AWS re:Invent

Get More Training

Deepen your technical expertise after re:Invent with AWS online and instructor-led courses.

aws.amazon.com/training

#AWSTraining

Take Hands-On Labs

Practice AWS in a live environment. Choose from 70+ labs, organized by service and use case. Open Tue-Fri.

Free Onsite

Get AWS Certified Onsite

Validate your AWS expertise. Exams available Mon-Fri. Visit the AWS Certification Information booth to register.

#AWSCertified

amazon web services

AWS re:Invent

Are you AWS Certified?

AWS Certification Appreciation Reception

Public House at the Venetian

Thursday, October 8

5-7 PM

Open to AWS Certified Individuals

RSVP: https://aws.amazon.com/reinvent/certified-appreciation-event-2015/

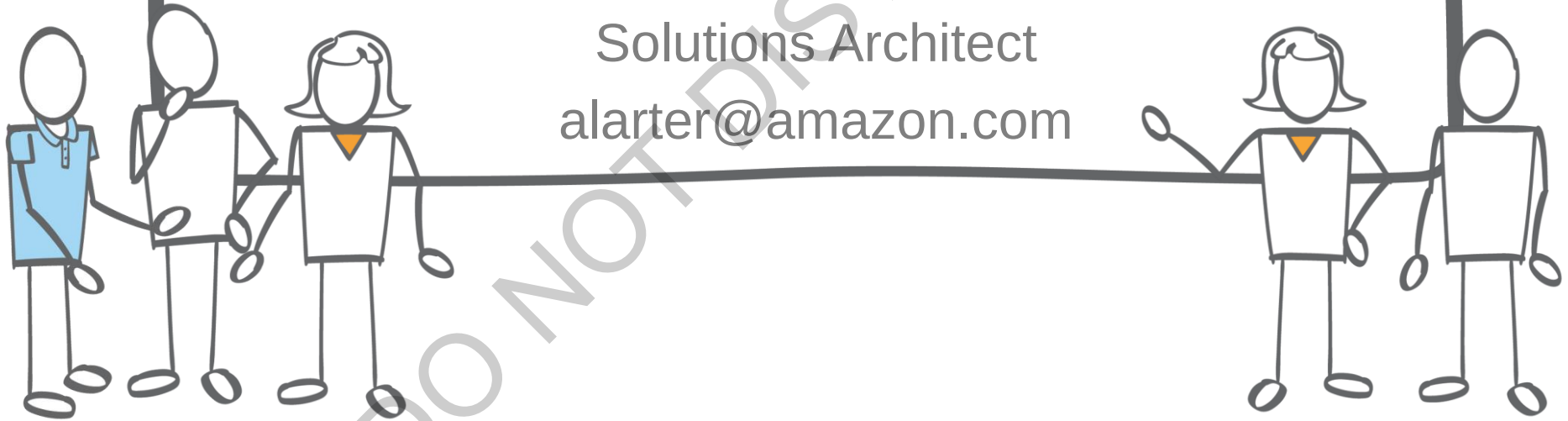
amazon web services

Creating applications for Mobile and IoT

Adam Larter

Solutions Architect

alarter@amazon.com



100 “Things” per second

The Internet of People

The Internet of Things is here on AWS

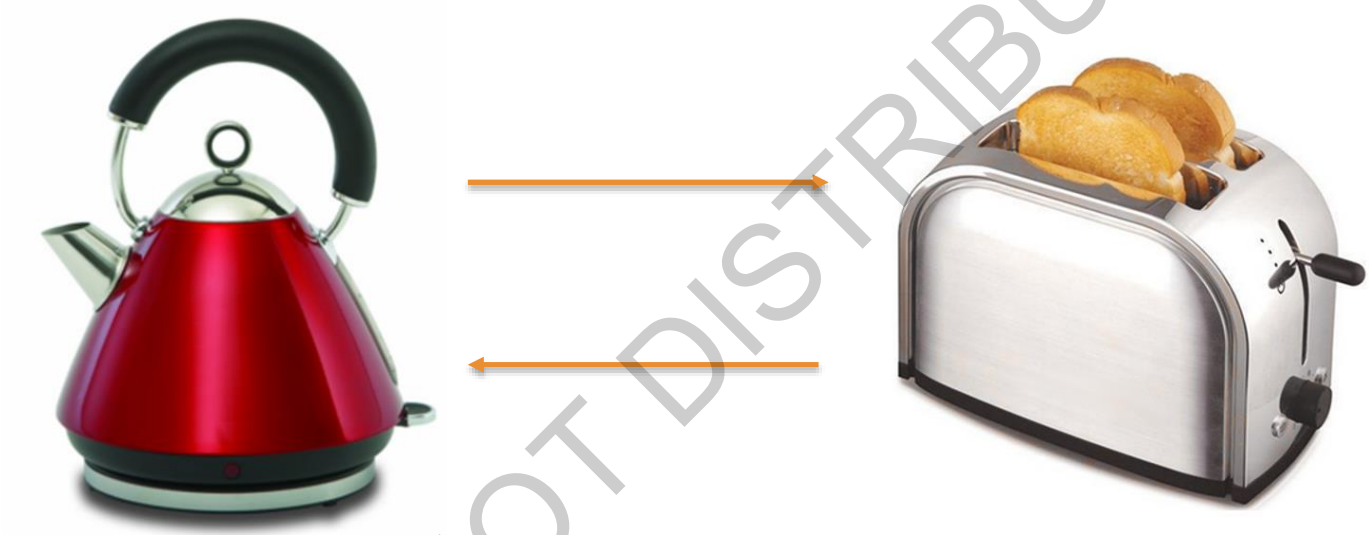
The Internet of Connected Devices – IoT & Mobile

- **Mobile devices**
  - iOS, Android, Kindle – smart phones and tablets
- **Maker devices**
  - Arduinos, Raspberry Pi's, Intel Edison
- **Embedded devices & wearables**
  - Health & fitness management, safety & tracking
- **Smart home**
  - Smoke alarms, temperature sensors, light globes & switches

On the Internet

nobody knows you're a toaster

Our goal: Internet-enabling consumer appliances



What we will build today

- ***Integrated appliance management system***

  - Connected appliances – toasters, kettles, etc
  - Using Intel Edison IoT devices
  - Two-way communication between appliances
  - Central website as our device registry
  - Companion mobile application to control the system

All this accomplished without running any compute instances

We are creating a system

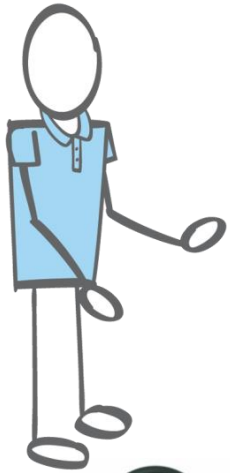
not a simple standalone application…

…so it's going to get

a little bit complicated

System Overview:
Tethering

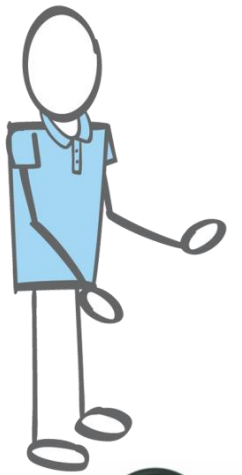
Bob



Connected Appliance

Companion mobile application

Bob



Both the app and the appliance are connected to AWS over **WiFi**

There is no direct connection between the mobile and the appliance

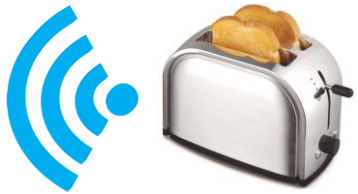
In order for the mobile app to learn about the presence of the appliance we ‘Tether’ them together using a QR code

http://bit.ly/mobile-iot-bootcamp

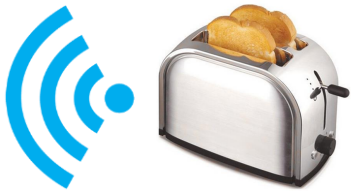
The mobile app can receive information from the appliance via AWS and vice versa

System Overview:
Discovery

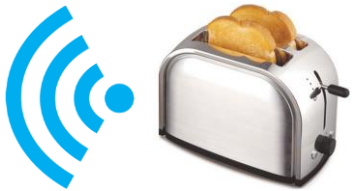
Bluetooth Low Energy (BLE) adertisements used for Discovery and Proximity detection



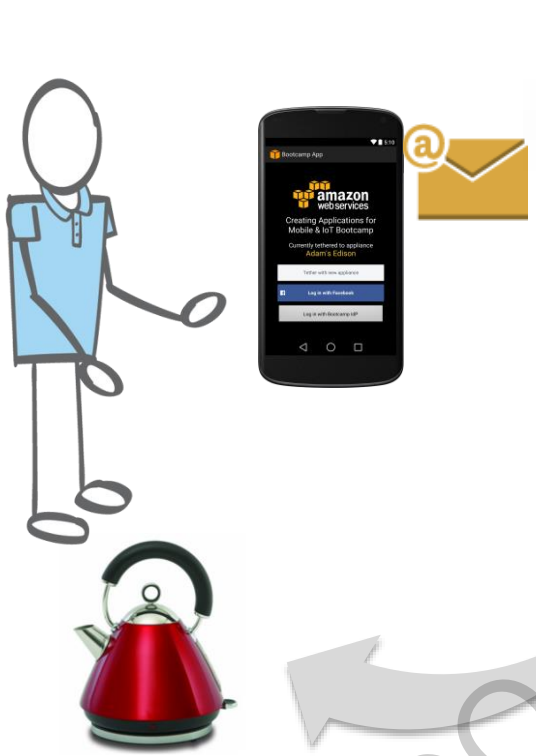
As each advertisement is received, a message is sent to the AWS cloud for processing and storage



Distance information and other metadata is stored in the AWS Cloud, ready to be accessed by the mobile app

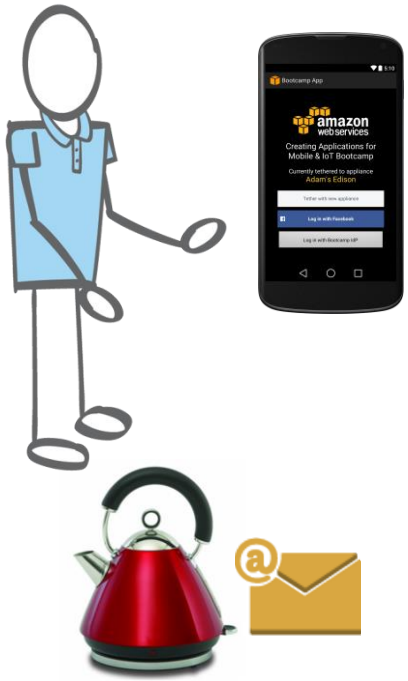


System Overview:
Pairing



User interacts with mobile app and commands the tethered appliance to ‘Pair’ with the other appliance

Note that the mobile app did not communicate directly with the appliance – a message was sent via the AWS Cloud



Messages are then exchanged between appliances using MQTT over the WiFi (not Bluetooth)

MQTT over WiFi

System Overview:
Chat

Bob

Virtual chat channel

Alice

Bob's tethered appliance

Alice's tethered appliance

1 Message is sent from app to AWS Cloud

2 AWS Cloud forwards message to appliance

Message sent between appliances using MQTT

MQTT over WiFi

5 Message is received and processed on the mobile app

4 Message sent to the mobile app via the AWS Cloud

MQTT over WiFi

Mobile apps never communicate directly

All communication between mobile apps passes between the appliances

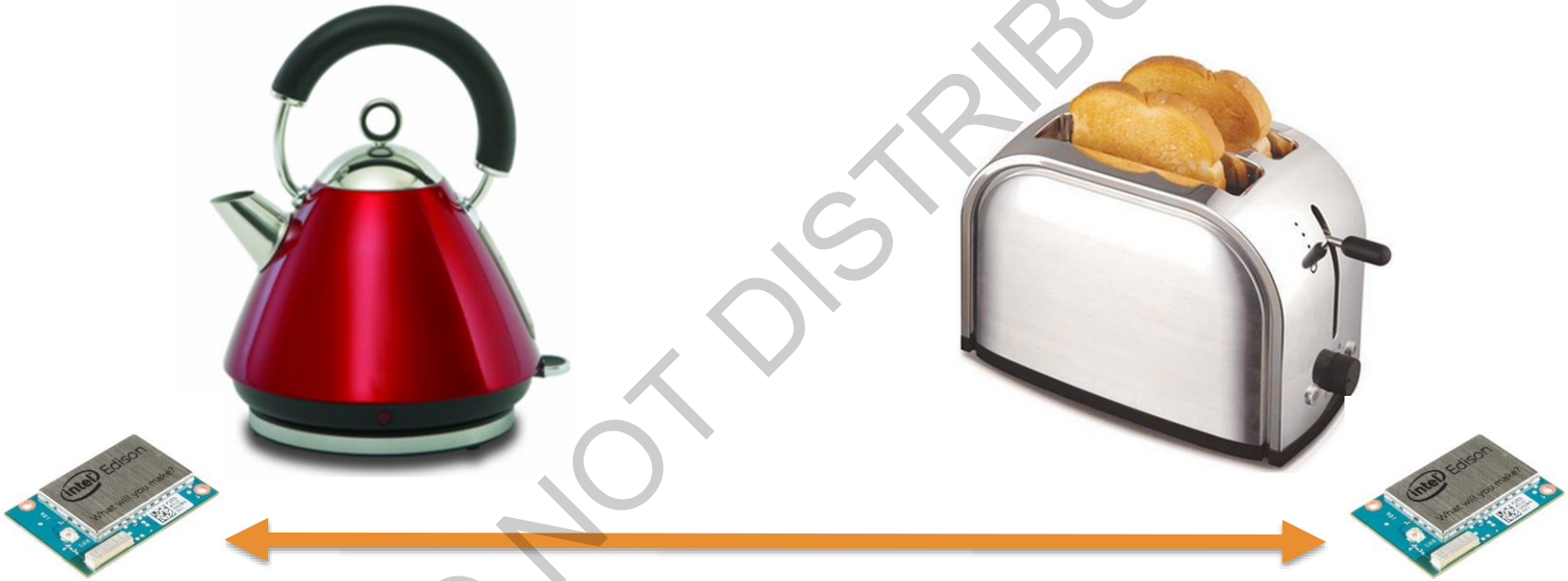
from and to the AWS Cloud

Mobile apps never communicate directly

All communication between mobile apps passes between the appliances

from and to the AWS Cloud

Internet-enabling appliances using Intel Edison



Introducing the Intel Edison

- Dual-core 500 MHz Atom CPU
- 4 GB storage
- 1GB RAM
- Yocto Linux
- WiFi 802.11 a/b/g/n
- Bluetooth 4.0

We will work in pairs

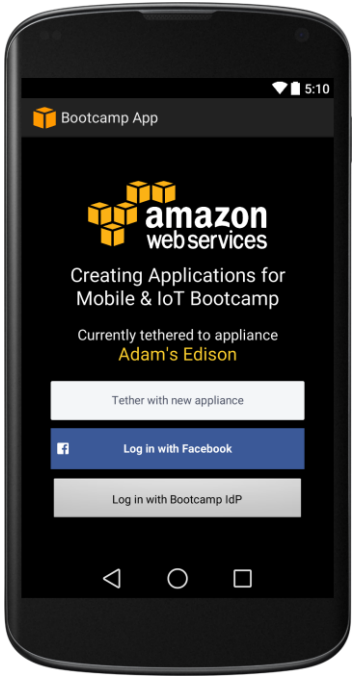
Everyone is a Bob!

…and an Alice!

Our system specification

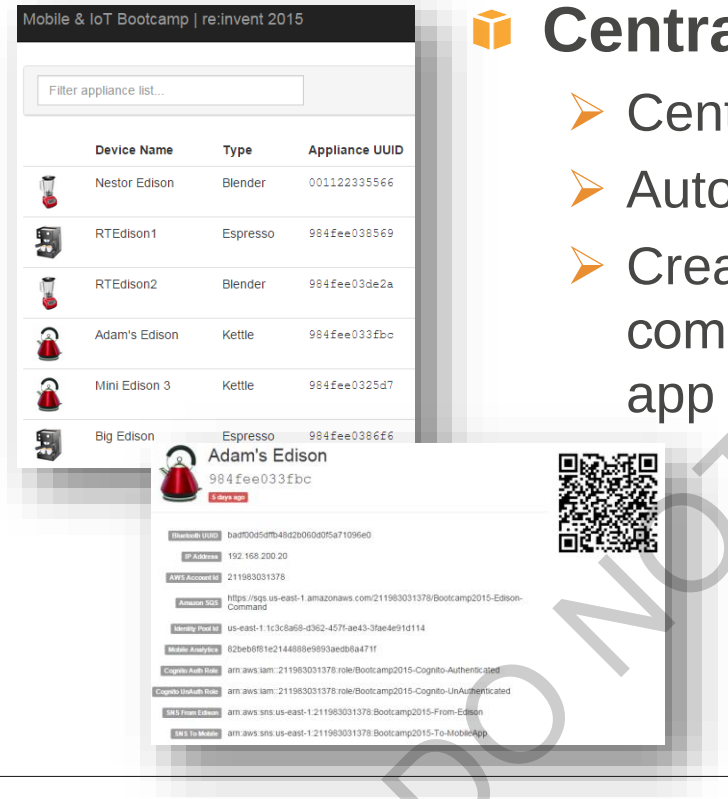
- **Appliances**
  - Represented by Intel Edison devices
  - Appliances implemented in NodeJS code
  - Discover each other using Bluetooth Low Energy advertisements – the Bluetooth 4.0 specification
  - Continually send telemetry data to the AWS cloud
  - Accept commands from the user via the AWS cloud
  - Send commands to other devices via MQTT
  - Send responses to the user via the AWS cloud

Our system specification



- **Companion mobile app**
  - Log in using our own Identity Provider (FB optional)
  - Dynamically ‘tether’ to an appliance (Edison)
  - Show list of appliances nearby our tethered appliance
  - Chat between one mobile app and another app – communication sent via the two tethered Edison appliances
  - Control panels for each appliance
  - Analytics in the mobile app to capture custom events, in-app purchases and user stickiness

Our system specification



Central web site

- Central repository for appliance configuration
- Automatically updated by Edison at startup
- Creates QR code to allow the user to scan from the companion mobile application to ‘tether’ the mobile app to an appliance

http://bit.ly/mobile-iot-bootcamp

AWS services we will use in our system

Amazon Cognito

- Identity & data synchronization
  - Determine if a device or user is allowed to do something
  - Share state between multiple instances of the app

AWS Lambda

- Event-driven architecture
  - Running code in response to events in the system

AWS services we will use in our system

AWS IAM

- **Identity & Access Management**
  Secure, fine-grained control of cloud resources

Amazon Kinesis

- **Telemetry**
  High-velocity data stream containing BLE detection data

Amazon SQS

- **Commands**
  Issuing commands to the appliance (chat/pair)

AWS services we will use in our system

Amazon
DynamoDB

- **Shared data store**
  High throughput NoSQL store for our detection data

Amazon SNS
Mobile Push

- **Push notification & synchronization**
  Chat messages and other commands
  in-bound to the mobile app

Amazon
API Gateway

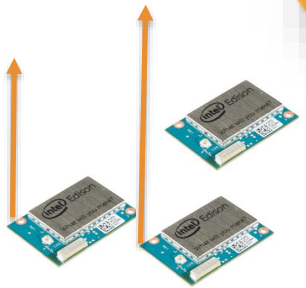
- **API Gateway**
  Our custom authentication mechanism will be implemented as Lambda functions exposed via the Amazon API gateway

Services used for Appliance Detection

Services used for Appliance Detection

Amazon Cognito

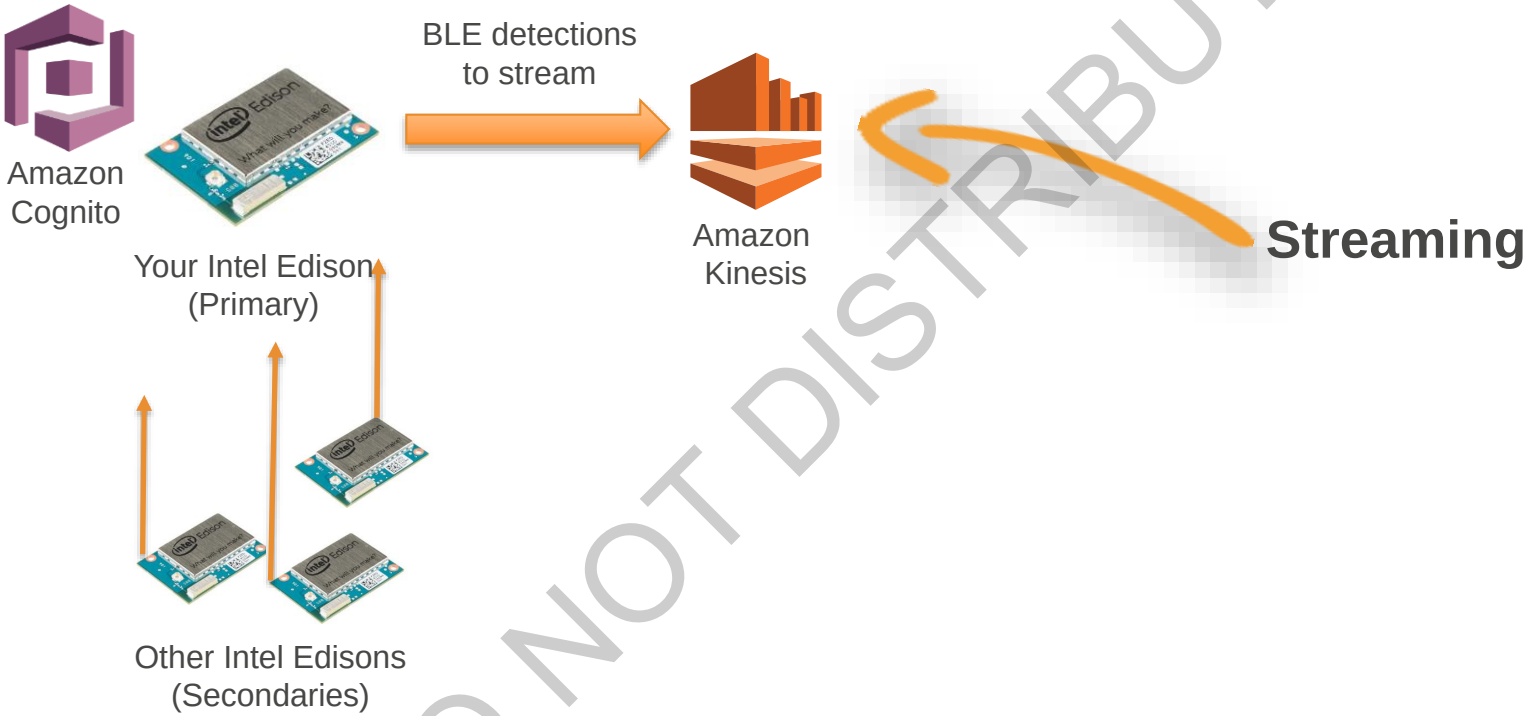
Your Intel Edison (Primary)



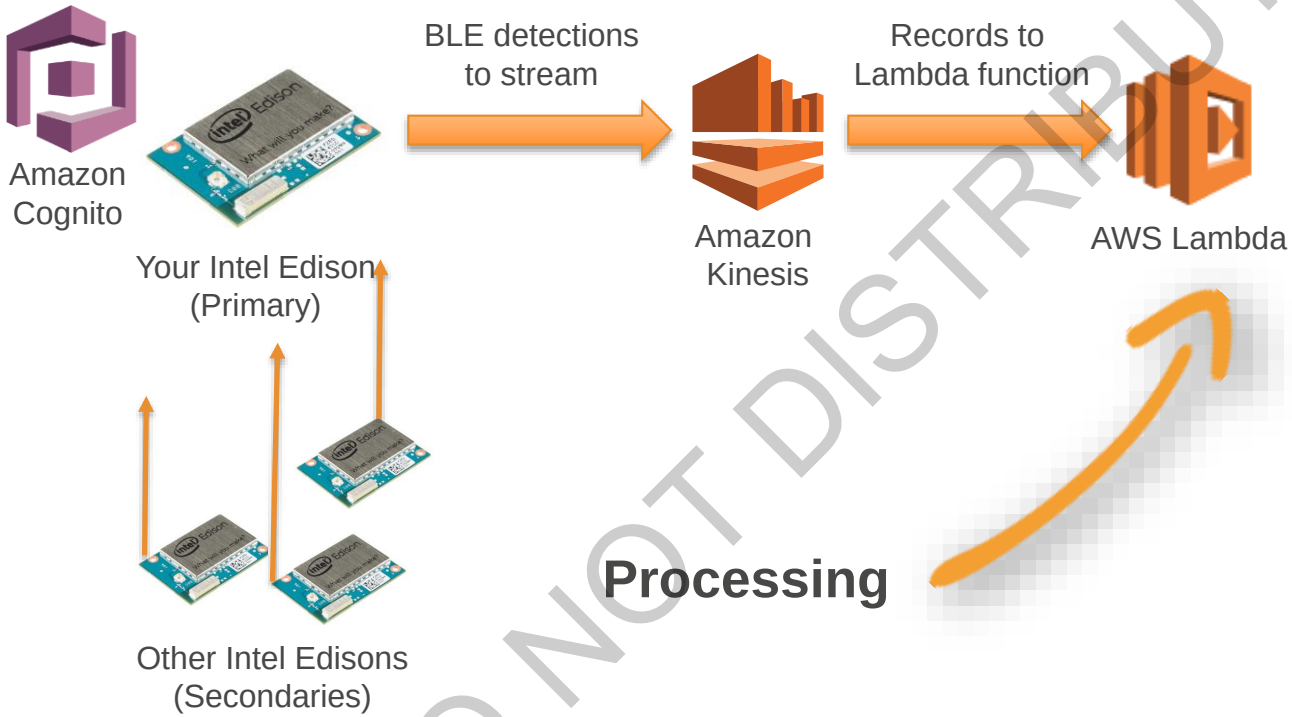
Other Intel Edisons (Secondaries)

