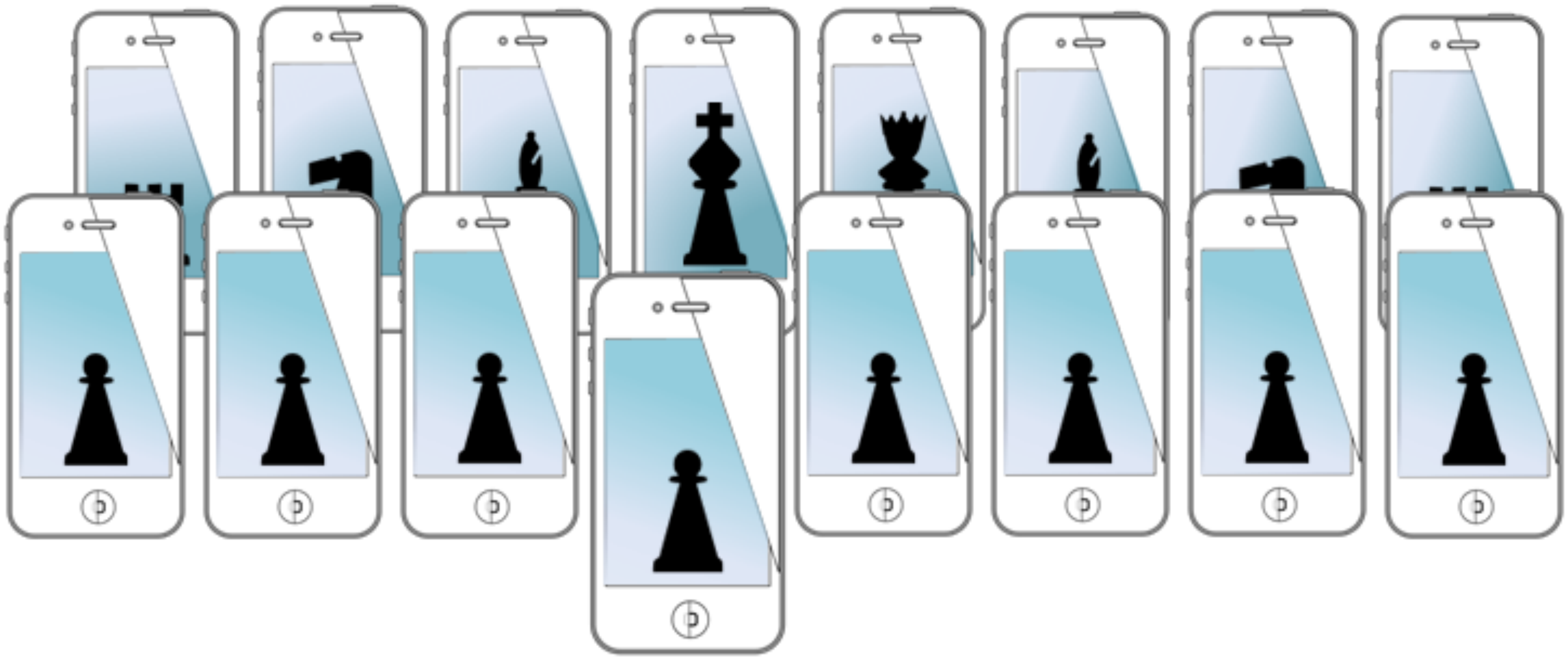
MOBILE SENSING LEARNING



CS5323 & 7323

Mobile Sensing and Learning

CoreML and ARKit

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

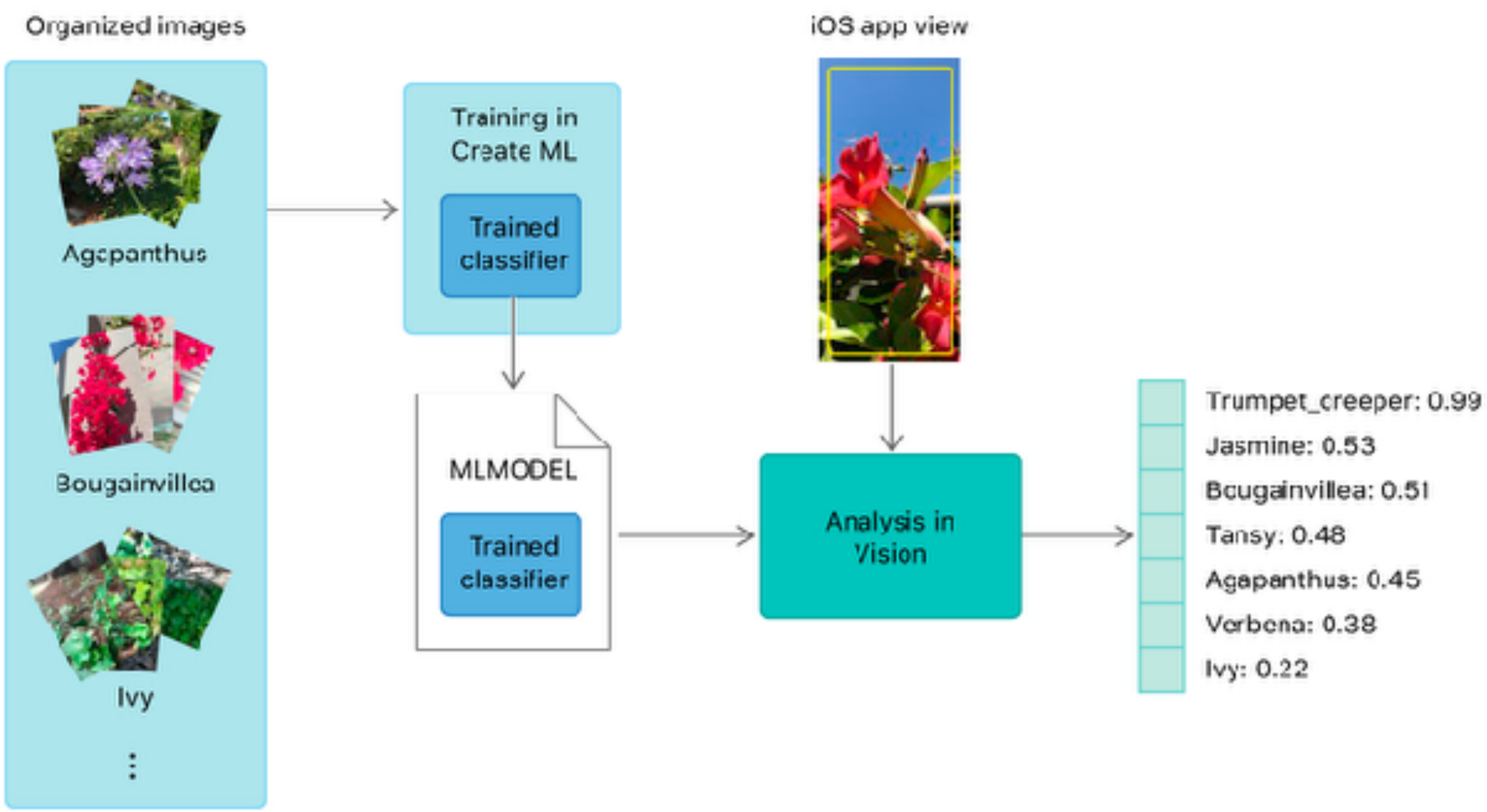
course logistics and agenda

- agenda:
  - vision API and CoreML demo
  - custom Trained CoreML demo
  - sceneKit review
  - ARKit
  - ARKit demo

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

Review: CoreML with vision API

- load ML model in Xcode
- wrap model
- create vision request
- wait for result in completion handler



Review: the vision API

```
// generate request for vision and ML model
let request = VNCoreMLRequest(model: self.model,
                completionHandler: resultsMethod)

// add data to vision request handler
let handler = VNImageRequestHandler(cgImage: cgImage!, options: [:])

// now perform classification
do{
     try handler.perform([request])
}catch _{
     …
}

func resultsMethod(request: VNRequest, error: Error?) {
        guard let results = request.results as? [VNClassificationObservation]
            else { fatalError() }

        for result in results {
            if(result.confidence > 0.05){
                print(result.identifier,result.confidence)
            }
        }
    }
```

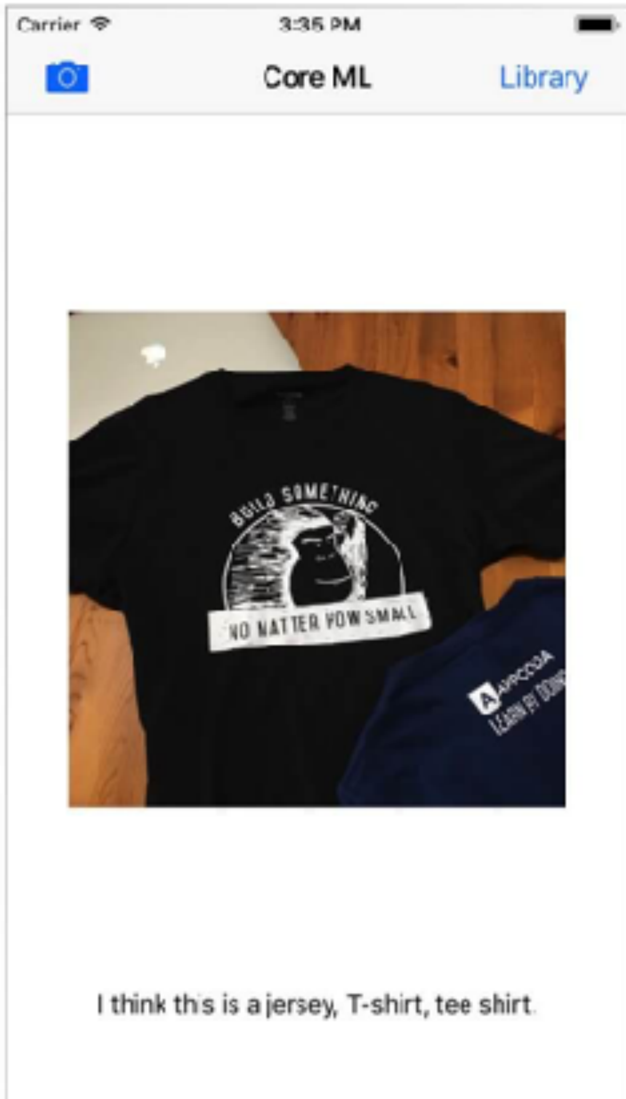
setup vision request with completion handler and ML model

