assignment 5 demo



all uploads handled via JSON formatted HTTP requests

most of code is already functional

Lab Five and Proposals

- Questions on lab five
- Questions on proposals

Members:

Design: No, NOT opting into the MOD

Description: I want to develop a guitar chord recognition program that allows users to identify various guitar chords through their mobile phones. The reason for the design is that guitar players often cannot find the right chord in the process of learning the guitar. When a user finds good music, they can use chord analysis software to analyze how to play accompaniments. The app will analyze a guitar chord in real time, send the data to the server for online analysis, and display the name and fingering of the current guitar chord at the bottom of the UI. There is an app on the app store that does this, called Chord AI. It is unclear if the app is very reliable.

Constraints:

1. The program will be able to detect and analyze the guitar chords collected by the microphone. Since there are nearly 1000 guitar chords, the accuracy of the analyzed chords will mainly be the main chords that are common and used by most people. These include major and minor variants of different base notes including the sharps and flats. The chord will always have the main note in the base (i.e., no inversions).
2. The only interface of this program is the logic of real-time analysis, so the Prediction function will be automatically opened and analyzed on the main page, always running.
3. The real-time analysis audio interface of the program will also display the guitar fingering diagram corresponding to the actual chord.
4. The program will use ML as a Service to run as a server and will send data HTTP requests to ML on the server to request prediction results.

Elements of Labs:

1. UI
2. Audio Processing
3. Machine Learning as a Service

work flow in 3D scenes

SCNScene

rootNode

SCNNode

Name
Camera
Light
Geometry

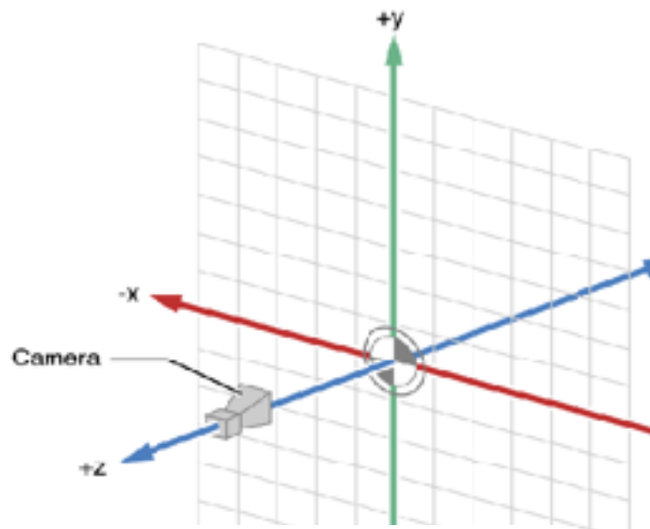
childNodes[]

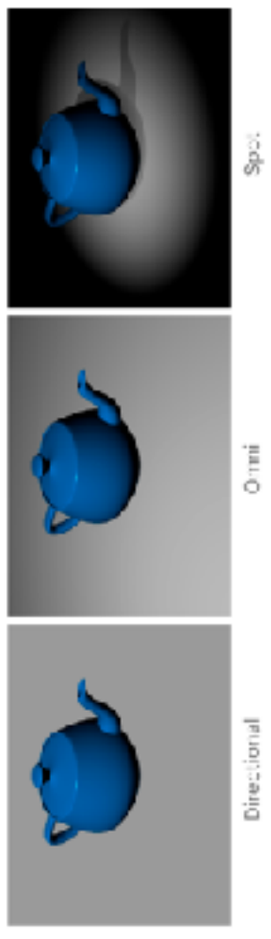
SCNGeometry

Name
Materials[]
Text
Shape (plane, box, sphere, cylinder…)

SCNMaterial

Name
Diffuse
Ambient
Specular
Reflective
…



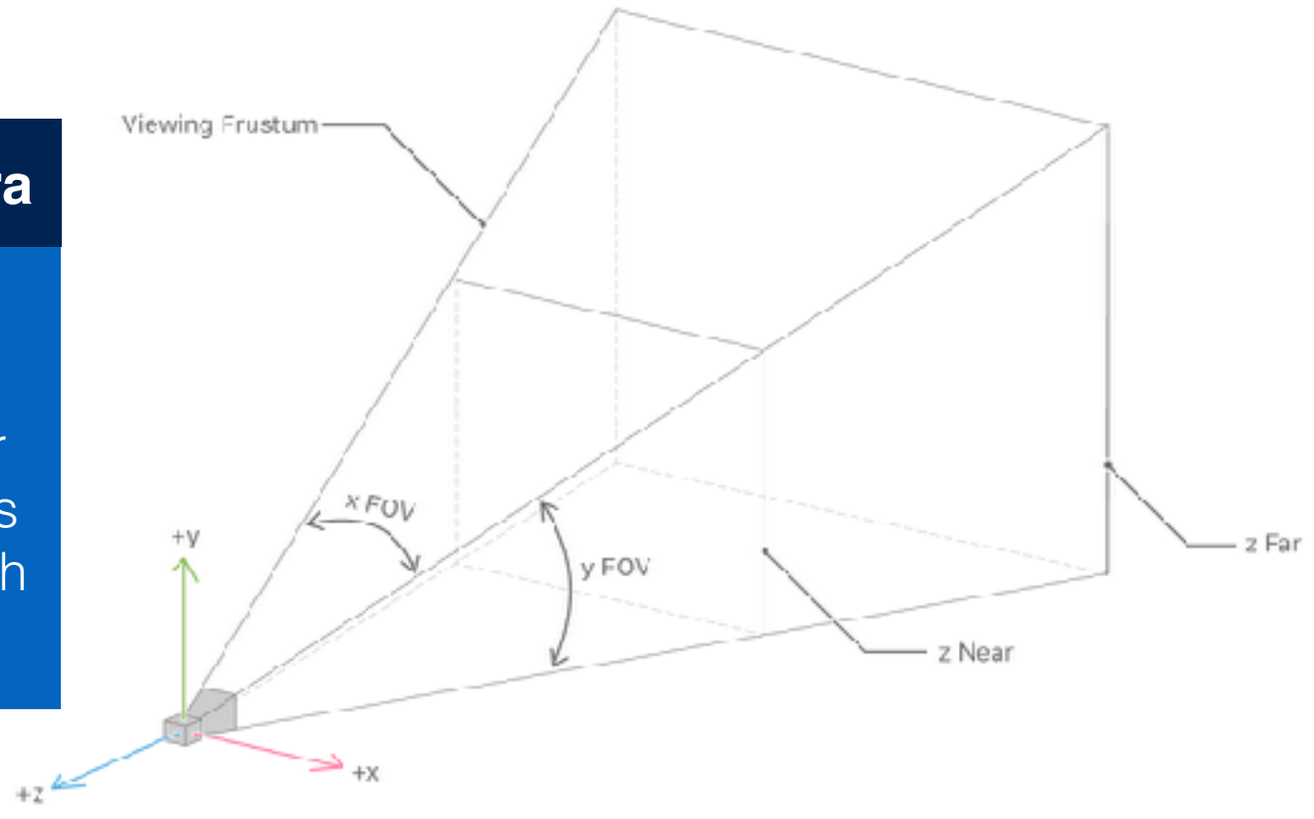


SCNLight

Type (ambient, omni, spot, directional)
Color
Temperature
Intensity
…

SCNCamera

Location
FoV
zNear / zFar
Euler Angles
Focal Length
…



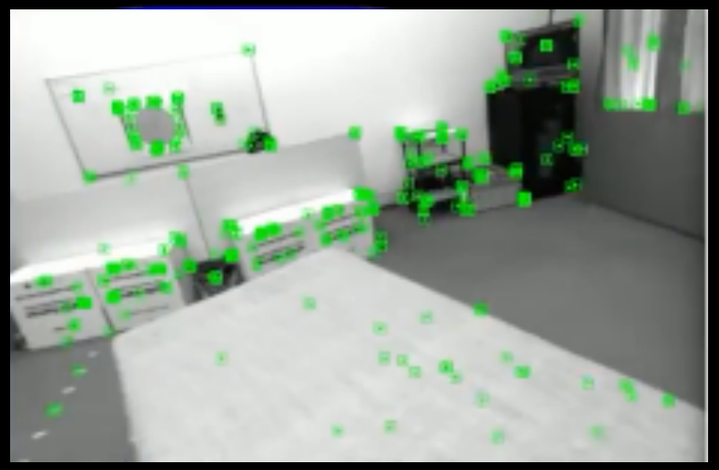
SCNNode is the base for nearly everything in simulation env.

but… we want the SCNScene
to be the real world
ARKit anchors the SCNScene to
objects in the immediate environment

```
// Setup view
let view = self.view as SCNView
view.scene = scene
```

```
// Setup view
let view = self.view as ARSCNView
view.scene = scene
```