Detection

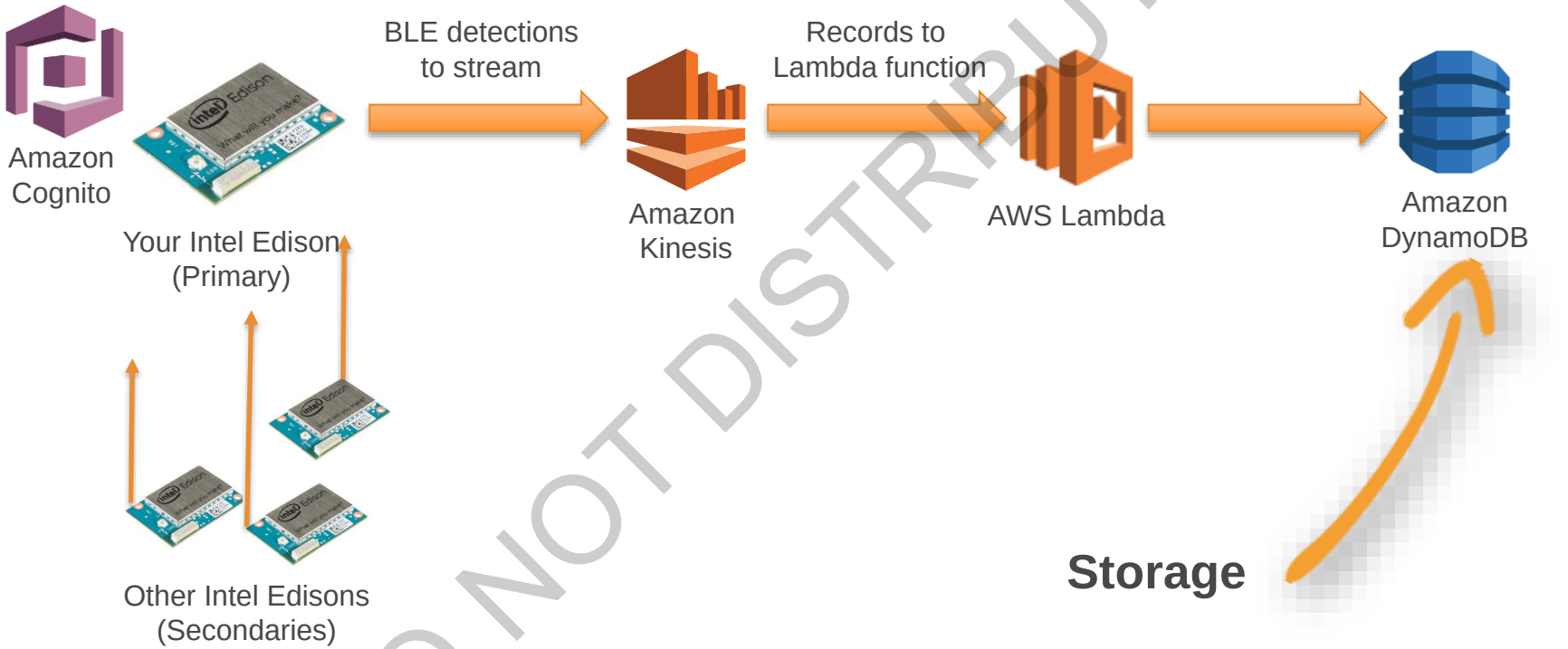
Services used for Appliance Detection



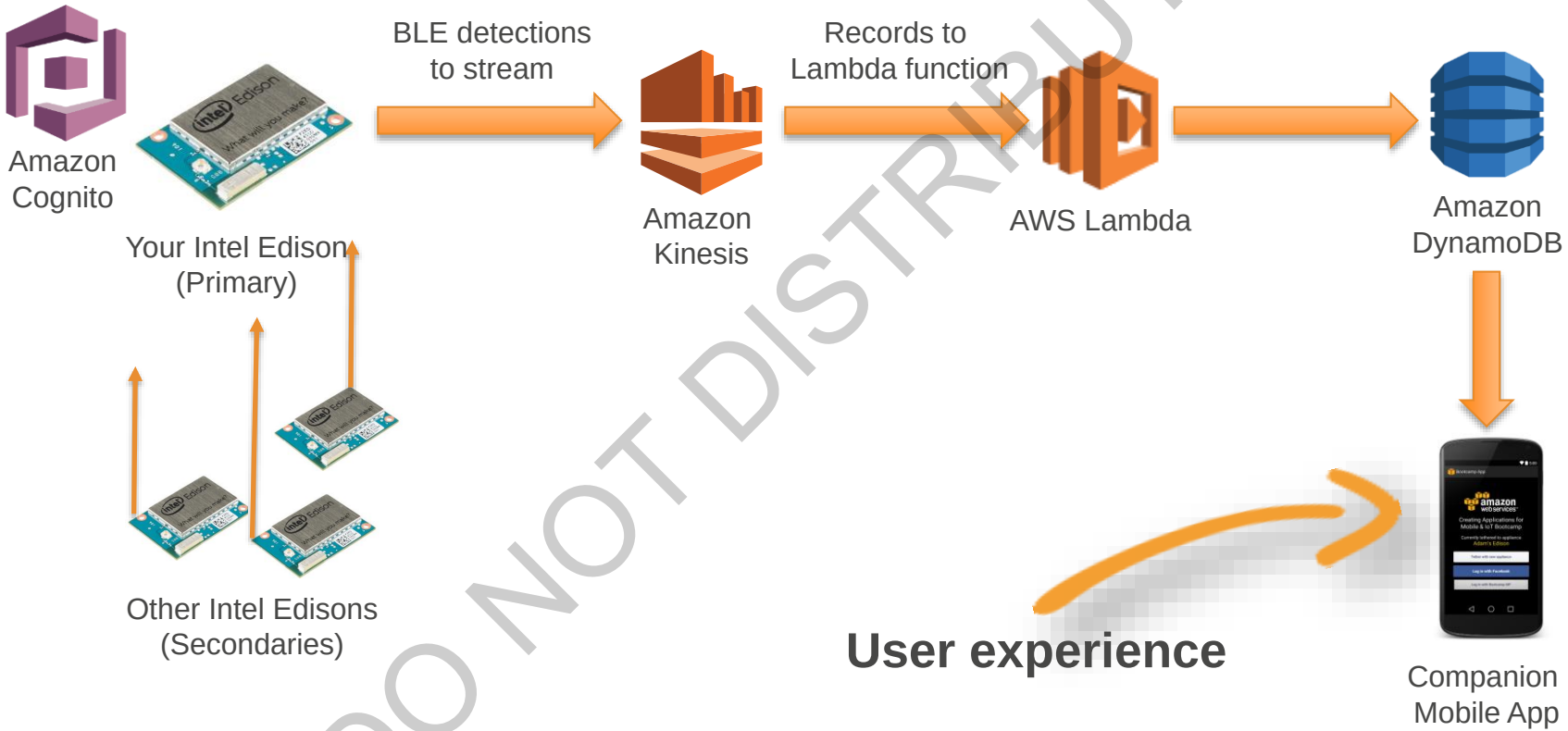
Services used for Appliance Detection



Services used for Appliance Detection



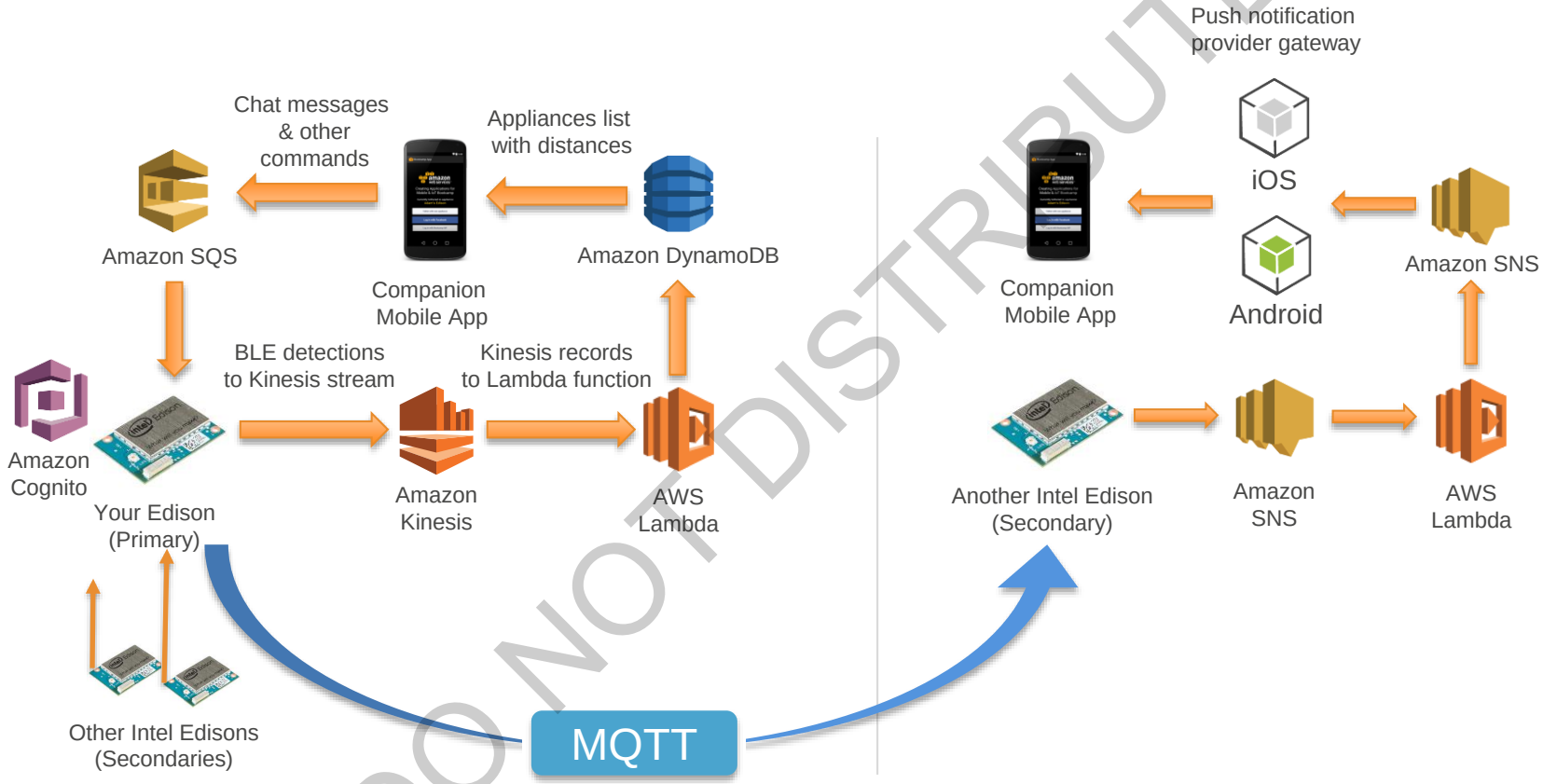
Services used for Appliance Detection



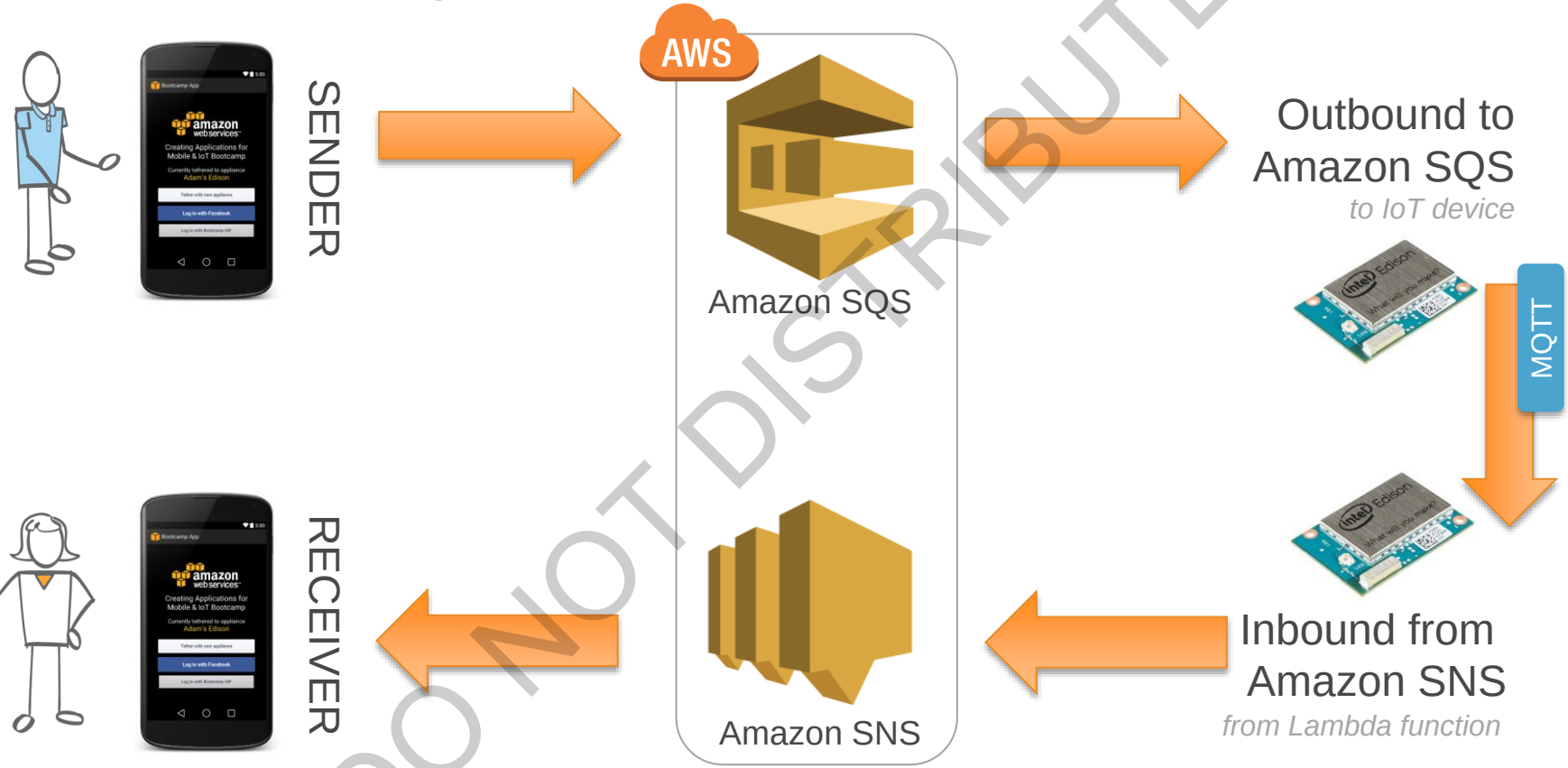
System design & architecture

Integrated Appliance Management System

System architecture



Data-flow is easy to remember!



Bluetooth is only used for

Discovery/Proximity

MQTT over WiFi is used for

cross-appliance communication

HTTP over WiFi is used for

everything else!

Mobile & IoT Challenges

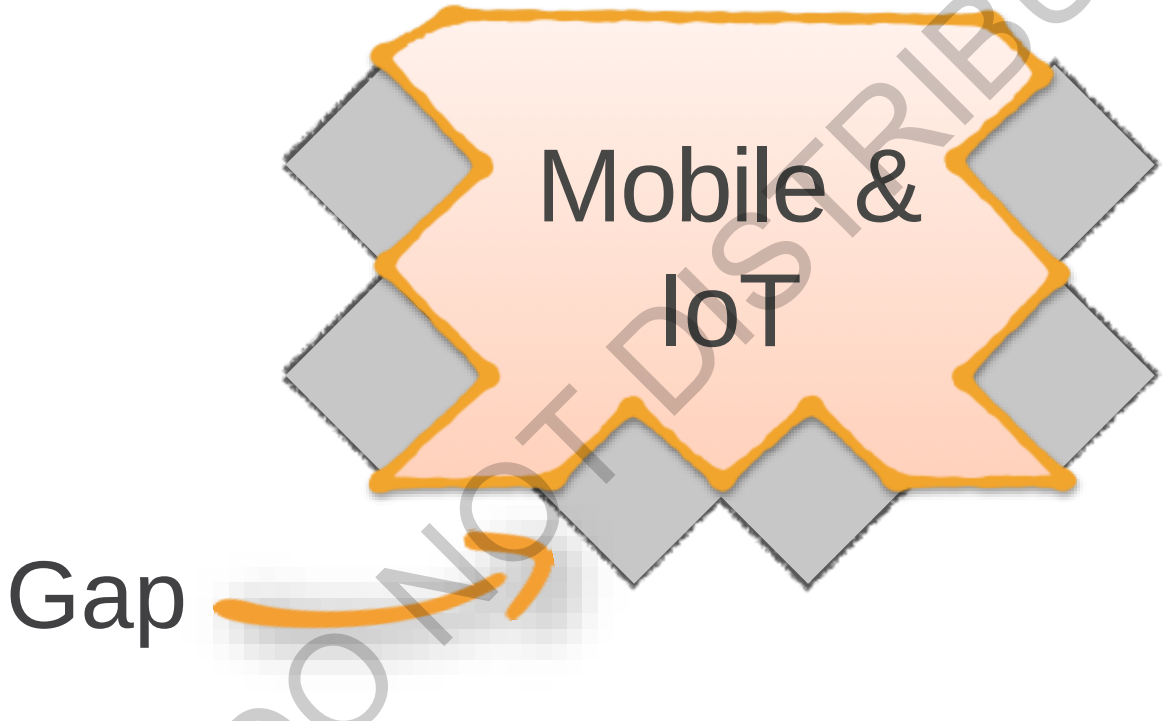
How AWS meets the challenges

IoT & Mobile challenges

- Devices constrained in at least one computing dimension
- Regularly deployed at counts in the thousands, millions or more
- May have minimal human interface or human operator interaction

Computers… not just ‘dumb sensors’

Resource constrained computing gaps



AWS helps fills the gaps in IoT/Mobile

IoT challenges

- **Authorization**
  Determine if a device should be allowed to do something

- **Telemetry**
  Remotely determining what a device senses

- **Commands**
  Issuing commands to the remote device

What about Mobile applications?

Authenticate users

Manage users and identity providers

Authorize access

Securely access cloud resources

Synchronize data

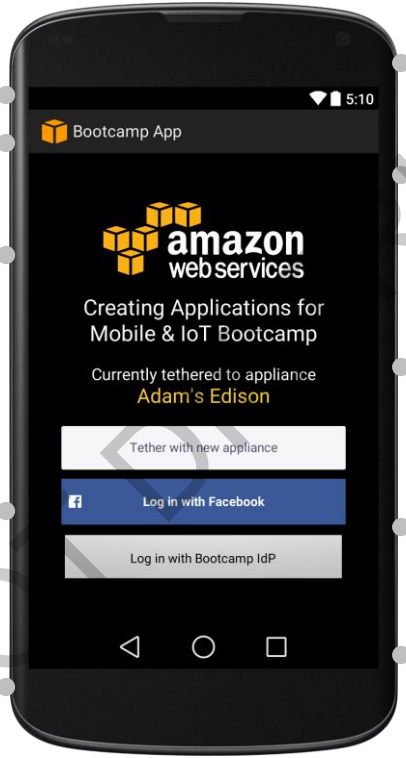
Sync user prefs across devices

Analyze User Behavior

Track active users, engagement

Track Retention

Manage funnels, Campaign performances



Store and share media

Store user-generated photos Media and share them

Deliver media

Automatically detect mobile devices Deliver content quickly globally

Send push notifications

Bring users back to your app by sending messages reliably

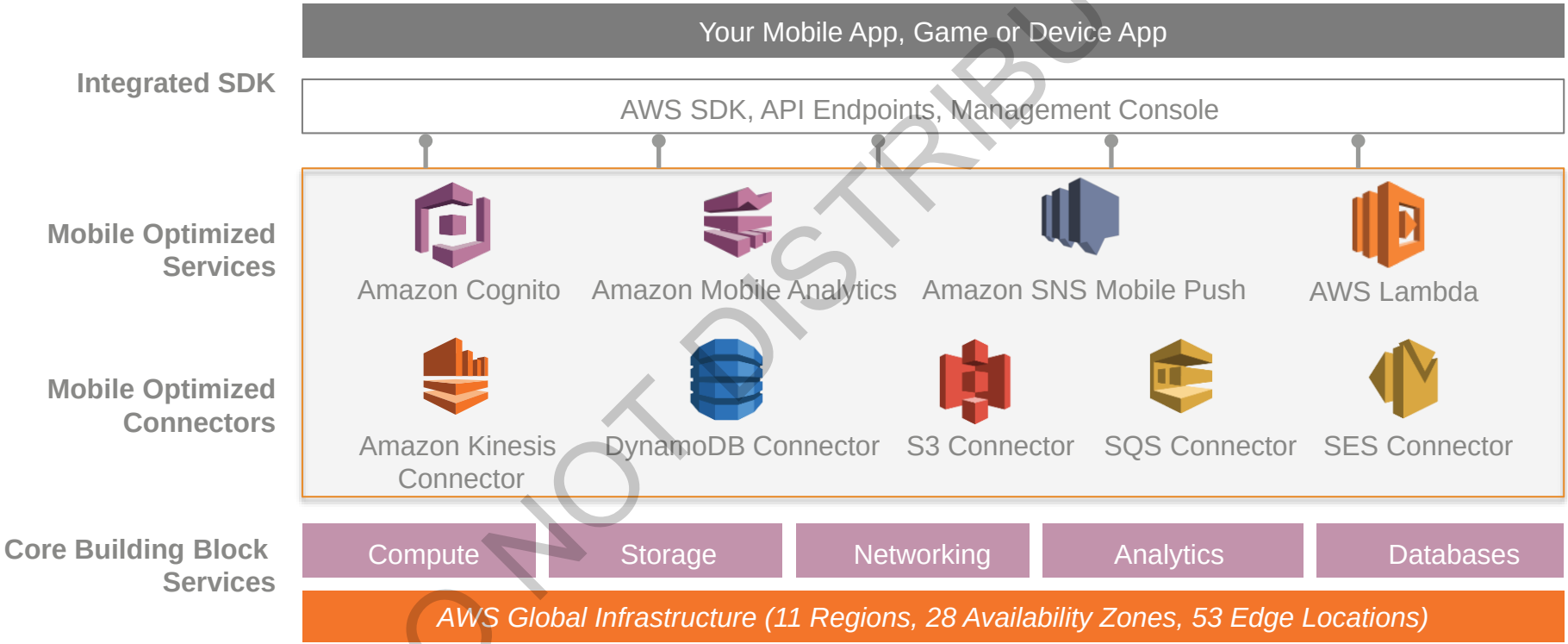
Store shared data

Store and query fast NoSQL data across users and devices

Stream real-time data

Collect real-time clickstream logs and take actions quickly

AWS Services for Mobile & IoT



AWS Mobile SDKs

Fully-integrated AWS mobile SDK

Cross-platform, optimized for native mobile OS

Automatically handles intermittent and latent network

Reduced memory footprint

Common authentication method across all services

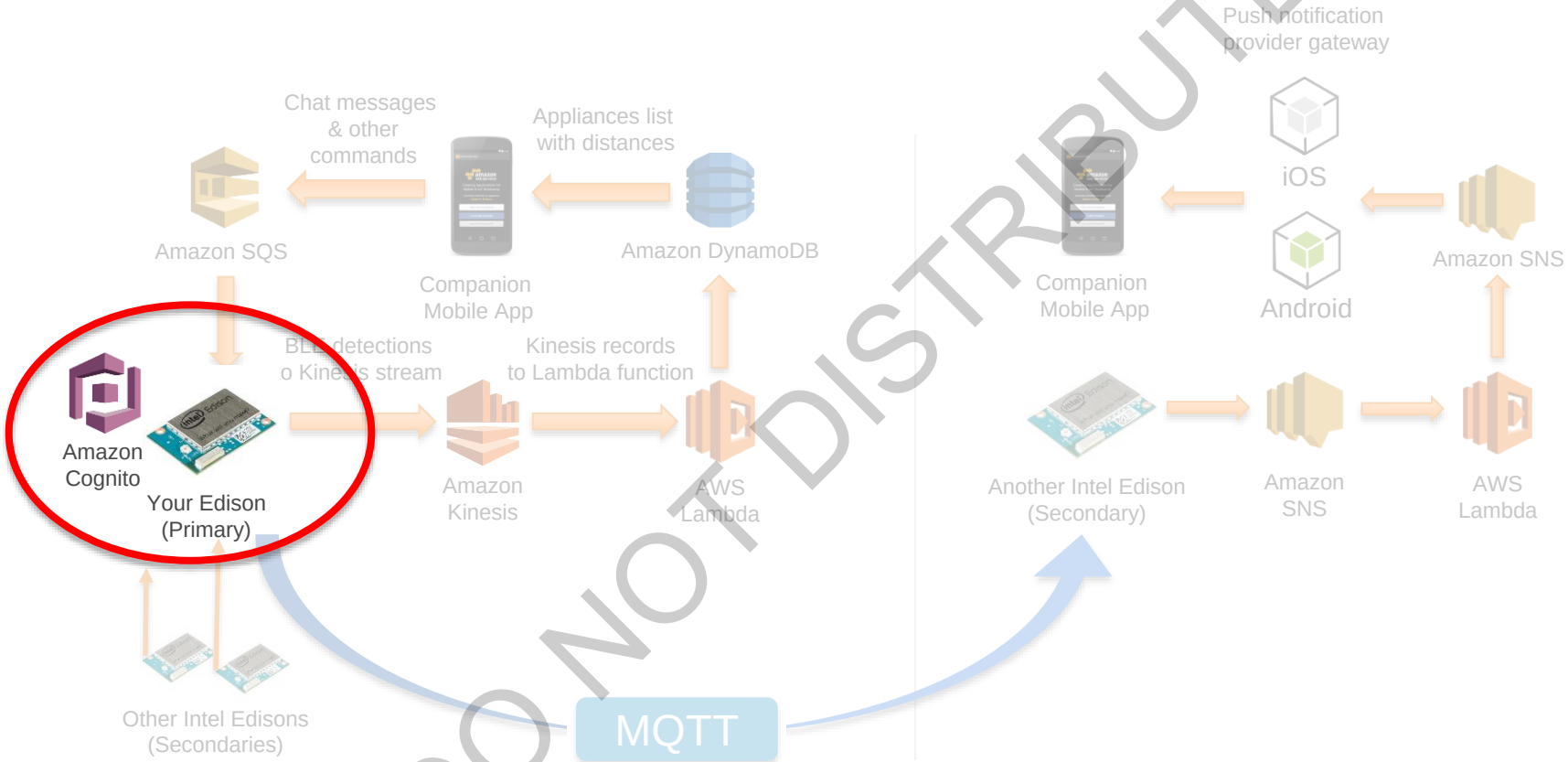
iOS

Android

Part 1

Authorization, Telemetry & Edison

Sign-posting

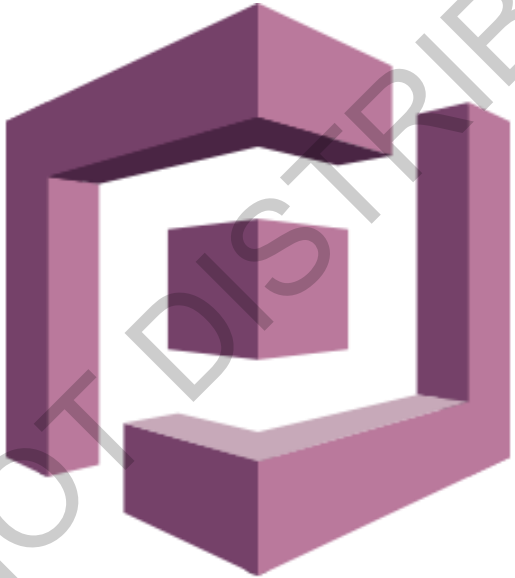


Amazon Cognito

Introduction to Amazon Cognito
& our first Lab

Amazon Cognito – two sides

Amazon Cognito
Identity Broker



Amazon Cognito
Sync Store

Authorization & Synchronization

Amazon Cognito - authorization

- Simplifies security - Integrated with IAM Roles
  Fully-integrated with IAM for security

- Supports multiple Login Providers
  Easily integrate with major login providers (FB, Google+, Login with Amazon) and supports Open ID Connect for authentication – or create your own Identity Provider

- Supports un-authenticated ‘guests’
  Your users can be authorized if you choose

Amazon Cognito – data synchronization

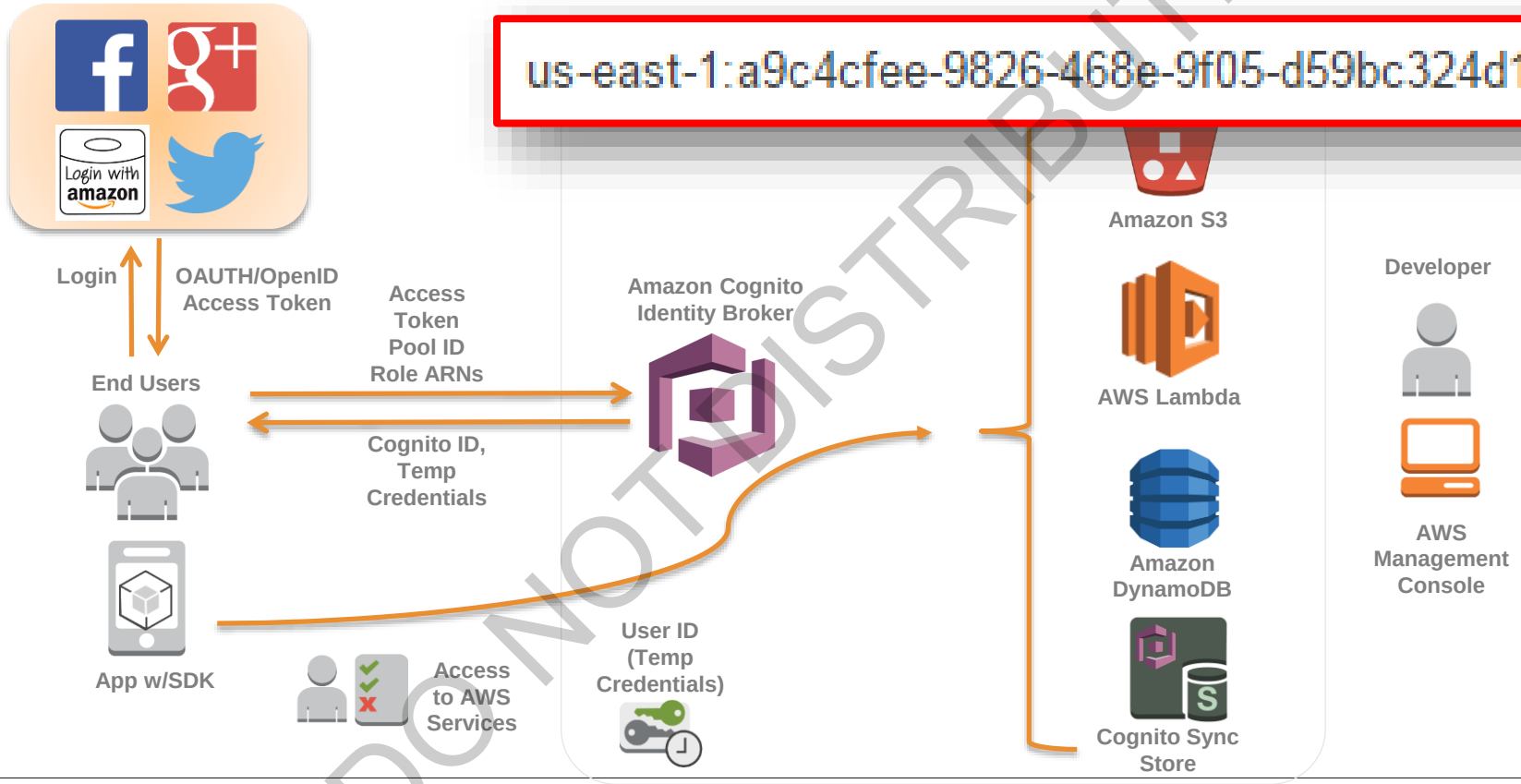
- **Cross-device and Cross-platform Push Sync**
  Synchronize user's data across devices and platforms