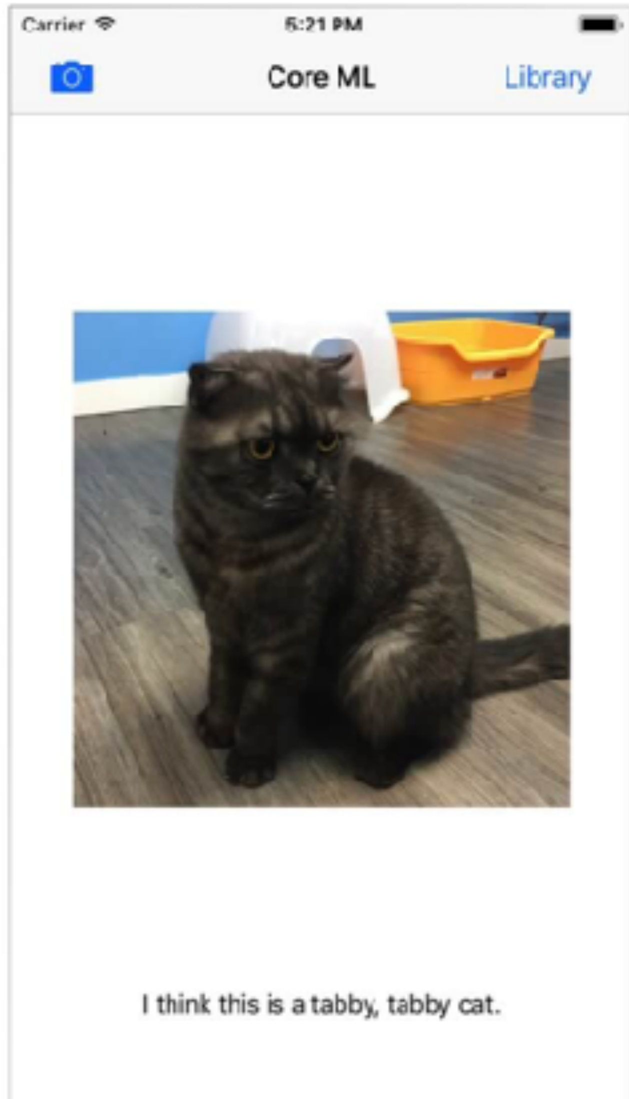
add data to request(s)