ARKit basics



Tracking

World tracking

Visual inertial odometry

No external setup

Rendering

Easy integration

AR views

Custom rendering

Scene Understanding

Plane detection

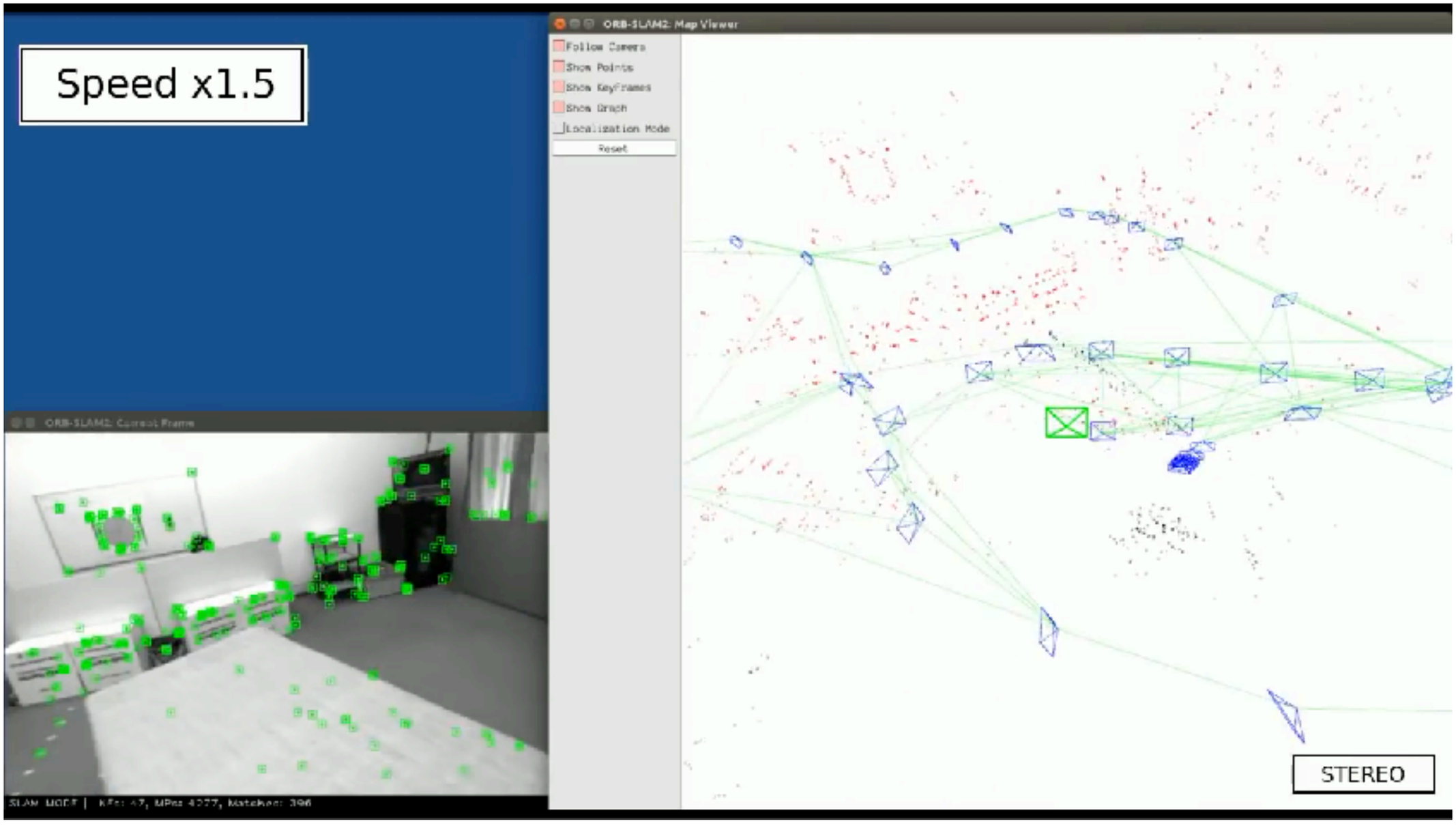
Hit-testing

Light estimation

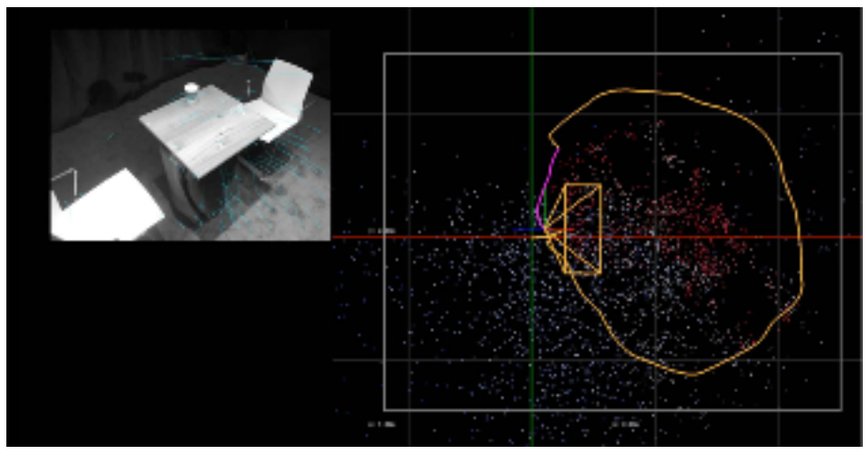
https://developer.apple.com/videos/play/wwdc2017/602/