- **Amazon Cognito Streams feature**
  Automatically stream changes in user data to Amazon Kinesis for processing

- **Sync Trigger – Amazon Cognito Event**
  to intercept changes in Amazon Cognito dataset

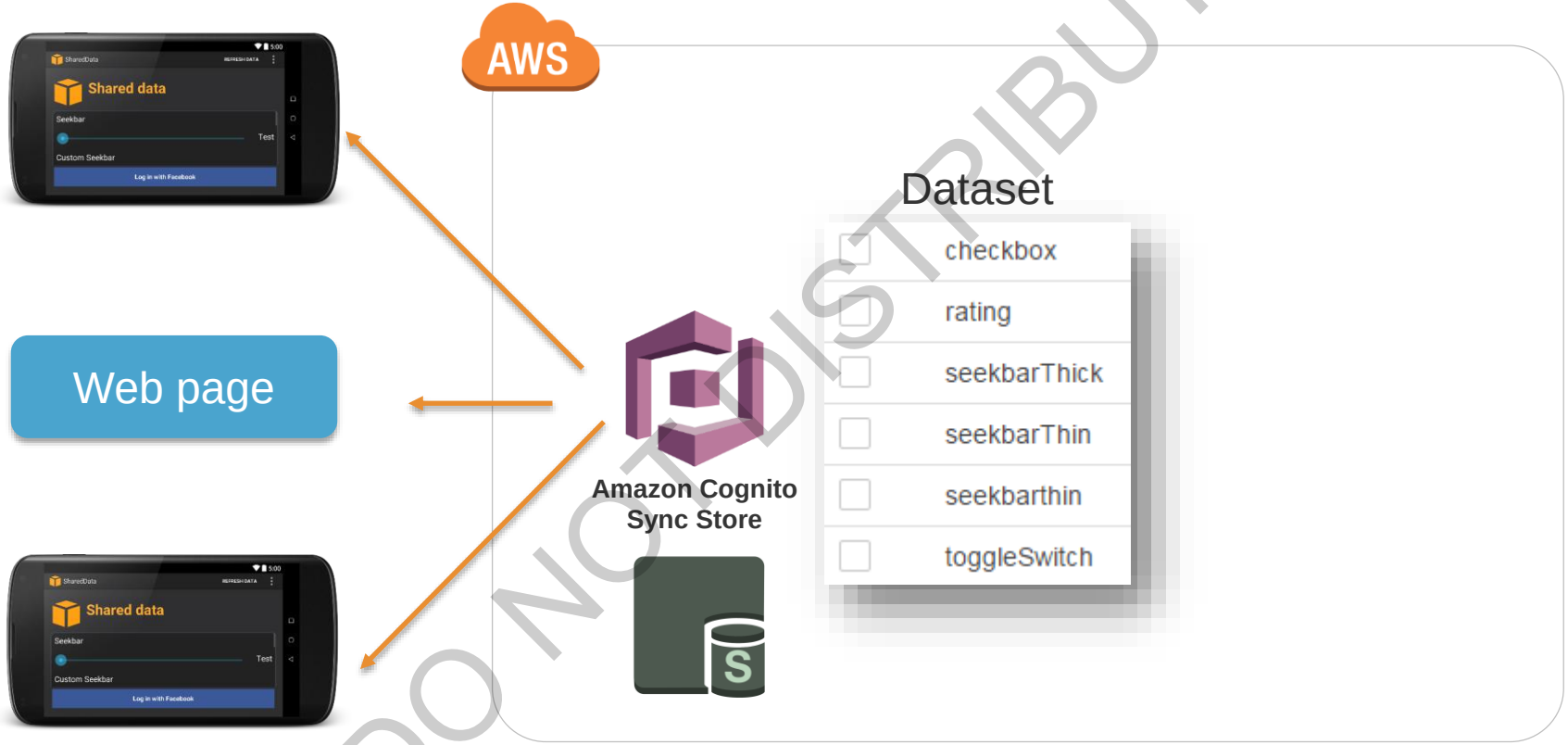
- **Programmatic access to sync store**
  Run back-end processes that modify in-app data

Amazon Cognito Security Architecture



Amazon Cognito – push sync

Amazon Cognito – push sync



Amazon Cognito – push sync

Amazon Cognito – push sync

Mobile App receives change

Web page

Mobile App initiates change

AWS

Amazon Cognito Sync Store

Dataset

- checkbox
- rating
- seekbarThick
- seekbarThin
- seekbarthin
- toggleSwitch

For our mobile app…

…and IoT device!

In our system we use Amazon Cognito to manage authorized access to all cloud resources, via AWS IAM Roles

Amazon Cognito

Accessing the Amazon Cognito Sync Store from a web page

Amazon Cognito from your web page

```
function setupCognito(oauthToken)
{
    var params =
    {
        AccountId       : bootcampConfig.AWS_ACCOUNT_ID,
        IdentityPoolId  : bootcampConfig.COGNITO_IDENTITY_POOL_ID
        RoleArn         : bootcampConfig.COGNITO_ROLE_AUTH;
        Logins          :
        {
            'graph.facebook.com': oauthToken
        }
    }

    AWS.config.region = bootcampConfig.COGNITO_REGION;
    AWS.config.credentials = new AWS.CognitoIdentityCredentials(params);

    AWS.config.credentials.get(function(err)
    {
        if (!err)
        {
            var cognitoId           = AWS.config.credentials.identityId;
            var cognitoSyncClient   = new AWS.CognitoSyncManager();
```

Amazon Cognito from your web page

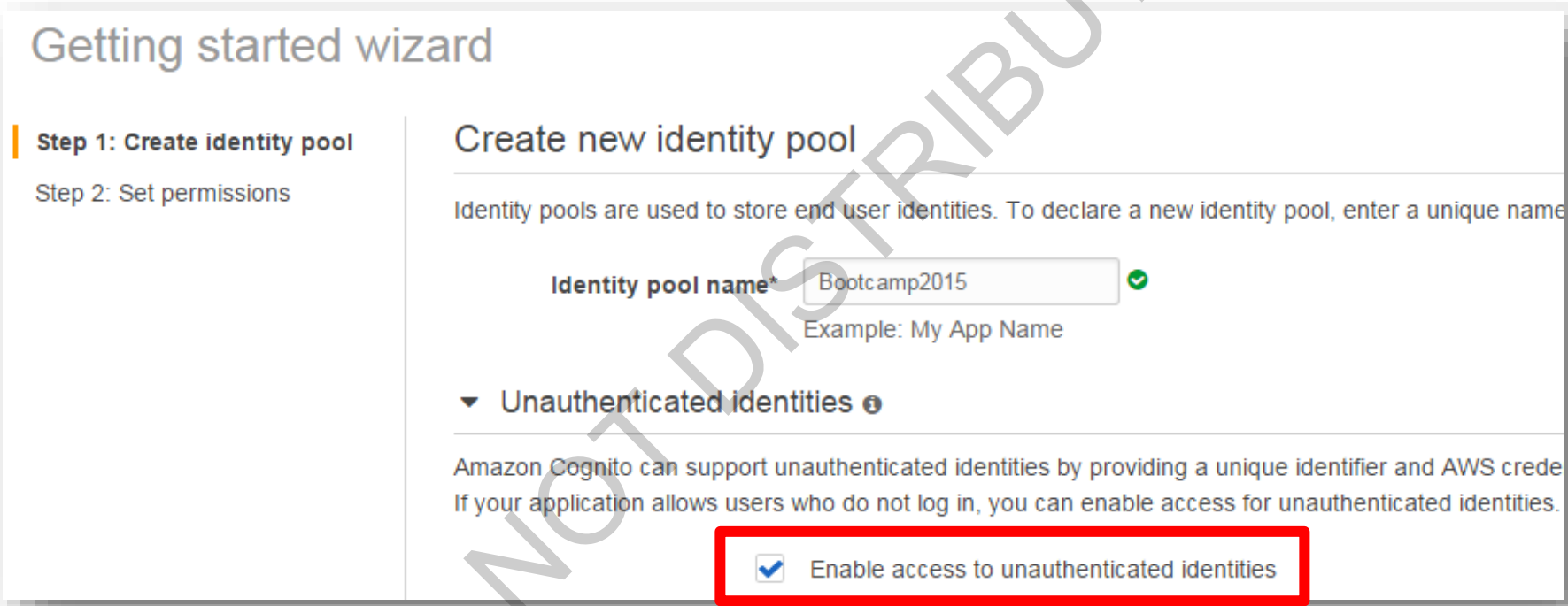
```
function setupCognito(oauthToken)
{
    var params =
    {
        AccountId       : bootcampConfig.AWS_ACCOUNT_ID,
        IdentityPoolId  : bootcampConfig.COGNITO_IDENTITY_POOL_ID
        RoleArn         : bootcampConfig.COGNITO_ROLE_AUTH;
        Logins          :
        {
            'graph.facebook.com': oauthToken
        }
    }

    AWS.config.region = bootcampConfig.COGNITO_REGION;
    AWS.config.credentials = new AWS.CognitoIdentityCredentials(params);

    AWS.config.credentials.get function(err)
    {
        if (!err)
        {
            var cognitoId           = AWS.config.credentials.identityId;
            var cognitoSyncClient   = new AWS.CognitoSyncManager();
```

```
        {
            'graph.facebook.com': oauthToken
        }
    }

    AWS.config.region = bootcampConfig.COGNITO_REGION;
    AWS.config.credentials = new AWS.CognitoIdentityCredentials(params);

    AWS.config.credentials.get(function(err)
    {
        if (!err)
        {
            var cognitoId           = AWS.config.credentials.identityId;
            var cognitoSyncClient   = new AWS.CognitoSyncManager();

            //
            // Sync data
            //
            openCognitoDataset();
        }
    });
}
```

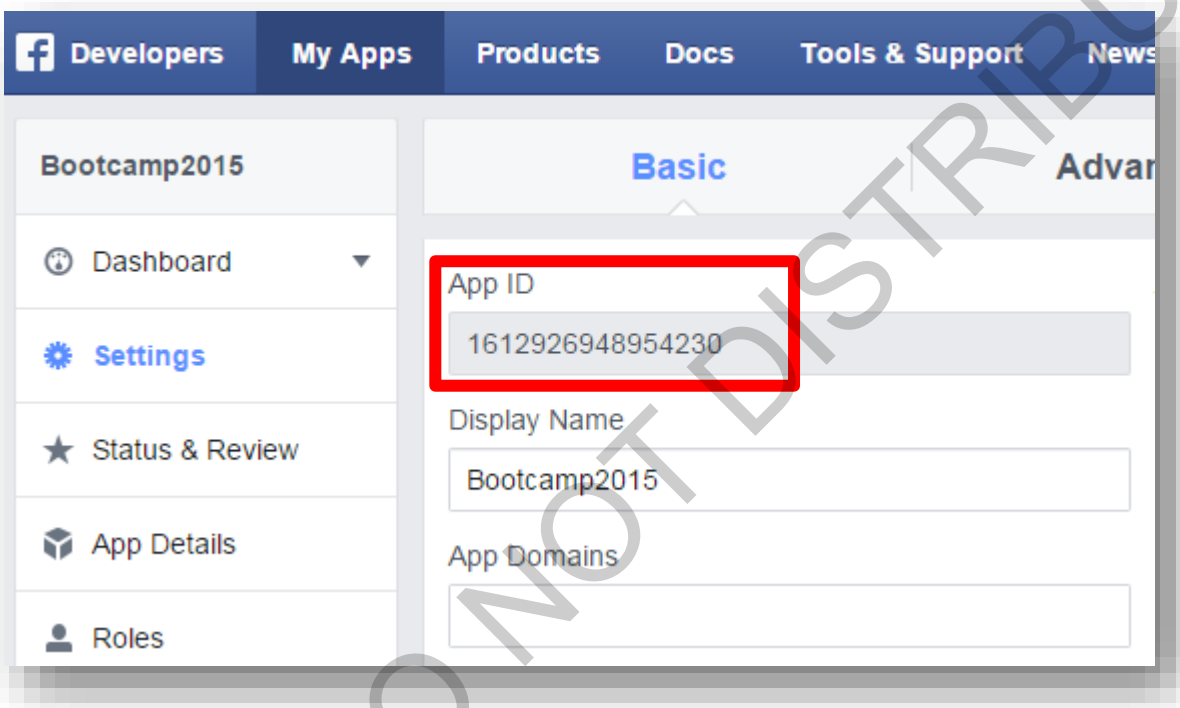
Amazon Cognito

Highlights of
Amazon Cognito Identity Pool
setup Lab

Create a new Amazon Cognito Identity Pool

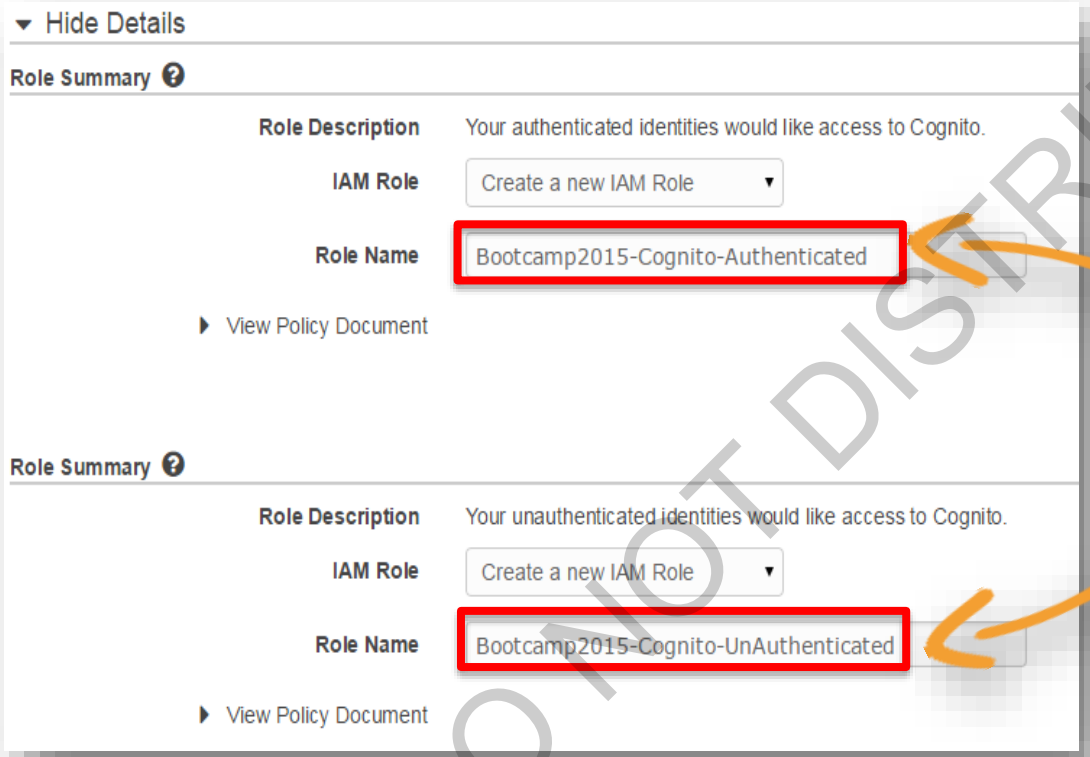


Create a new Facebook Application



Assign the Facebook Id to the identity pool

Create new AWS IAM Roles

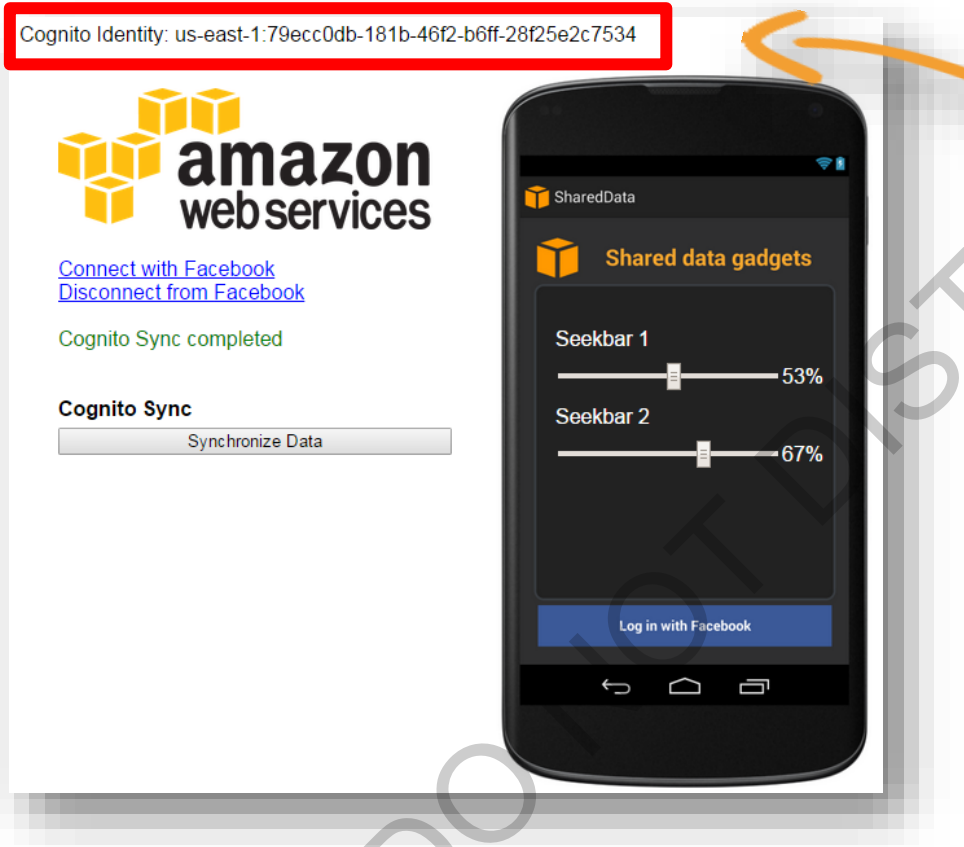


IAM Roles

Just use the default privileges assigned by the IAM Role Creation Wizard when you create these roles

But rename the roles as mentioned in the lab guide!

Test Amazon Cognito in your web browser



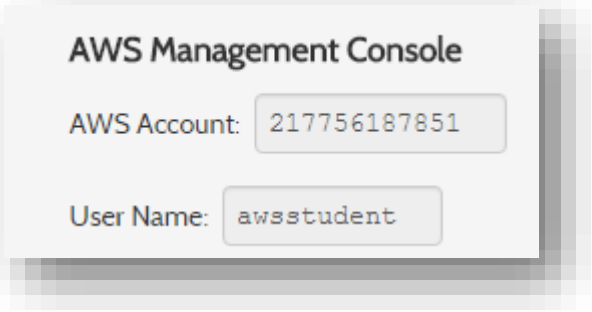
Your Amazon Cognito Identity Id

In this application, we show the Amazon Cognito Identity Id in the web page to make it easier for you to make a manual change to the shared data later in the Lab

Edit the configuration file

```
//////////////////////////////////////////
//
// Replace the "XXXXXXXXXXXX" in the
// line below, to be your Qwiklabs Account Id
// without any dashes
//
// It will look something like this "217756187851"
//
//////////////////////////////////////////

this.AWS_ACCOUNT_ID              = "XXXXXXXXXXXX";

//////////////////////////////////////////
//
// Replace the "XXXXXXXXXXXX" in the
// line below, to be the Cognito Identity pool Id
// exactly as it is shown in the AWS Console
//
// It will look something like this "us-east-1:223e168c-5ddb
//
//////////////////////////////////////////

this.COGNITO_IDENTITY_POOL_ID    = "XXXXXXXXXXXX";
```

Replace this with your AWS Account Id you are using in Qwiklabs

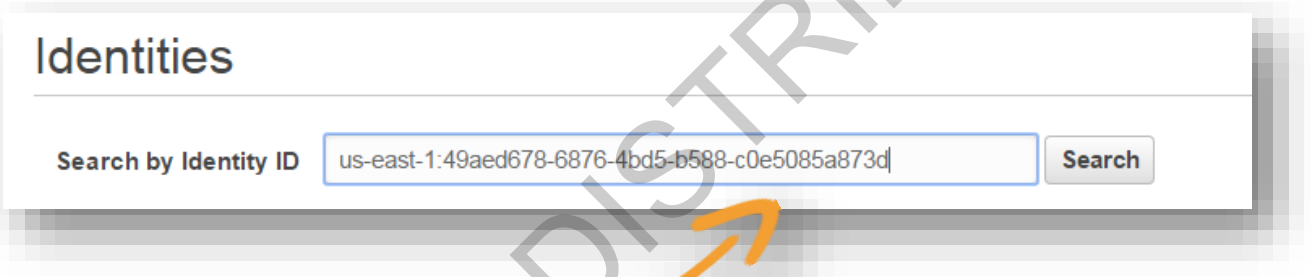


Edit the inline configuration

```
////////////////////////////////////////////
//
// Replace the "XXXXXXXXXXXX" in the
// line below, to be your Qwiklabs Account Id
// without any dashes
//
// It will look something like this "217756187851"
//
////////////////////////////////////////////

this.AWS_ACCOUNT_ID              = "XXXXXXXXXXXX";

////////////////////////////////////////////
//
// Replace the "XXXXXXXXXXXX" in the
// line below, to be the Cognito Identity pool Id
// exactly as it is shown in the AWS Console
//
// It will look something like this "us-east-1:223e168c-5ddb
//
////////////////////////////////////////////

this.COGNITO_IDENTITY_POOL_ID    = "XXXXXXXXXXXX";
```

Replace this with the Amazon Cognito Identity Pool's ID – you cut and paste that from the AWS console after you have created the Identity Pool

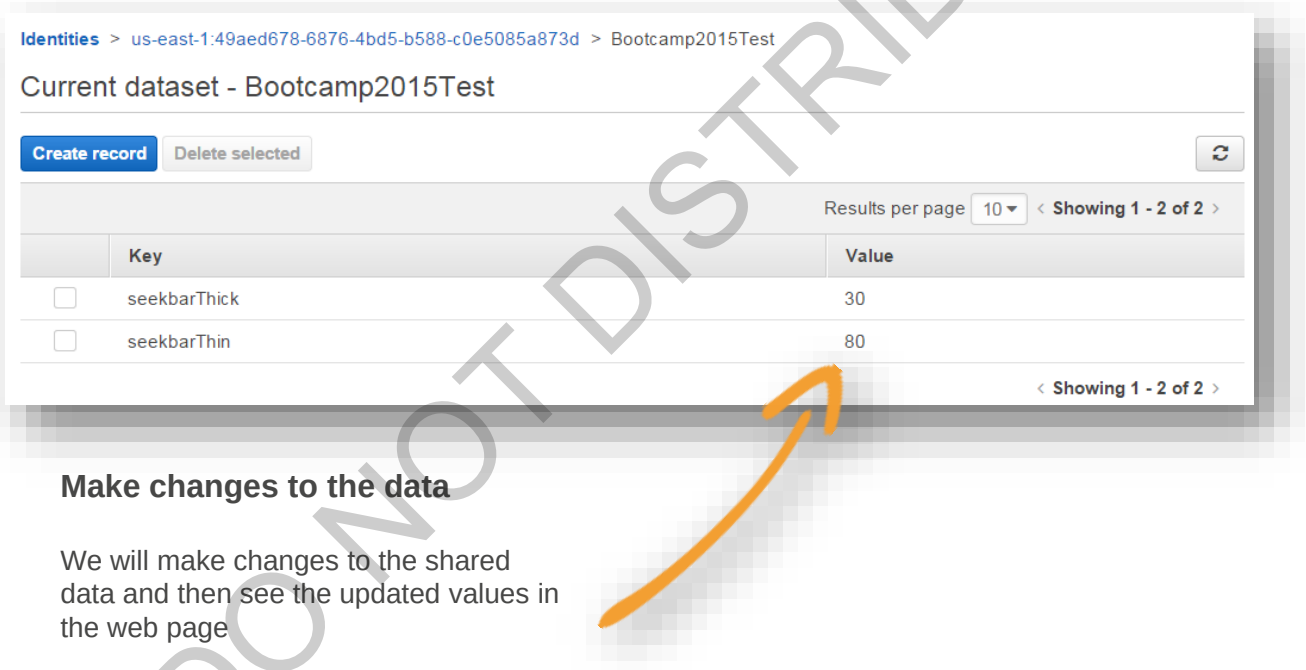
Modify shared data



Find the identity in the Amazon Cognito Sync Store

Using the Amazon Cognito Identity Id, we will find the shared data for this identity and modify it

Modify shared data



Make changes to the data

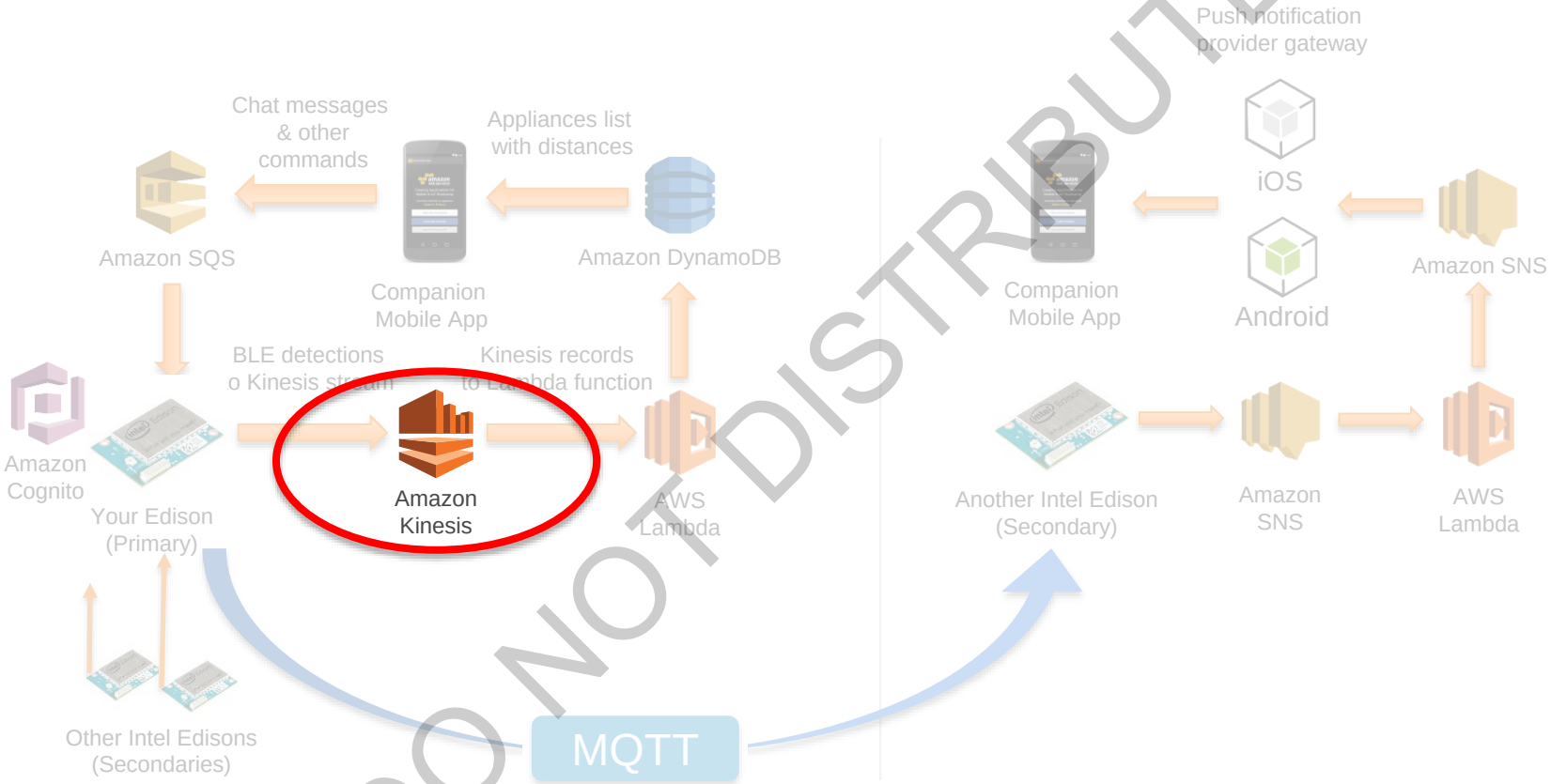
We will make changes to the shared data and then see the updated values in the web page

LAB 1

Set Up Amazon Cognito Identity Pool

Test Amazon Cognito Sync in the Browser

Sign-posting



Amazon Kinesis

Collect and process large streams of data records in real time

What is telemetry streaming?