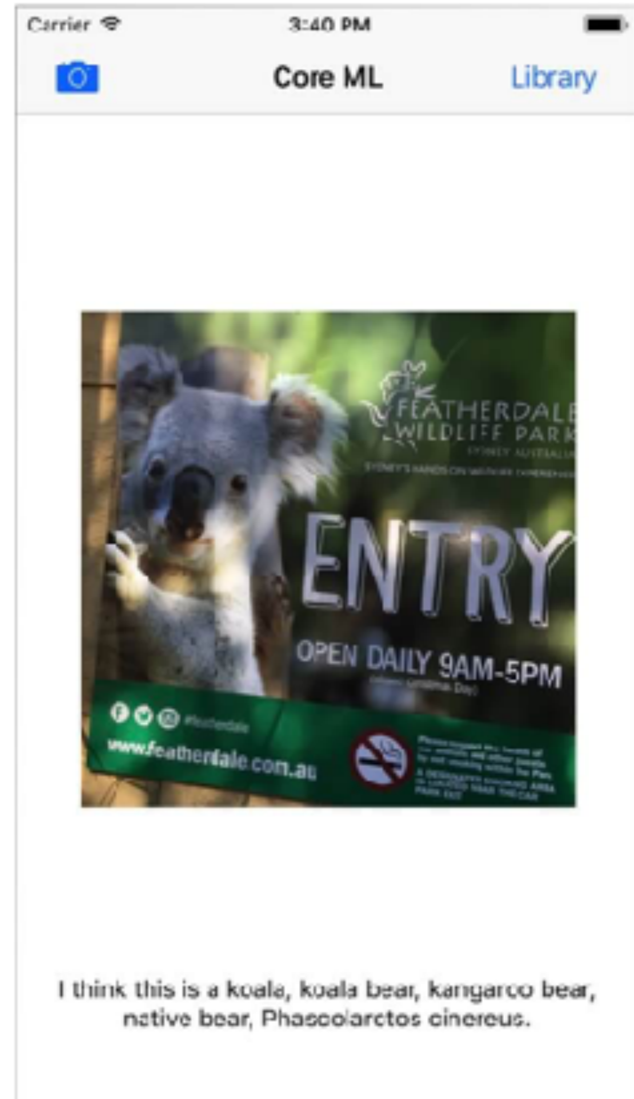
perform request

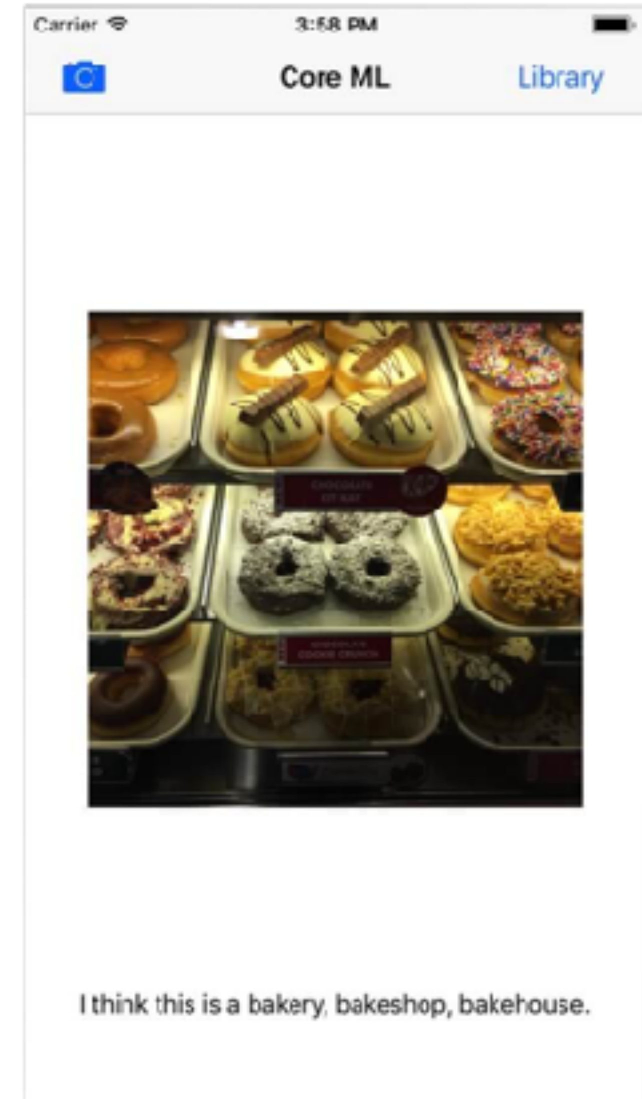
interpret request results

CoreML with Vision



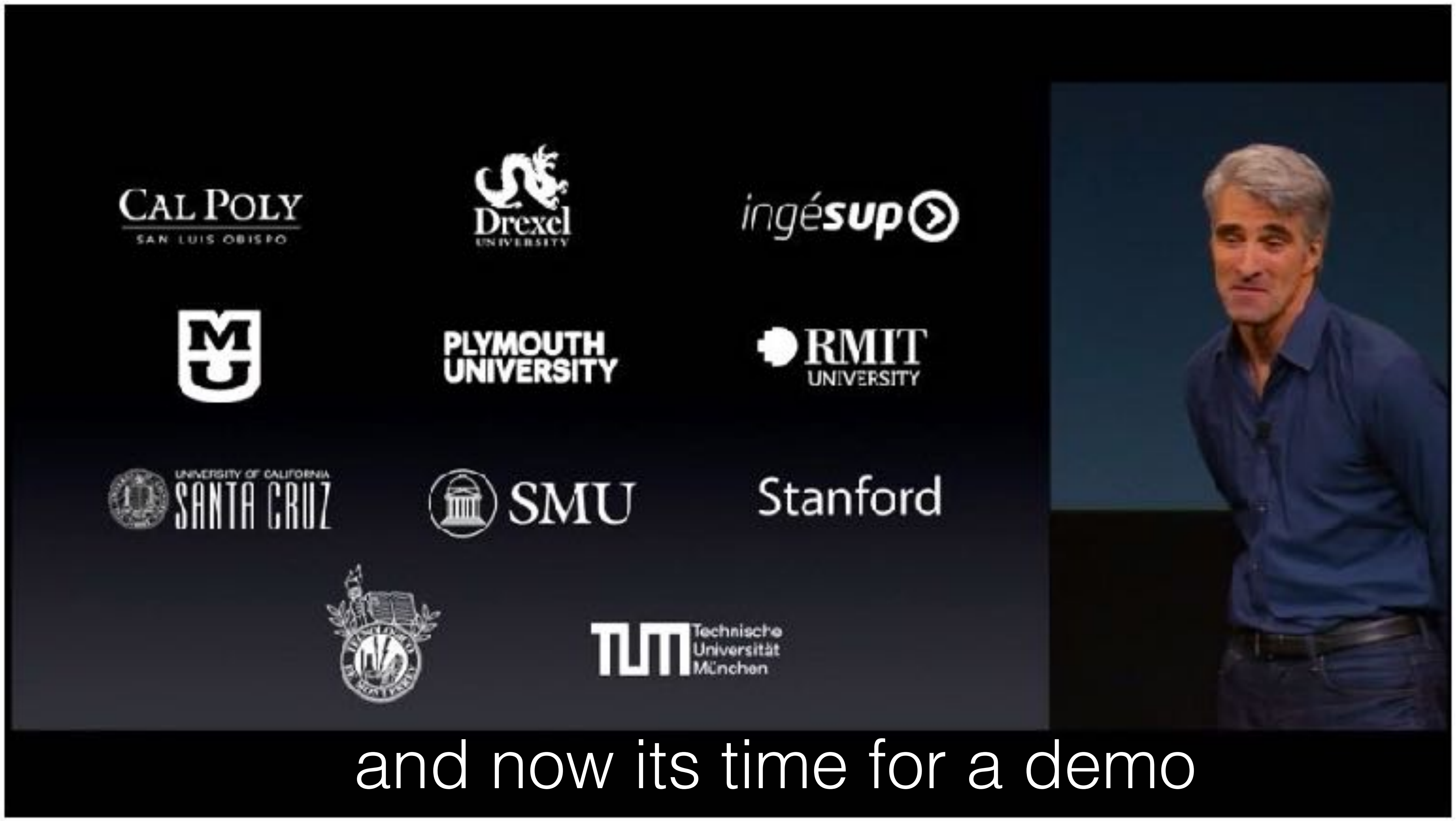




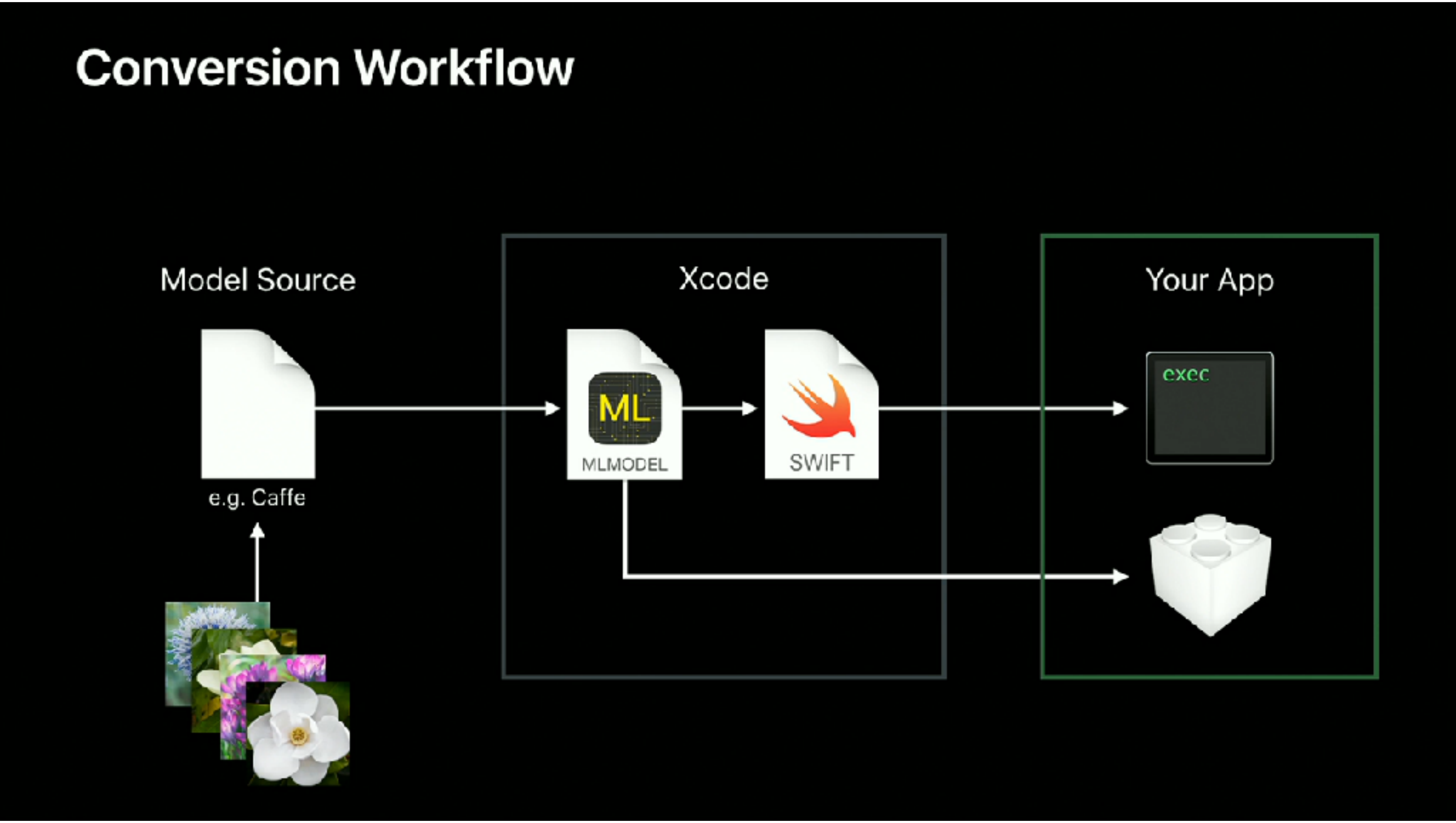


CoreML, a taste

- demo using vision API

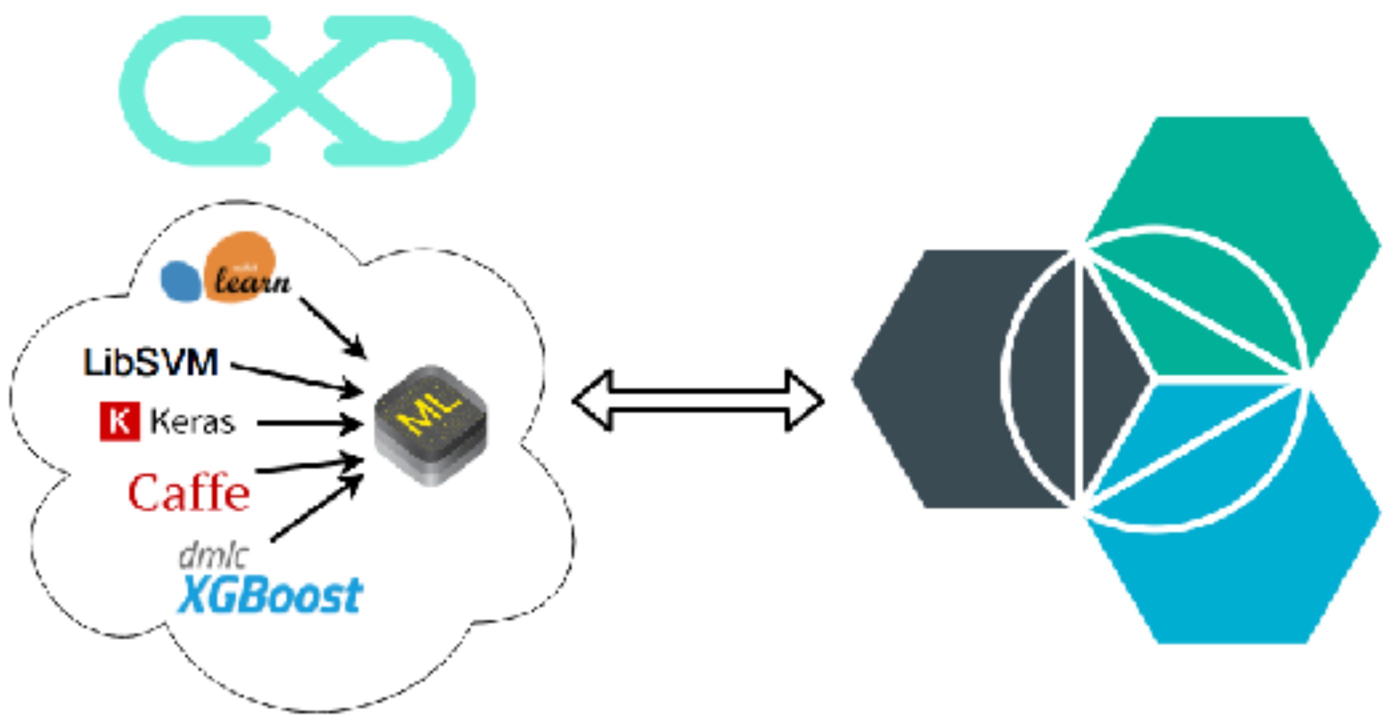


CoreML review



CoreML review

getting trained models:



Working with Models

Build your apps with the ready-to-use Core ML models below, or use Core ML Tools to easily convert models into the Core ML format.

Models

MobileNet

MobileNets are based on a streamlined architecture that have depth-wise separable convolutions to build lightweight, deep neural networks.

Detects the dominant objects present in an image from a set of 1000 categories such as trees, animals, food, vehicles, people, and more.