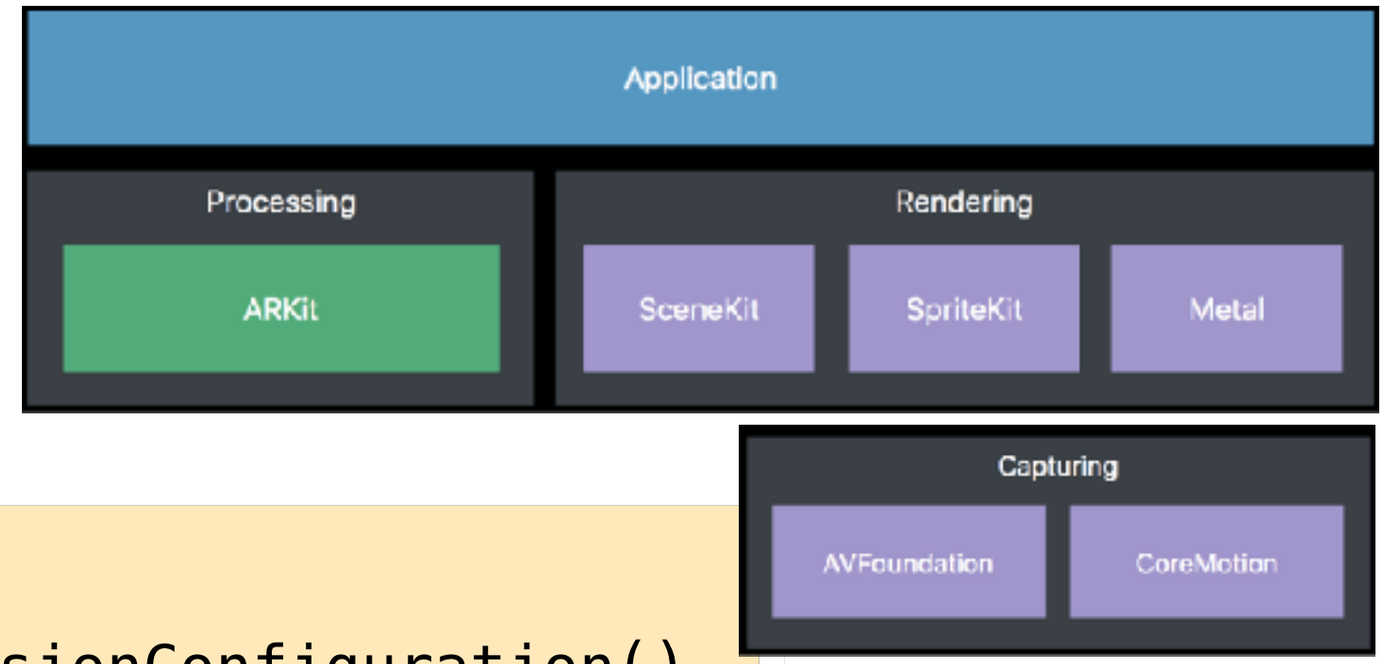
SLAM

- ORB-SLAM: Uses Key points in Image
- Simultaneous Localization and Mapping

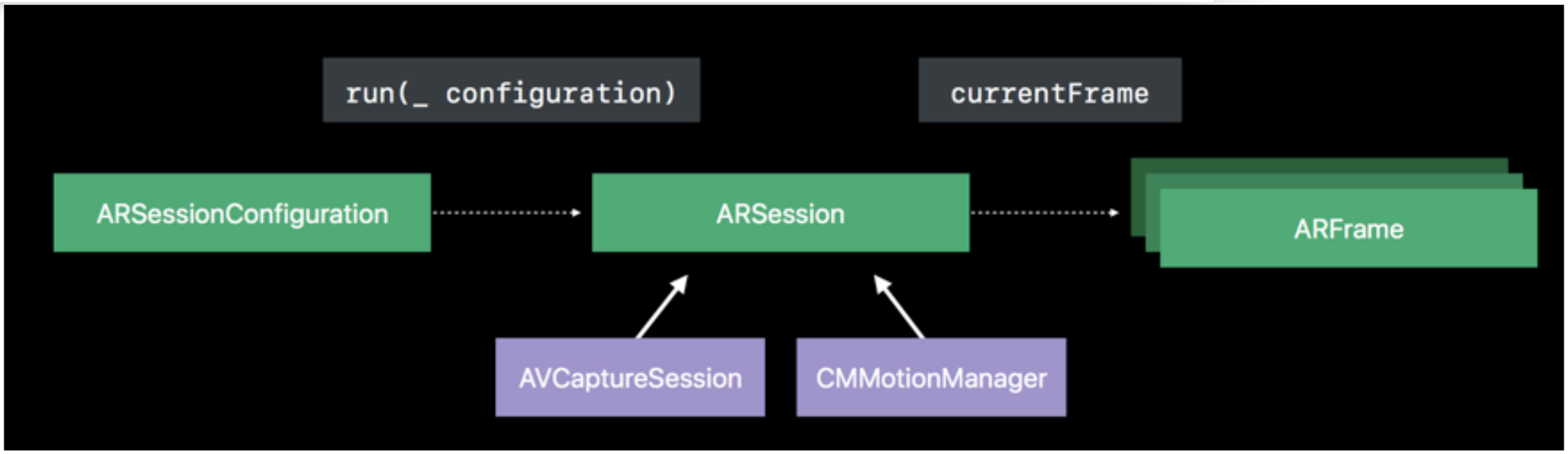


ARKit system integration



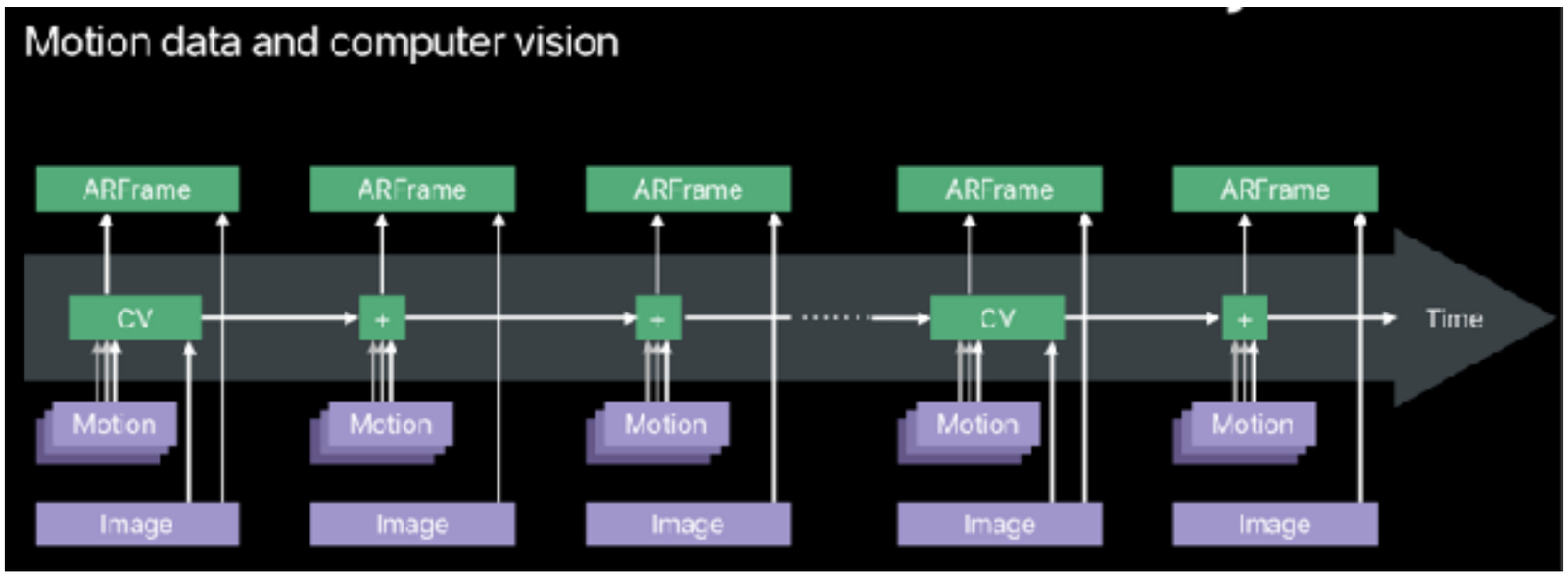


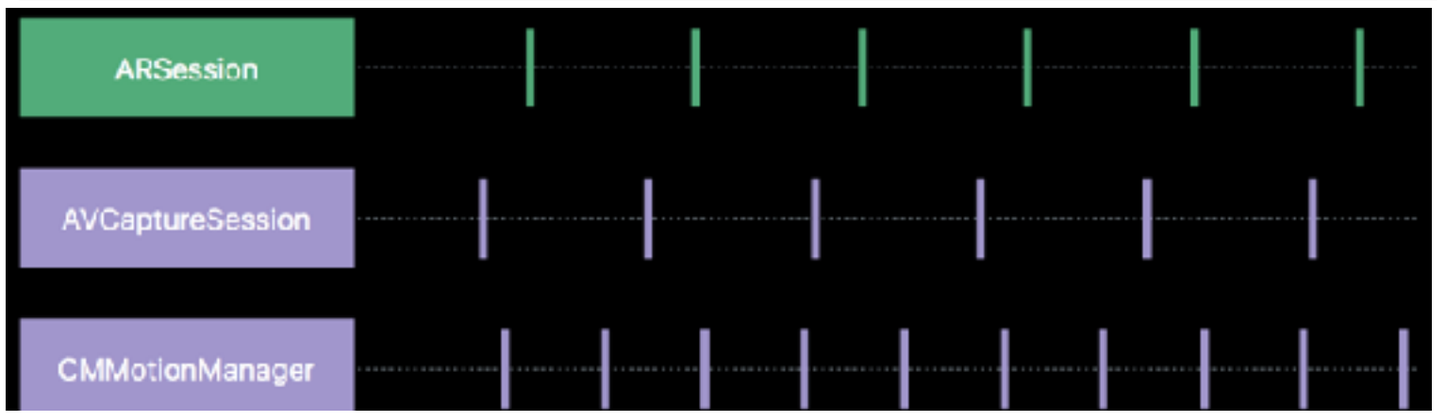

```
let session = ARSession()
let conf = ARWorldTrackingSessionConfiguration()
session.run(conf)
```



https://developer.apple.com/videos/play/wwdc2017/602/

ARKit Over Time





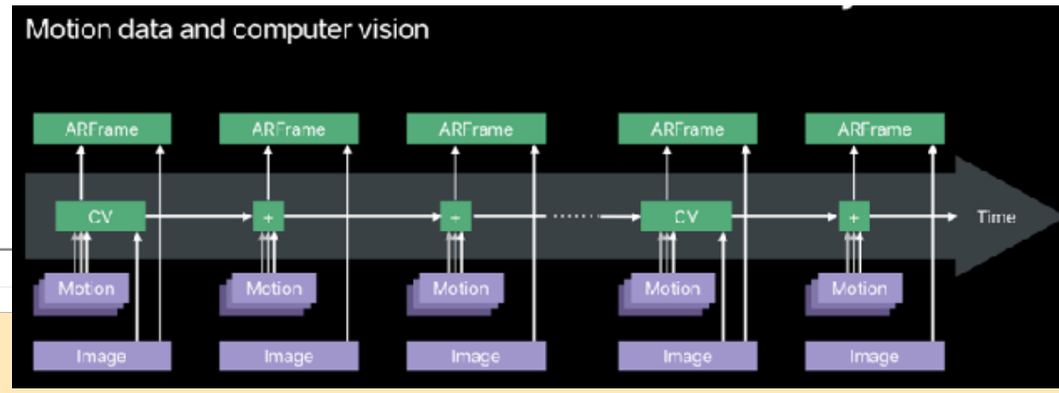
unified sampling

```
// Access the latest frame
func session(_: ARSession, didUpdate: ARFrame)

// Handle session errors
func session(_: ARSession, didFailWithError: Error)
```

delegation

https://developer.apple.com/videos/play/wwdc2018/610/

the AR world



poll ARSession, get ARFrame

```
// grab the current AR session frame from the scene, if possible
guard let currentFrame = sceneView.session.currentFrame else {
    return
}

// setup some geometry for a simple plane
let imagePlane = SCNPlane(width:sceneView.bounds.width/6000,
                          height:sceneView.bounds.height/6000)
```

create a plane

```
// add the node to the scene
let planeNode = SCNNode(geometry:imagePlane)
sceneView.scene.rootNode.addChildNode(planeNode)

// update the node to be a bit in front of the camera inside the AR session


// step one create a translation transform
var translation = matrix_identity_float4x4
translation.columns.3.z = -0.1

// step two, apply translation relative to camera for the node
planeNode.simdTransform = matrix_multiply(currentFrame.camera.transform, translation )
```

add to world

translate

operations

Figure 1-8 Matrix configurations for common transformations

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$
Identity

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ tx & ty & tz & 1 \end{bmatrix}$$
Translate

$$\begin{bmatrix} sx & 0 & 0 & 0 \\ 0 & sy & 0 & 0 \\ 0 & 0 & sz & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$
Scale

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & \sin\theta & 0 \\ 0 & -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$
Rotate around X axis

$$\begin{bmatrix} \cos\theta & 0 & -\sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ \sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$
Rotate around Y axis

$$\begin{bmatrix} \cos\theta & \sin\theta & 0 & 0 \\ -\sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$
Rotate around Z axis

```
// step one create a translation transform
var translation = matrix_identity_float4x4
translation.columns.3.z = -0.1

// step two, apply translation relative to camera for th
planeNode.simdTransform = matrix_multiply(currentFrame.c
```

SIMD: single instruction, multiple data

Creating Transform Matrices

`func SCNMatrix4MakeTranslation(Float, Float, Float)`

Returns a matrix describing a translation transformation.

`func SCNMatrix4MakeRotation(Float, Float, Float, Float)`

Returns a matrix describing a rotation transformation.