- **Sensor networks**
  High-velocity and low-velocity telemetry data streams from sensors and devices (like our appliances!)

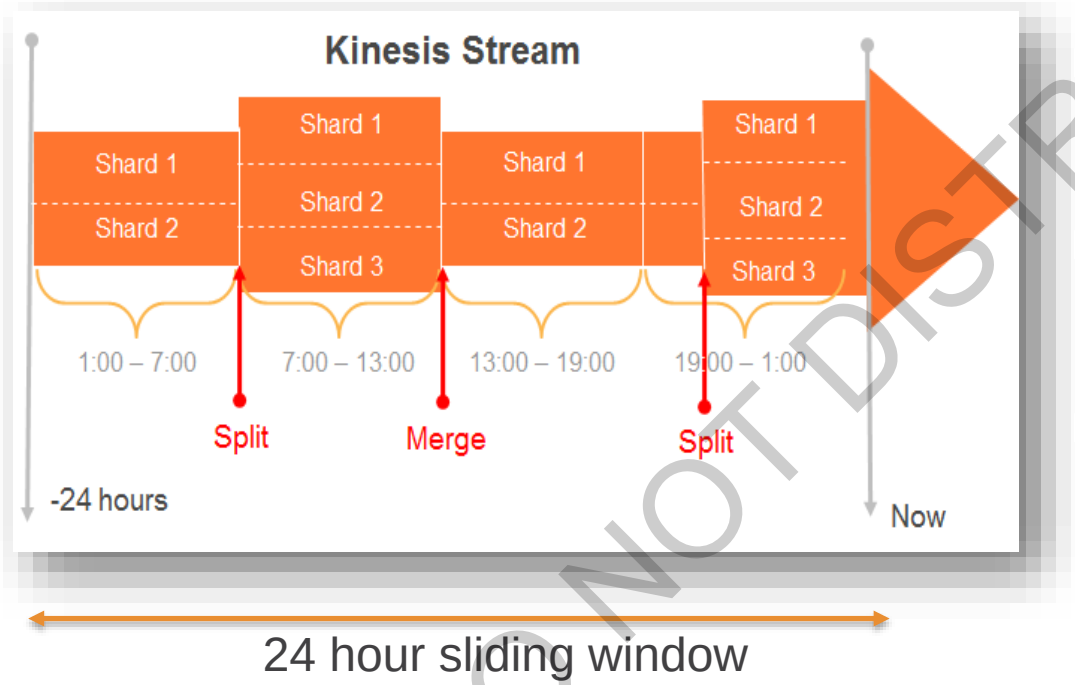
- **Ad network analytics & click streams**
  Tracking and monitoring ad views and user interaction in the digital space as they click around web pages

- **Log shipping and centralization**
  In-bound log data sent to a central store for analysis, alerting, etc

When to use stream processing

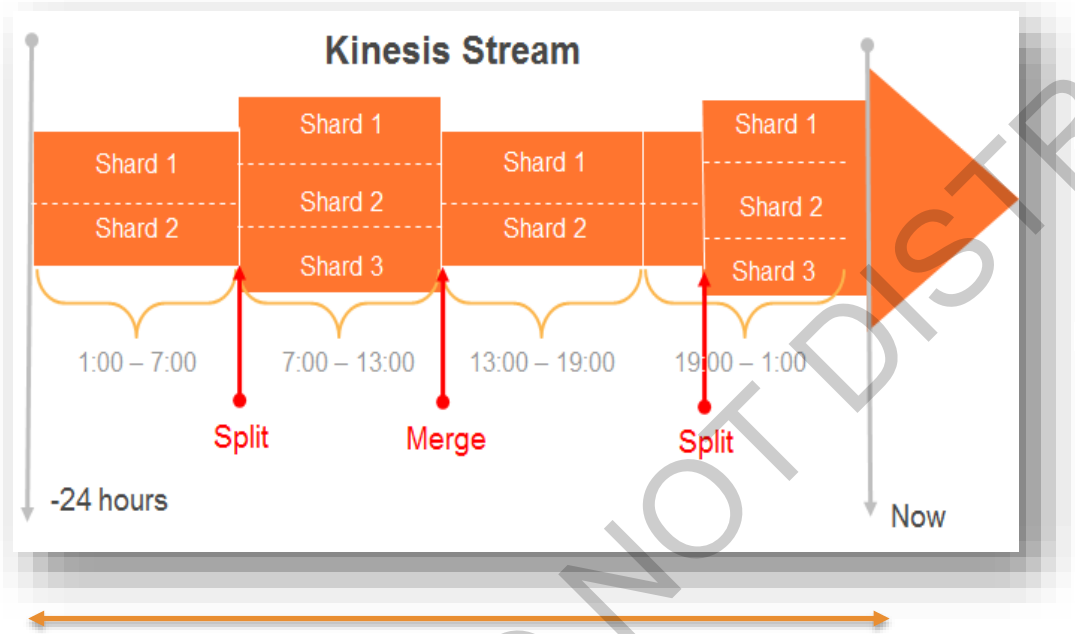
- You need ‘Real-time’ responses to telemetry data
  In-bound telemetry needs to be dealt with just-in-time or real-time
- Batch processing won’t suffice
  Your use case can’t afford to wait for batch processing done asynchronously
- Records can be thrown away after processing
  The records are just a means to an end – can be archived after processing has been completed

Amazon Kinesis - concepts



- **Streams** are made of **Shards**
- Each **shard** ingests < 1mbps of data and emits < 2mbps
- All captured data is stored for **24 hours**
- You can scale streams by splitting or merging Shards

Amazon Kinesis - concepts

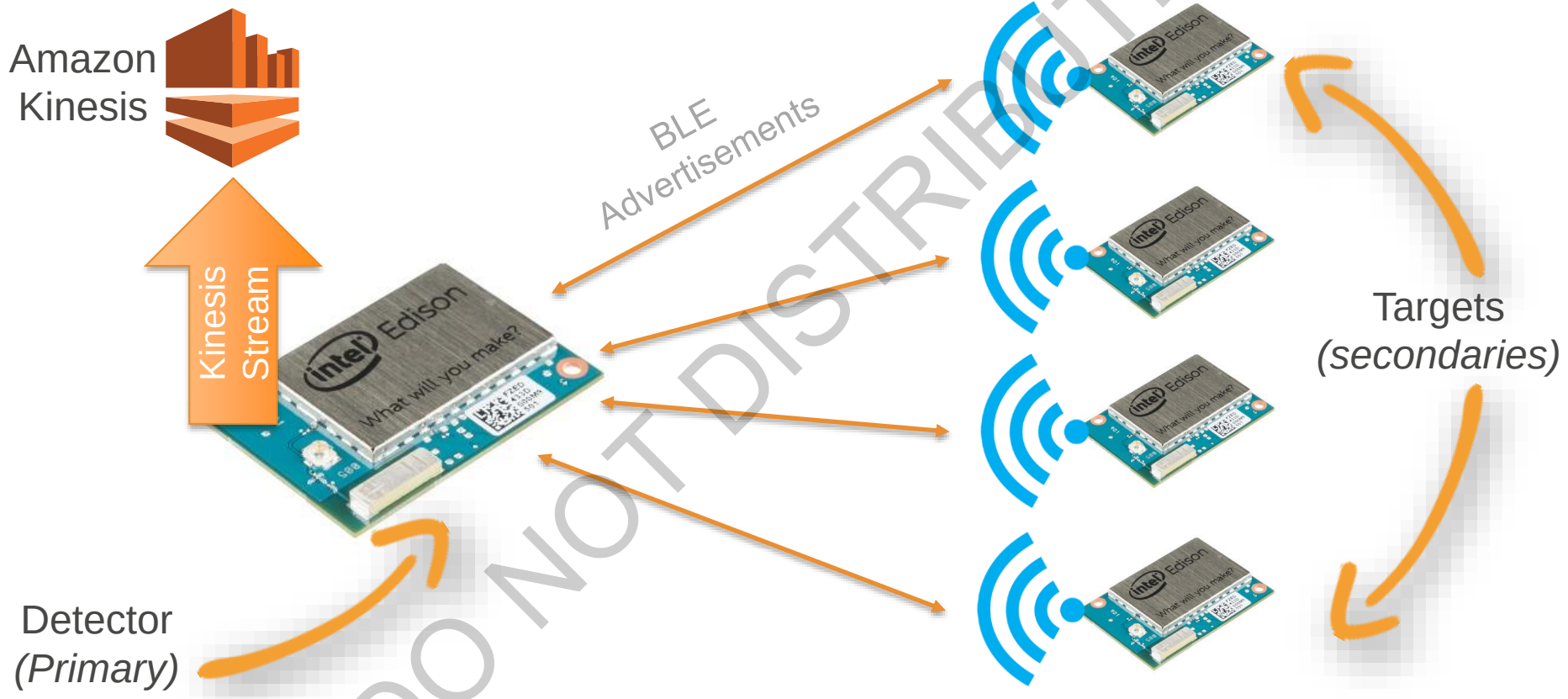


- **Producers** use a **PUT** call to store data in a **Stream**.
(Each record <= 50KB)
- A **Partition Key** is supplied by producer and used to **distribute** the PUTs across **Shards**
- You can have multiple **Amazon Kinesis consumers** operating on the stream at the same time – independently
- We will implement the **Amazon Kinesis consumers** as **Lambda functions**

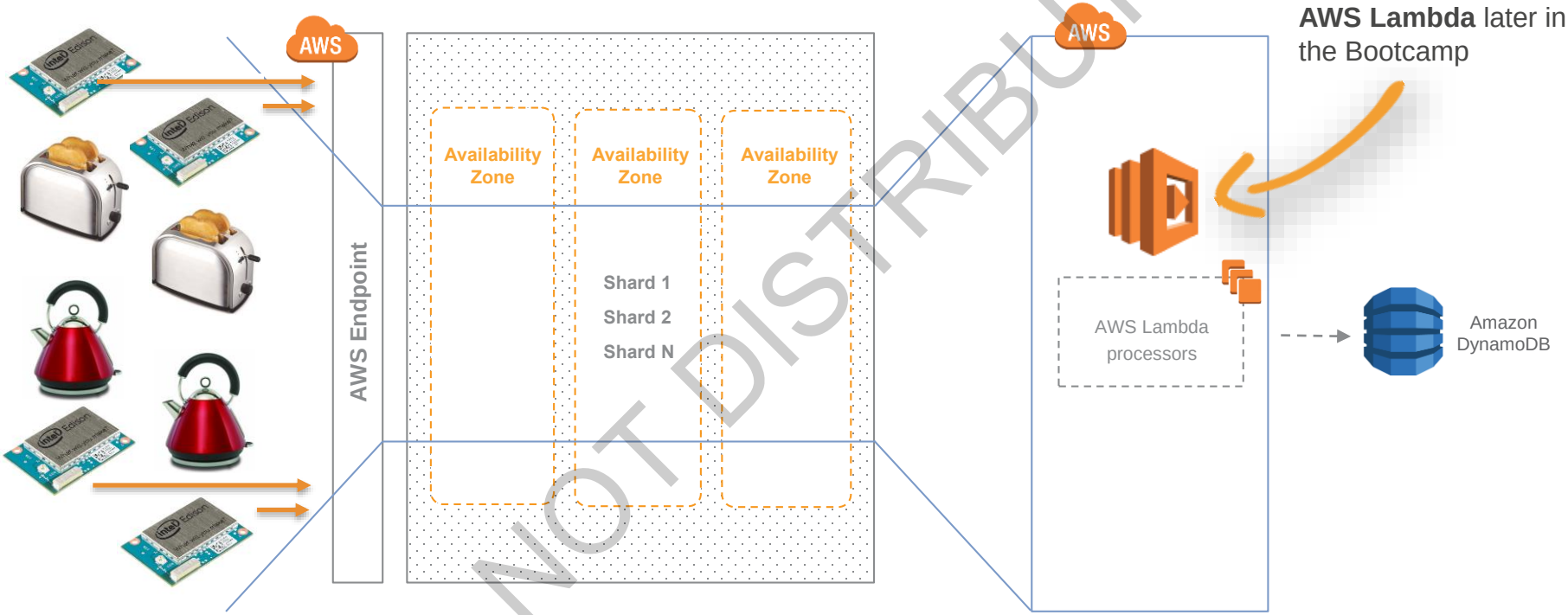
Amazon Kinesis features

- **Scale shards elastically**
  - Scale Up or Down without losing sequencing
  - Scale up to GB/sec (using multiple **shards**) without losing durability
- **Data accessible for 24 hours**
  - Workers/processors have access to data for 24 hours
- **Process high-velocity data in parallel**
  - Multiple producers, multiple shards, multiple independent consumers

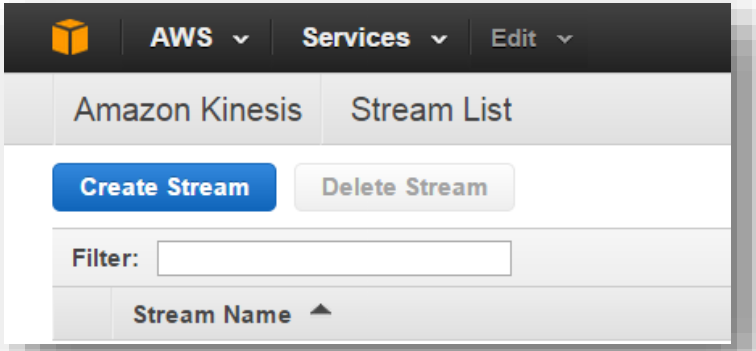
How Amazon Kinesis relates to our appliance network



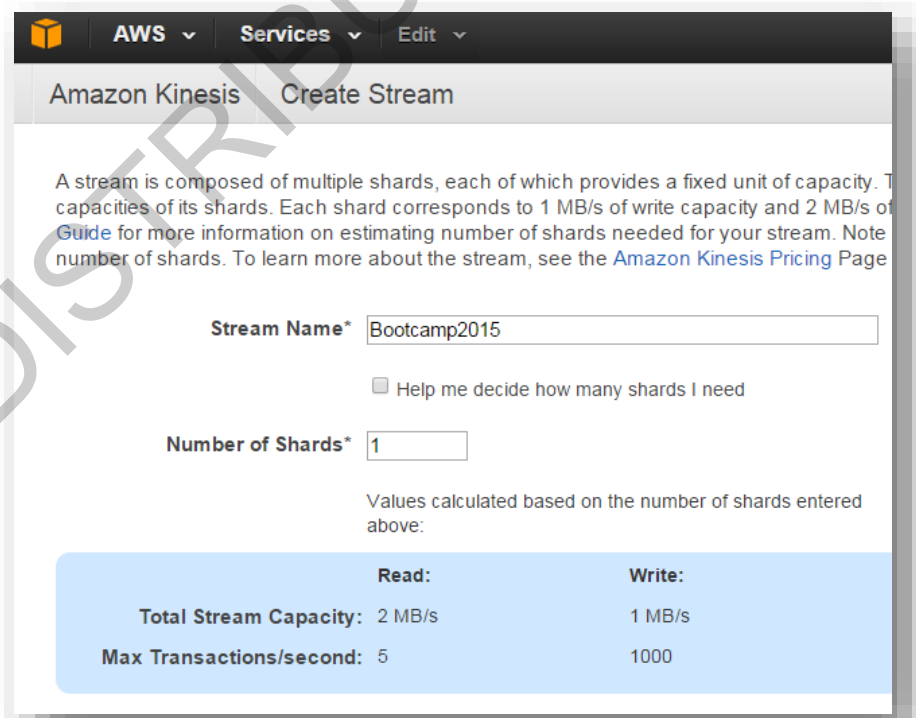
How Amazon Kinesis relates to our appliance network



Next Lab - Setting up a Kinesis stream



The Lab Guide will also update the IAM roles so that you can access Kinesis securely via Amazon Cognito



Using AWS CloudFormation

AWS
CloudFormation

```
{
    "AWSTemplateFormatVersion": "2010-09-09",

    "Parameters": ...,

    "Resources": {

        "Bootcamp2015CallLambdaInvokeFunction" :          ... AWS::IAM::Policy ...,

        "Bootcamp2015CallLambdaInvokeFunctionAuth" :      ... AWS::IAM::Policy ...,

        "Bootcamp2015SNSPublishFromEdison":               ... AWS::IAM::Policy ...,

        "Bootcamp2015SQSSendXAccount":                    ... AWS::IAM::Policy ...,

        "Bootcamp2015SQSToEdison":                        ... AWS::IAM::Policy ...,

        "Bootcamp2015KinesisPutRecords":                  ... AWS::IAM::Policy ...,

        "Bootcamp2015LambdaAccessKinesis":                ... AWS::IAM::Policy ...,

        "Bootcamp2015LambdaToMobile":                     ... AWS::IAM::Policy ...,
```

Using AWS CloudFormation

AWS
CloudFormation

```
"Bootcamp2015CallLambdaInvokeFunction" :        {

    "Type": "AWS::IAM::Policy",
    "Properties": {
        "Roles": [
            { "Ref": "CognitoUnAuthenticatedRole" }
        ],
        "PolicyName": "Bootcamp2015-Call-Lambda-InvokeFunction",
        "PolicyDocument": {
            "Version": "2012-10-17",
            "Statement": [
                {
                    "Effect": "Allow",
                    "Action": [
                        "lambda:InvokeFunction"
                    ],
```

Lab 2: Creating a stream & updating IAM roles

- Create stream
  Use the console to create a stream with 1 shard

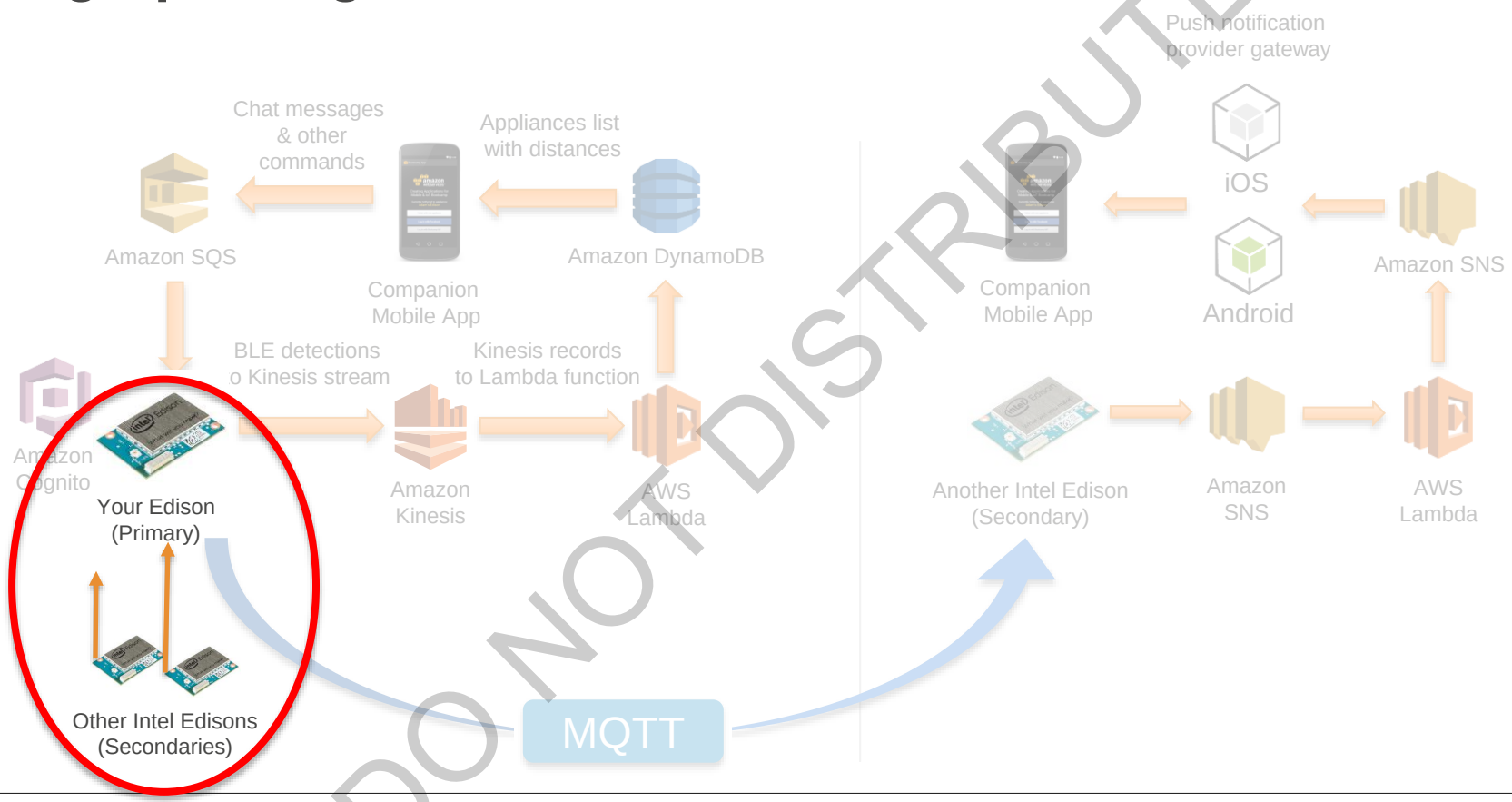
- Create a new role for use later by AWS Lambda
  We create the role manually and set managed policies

- Use AWS CloudFormation to automate update of IAM roles
  We demonstrate the use of AWS CloudFormation to automate setting all the policies and permissions we need for our Bootcamp

LAB 2

Create Amazon Kinesis stream &
set up all IAM policies for the
Bootcamp using AWS CloudFormation

Sign-posting



Intel Edison

Setting up the Intel Edison

developer kit

A fast, flexible and scalable path to commercial IoT Solutions

Come visit us at booth #1000 and #751 to see Intel's IoT solutions in action

intel®

Recap: the Intel Edison

- Dual-core 500 MHz Atom CPU
- 4 GB storage
- 1GB RAM
- Yocto Linux
- WiFi 802.11 a/b/g/n
- Bluetooth 4.0

Recap: the Intel Edison

All our ‘appliance’ code will run on the Intel Edison

Code is implemented in NodeJS

Plugging in the hardware

Snap the Intel Edison SoC onto the breakout board

Line up the posts

Be gentle!

Plugging in the hardware

This is the USB connector for SERIAL

Connect this to a spare USB slot on your laptop

Plugging in the hardware

This is the USB connector for POWER

Connect this to another spare USB slot on your laptop

Our next Lab: Setup Edison & Bootcamp configuration

- Set the name, root password & connect Edison to WiFi

  Use the serial interface to gain first access to the device. Once you have connected to WiFi, you can use SSH to connect

- Download & Unpack the ***Edison.tar*** bundle
  - The Edison.tar bundle contains several system and application files
  - You move the files into the relevant locations
  - Build/install Node Package dependencies
  - You will edit the ***bootcamp-config.js*** file

Our next Lab: Setup Edison & bootcamp configuration

- Initial config of the ***'bootcamp-config.js'*** file

  The ***bootcamp-config.js*** file is the central configuration for your device and your entire domestic appliance system

  You will set these values in this Lab

  - **AWS Account Id** – from the Qwiklabs dashboard
  - **Cognito Identity Pool Id** – you created this in Lab 1
  - **Device Name –** give your device a name to recognize it from the list of all devices
  - **Device Type** – you can choose to be a kettle, toaster, coffee machine and more!
  - **Bluetooth BLE UUID** – the BLE 'channel'. Make sure you use the same UUID as your partner
  - **Mobile Push Provider** – either GCM for Android or APNS_SANDBOX for iOS

The *bootcamp-config.js* configuration file

```
////////////////////////////////////////////////////////////////////////////////
//
// Configure these values
//
////////////////////////////////////////////////////////////////////////////////

var DEVICE_NAME                 = "Mini Edison";
var AWS_ACCOUNT_ID              = "0000000000";
var COGNITO_IDENTITY_POOL_ID    = "us-east-1:XXXXXXXXXXXX";
var MOBILE_ANALYTICS_APP_ID     = "0000000000";
var API_GATEWAY_ENDPOINT_URL    = "https://REPLACE_THIS_WITH_API_GATEWAY_URL_FROM_LAB_GUIDE";
var APPLIANCE_TYPE              = 5;
var BLE_UUID                    = "badf00d5dffb48d2b060d0f5a71096e0";
var GOOGLE_PROJECT_NUMBER       = "0000000000";
var MOBILE_PUSH_PROVIDER        = "GCM"; // Set to GCM for Android or APNS for iOS mobile push
```

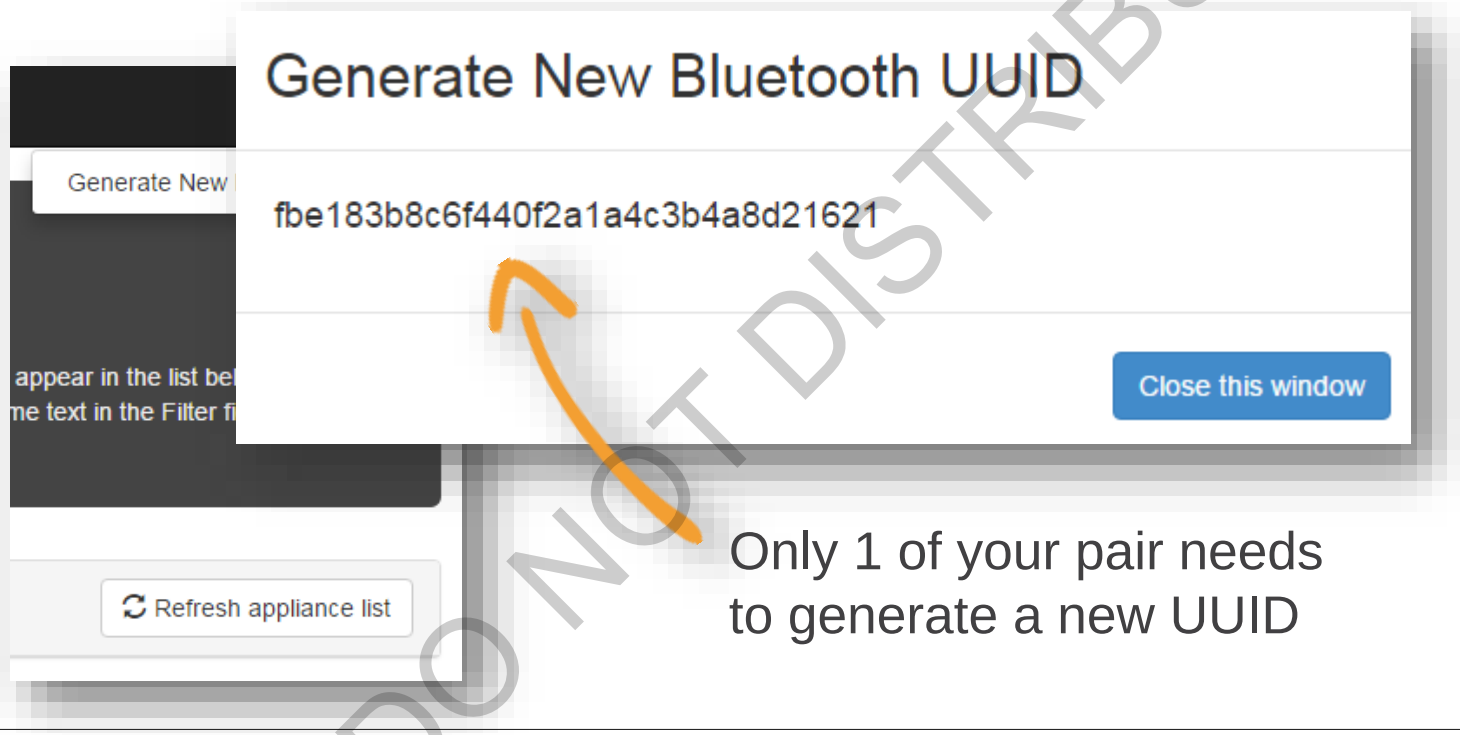
Path on your Intel Edison `/etc/bootcamp-config.js`