View original model details ›

Download Core ML Model

File size: 17.1 MB

https://developer.apple.com/machine-learning/

but… we want more than **pre-trained** models

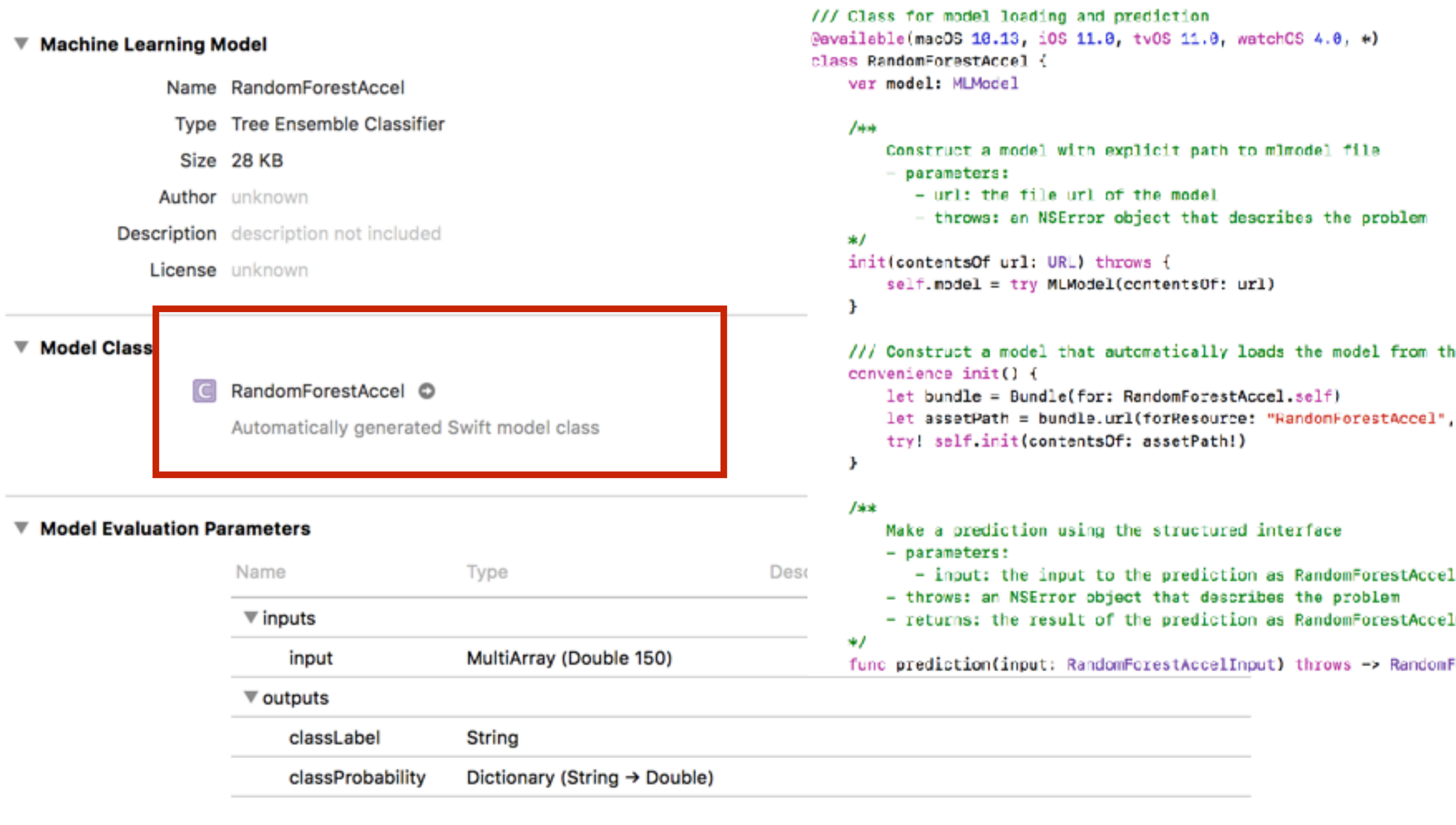
installing CoreMLTools

- built into TuriCreate
  `model.export_coreml("MyModel.mlmodel")`
- also available for other libraries: so create a conda environment and install coremltools
  - `conda install sklearn numpy (+others) …`
  - `pip install coremltools`

```
clf = RandomForestClassifier(n_estimators=50)
print("Training Model", clf)

clf.fit(X,y)

print("Exporting to CoreML")

coreml_model = coremltools.converters.sklearn.convert(
    clf,
    ["accelX"]*50+["accelY"]*50+["accelZ"]*50, # feature names (optional)
    "Direction") # label name (optional)

# save out as a file
coreml_model.save('rf.mlmodel')
```

using CoreML

- drag into project



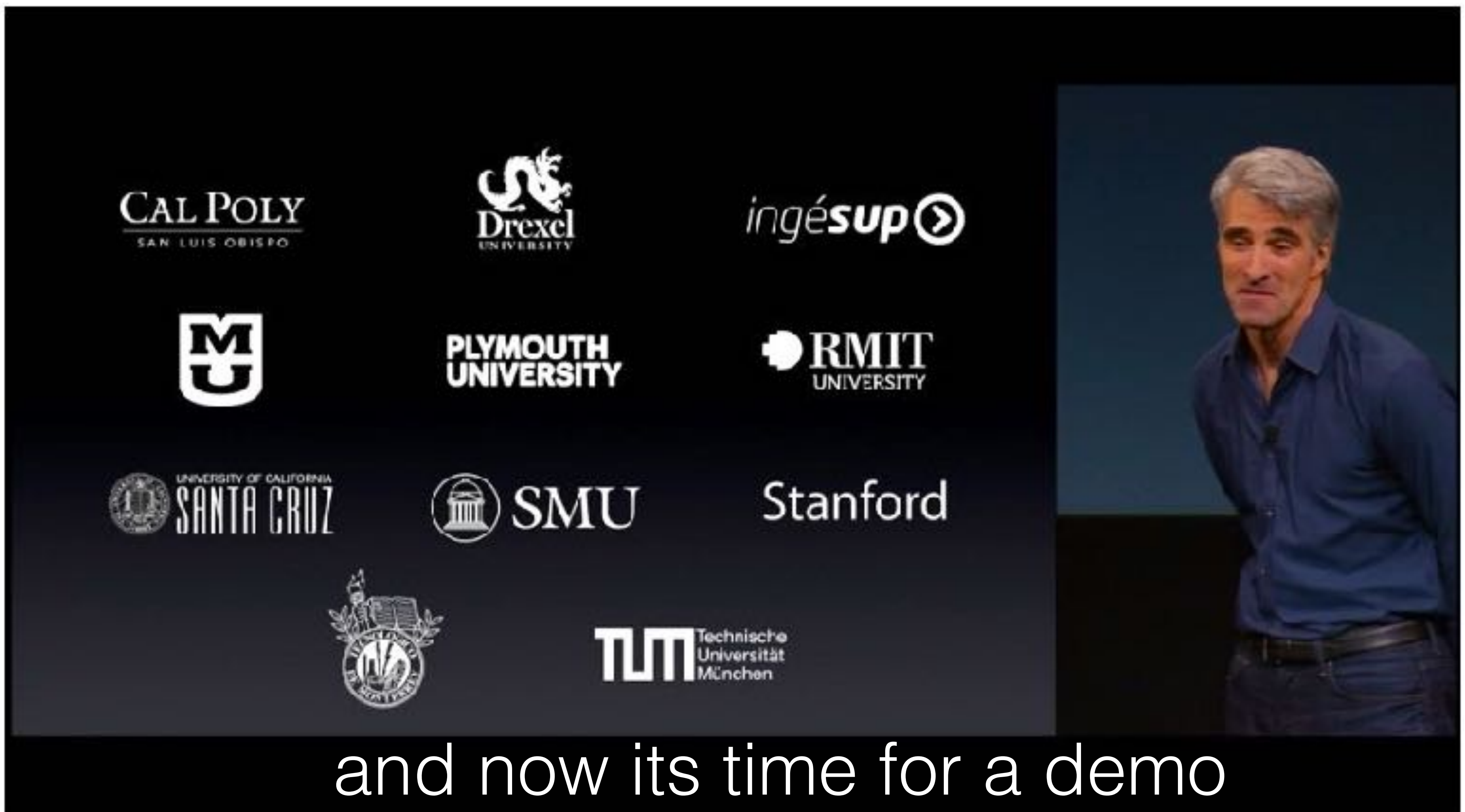
using CoreML

- drag into project
- use like previously in HTTP model

```
var model = RandomForestAccel()


//predict a label
let seq = toMLMultiArray(self.ringBuffer.getDataAsVector())
guard let output = try? model.prediction(input: seq) else {
    fatalError("Unexpected runtime error.")
}

displayLabelResponse(output.classLabel)
```

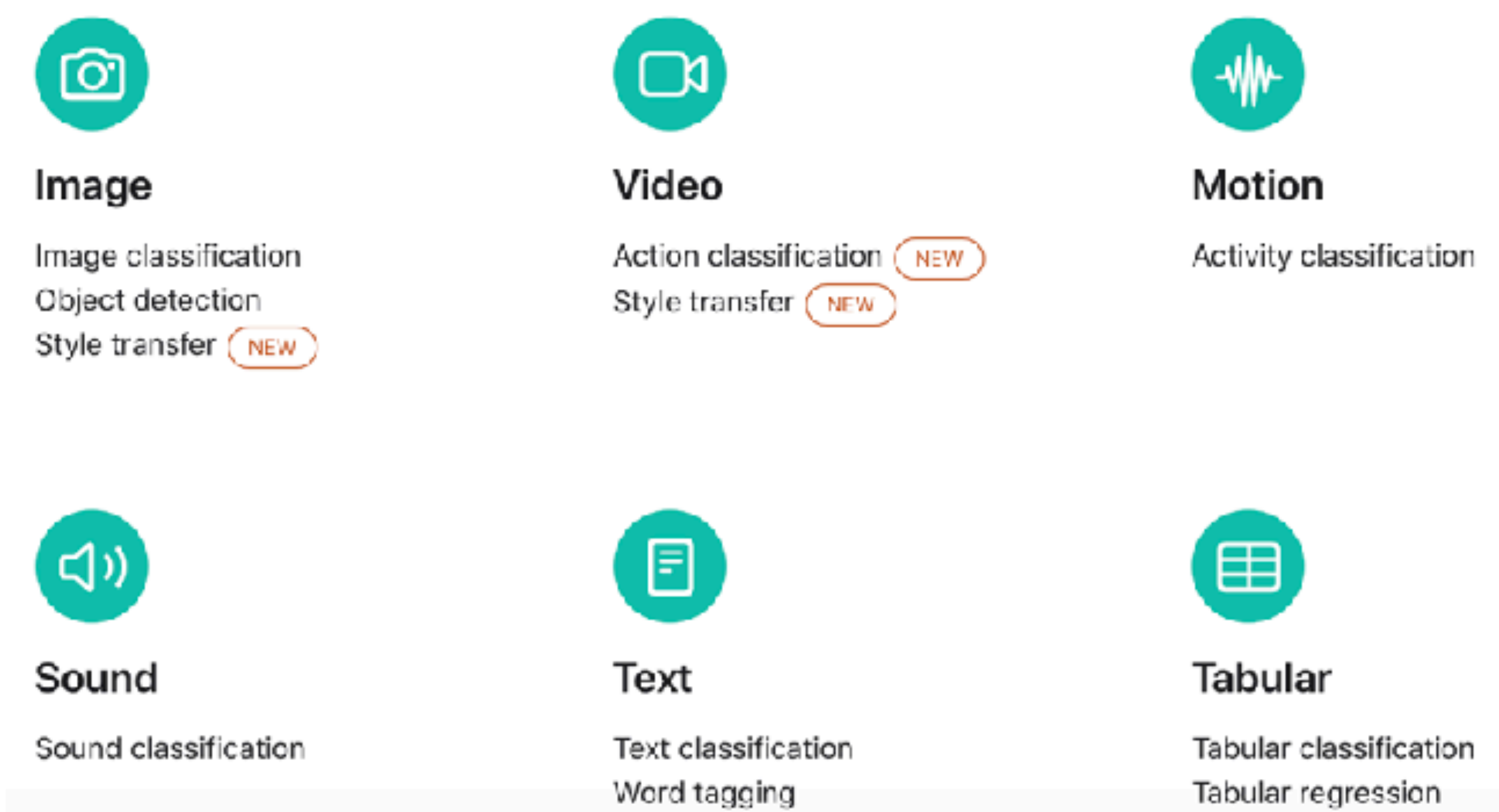
CoreML

- extended demo from beginning to end
- let’s add support for another feature from the user



more advanced: Create ML

making machine learning easy to use for developers that are not data scientists



could be good for final projects!!!