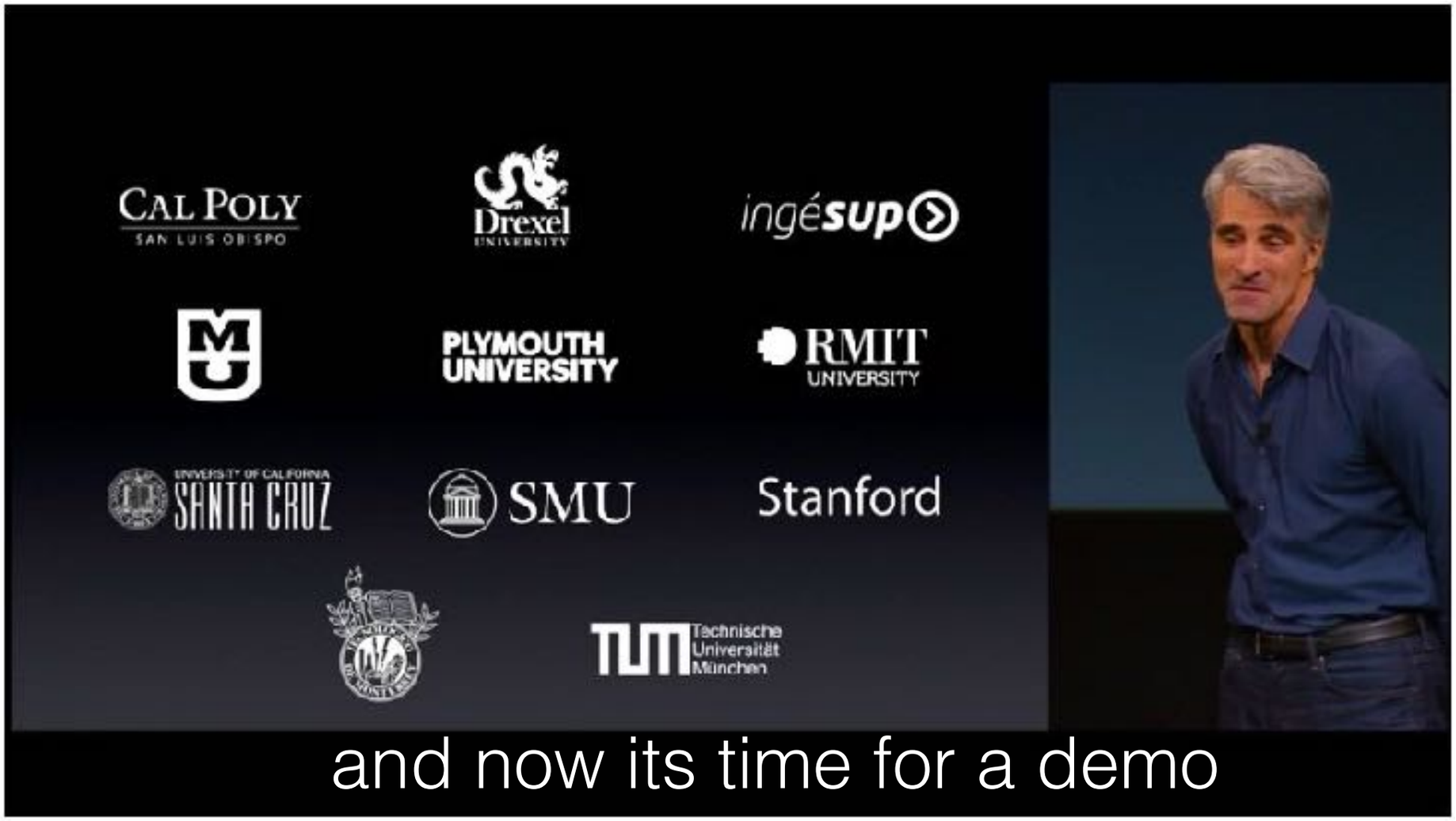
`func SCNMatrix4MakeScale(Float, Float, Float)`

Returns a matrix describing a scale transformation.

ARKit and SceneKit

- extended demo

branches:
(1) bare implementation,
(2) image extension,
(3) stylization (with SCNActions)



object detection in AR

- need to perform detection of images and 3D objects
  - to anchor AR in the world
  - for shared AR experiences
- keep in mind
  - automatic object detection is a computational process
  - …but is very easy to get started

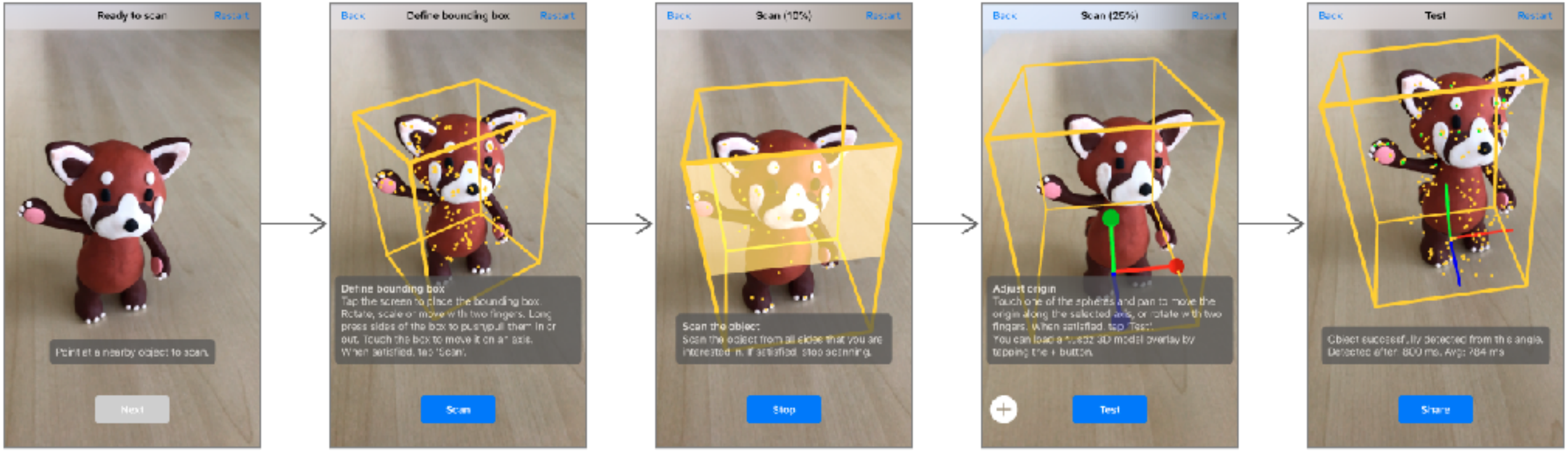
image recognition in ARKit

- detection based on scans of keypoints in example image, projected onto detected planes
- drag and drop image into `Assets` file as an “ARImage”

https://developer.apple.com/videos/play/wwdc2018/610/

object recognition in ARKit

- detection based on scans of an object
- scanning app available here:
  - https://developer.apple.com/documentation/arkit/scanning_and_detecting_3d_objects



object recognition in ARKit

- drag **arobject** into assets repository
- tell ARKit to begin looking for object

```
override func viewWillAppear(_ animated: Bool) {
        super.viewWillAppear(animated)

        // Create a session configuration
        let configuration = ARWorldTrackingConfiguration()

        // here is where we setup detection of 3D point clouds, drag into AR assests
        guard let referenceObjects = ARReferenceObject.referenceObjects(inGroupNamed:
        "gallery", bundle: nil) else {
            fatalError("Missing expected asset catalog resources.")
        }
        configuration.detectionObjects = referenceObjects

        // Run the view's session
        sceneView.session.run(configuration)
    }
```

load AR models

set models of objects to look for

object recognition in ARKit

- use delegation to respond when object is found
- add new nodes anchored to the ARAnchor

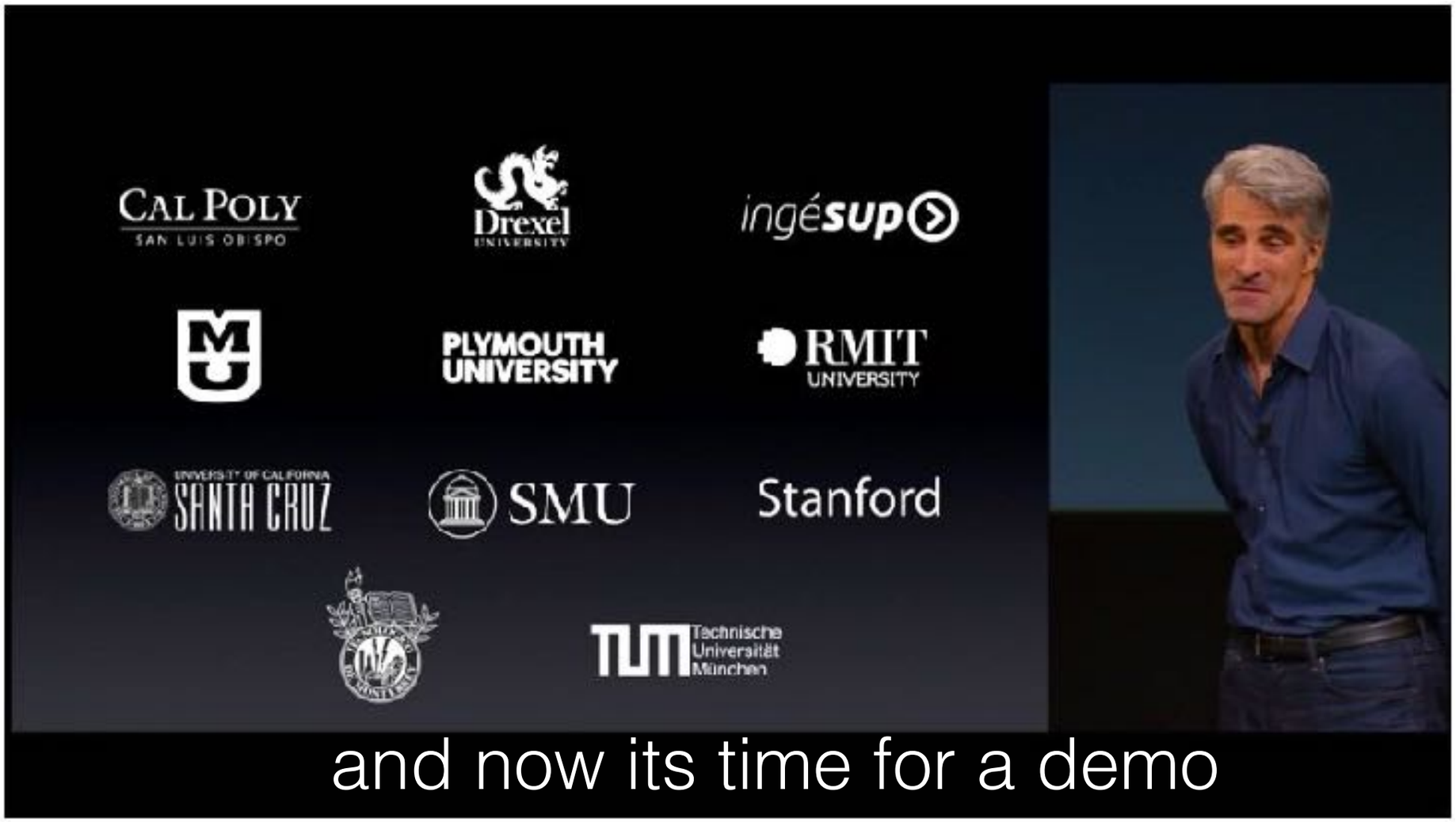
```
func renderer(_ renderer: SCNSceneRenderer,
        didAdd node: SCNNode,
        for anchor: ARAnchor) {

        if let objectAnchor = anchor as? ARObjectAnchor {

            newNode = … what you want it to be …
            print(objectAnchor.name! + " found”)
            node.addChildNode(newNode)
        }
}
```

delegate function called when new node is added to ARSCNScene

true if recognized as ARObject from gallery

ARKit with ObjectRecognition

- find the object!
- place images around the object!