Appliance definitions

```
//
// CONSTS
//
define({
    APPLIANCE_TYPE_UNKNOWN   : 0,
    APPLIANCE_TYPE_TOASTER   : 1,
    APPLIANCE_TYPE_MICROWAVE : 2,
    APPLIANCE_TYPE_ESPRESSO  : 3,
    APPLIANCE_TYPE_BLENDER   : 4,
    APPLIANCE_TYPE_KETTLE    : 5,
});
```

Generating the UUID for BLE

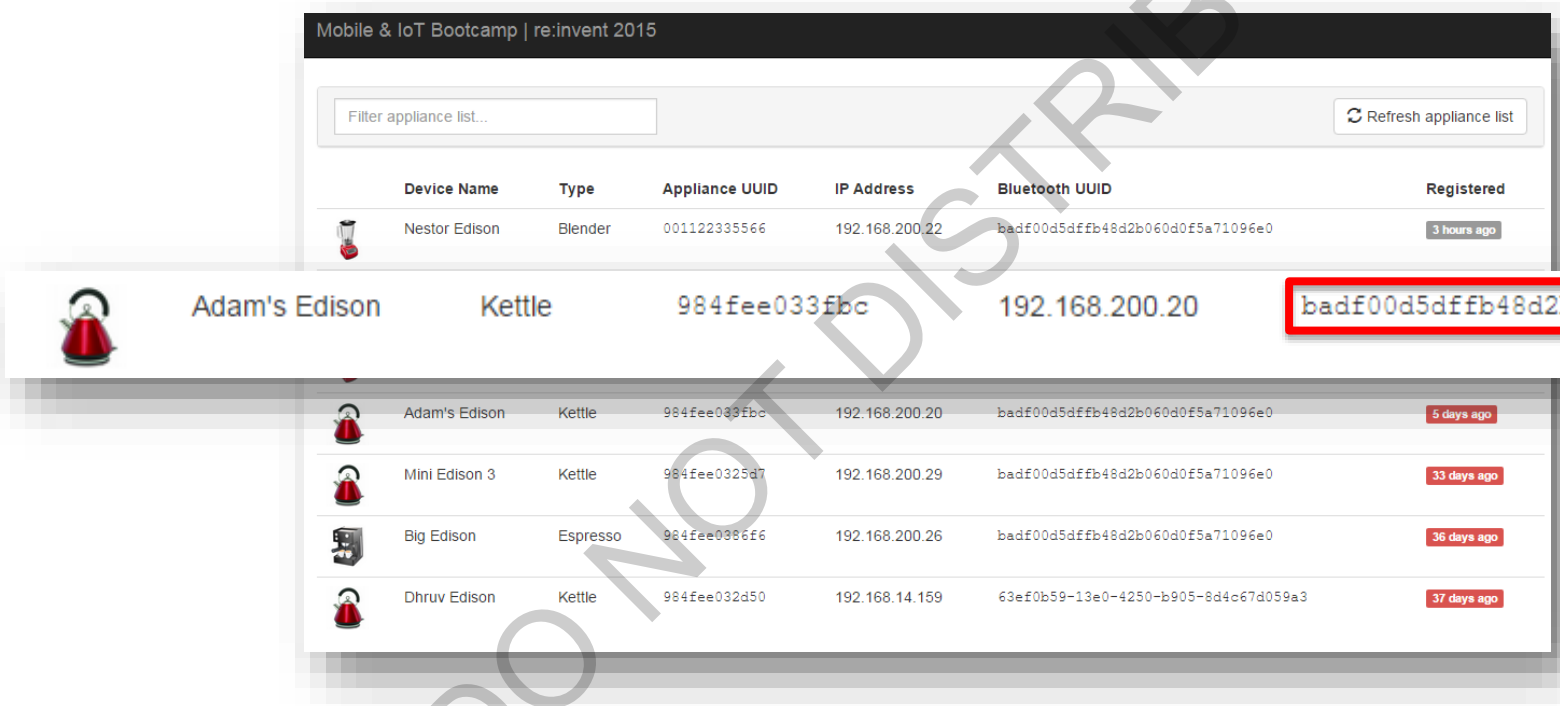
http://bit.ly/mobile-iot-bootcamp



Only 1 of your pair needs to generate a new UUID

Copying your partner's UUID for BLE

http://bit.ly/mobile-iot-bootcamp



We use *SSH* to remote onto the IoT device and use *vi* as a text editor

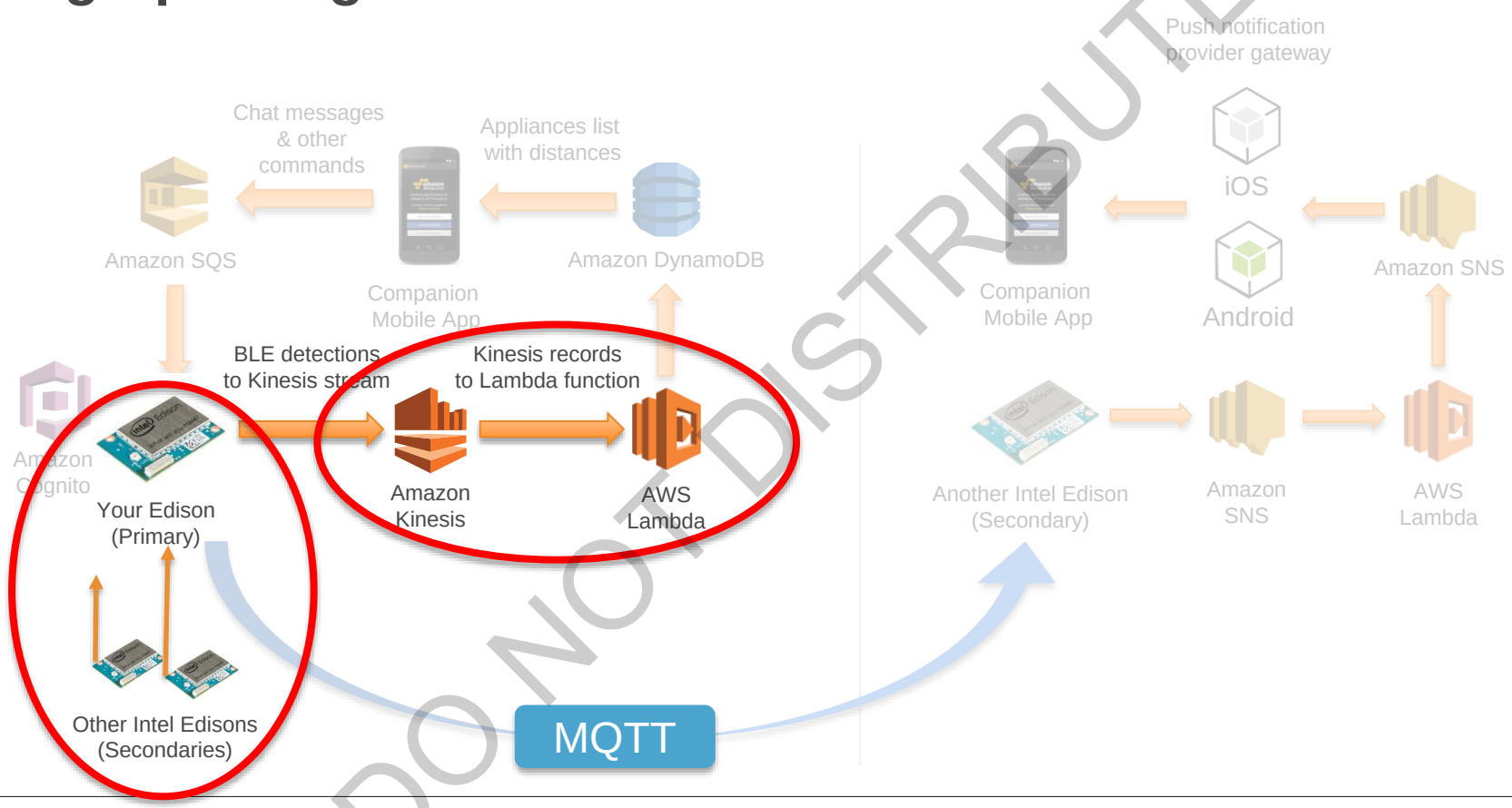
LAB 3

Configure your Intel Edison
to run our system

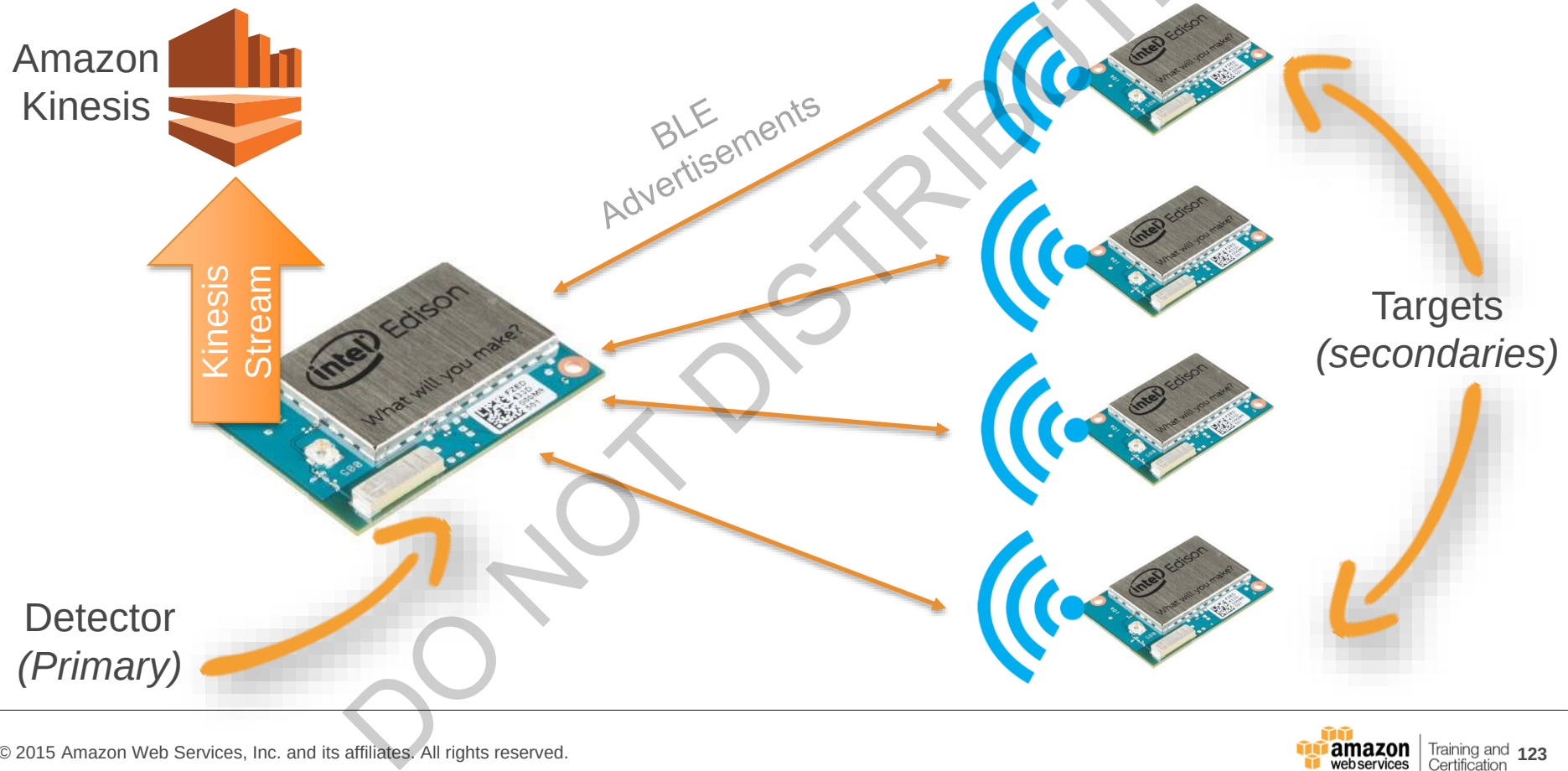
Part 2

Detecting appliances in range using Bluetooth Low Energy, Amazon Kinesis & AWS Lambda

Sign-posting



Detecting peers using Bluetooth LE



Bluetooth Low Energy

- Light-weight subset of classic Bluetooth
- Part of the Bluetooth 4.0 specification
- Advertisement packets sent at ~1Hz
- Technology behind iBeacon by Apple
- Key fields in advertisement
  - UUID – the ‘channel’ the device belongs to. A group of related devices.
  - RSSI – received signal strength indicator – measurement of the strength of the signal being received

***iBeacon spec** also has MAJOR & MINOR – 2-byte integers that can be used to identify an individual device in a group of related devices*

Bluetooth Low Energy - RSSI

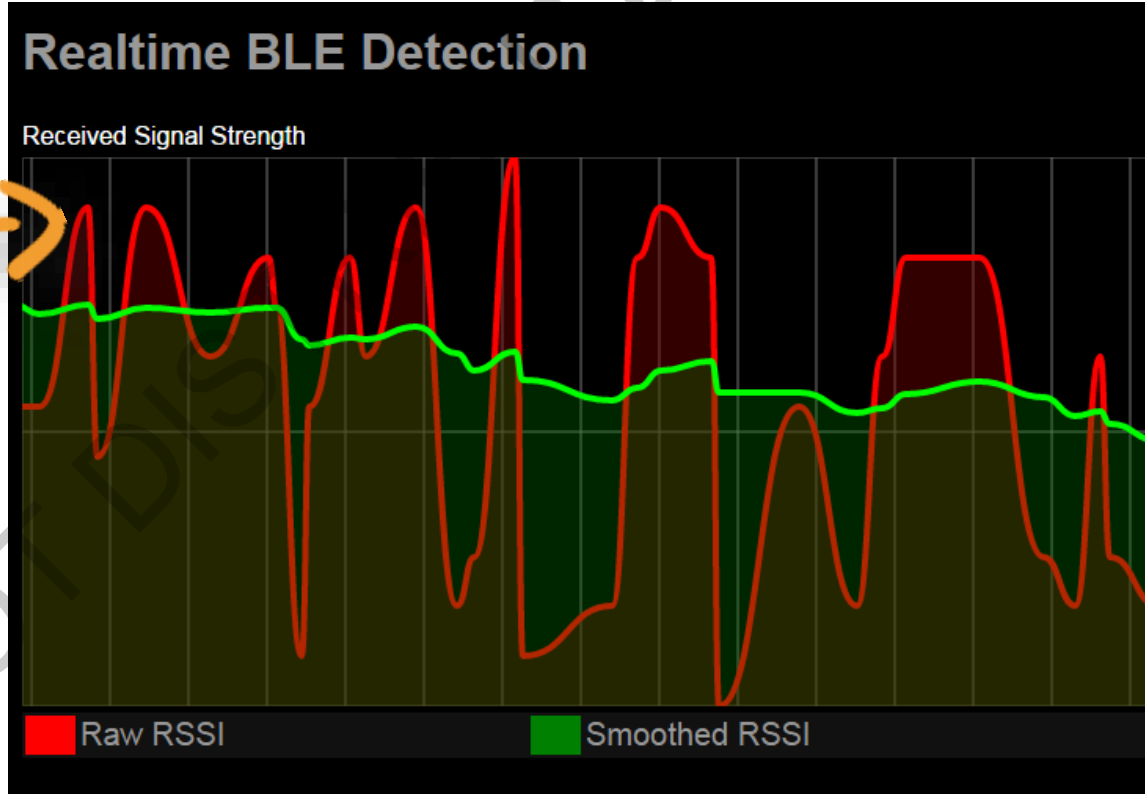
BLE Target

Kalman filtering, also known as linear quadratic estimation (LQE), is an algorithm that uses a series of measurements observed over time, containing noise (random variations) and other inaccuracies, and produces estimates of unknown variables that tend to be more precise than those based on a single measurement alone. Its purpose in our system is to smooth out the radio frequency noise and provide more accuracy in the distance measurements

BLE Detector → Kalman Filter → Kinesis Stream

Bluetooth Low Energy

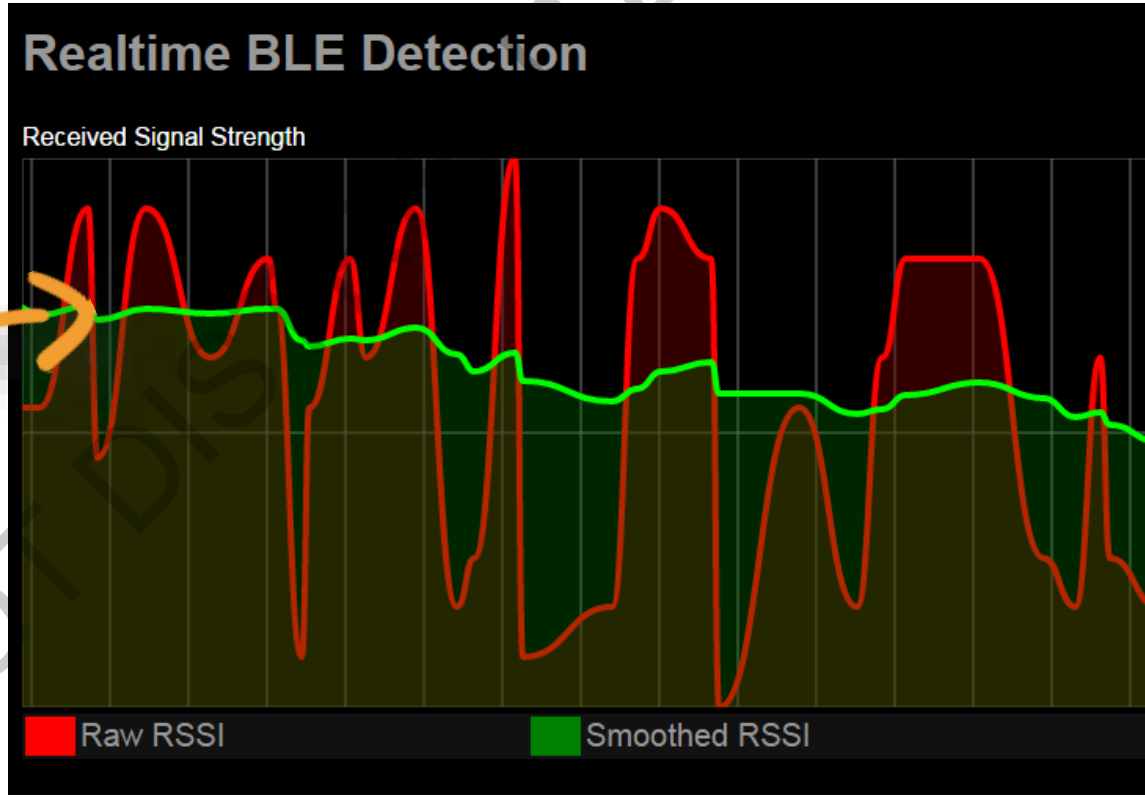
The red line is the ‘raw’ RSSI of the Bluetooth Low Energy advertisements as received by the Edison



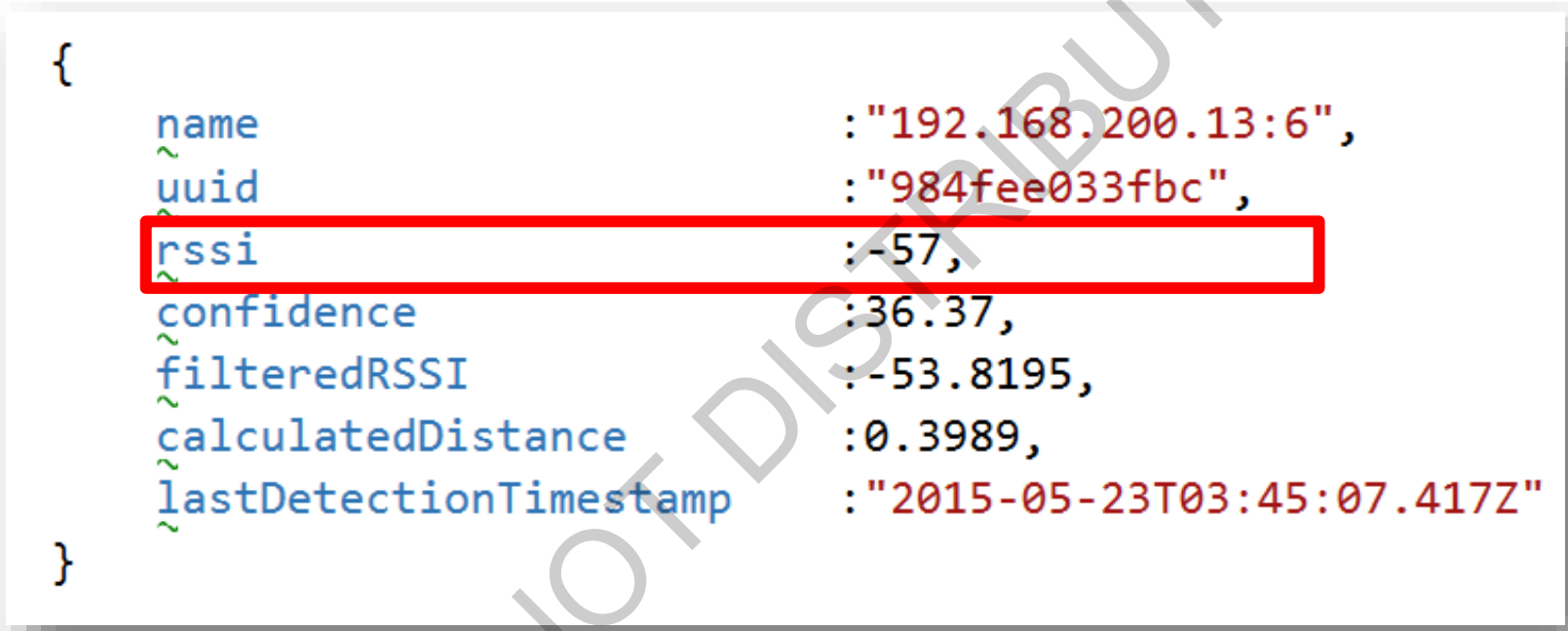
Bluetooth Low Energy

The green line is the same data run through the Kalman filter and then sent to Amazon Kinesis.

‘Noise’ in the signal has been minimized

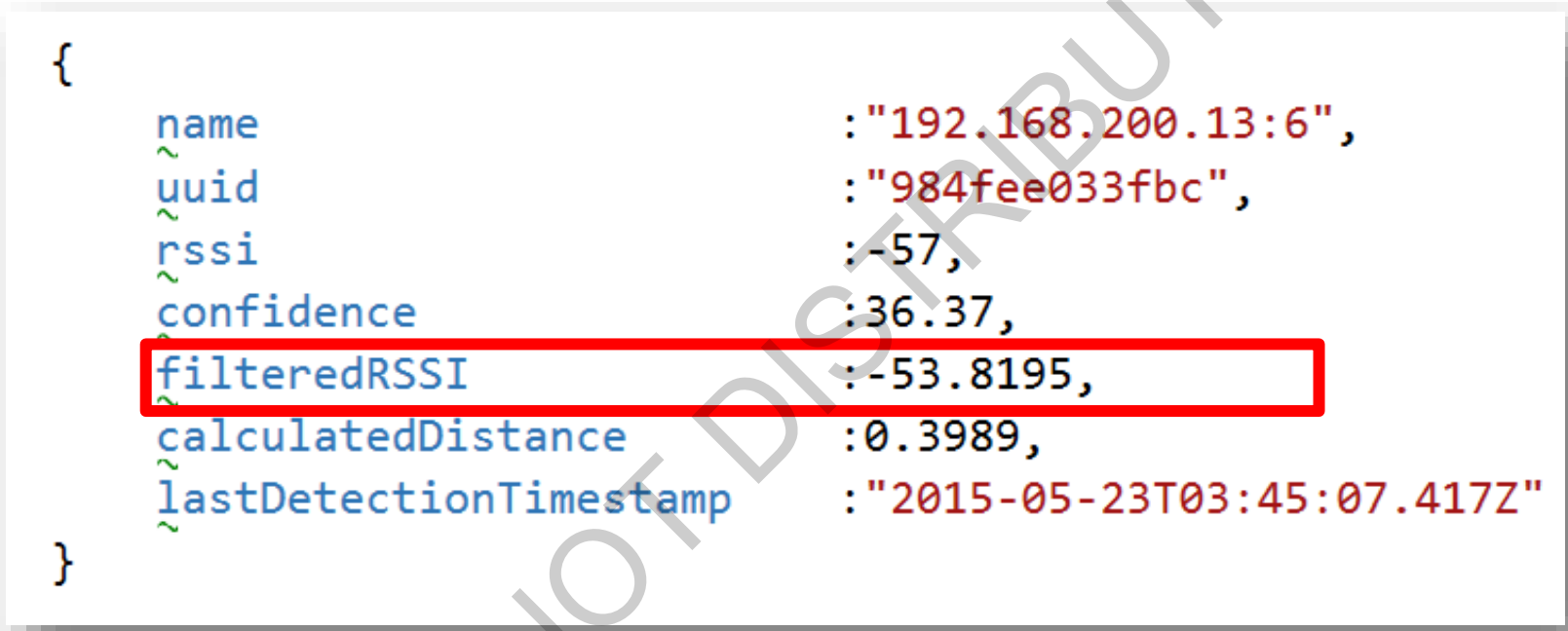


Bluetooth Low Energy



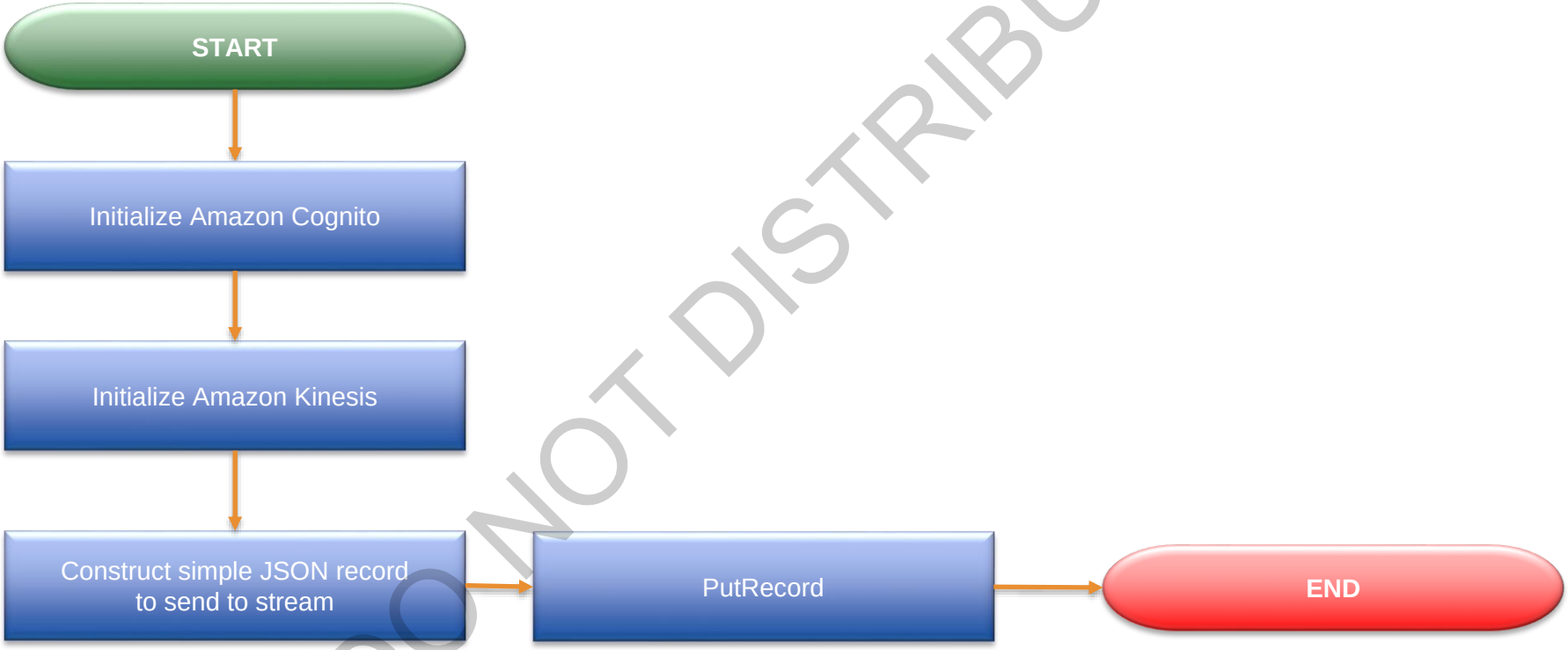
Raw Received Signal Strength as detected