Task focused toolkits

- Recommender Systems
- Image Classification
- Drawing Classification
- Sound Classification
  - How it works
  - Advanced Usage
  - Deployment to Core ML
- Image Similarity
- Object Detection
- One-Shot Object Detection
- Style Transfer
- Activity Classification
- Text Classifier

How Does This Work?

Training and making predictions for a sound classifier model is a three stage process:

1. Signal preprocessing
2. A pretrained neural network is used to extract deep features
3. A custom neural network is used to make the predictions

Details below about each stage.

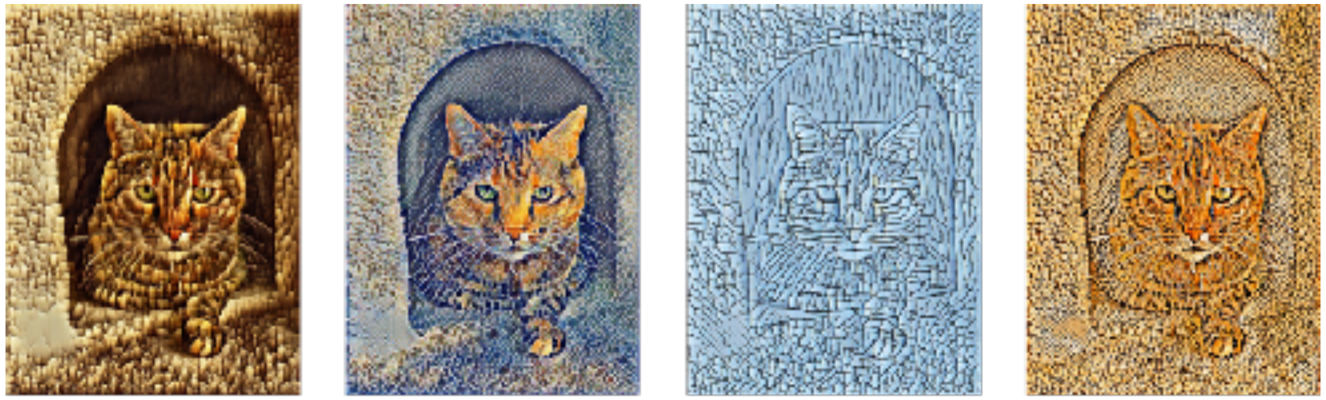
At a high level, the preprocessing pipeline does the following:

- The raw pulse code modulation data from the wav file is converted to floats on a [-1.0, +1.0] scale.
- If there are two channels, the elements are averaged to produce one channel.
- The data is resampled to only 16,000 samples per second.
- The data is broken up into several overlapping windows.
- A Hamming Window is applied to each windows.
- The Power Spectrum is calculated, using a Fast Fourier Transformation.
- Frequencies above and below certain thresholds are dropped.
- Mel Frequency Filter Banks are applied.
- Finally the natural logarithm is taken of all values.

The preprocessing pipeline takes 975ms worth of audio as input (exact input length depends on sample rate) and produces an array of shape (96, 64).

https://apple.github.io/turicreate/docs/userguide/sound_classifier/

could be good for final projects!!!



Style transfer model

The technique used in Turi Create is based on *"A Learned Representation For Artistic Style"*. The model is compact and fast and hence can run on mobile devices like an iPhone. The model consists of 3 convolutional layers, 5 residual layers (2 convolutional layers in each) and 3 upsampling layers each followed by a convolutional layer. There are a total of 16 convolutional layers.

There are three aspects about this technique that are worth noting:

- It is designed to be incredibly fast at stylizing images, allowing deployment on device. As a trade off, the model creation takes longer.
- A single model can incorporate a large number of styles without any significant increase in the size of the model.
- The model can take input of any size and output a stylized image of the same size.

During training, we employ Transfer Learning. The model uses the visual semantics of an already trained VGG-16 network to understand and mimic stylistic elements.

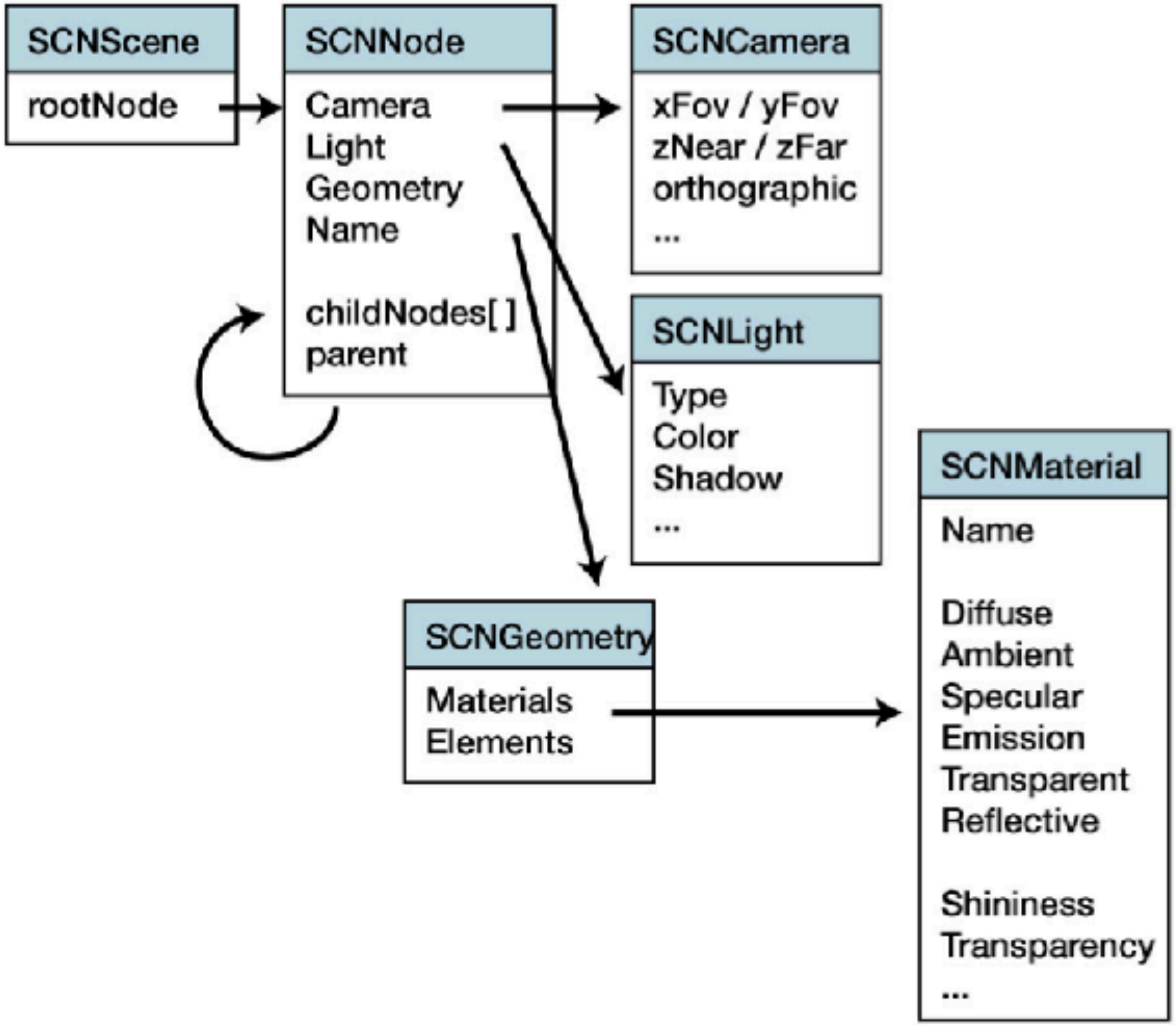
https://apple.github.io/turicreate/docs/userguide/sound_classifier/

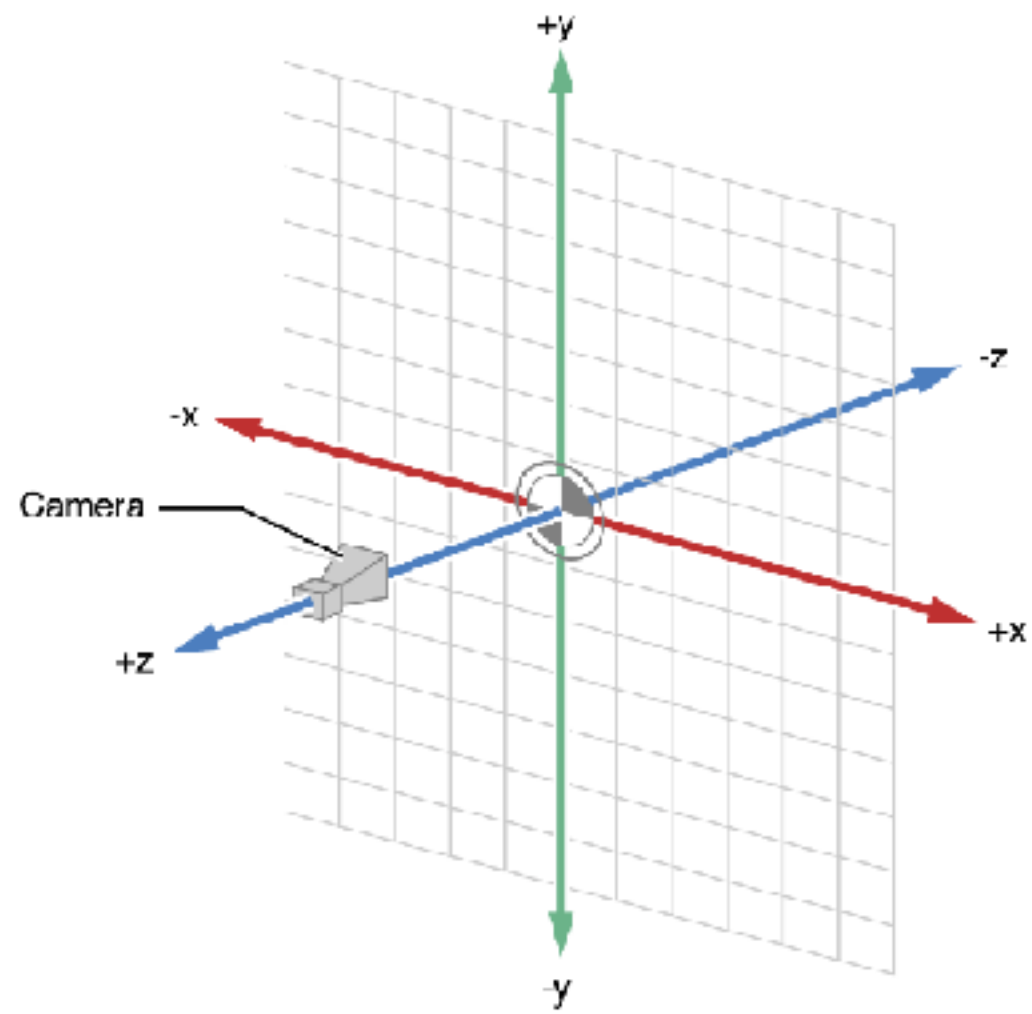
and **now**… moving on to
3D scenes…
(we will ***return*** to CoreML)