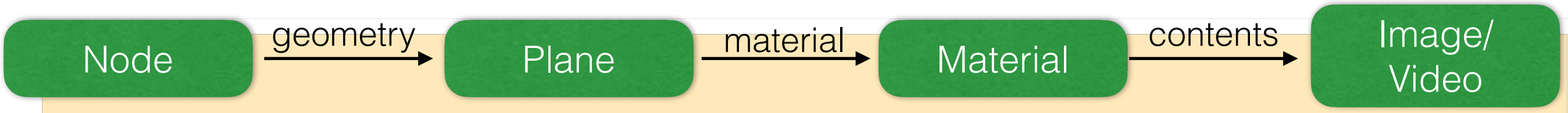
additional UI elements

if time

- planes
- simple nodes
- actions
- UI Text

Planes as Content Holders

- planes are placeholders for flat content like images and videos
- planes are the geometry for a node in SceneKit




```
// add in a video near the detected object
let planarNode = SCNNode()
let contentPlane = SCNPlane(width: CGFloat(0.1), height: CGFloat(0.05))
let avMaterial = SCNMaterial()

avMaterial.diffuse.contents = …image or video…
contentPlane.materials = [avMaterial]

planarNode.geometry = contentPlane
node.addChildNode(planarNode)
```

setup image or video

set material of the plane

plane is geometry of node

Simple Nodes in SceneKit

- define geometry with pre-made shape, like box
- make new node with given geometry


```
let boxNode:SCNNode? = SCNNode()

let box = SCNBox(width: CGFloat(0.1), height: CGFloat(0.1),
                 length: CGFloat(0.1), chamferRadius: 0.01)
box.firstMaterial?.diffuse.contents = UIColor(white: 1.0, alpha: 0.8)
box.firstMaterial?.isDoubleSided = true

boxNode!.geometry = box // make this node a box!

node.addChildNode(box!)


let alphaAction = SCNAction.fadeOpacity(to: 0.1, duration: 5)
box?.runAction(alphaAction)
```

setup initial color

add animation, change alpha!

Actions in SceneKit

- any movement scaling, or change of added nodes should be done via actions
- actions can be reused on different nodes!

```
// add some animation
let prevScale = uiObject.scale
uiObject.scale = SCNVector3(CGFloat(prevScale.x)/3, CGFloat(prevScale.x)/3,
                            CGFloat(prevScale.x)/3)

let scaleAction = SCNAction.scale(to: CGFloat(prevScale.x), duration: 2)
scaleAction.timingMode = .easeIn

node.addChildNode(uiObject)

uiObject.runAction(scaleAction, forKey: "scaleAction")
```

not yet on screen: will start smaller, then scale back to original size

make visible

define scaling action and params

run action immediately

UI Text Elements in SceneKit

Node → geometry → Text → material → Material → contents → Color

```
func createTextNode(textString: String)->SCNNode?{

    let textNode:SCNNode? = SCNNode()
    textNode!.geometry = setupTextParameters(textString: textString)
    textNode!.scale = SCNVector3Make(0.001, 0.001, 0.001)
    textNode!.position = SCNVector3Make(-0.1, 0.1, 0.0)
    textNode?.castsShadow = true

    return textNode
}

func setupTextParameters(textString: String)->SCNText{
    let text = SCNText(string: textString, extrusionDepth: 1)

    let material = SCNMaterial()
    material.diffuse.contents = UIColor.white

    text.flatness = 0
    text.isWrapped = true
    text.materials = [material]
    return text
}
```

lower left corner

if plane detection is enabled

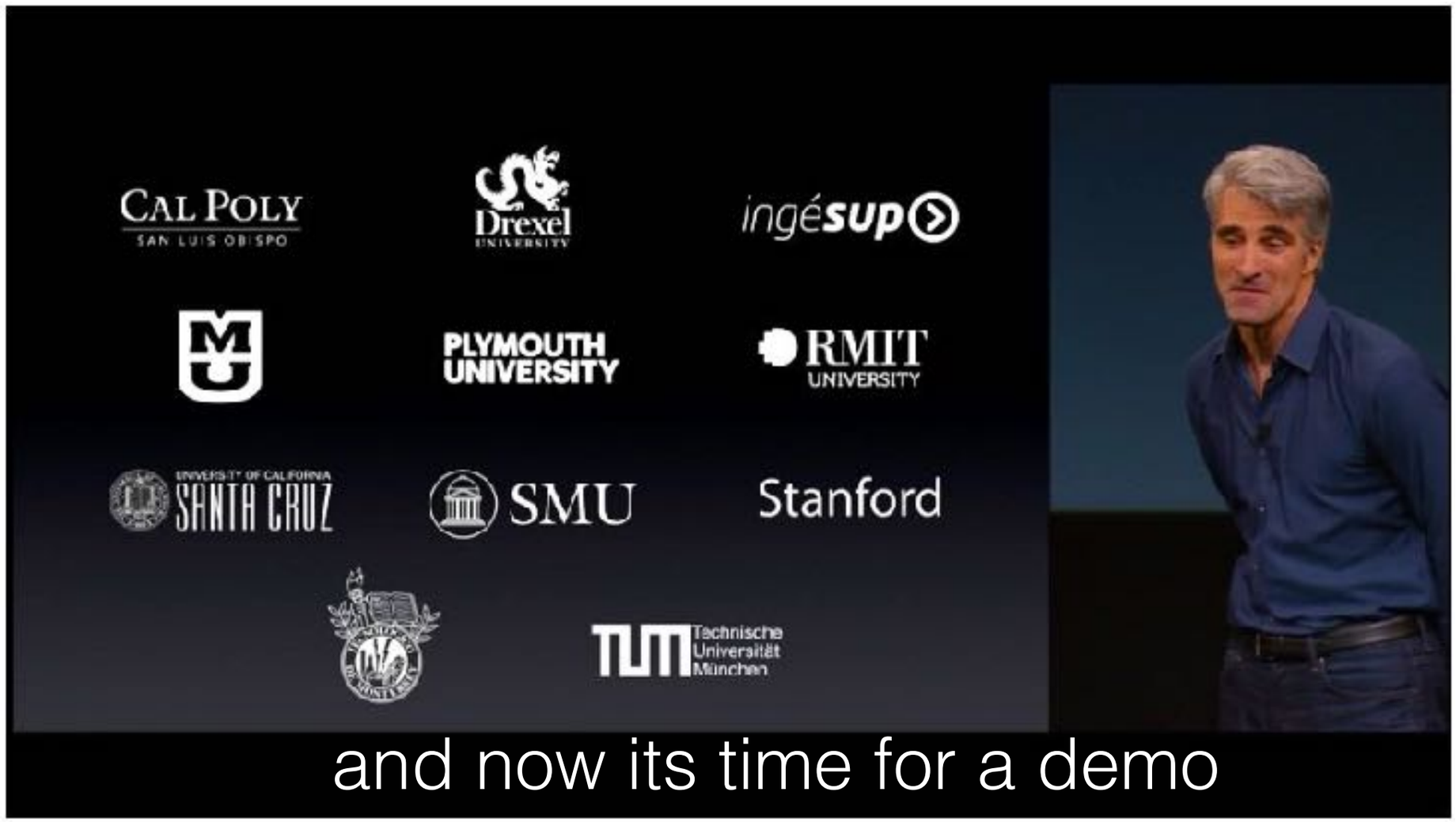
extrusion defines depth of the text (flat versus extended)

color and other properties are set in the material of the node

https://developer.apple.com/documentation/scenekit/scntext

ARKit with ObjectRecognition

- add to demo some more interesting UI elements



for next time…

- Speech Recognition
- Pitching (Required)
- ~Fin~

Back Up Slides

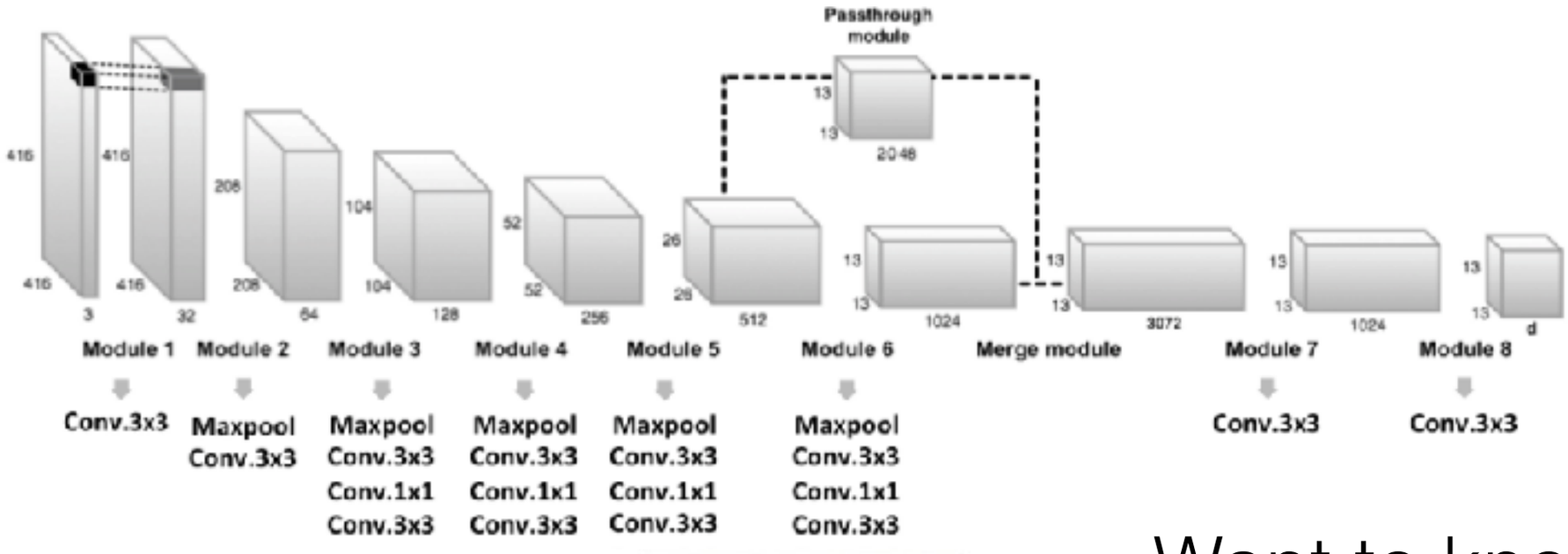
Detecting Objects in Images

- Using Turi Create
  - SFrames
  - Simple Machine Learning
- Object Detection with YOLO
  - Creation
  - Exporting to CoreML
  - Incorporating into ARKit