Bluetooth Low Energy



Calculated RSSI after Kalman filtering

Sending a test record to Amazon Kinesis from NodeJS



Sending a record to Amazon Kinesis from NodeJS

```
function initialiseCognito()
{
    console.log("Initialising Cognito...");

    var params = {
        AccountId:          configuration.aws.AWS_ACCOUNT_ID,
        RoleArn:            configuration.aws.COGNITO_ROLE_UNAUTH,
        IdentityPoolId:     configuration.aws.COGNITO_IDENTITY_POOL_ID
    };

    // initialize the region for the SDK
    aws.config.region      = configuration.aws.COGNITO_REGION;

    // initialize the Credentials object
    aws.config.credentials = new aws.CognitoIdentityCredentials(params);

    // Get the credentials for our user
    aws.config.credentials.get(function(err)
    {
        if (err)
        {
            console.log("## credentials.get: " + err, err.stack); // an error occurred
        }
        else
        {
            console.log("Cognito IdentityId => " + aws.config.credentials.identityId);
```

Initialize Amazon Cognito

Sending a record to Amazon Kinesis from NodeJS

```
function sendRecordToKinesis()
{
    var partitionKey    = aws.config.credentials.identityId;
    var recordParams    =
    {
        Data:           "Hello Kinesis World!",
        PartitionKey:   partitionKey,
        StreamName:     configuration.aws.KINESIS_STREA
    };

    console.log(JSON.stringify(recordParams));

    kinesis.putRecord(recordParams, onKinesisPutRecord);
}
```

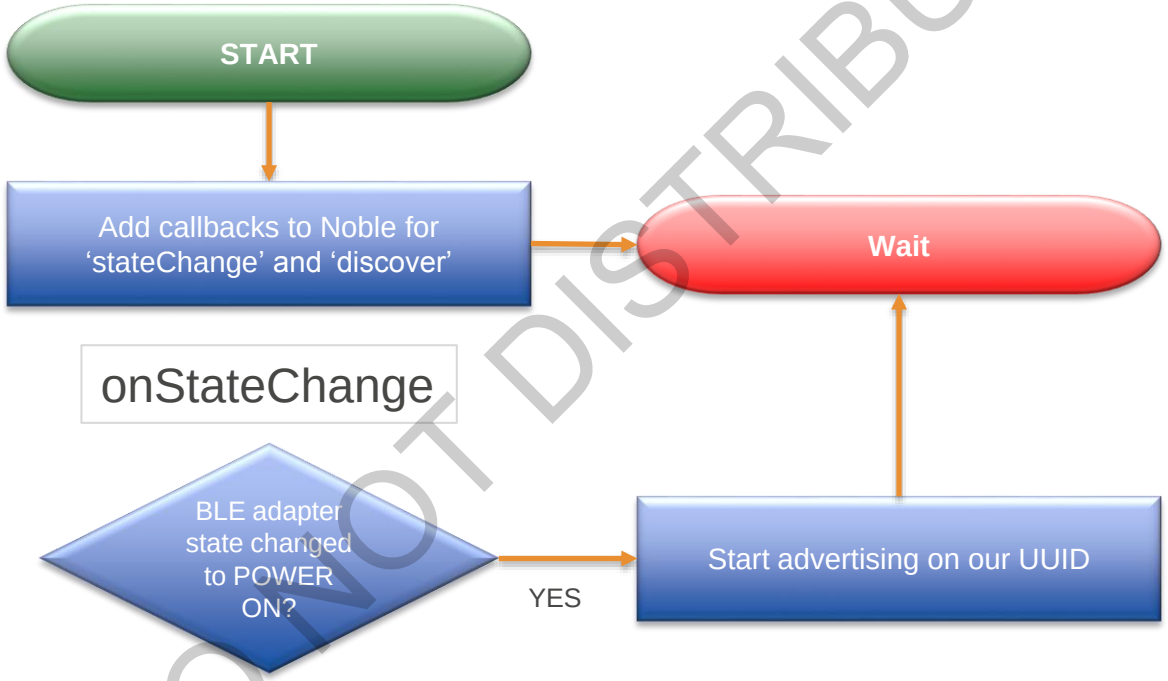
Construct simple record
& send to stream

Node Libraries for BLE

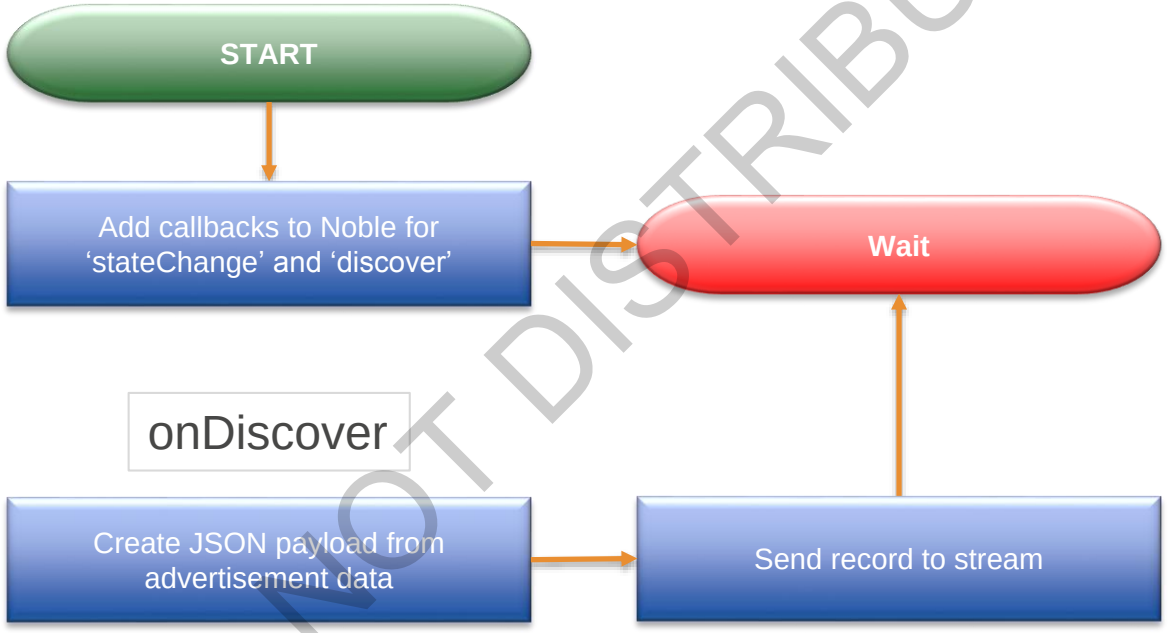
- **Bleno**
  - A node.js module for implementing BLE (Bluetooth low energy) peripherals
  - Allows us to easily advertise our appliance using BLE
  - Acts as a ‘server’ (Bluetooth peripheral)

- **Noble**
  - A node.js BLE central module
  - Allows us to easily detect other devices in range using BLE
  - Callbacks for state change in the Bluetooth adapter and when a new appliance (Bluetooth peripheral) is detected

How we ‘wire up’ BLE using Bleno and Noble in NodeJS



How we ‘wire up’ BLE using Bleno and Noble in NodeJS



main() – wire up callbacks

```
function main()
{
  noble.on('stateChange', onNobleStateChange);
  noble.on('discover',    onNobleDiscoverPeripheral);
  noble.on('warning',     onNobleWarning);

  setInterval(reportKinesisInFlight, 1000);
}
```

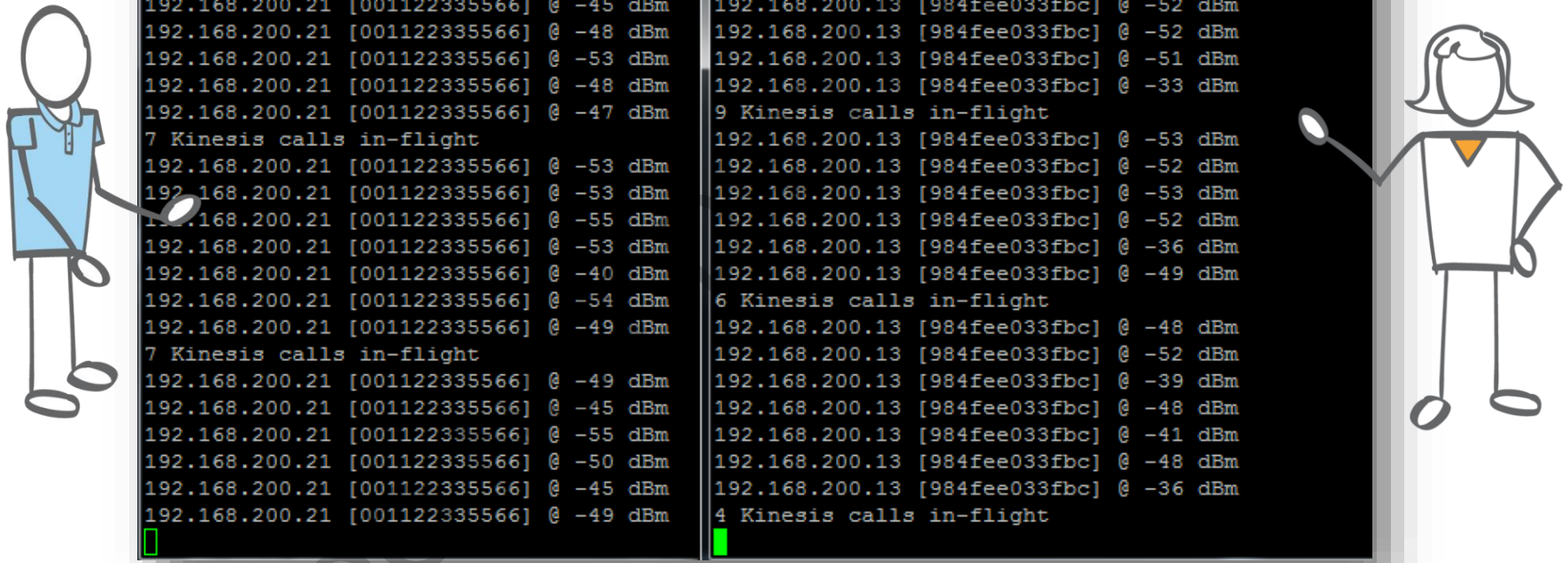
onNobleStateChange() – start advertising

```
function onNobleStateChange(state)
{
  console.log("onNobleStateChange -> " + state);

  if ( state == "poweredOn" )
  {
    console.log("Starting to Advertise this Edison...");
    bleno.startAdvertising( getApplianceName(), configuration.bluetooth.ble.uuid);

    console.log("Initialising Cognito...");
    initialiseCognito();
  }
  else
  {
    console.log("onNobleStateChange -> " + state);
  }
}
```

onNobleDiscoverPeripheral() – send record to stream

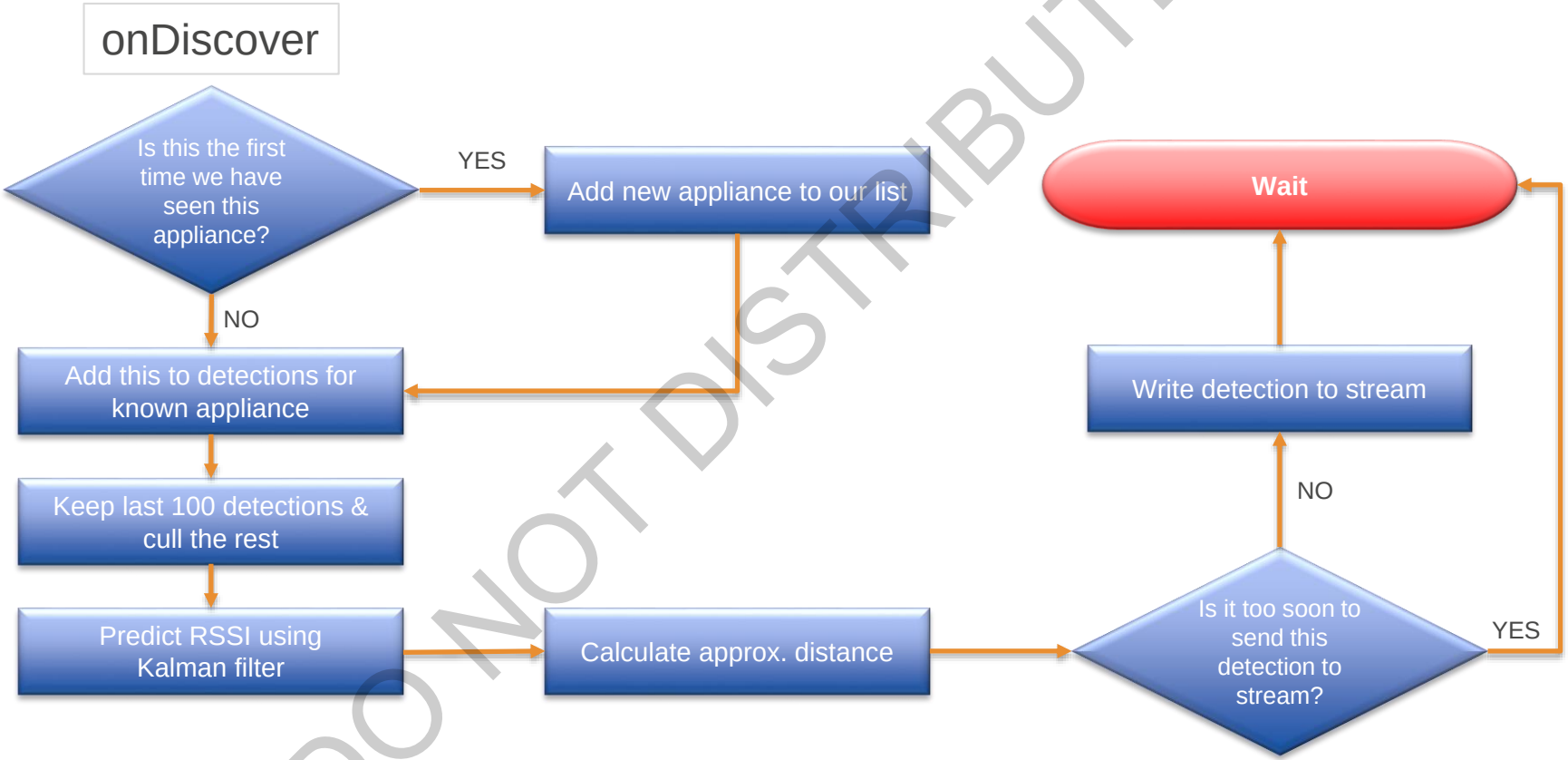
```
function onNobleDiscoverPeripheral(peripheral)
{
    var objPayload = new bleadv(new Date(), peripheral.uuid, peripheral.advertisement.localName, peripheral.rssi);
    var strPayload = JSON.stringify(objPayload);
    console.log(peripheral.advertisement.localName + " [" + peripheral.uuid + "] @ " + peripheral.rssi + " dBm");
    sendRecordToKinesis(strPayload);
}
```

Lab 4 – console output



Lab 4 – Amazon CloudWatch metrics

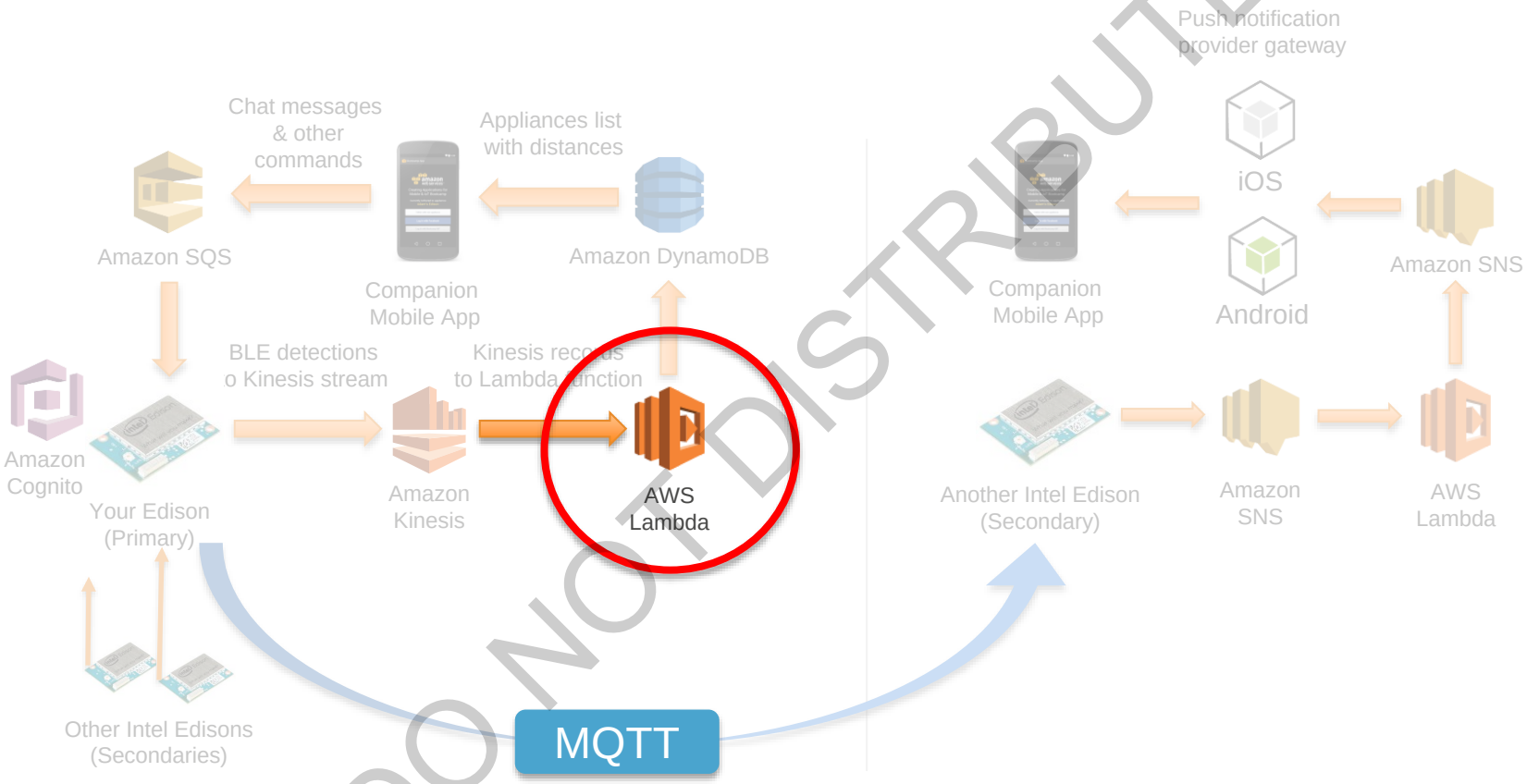
Lab 4 – ‘Smoothed & batched’ detections

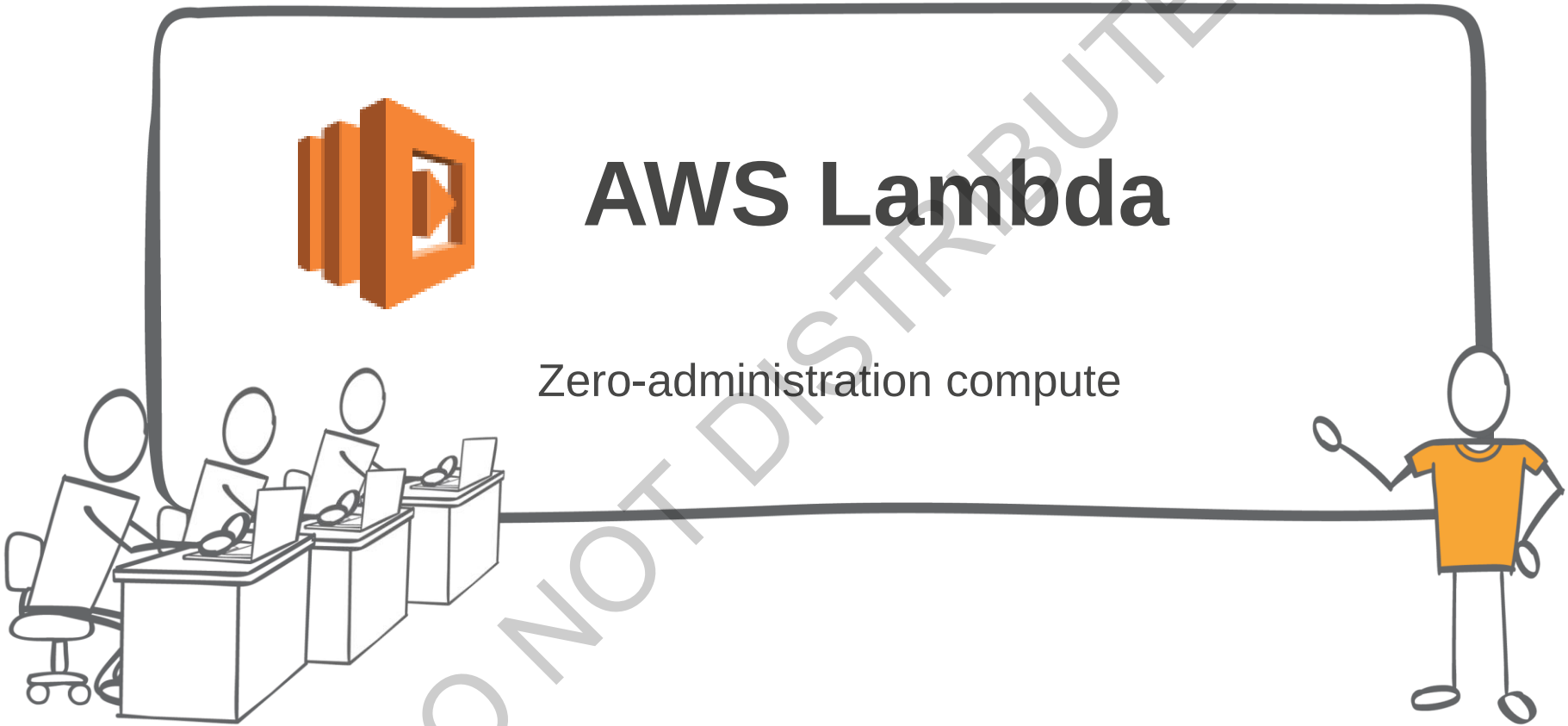


LAB 4

Write BLE detection data
to Amazon Kinesis Stream – ‘raw’ mode
& ‘smoothed and batched’ mode

Sign-posting





AWS Lambda

Zero-administration compute

AWS Lambda
is connective tissue for AWS services


Amazon S3


Amazon DynamoDB


Amazon SNS


Amazon Kinesis


Amazon Cognito

AWS Lambda

- **Zero-administration**
  Focus on business logic, not infrastructure.
  Just upload your code & Amazon Lambda handles the rest

- **Auto-scaling**
  AWS Lambda scales the infrastructure as needed to match the event rate and pay as you go

- **Bring your own Code**
  Node.JS & Java 8 available now with other languages to follow.
  Create threads and processes, run batch scripts **or other exe's**

AWS Lambda

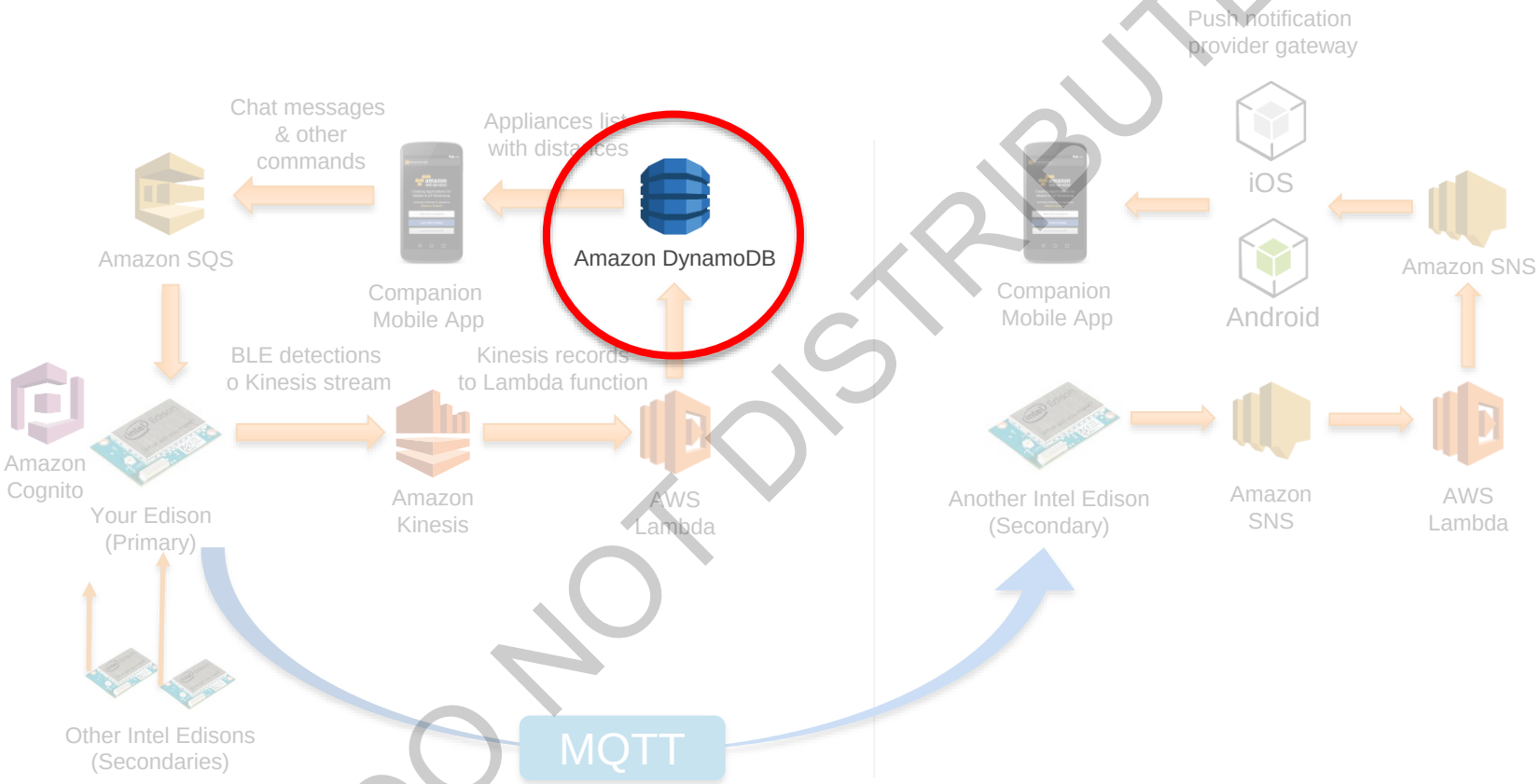
- **Sync & Async events**
  Respond to application calls with low-latency real-time functions

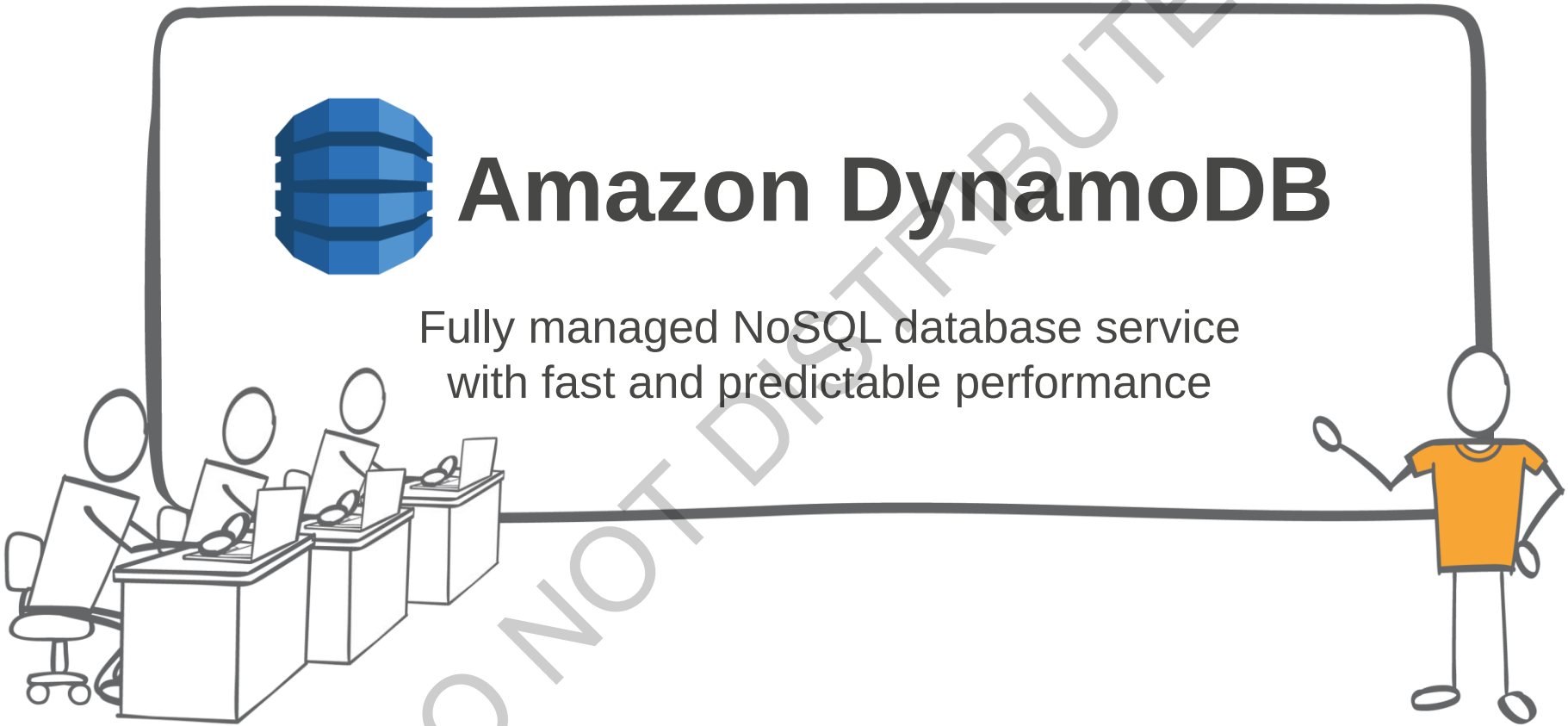
- **Trigger Lambda Functions Using Amazon SNS**
  Respond dynamically and automatically to SNS notifications