SceneKit review: 3D scenes

- need some basic knowledge to understand augmented reality and digital scene
- SceneKit allows you to create a 3D world and add physics, nodes, lighting, etc.
    - very powerful
- basic workflow:
    - setup world
    - add nodes

work flow in 3D scenes

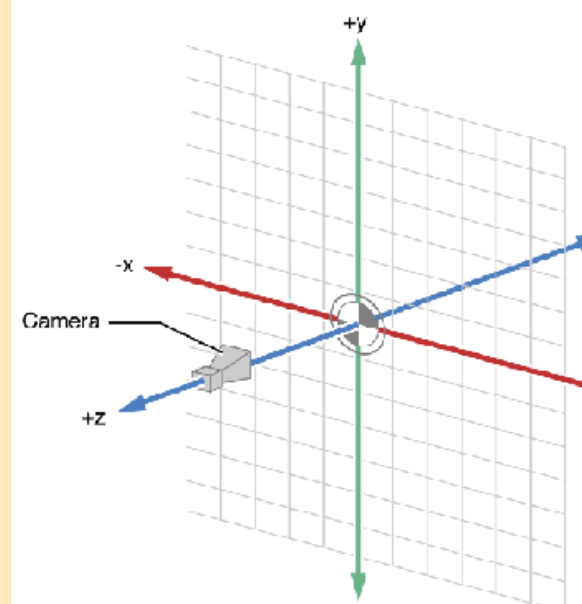




setting up a world

```
// Setup scene
scene = SCNScene()
scene.physicsWorld.speed = 1

// Setup camera position
cameraNode = SCNNode()
cameraNode.camera = SCNCamera()
cameraNode.position = SCNVector3(x: 0, y: 0, z: 30)
scene.rootNode.addChildNode(cameraNode)

// add a plane to the view that users must bounce the ball on
//setup the geometry of node (as a plane)
let wall = SCNPlane(width: 10.0, height: 10.0)
wall.firstMaterial?.doubleSided = true
wall.firstMaterial?.diffuse.contents = UIColor.redColor() // make it red!!

// add the plane to the world as a static body (no dynamic physics)
wallNode = SCNNode()
wallNode.geometry = wall
wallNode.physicsBody = SCNPhysicsBody.staticBody()
wallNode.position = SCNVector3(x: 0.0, y: 0.0, z: -5)

scene.rootNode.addChildNode(wallNode)

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

add camera



add plane

make this scene the world

add to world

```
func addBall() {

        // add a sphere to the world
        let ballGeometry = SCNSphere(radius: 1.0)

        // make it have texture
        let ballMaterial = SCNMaterial()
        ballMaterial.diffuse.contents = UIImage(named: "texture")

        // adjust physics to make it slightly highly bounc
        let ball = SCNNode(geometry: ballGeometry)
        ball.geometry?.firstMaterial = ballMaterial;
        ball.physicsBody = SCNPhysicsBody.dynamicBody()
        ball.physicsBody?.restitution = 2.5
        ball.position = SCNVector3(x: 0, y: 0, z: 0)

        scene.rootNode.addChildNode(ball)