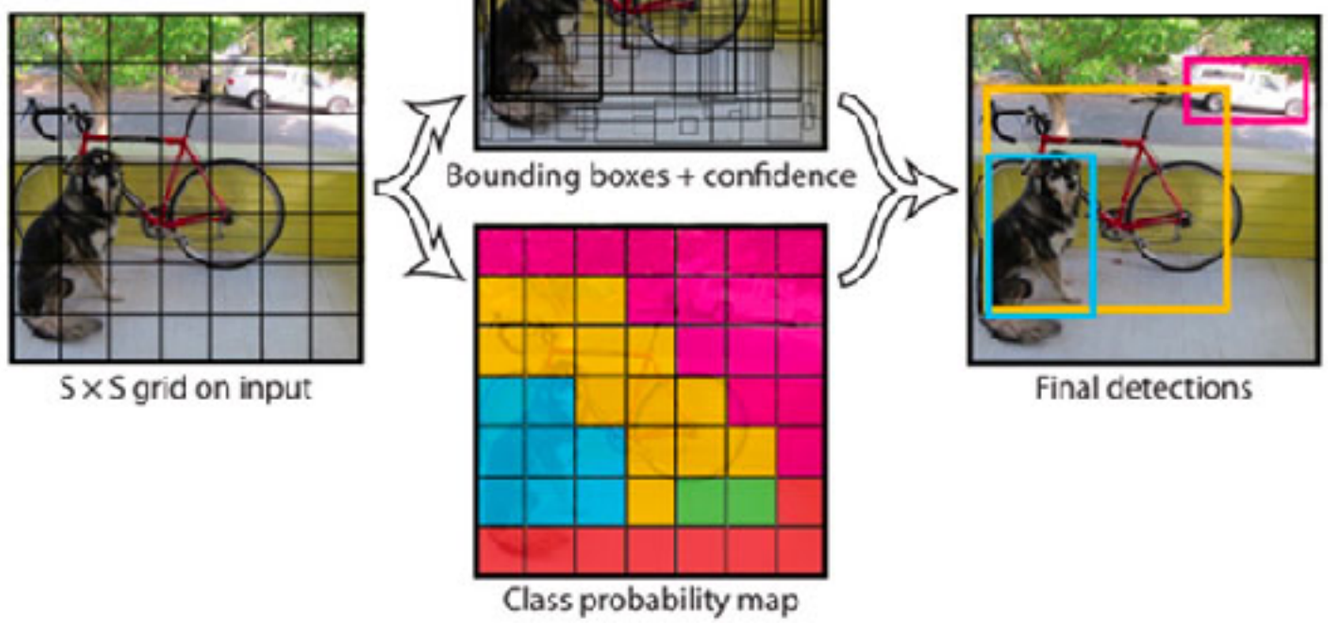
didn’t we already detect objects?

- with ARKit, yes
- but what if we want to detect all sorts of objects
- and we need to do it very fast (multiple times per second)
- and we only have examples of the types of objects we want, not a scan of the exact object
- enter… Turi Create

YOLO



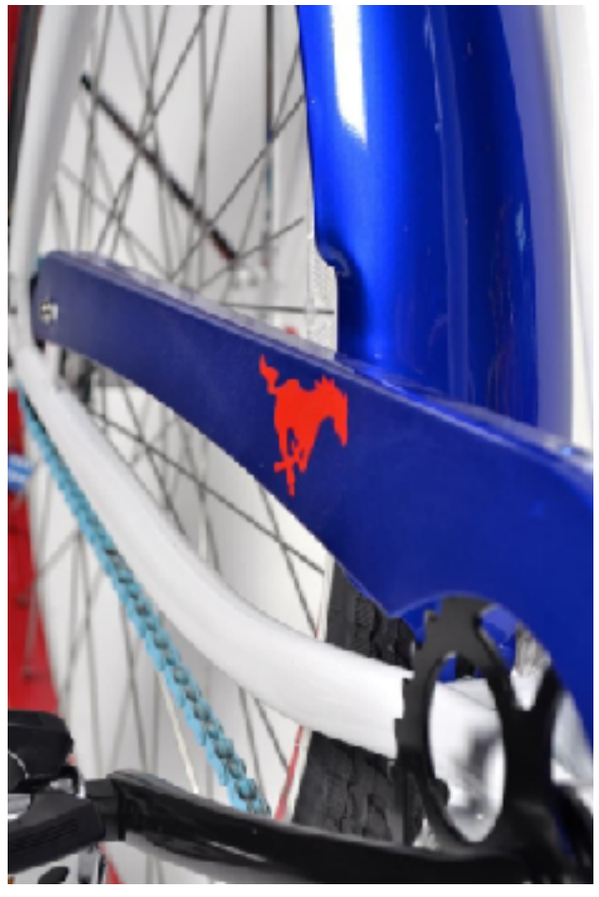
Want to know more?
Take CS8321…



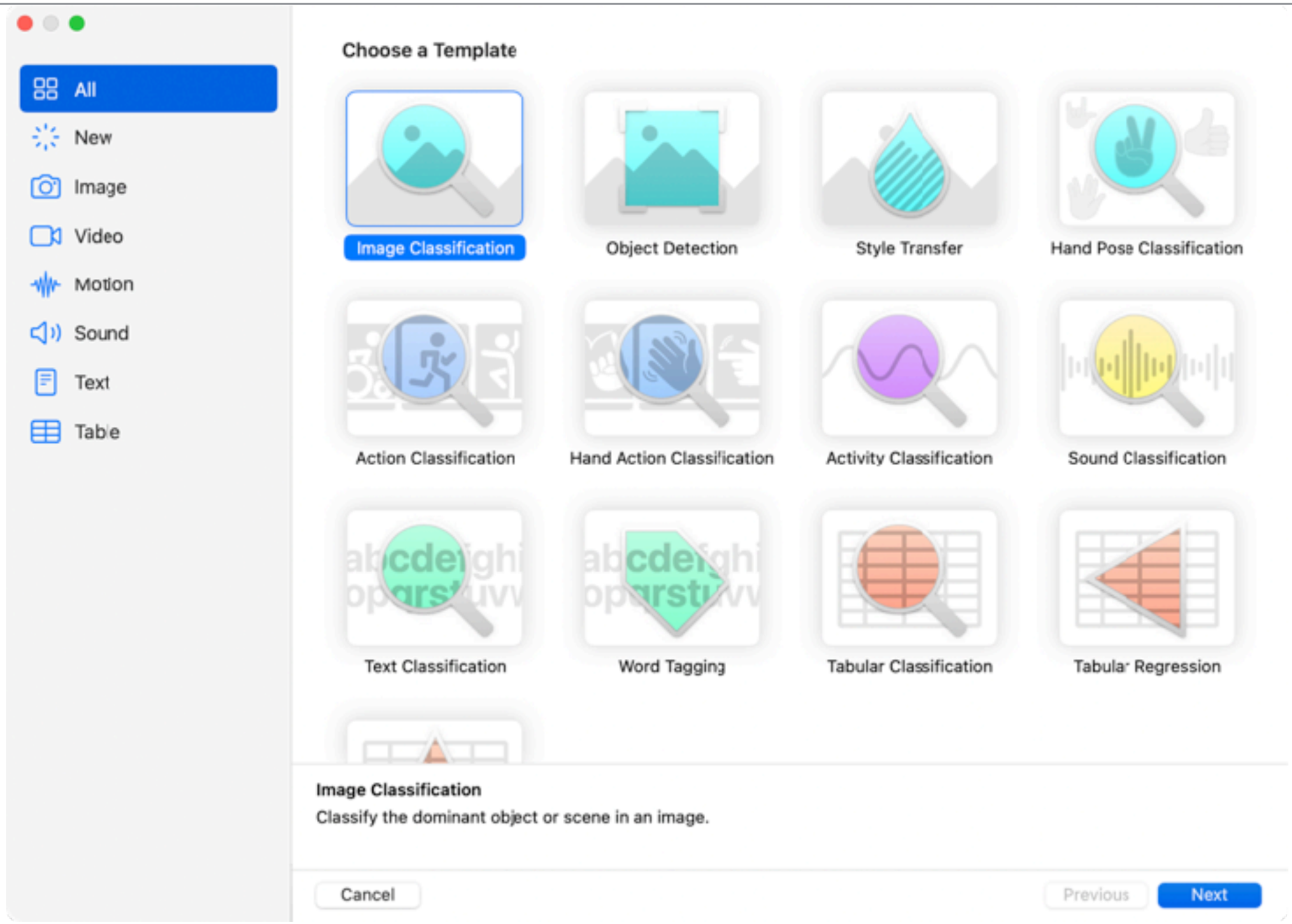
https://datascience.stackexchange.com/questions/42509/yolo-algorithm-understanding-training-data