- **Multiple AWS Lambda Functions for Amazon Kinesis Streams**
  Attach one or more Lambda functions to handle stream events

- **Integrate AWS Lambda with Amazon Cognito**
  Intercept changes to datasets with **Cognito Sync Triggers**, or trigger code to run by subscribing to changes in datasets with **Cognito Streams**

Sign-posting





Amazon DynamoDB

Fully managed NoSQL database service
with fast and predictable performance

Amazon DynamoDB: Managed NoSQL in the Cloud

Non-Relational Managed NoSQL Database Service

- **Schema-less** data model
- Consistent **low-latency** performance
- **Predictable** provisioned throughput
- Seamless **scalability**
- **No** storage limits
- **High-durability** and **availability**
- Easy Administration – **We scale for you!**
- **Low** Cost
- Cost modeling on **throughput and size**

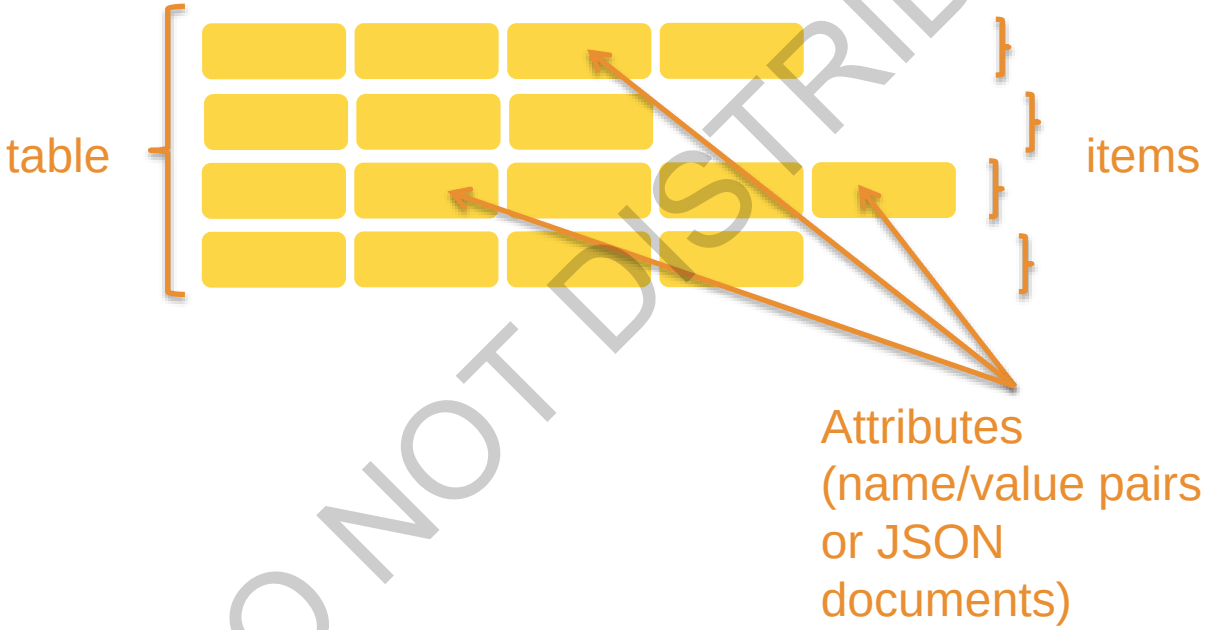
Amazon DynamoDB: Managed NoSQL in the Cloud

Amazon Cognito AWS IAM

Non-Relational Managed NoSQL Database Service

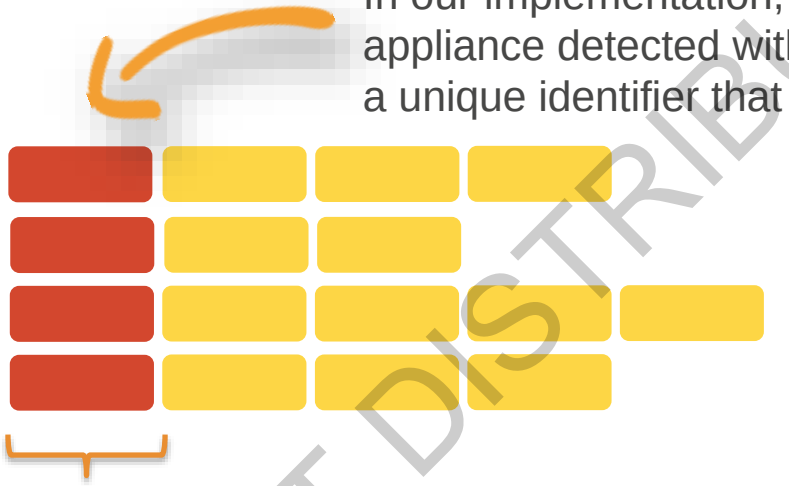
- Fully integrated with AWS Identity & Access Management
  - **Security is native** to Amazon DynamoDB
  - Implement **Fine-grained access control** of individual records – even which fields of records can be accessed
  - For example, make use of the **Cognito Identity** for a user as a key in Amazon DynamoDB to create partitioned views and control access to records based on this key

Amazon DynamoDB is a schemaless database



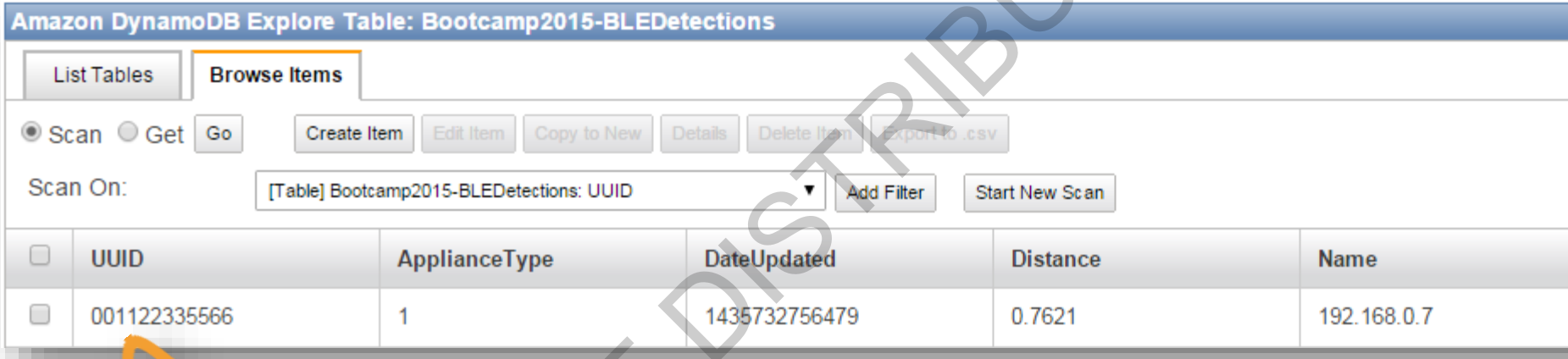
Amazon DynamoDB – Hash Key

In our implementation, each row represents an appliance detected within range and is identified by a unique identifier that we use as the Hash Key



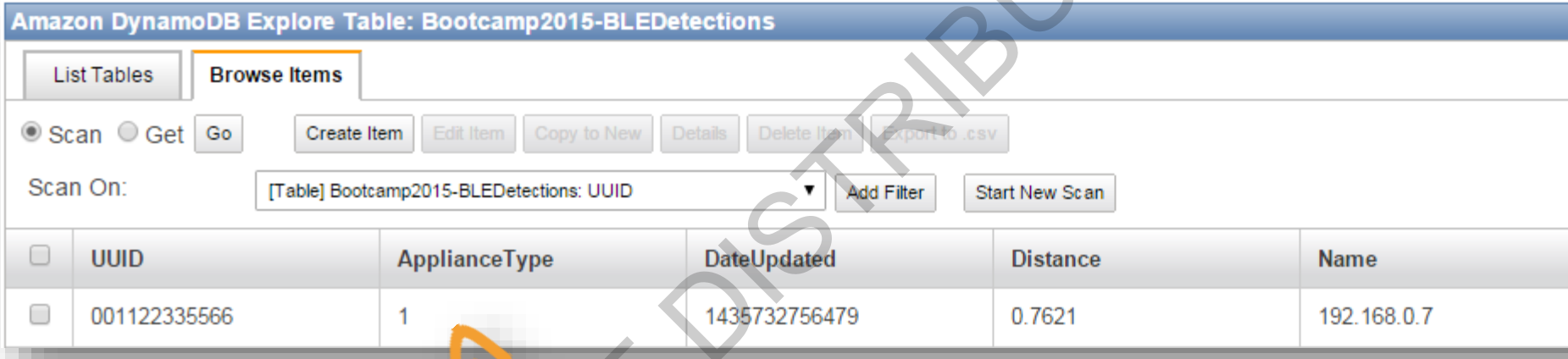
Hash key
(Amazon DynamoDB maintains an unordered index)

How we use Amazon DynamoDB in our system



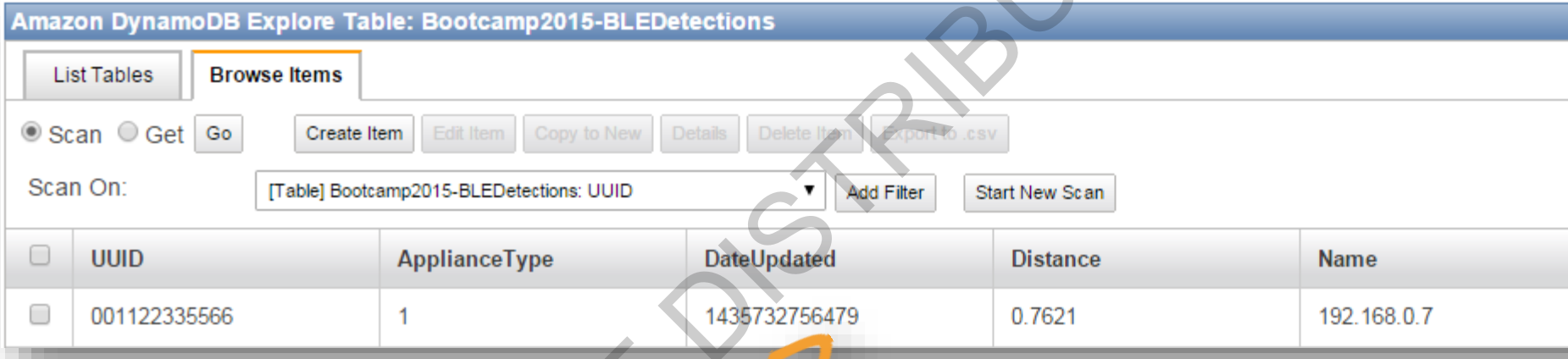
UUID
The Bluetooth MAC address that uniquely identifies this appliance on the network

How we use Amazon DynamoDB in our system



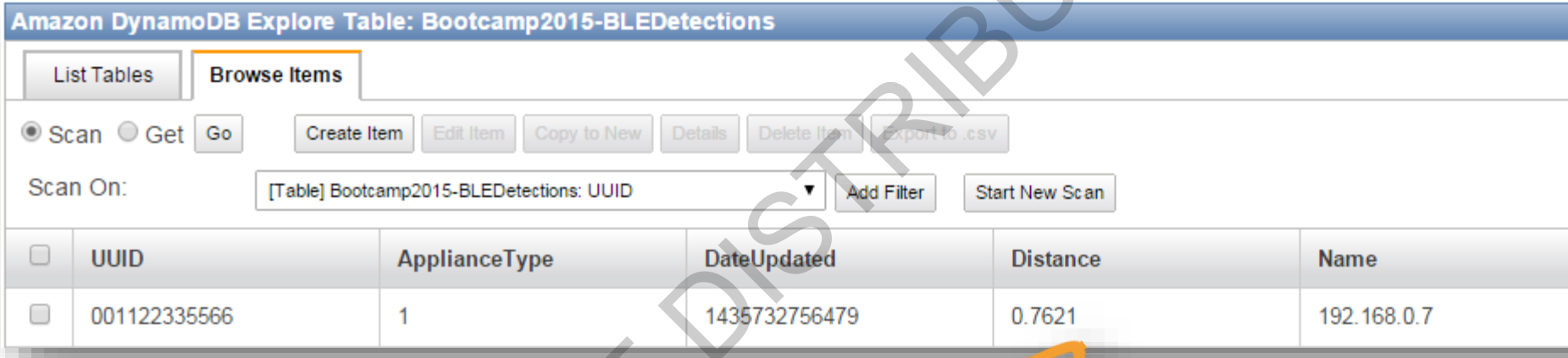
ApplianceType
ENUM value that identifies the type of appliance this device represents (configurable)

How we use Amazon DynamoDB in our system



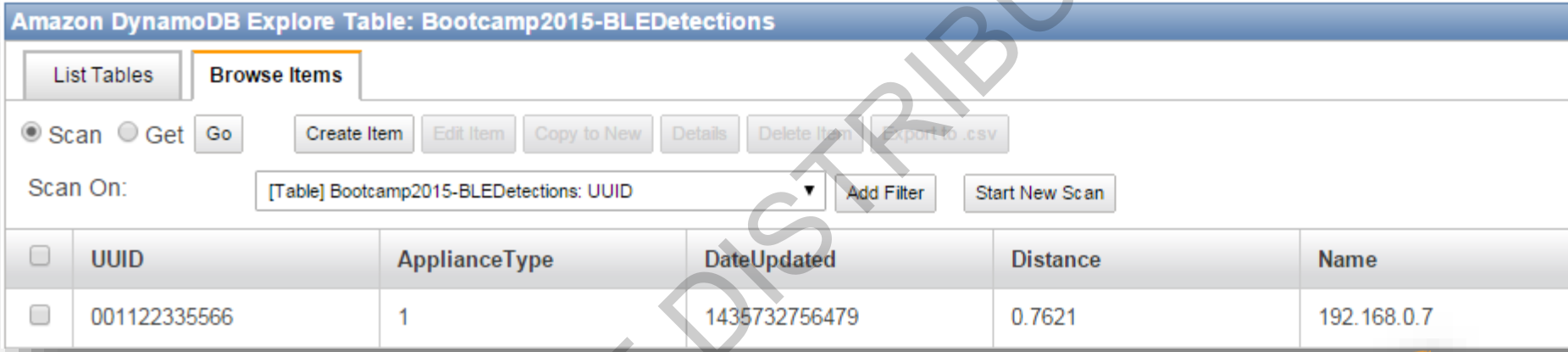
DateUpdated
Timestamp indicating in seconds since epoch, the last time this record was updated

How we use Amazon DynamoDB in our system



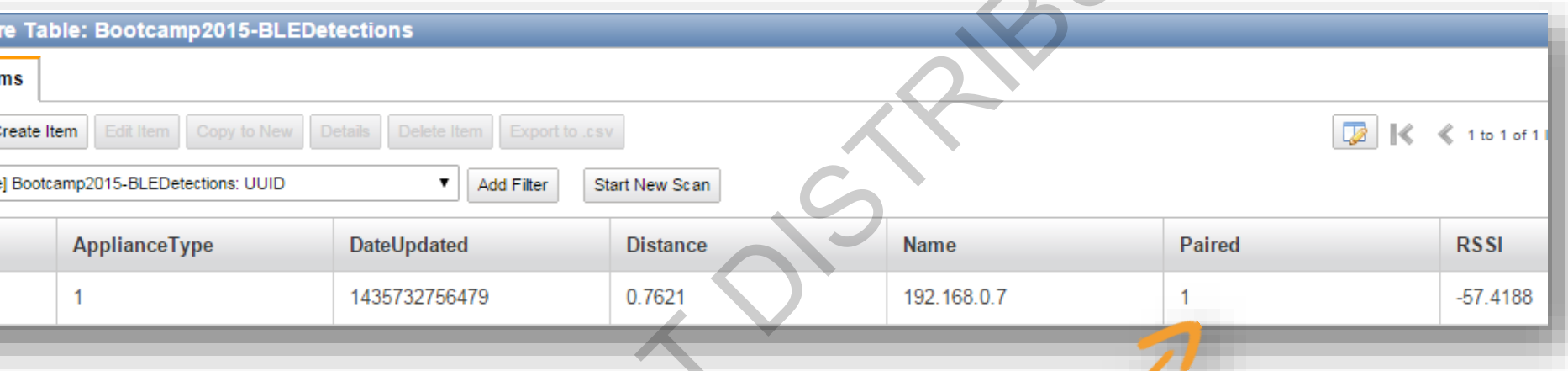
Distance
Distance in meters between the primary appliance, and this secondary appliance

How we use Amazon DynamoDB in our system



Name
The name that the appliance is known as on the network. We use IP addresses in our system because we are not registering devices with DNS

How we use Amazon DynamoDB in our system



Paired
Boolean flag indicating if the primary appliance is paired with this secondary appliance. Used to indicate this in the user interface on the mobile app

How we use Amazon DynamoDB in our system

re Table: Bootcamp2015-BLEDetections

ms

Create Item | Edit Item | Copy to New | Details | Delete Item | Export to .csv

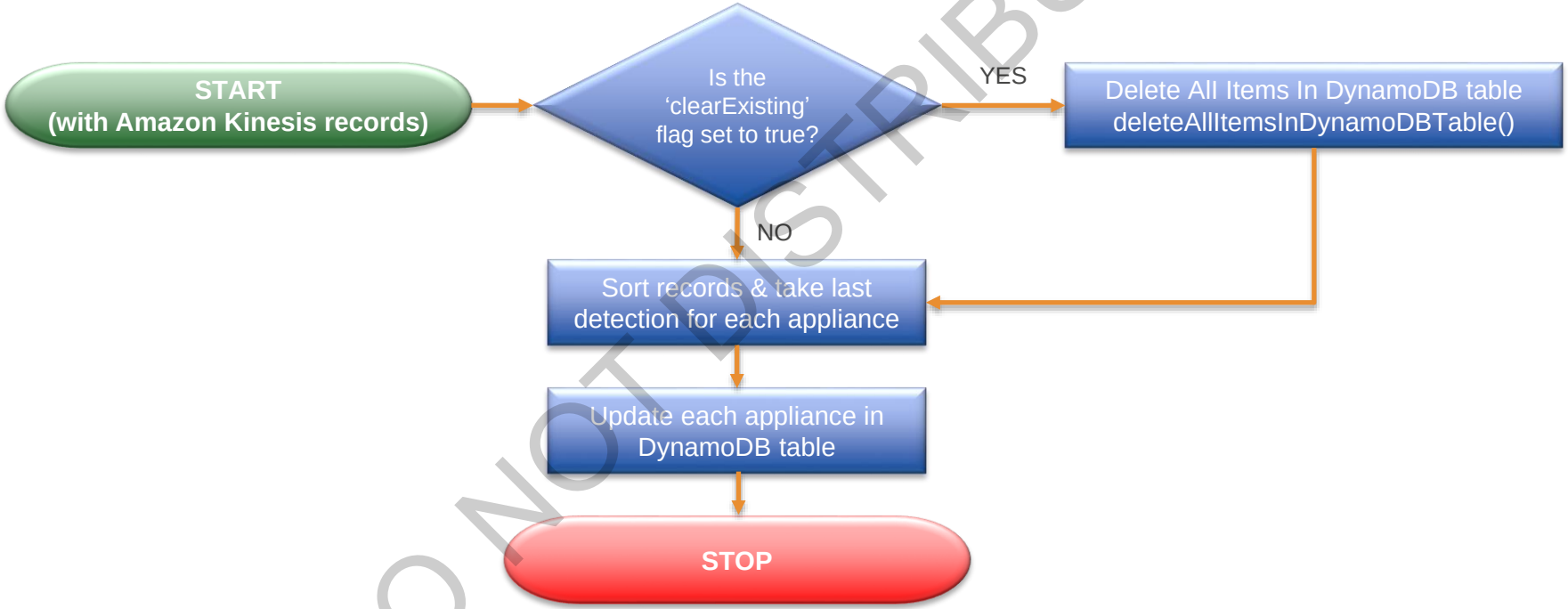
1 to 1 of 1

] Bootcamp2015-BLEDetections: UUID ▼ Add Filter Start New Scan

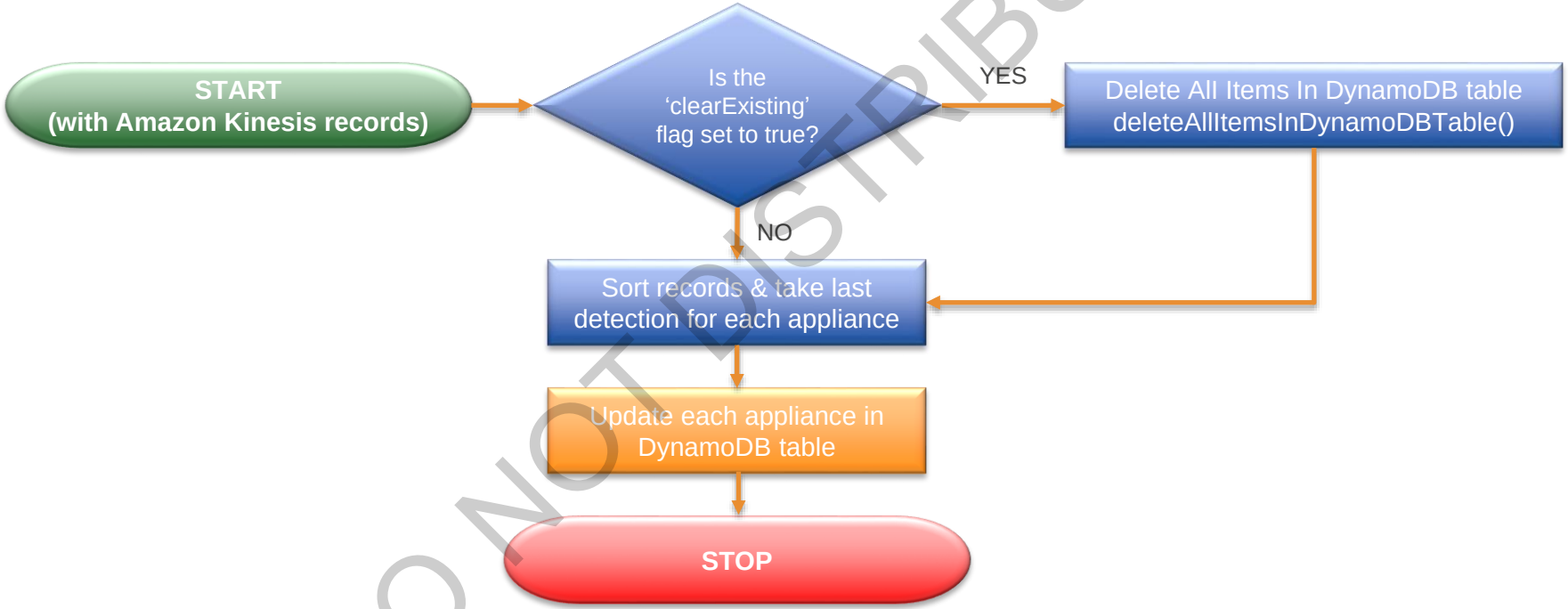
	ApplianceType	DateUpdated	Distance	Name	Paired	RSSI
	1	1435732756479	0.7621	192.168.0.7	1	-57.4188

RSSI
The calculated Receive Signal Strength Indicator, based on the result of the Kalman filter, and distance calculation on the primary appliance

Lab 5 – Write BLE detections to Amazon DynamoDB using AWS Lambda



Lab 5 – Write BLE detections to Amazon DynamoDB using AWS Lambda



```
new AWS.DynamoDB().updateItem(
{
  "TableName": TABLE_NAME_DETECTIONS,

    "Key" :
    {
        "UUID" : {"S" : mostRecentDetection.uuid }
    },

    "ExpressionAttributeNames" :
    {
        "#NAME" : "Name",
        "#RSSI" : "RSSI",
        "#DIST" : "Distance",
        "#APPL" : "ApplianceType",
        "#DATE" : "DateUpdated"
    },

    "ExpressionAttributeValues" :
    {
        ":name" : {"S"  :  GetIPAddress(mostRecentDetection.name) },
        ":rssi" : {"N"  :  mostRecentDetection.filteredRSSI },
        ":dist" : {"N"  :  mostRecentDetection.calculatedDistance },
        ":appl" : {"S"  :  GetApplianceType(mostRecentDetection.name) },
        ":date" : {"N"  :  new Date().getTime().toString() }
    },

    "UpdateExpression" : "SET #NAME = :name, #RSSI = :rssi, #DIST = :dist, #APPL = :appl, #DATE = :date"
```