  }
```

make ball

add physics

make bouncy

add to world

but… we want the SCNScene

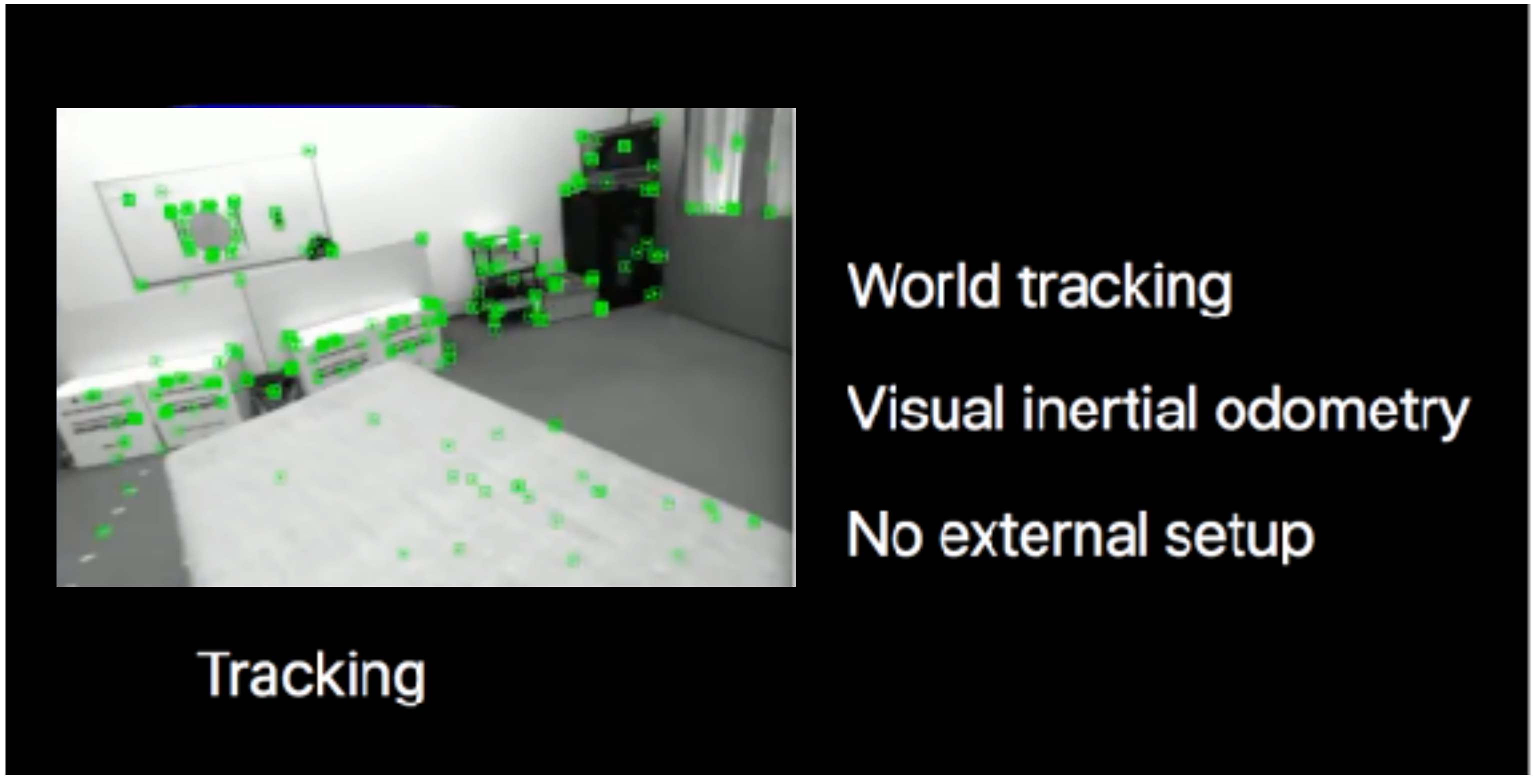
to be the real world

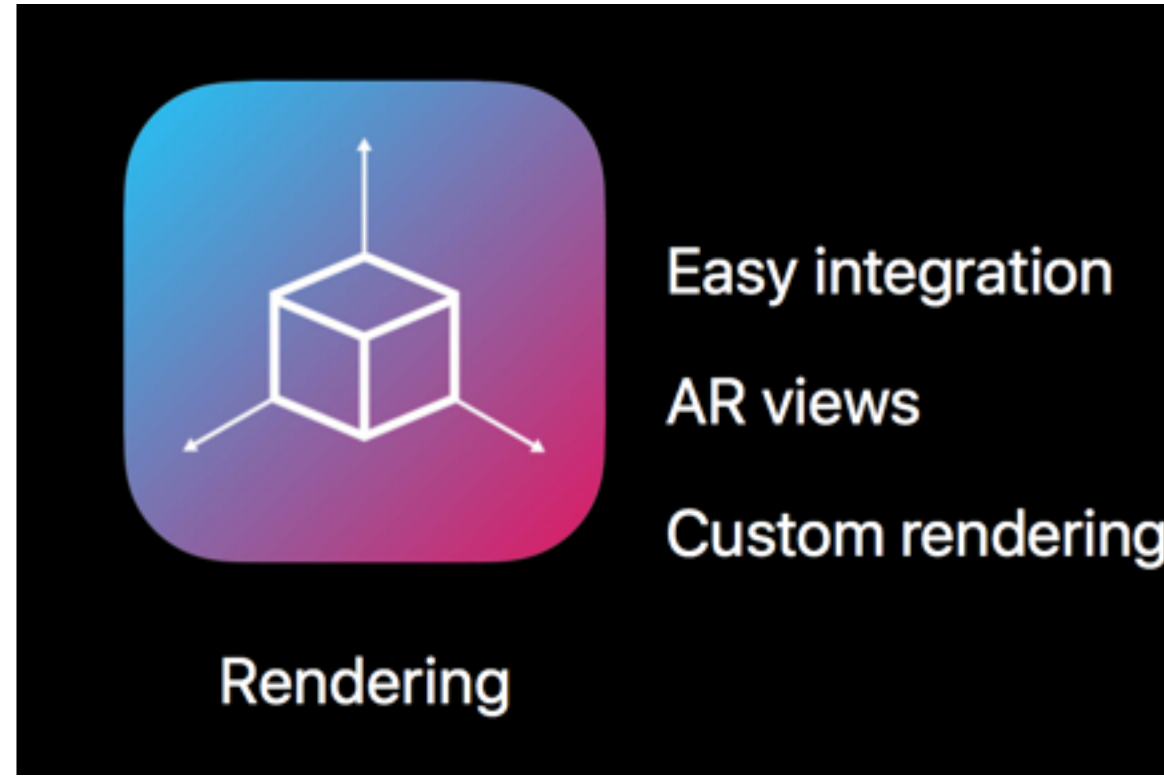
ARKit anchors the SCNScene to objects in the immediate environment

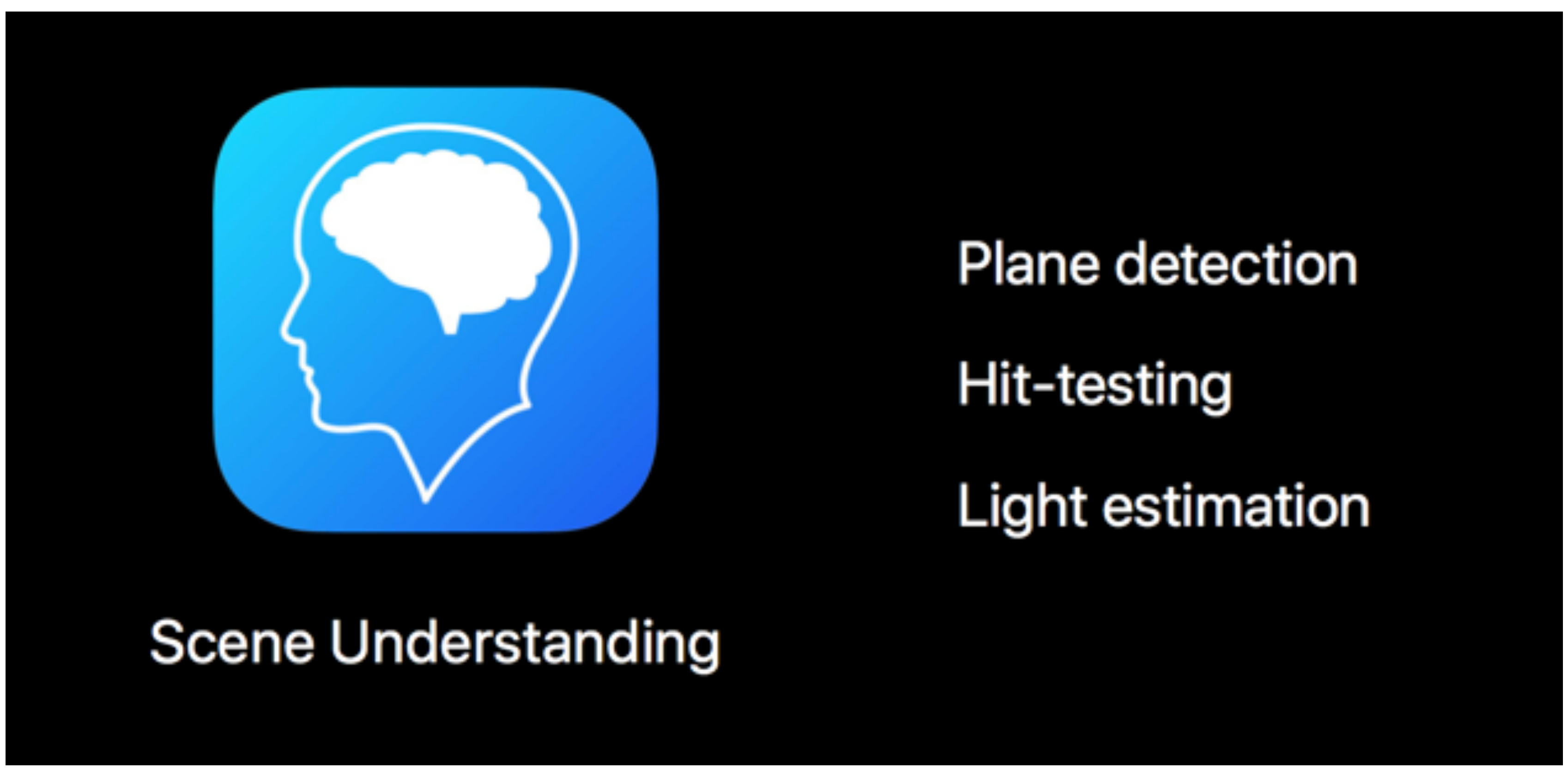
```
// Setup view
let view = self.view as SCNView
view.scene = scene
```

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

ARKit basics



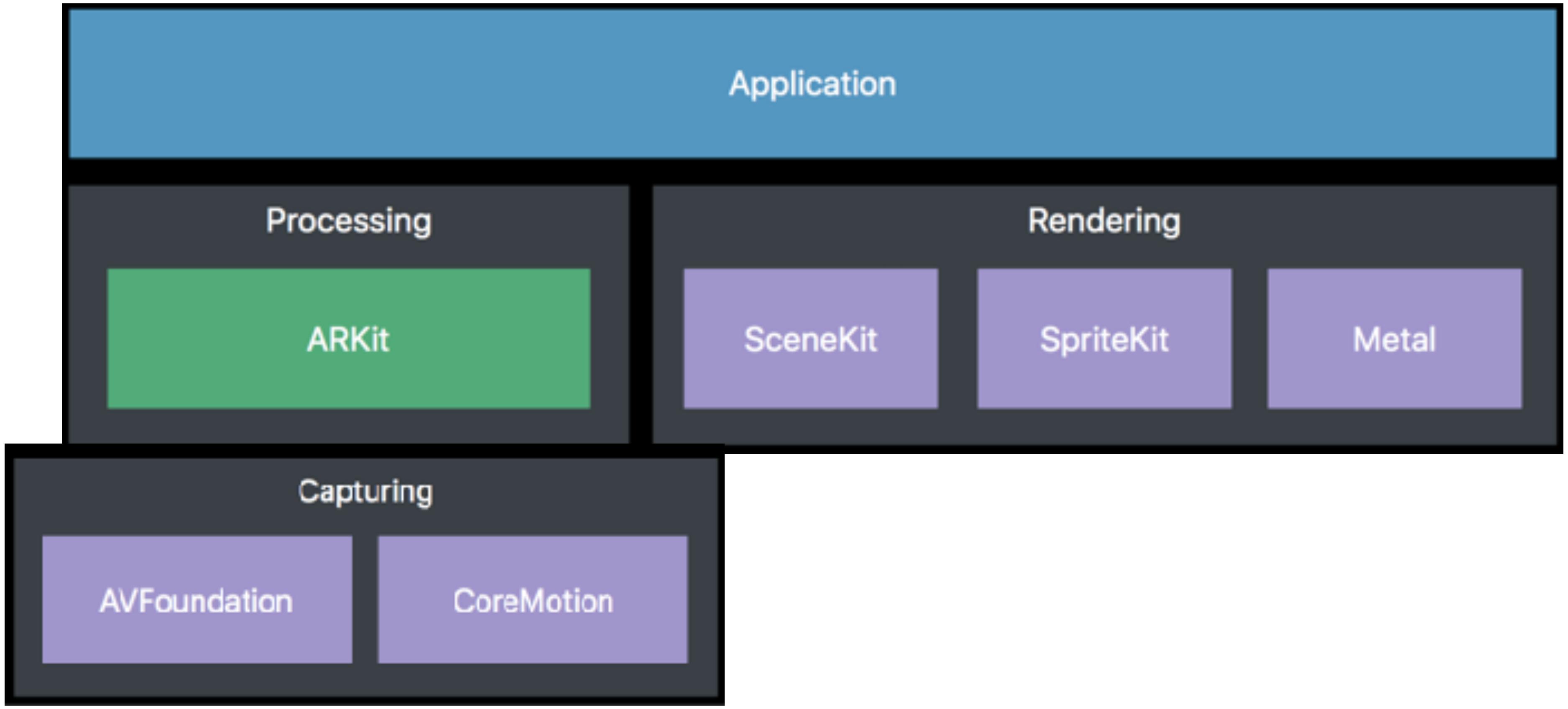


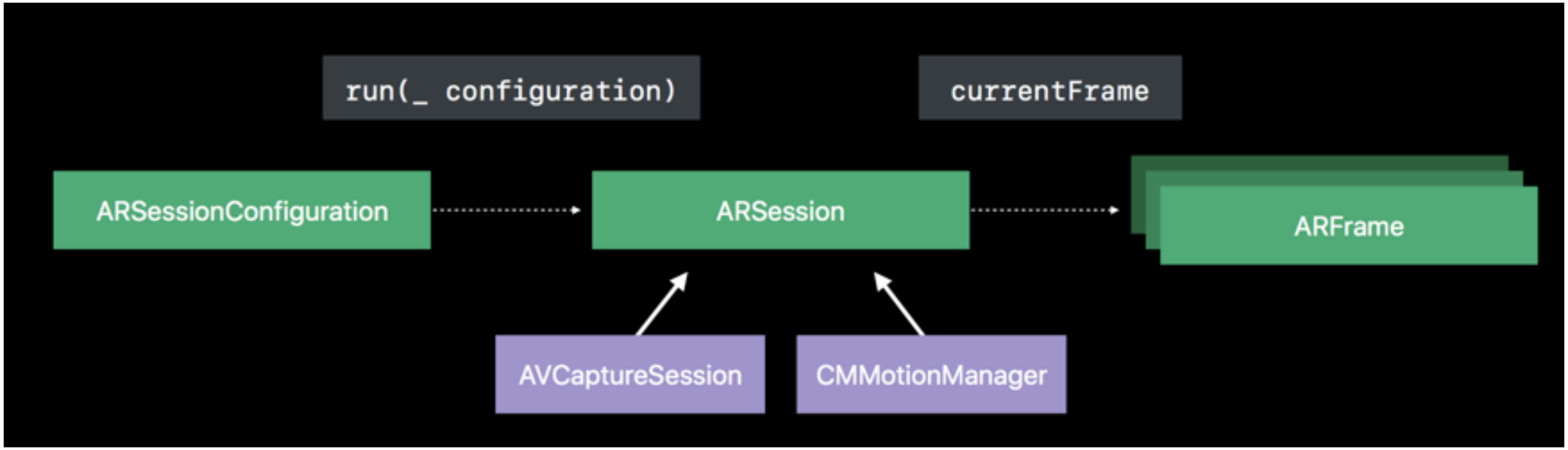


https://developer.apple.com/videos/play/wwdc2017/602/

ARKit system integration

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





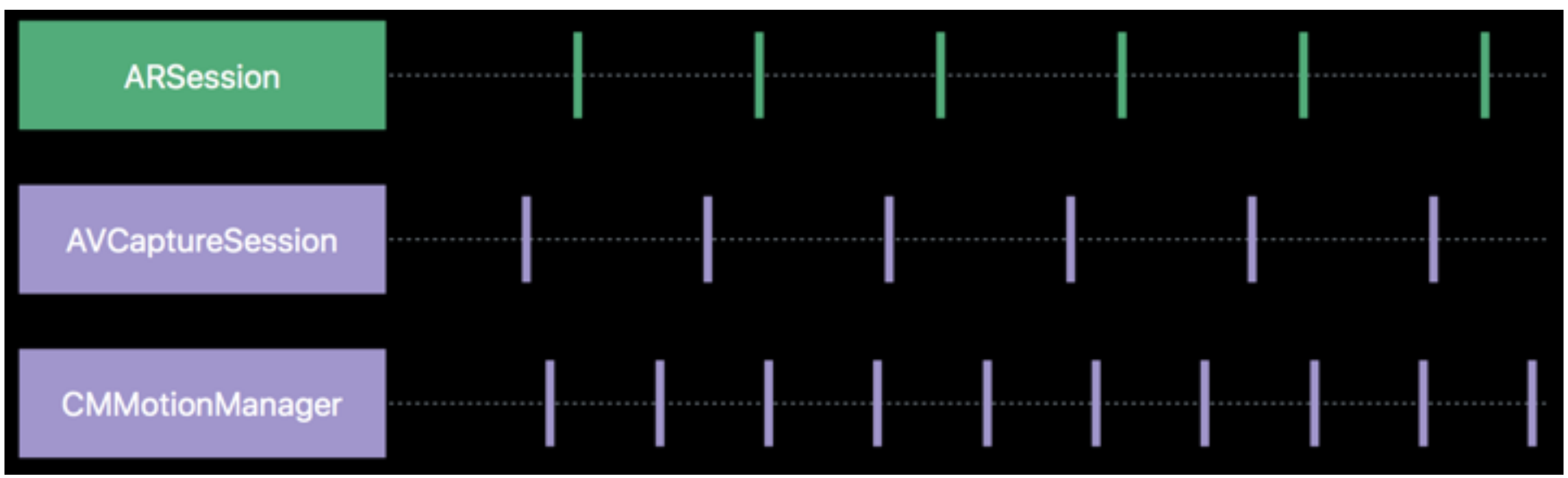
session basics

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

// Run your session
session.run(configuration)

// Pause your session
session.pause()

// Resume your session
session.run(session.configuration)

// Change your configuration
session.run(otherConfiguration)
```