create an object detector with Turi

- Turi Create, high level overview
  - SFrames and Object Detectors

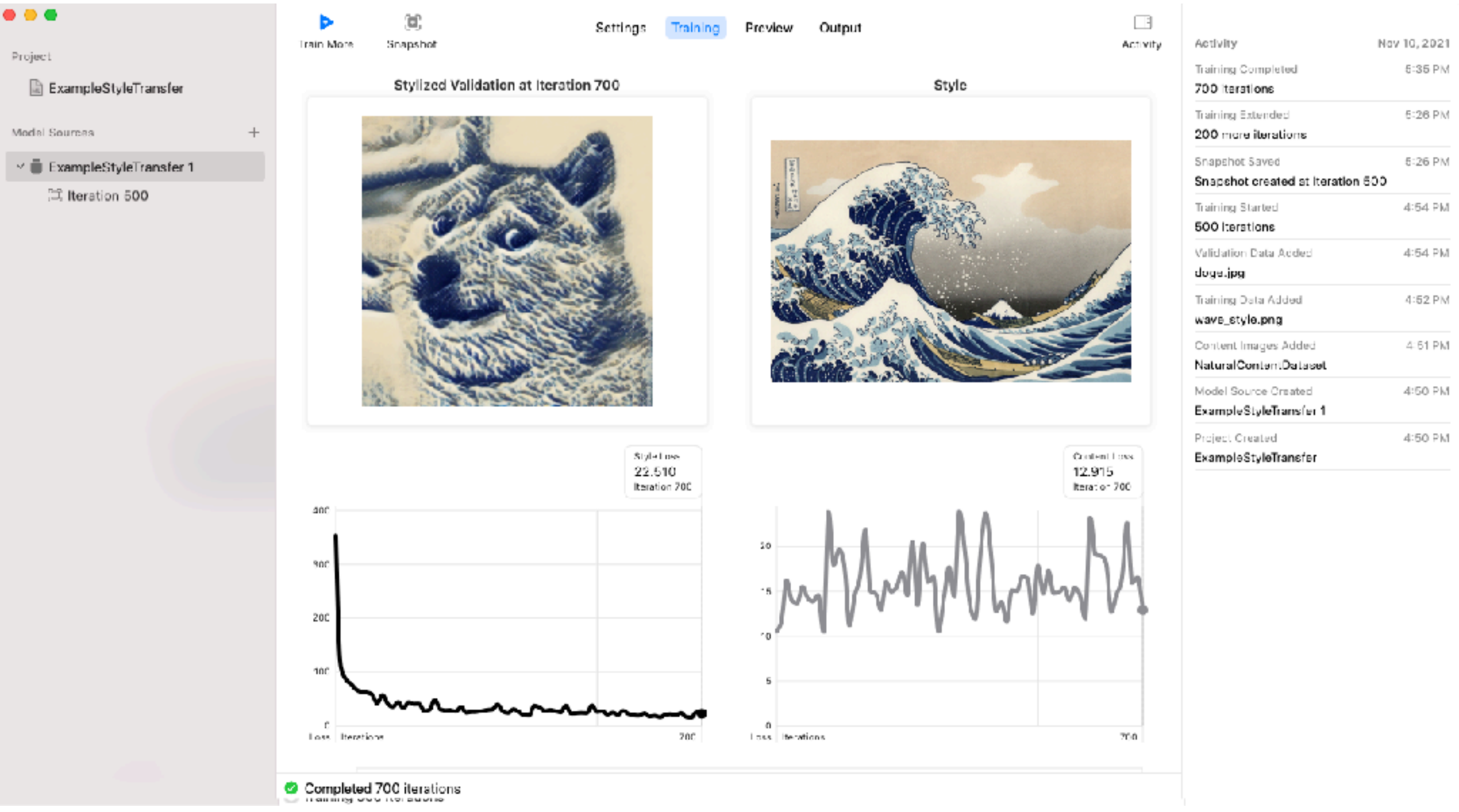


Bonus: Create ML (iOS 15.0+)



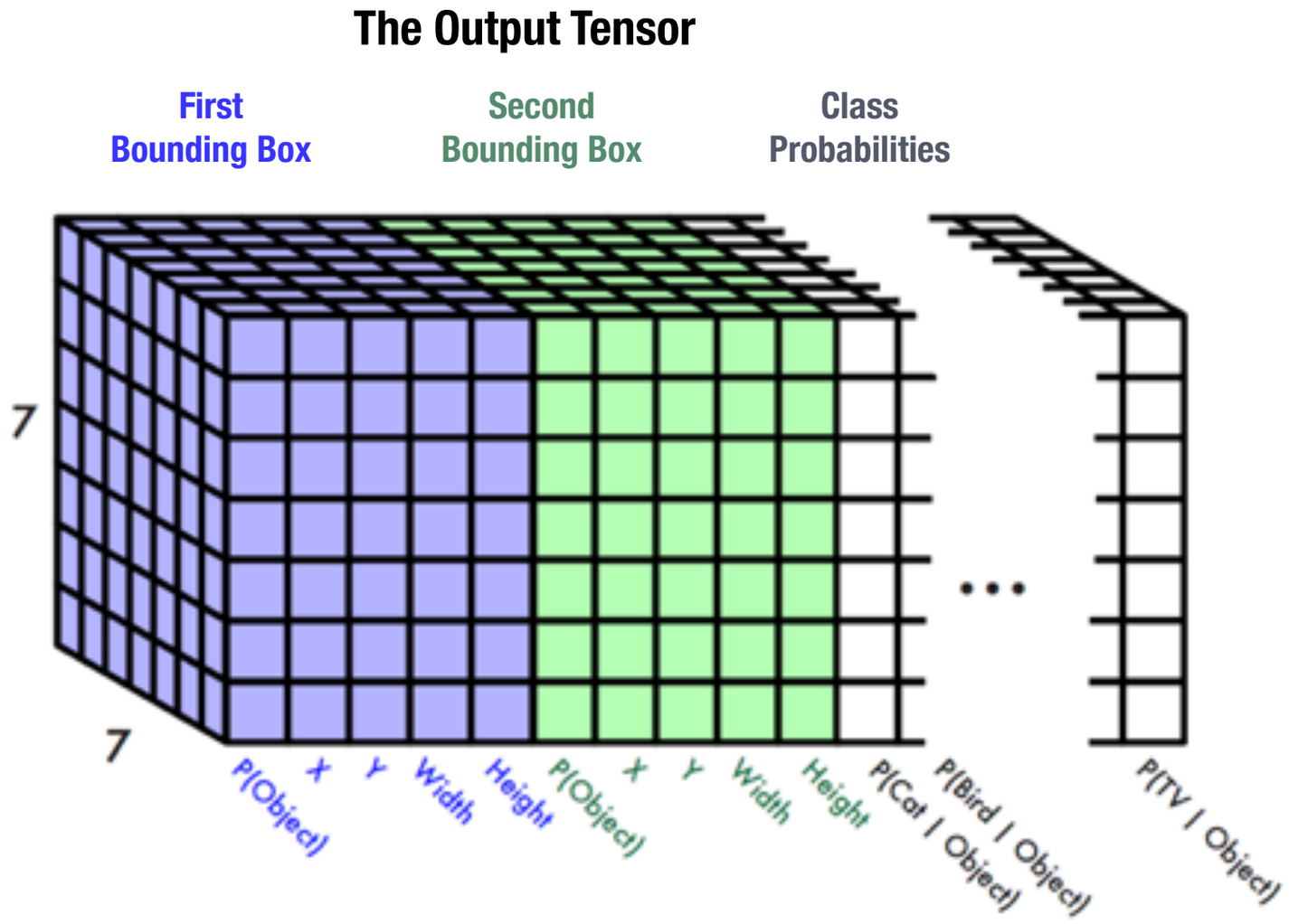
https://developer.apple.com/documentation/createml/

Create ML Stylizer



YOLO output tensor

- is there an object?
- where is the object?
- how large is the Object?
- what is the object?
- if competing, which
- object is most likely?



using the vision API

- Overview:
  - streaming video will require constant images from the AR session, **use delegation**
  - inside delegate, need to **grab** ARFrame, **convert** to pixel buffer, and **process** on a **background** thread
  - **handle vision** results in a new function
  - **update** the UI on the **main thread**

setting up vision

```
let model = PersonBike()
private var requests = [VNRequest]()

@discardableResult
func setupVision(useCPUOnly:Bool) {

  //MARK: One, Setup Vision
  let visionModel = try VNCoreMLModel(for: model.model)

  // use this request to setup the object recognition with Vision
  let objectRecognition = VNCoreMLRequest(model: visionModel,
                                          completionHandler:
                                              self.handleObjectRecognitionResult)

      objectRecognition.imageCropAndScaleOption = .scaleFill
      objectRecognition.usesCPUOnly = [True / False]
      self.requests = [objectRecognition]

  }
```

load model from Turi

setup vision wrapper

save this request for later

using vision with AR

ARDelegate function, called at 60 FPS

```
func renderer(_ renderer: SCNSceneRenderer, updateAtTime time: TimeInterval) {
        //MARK: Two, Get Image Frame from AR
       guard let frame = sceneView.session.currentFrame else { return }

       guard self.currentBuffer == nil, else { return }
      self.currentBuffer = frame.capturedImage // the pixels to process

       // run in the background so that AR doesn't suffer performance
       DispatchQueue.global(qos: .background).async {

           // setup input image for the request
           let imageRequestHandler = VNImageRequestHandler(
                                     cvPixelBuffer: self.currentBuffer,
                                       orientation: ORIENTATION,
                                           options: [:])

           imageRequestHandler.perform(self.requests)

       }
   }
```