session basics

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

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

delegation



unified sampling

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

adding to the AR world

poll ARSession

create a plane

add to world

translate

```
// 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)

// 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 )
```

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.

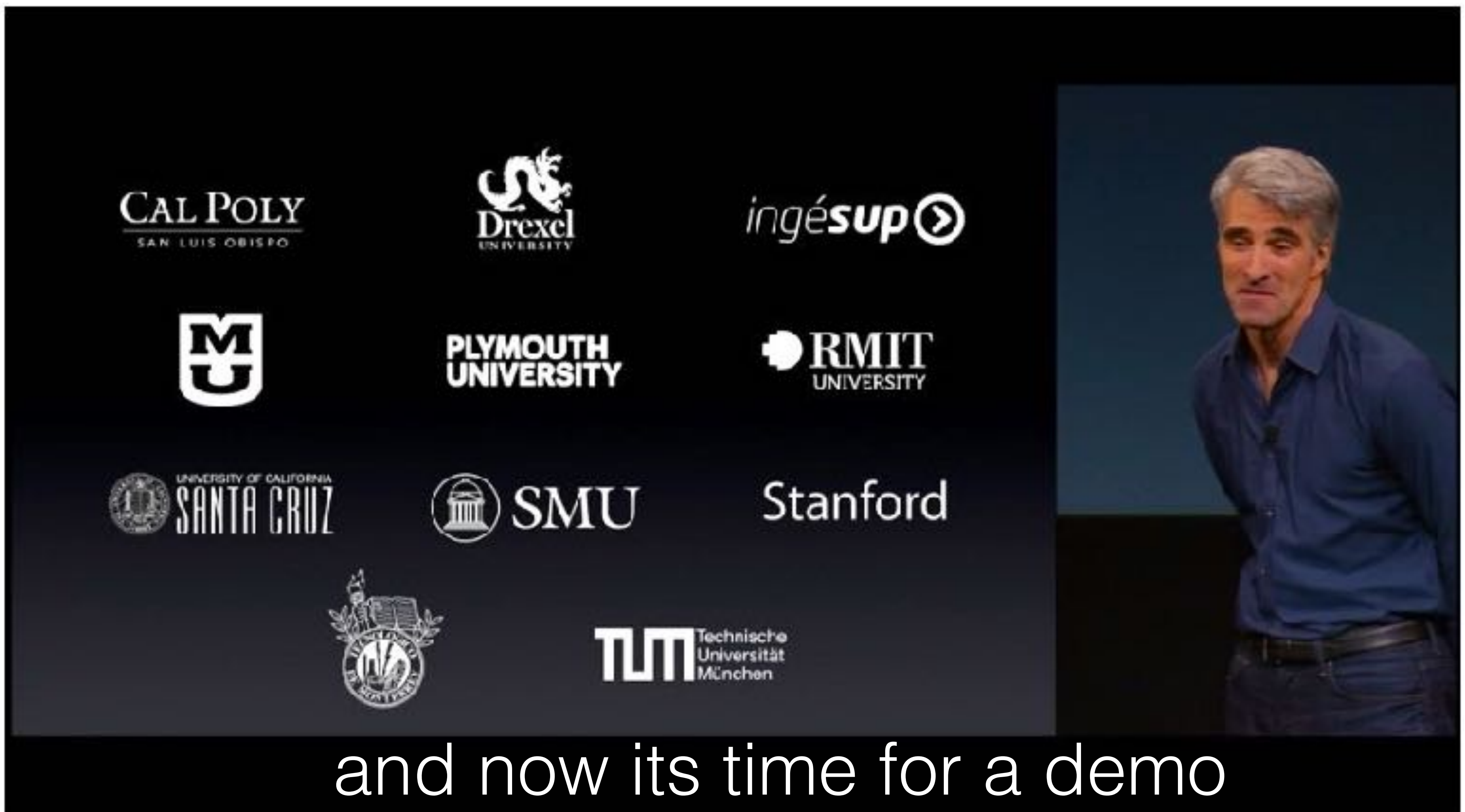
```
func SCNMatrix4MakeScale(Float, Float, Float)
```

Returns a matrix describing a scale transformation.

ARKit and SceneKit

- extended demo

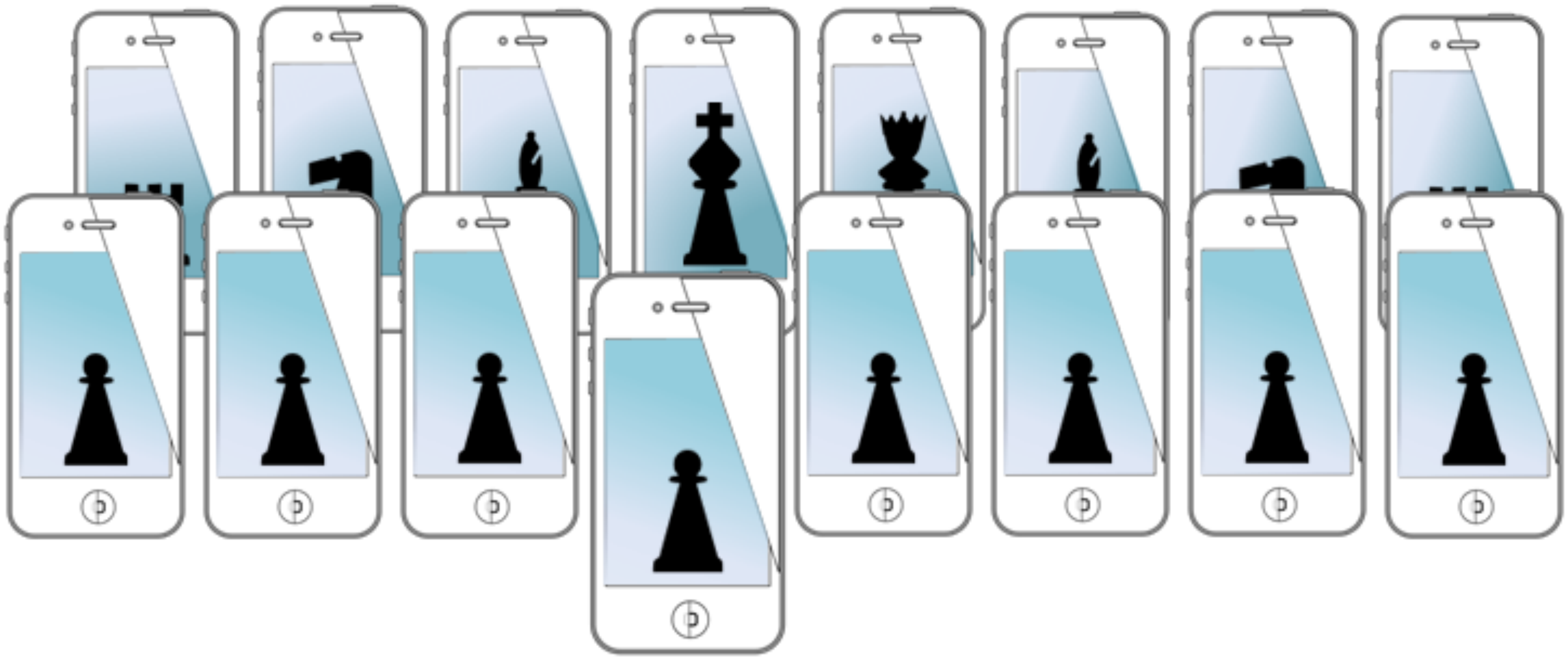
branches: (1) bare implementation,
(2) image extension, (3) stylization



for next time…

- ARKit with Object Recognition and YOLO
- Speech
- Pitching
- ~Fin~

MOBILE SENSING LEARNING



CS5323 & 7323

Mobile Sensing and Learning

ARKit and SceneKit

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