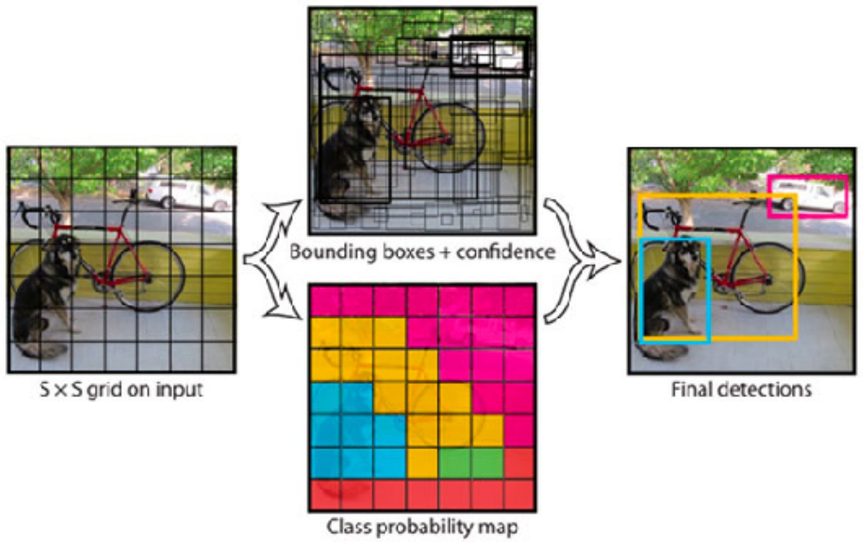
limit FPS of detection

setup image buffer

tell request what pixels to process

start processing request that we setup previously

handling output request



YOLO Output:

- list of bounding boxes for every square in grid
- list of all possible classes (bike, person)

can have multiple objects in the scene

```
func handleObjectRecognitionResult(_ request:VNRequest, error:Error?){
        // perform all the UI updates on the main queue
        if let results = request.results { // if we have valid results, else its nil
            DispatchQueue.main.async(execute: {

                self.drawVisionRequestResults(results)
                self.updateOverlay()

                // set as nil so we can process next ARFrame Image
                self.currentBuffer = nil
            })
        }
    }
```

go over, if time!

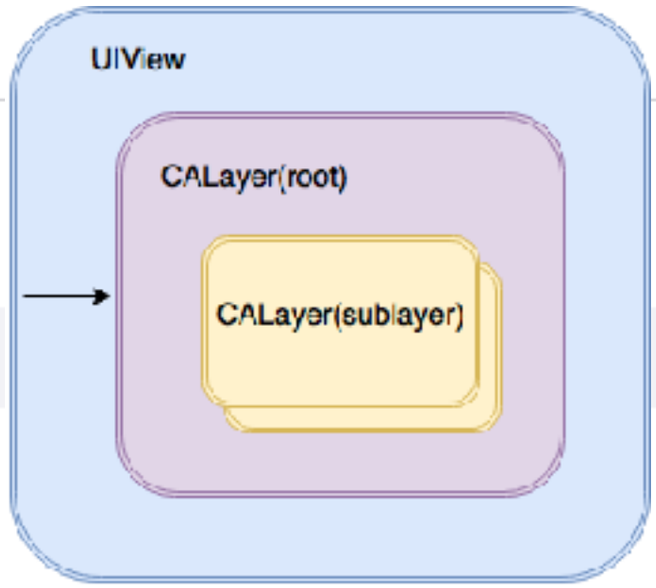
process the next captured frame now

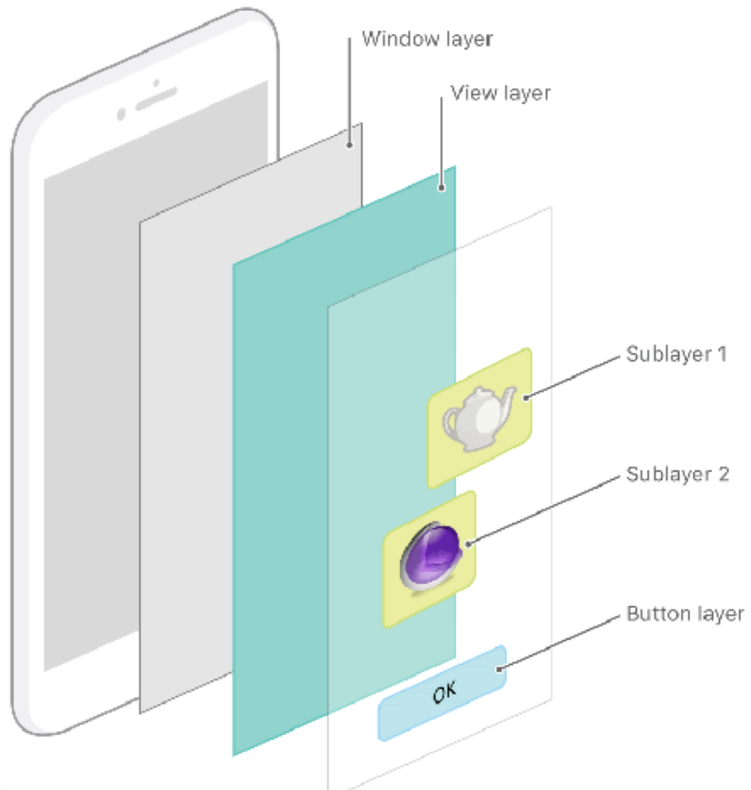
object detection

- YOLO demo
- If time, go over creating overlays with CATransactions

CATransactions

A mechanism for grouping multiple layer-tree operations into atomic updates to the render tree.





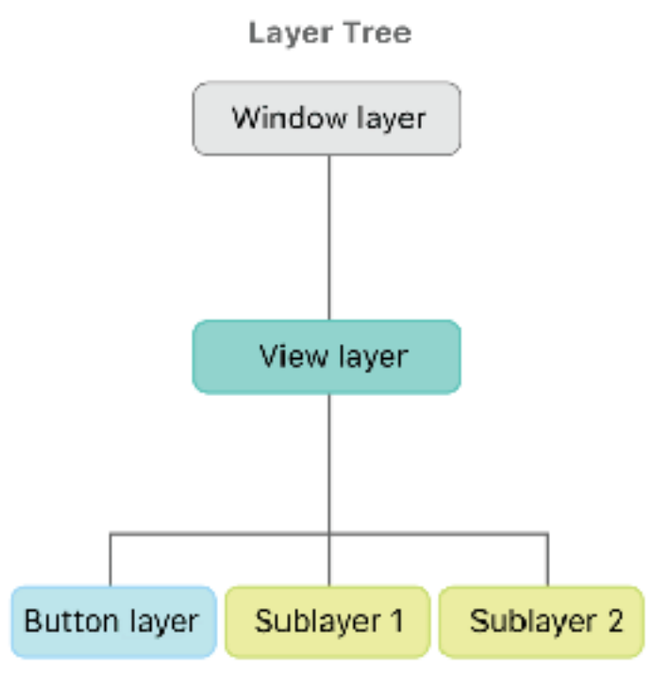
Declaration

```
class CATransaction : NSObject
```

Overview

`CATransaction` is the Core Animation mechanism for batching multiple layer-tree operations into atomic updates to the render tree. Every modification to a layer tree must be part of a transaction. Nested transactions are supported.

Core Animation supports two types of transactions: *implicit* transactions and *explicit* transactions. Implicit transactions are created automatically when the layer tree is modified by a thread without an active transaction and are committed automatically when the thread's runloop next iterates. Explicit transactions occur when the the application sends the `CATransaction` class a `begin()` message before modifying the layer tree, and a `commit()` message afterwards.

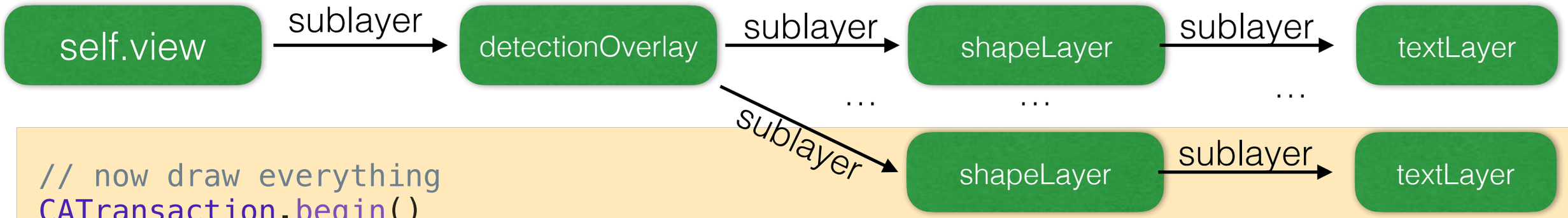


Overlays from YOLO

```
detectionOverlay = CALayer() // container layer that has all the renderings
detectionOverlay.name = "DetectionOverlay"

// set the initial bounds, will transform when we know more about the image
detectionOverlay.bounds = CGRect(x: 0.0, y: 0.0,
                                 width: self.view.bounds.width,
                                height: self.view.bounds.height )

self.sceneView.layer.addSublayer(detectionOverlay)
```

```
func updateOverlay() {

   let bounds = self.view.bounds

   // … Some magic transforms to the view  …
   // this tries to get the best mapping we can from the cropping that
   // Core Vision used, but It may not be 100% perfect
   // Scales, and adds magic numbers to the X and Y positions

   detectionOverlay.setNeedsDisplay() // sets display for all subviews
}
```

Overlays from YOLO




```
// now draw everything
CATransaction.begin()
detectionOverlay.sublayers = nil // remove all the old recognized objects

// for each result, create an overlay layer to display on it
for observation in results where observation is VNRecognizedObjectObservation {

   // Select the label with the highest confidence and get bounds
   let topLabelObservation = objectObservation.labels[0]
   let objectBounds = VNImageRectForNormalizedRect(objectObservation.boundingBox,
                        Int(captureImageSize.width),Int(captureImageSize.height))

   // get UI box for object
   let shapeLayer = self.createRoundedRectLayerWithBounds(objectBounds, {…})

   // show the label and confidence in the UI Box
   let textLayer = self.createTextSubLayerInBounds(objectBounds,  {…})

   shapeLayer.addSublayer(textLayer)// add text to box
   detectionOverlay.addSublayer(shapeLayer) // add box to the UI
}
CATransaction.commit() // now commit everything so that it displays
```