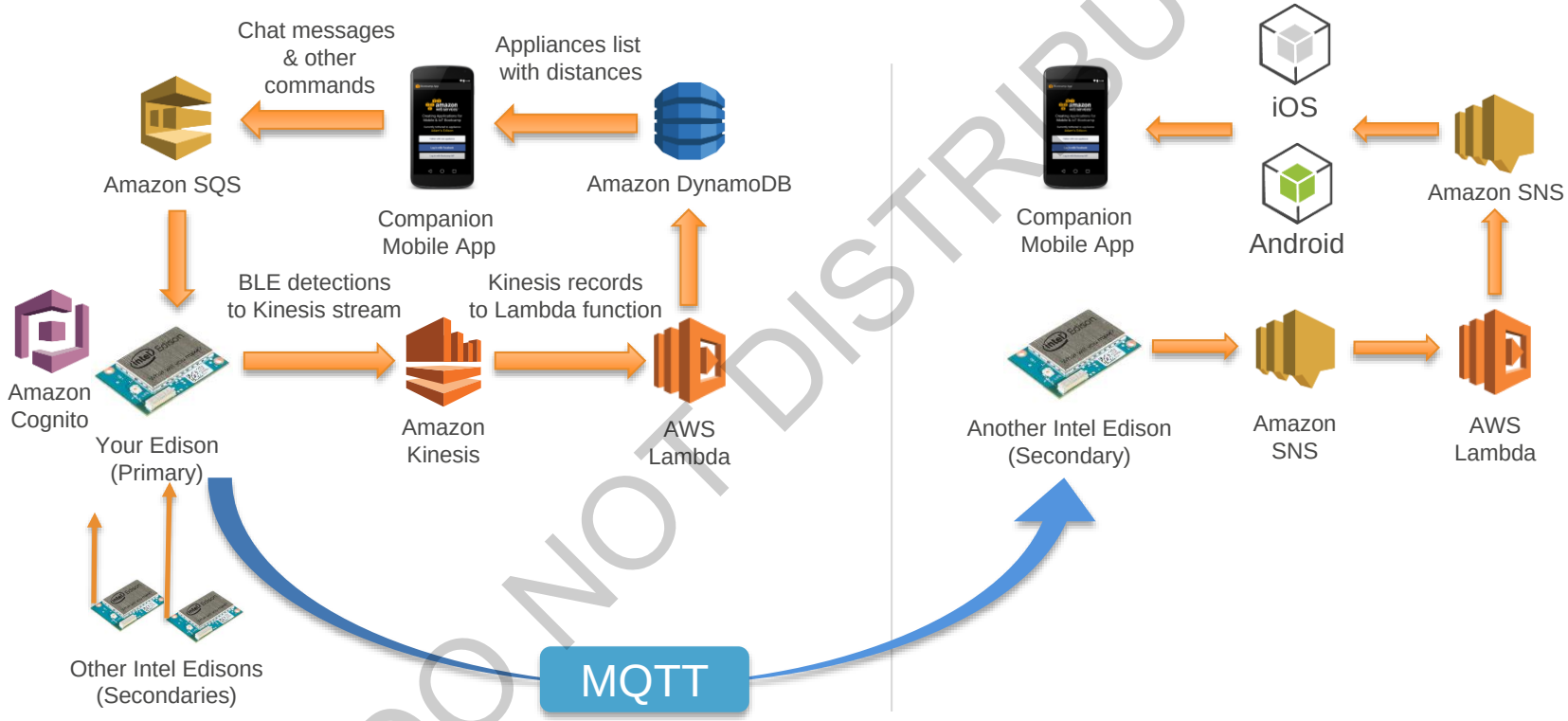
LAB 5

Using AWS Lambda to write Amazon Kinesis records to Amazon DynamoDB

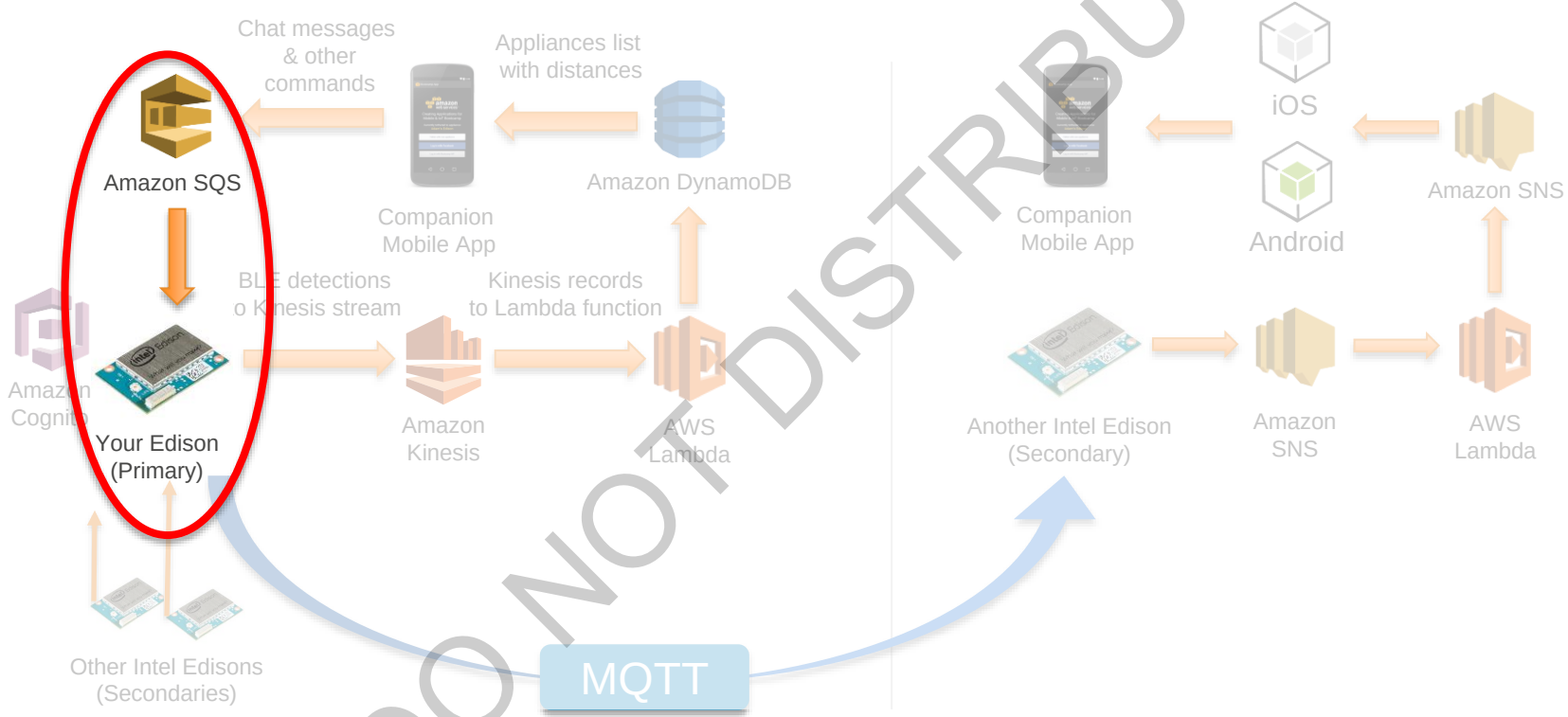
Part 3

Commanding the IoT device:
messaging with Amazon SQS,
Amazon SNS & the MQTT protocol

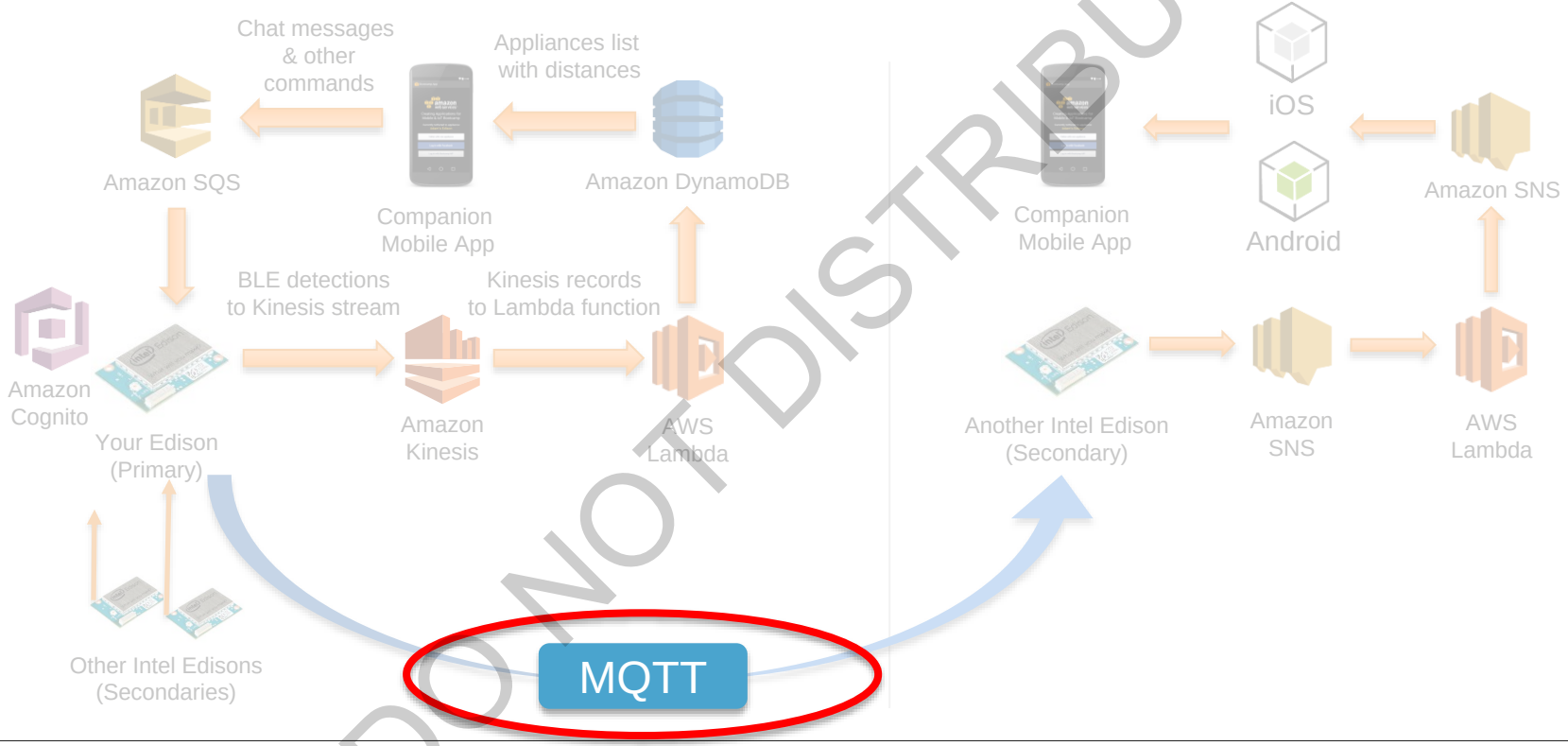
System architecture



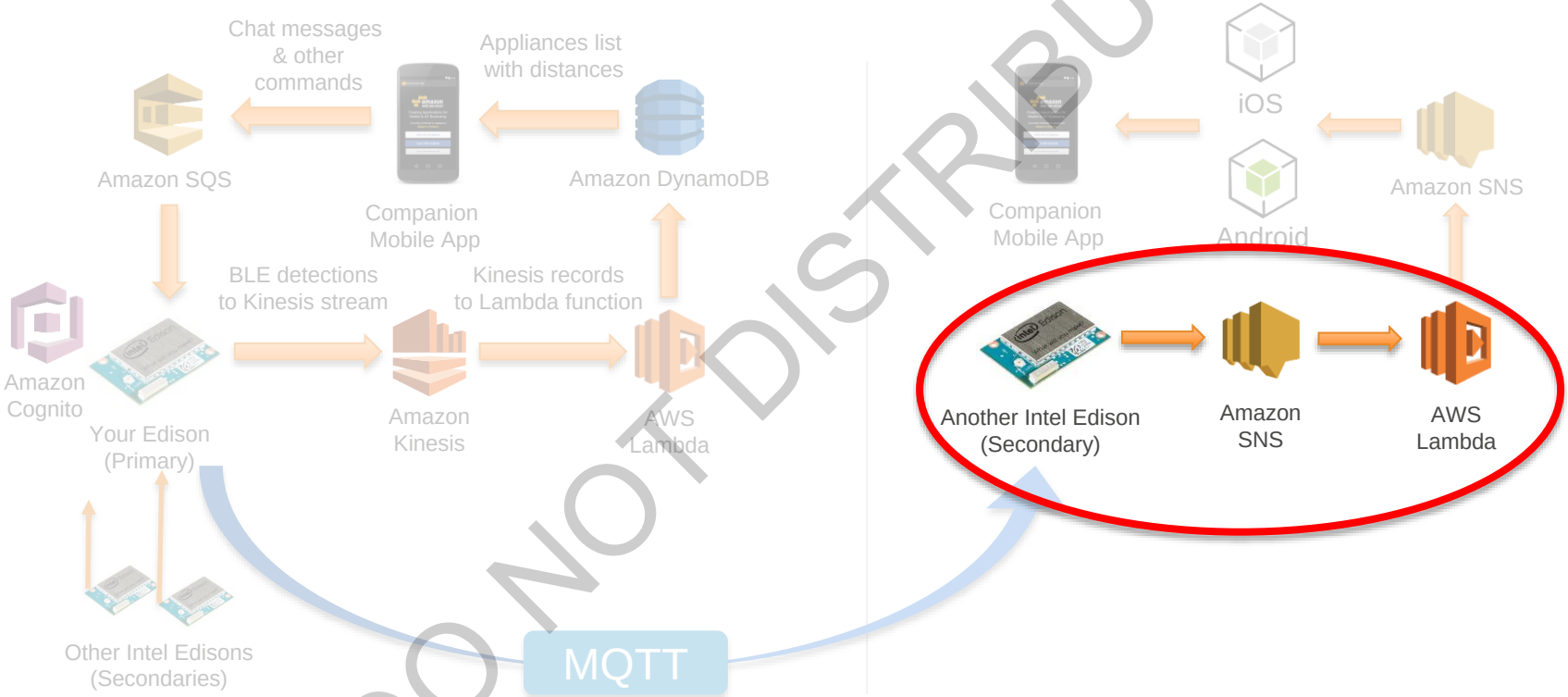
System architecture



System architecture



System architecture



Amazon SQS

Fast, reliable, scalable, fully managed message queuing service

Amazon SQS

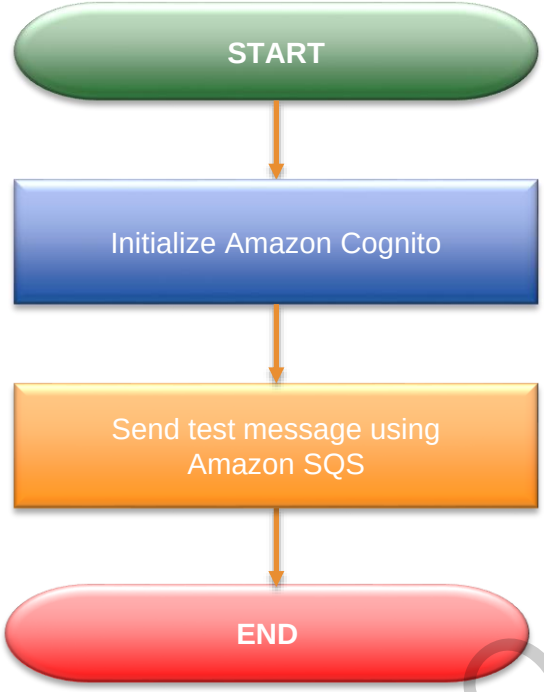
- **Reliable**
  Messages stored redundantly across servers and data centers

- **Scalable**
  Designed to enable an unlimited number of messaging services
  read and write an unlimited number of messages at any time

- **Simple and inexpensive**
  Only 5 API calls to manage queues and send/receive messages.
  No up-front costs – just pay for what you use

Lab 6: How we use Amazon SQS – sending a message

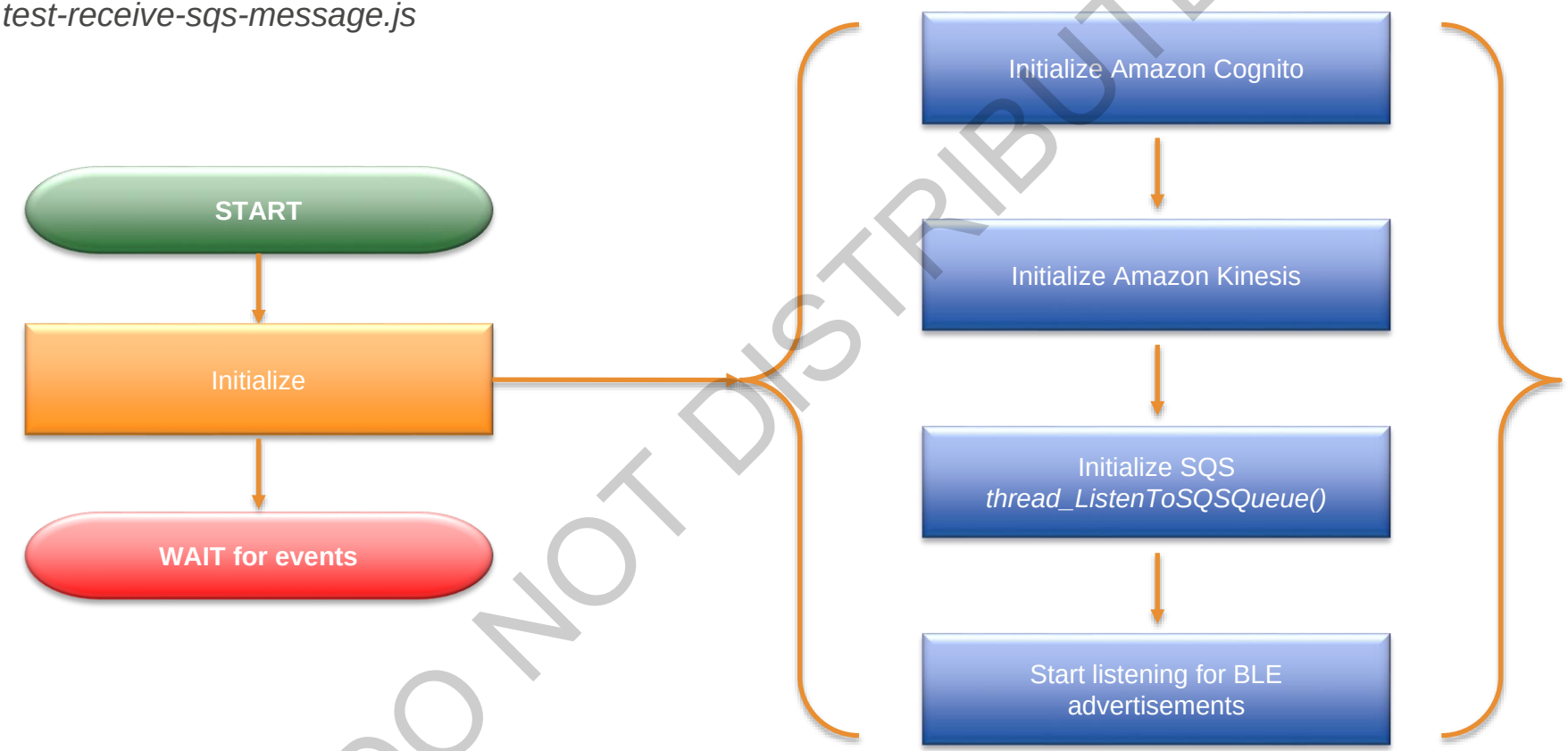
test-send-sqs-message.js




```
function sendSQSMessage()
{
  console.log("Sending message to SQS queue...");
  sqs = new aws.SQS();

  var messageToSend = new compak(compak.COMMAND_TEST, "Hello, World!");

  sqs.sendMessage({
       QueueUrl:              configuration.aws.SQS_URL_QUEUE_TO_EDISON,
       MessageBody:           JSON.stringify(messageToSend)
     }, onSQSMessageSent );

  console.log("     sent");
}
```

Lab 6: How we use Amazon SQS – receiving a message

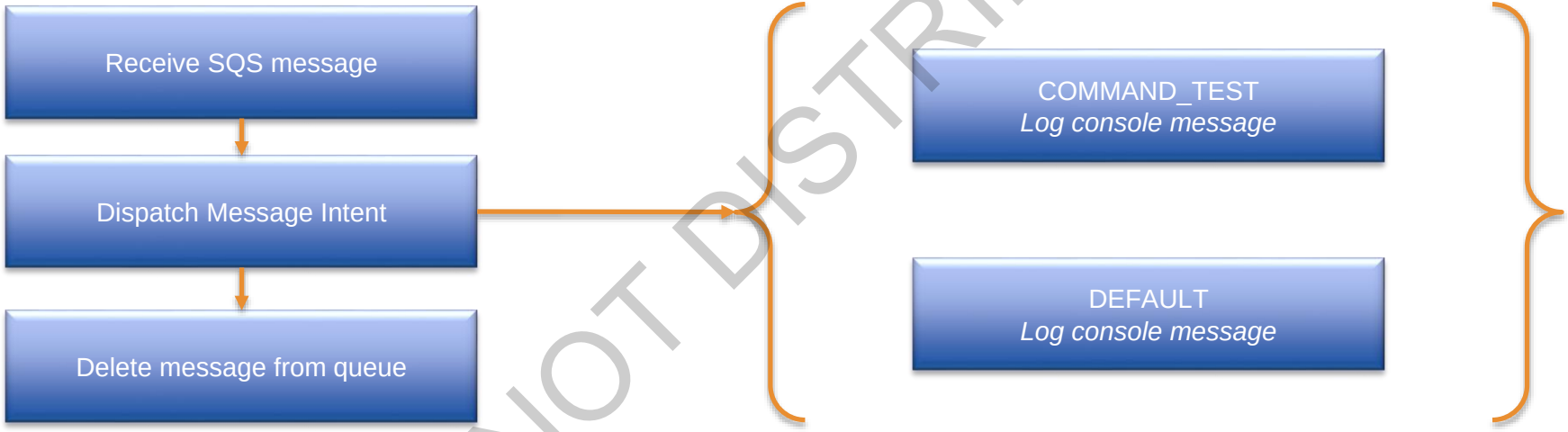
test-receive-sqs-message.js



Lab 6: How we use Amazon SQS – receiving a message

test-receive-sqs-message.js

onSQSMessageReceived



Lab 6: How we use Amazon SQS – receiving a message

test-receive-sqs-message.js

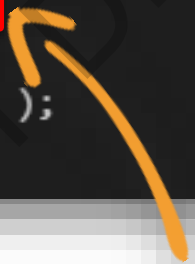
```
function thread_ListenToSQSQueue()
{
    sqs.receiveMessage({
       QueueUrl:              configuration.aws.SQS_URL_QUEUE_TO_EDISON,
       MaxNumberOfMessages:   1,
       VisibilityTimeout:     60,
       WaitTimeSeconds:       2
     }, onSQSMessageReceiveReturned);
}
```

Maximum time this 'long poll' will wait before returning without any messages. Will return earlier if any messages arrive during the wait time

Lab 6: How we use Amazon SQS – receiving a message

test-receive-sqs-message.js

```
function thread_ListenToSQSQueue()
{
    sqs.receiveMessage({
       QueueUrl:              configuration.aws.SQS_URL_QUEUE_TO_EDISON,
       MaxNumberOfMessages:   1,
       VisibilityTimeout:     60,
       WaitTimeSeconds:       2
     }, onSQSMessageReceiveReturned );
}
```



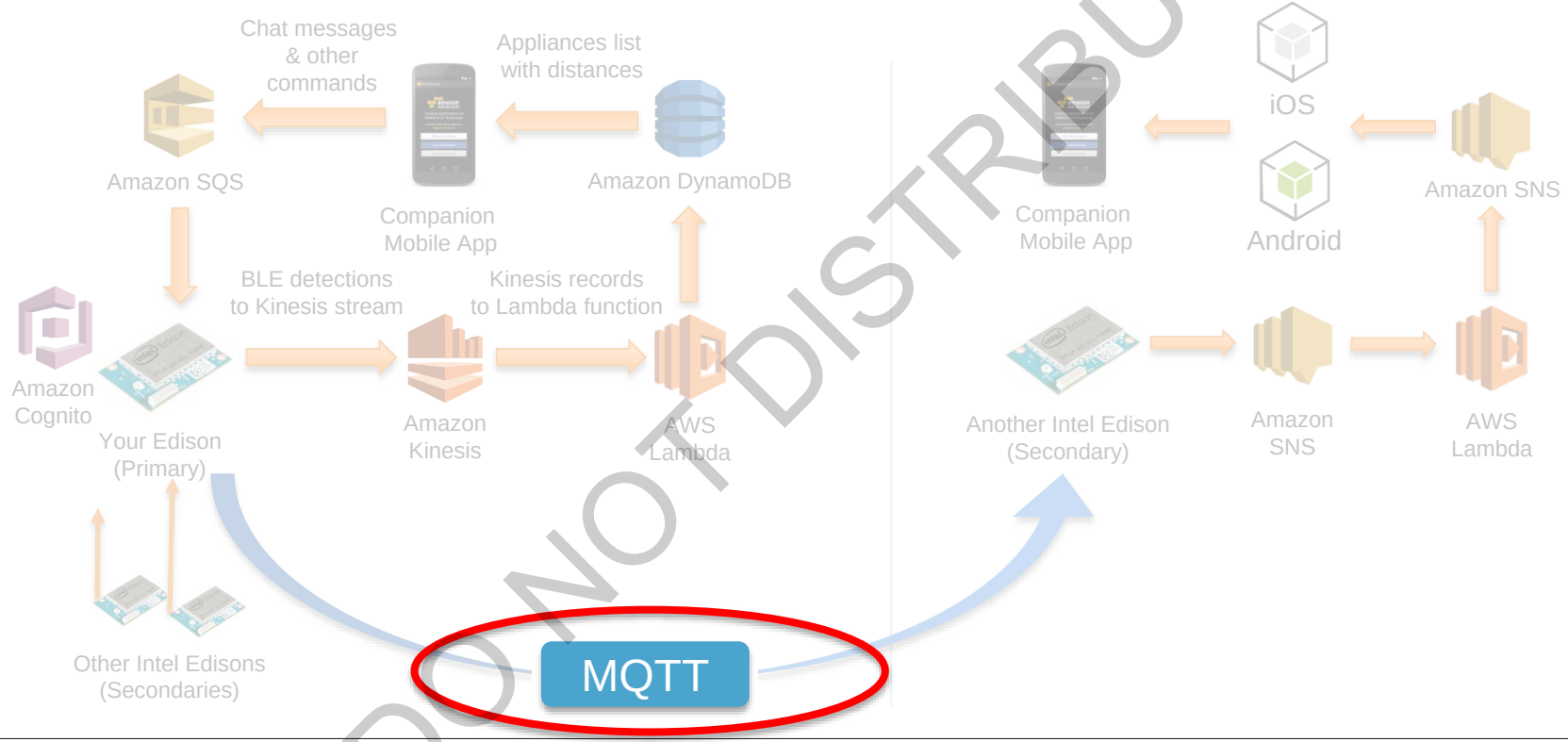
Maximum time a message will be 'in flight' before it is returned to the message queue for processing

Lab 6: How we use Amazon SQS – receiving a message

test-receive-sqs-message.js

```
function onSQSMessageReceiveReturned(err, data)
{
     if (err)
     {
       console.log("ERROR: onSQSMessageReceiveReturned() -> " + err);
     }
     else
     // If there are any messages to get
     if (data.Messages)
     {
        //
        // Process the message
        //
        DispatchQueueMessage(data.Messages);
     }

     setTimeout(thread_ListenToSQSQueue, 1000);
}
```

System architecture



What is MQTT?

- “MQ Telemetry Transport”
  Designed for message exchange between constrained devices and low-bandwidth, high-latency or unreliable networks

- Simple & Lightweight
  Uses TCP, long-lived connection to an MQTT broker. A broker is built in to the base Yocto image on Edison
  (RSMB - the *Really Simply Message Broker*)

- Publish/Subscribe model
  Simply subscribe to a topic and publish messages.
  Many-to-many protocol.
  Delivery modes: Fire & Forget; At least Once; Only Once

Using NodeJS libraries for MQTT

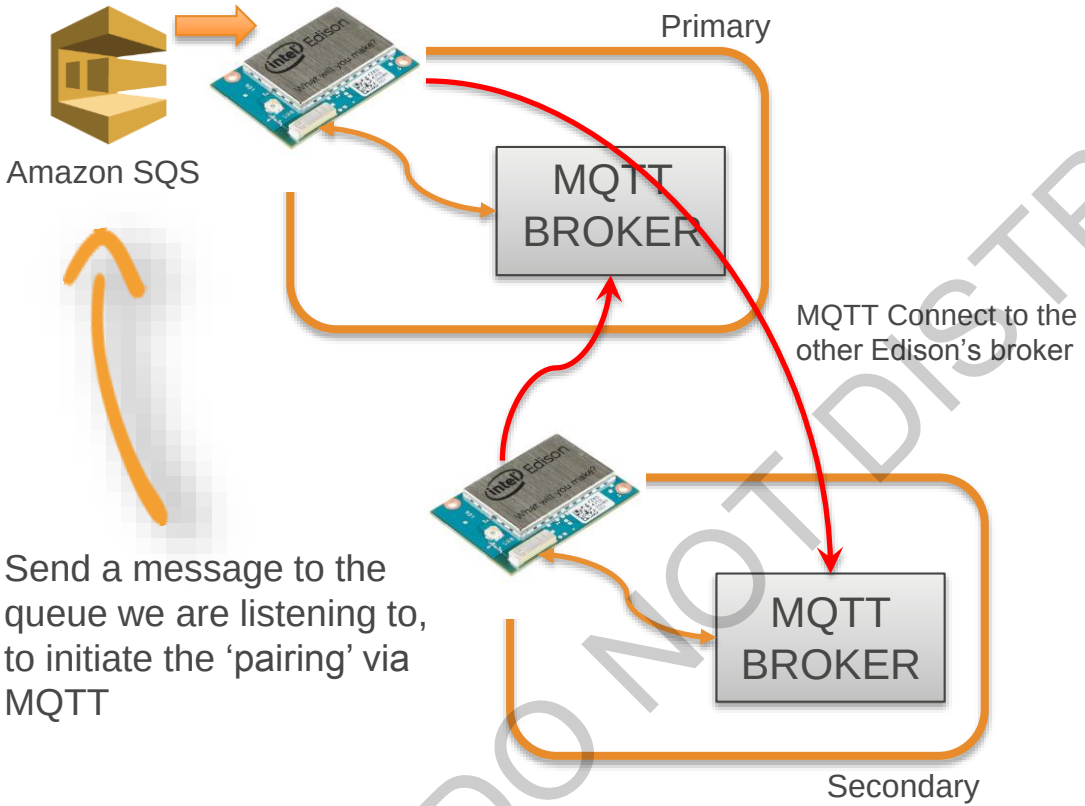
- mqtt
  - A node.js module for implementing MQTT
  - Simple to use
  - Async, callback-based

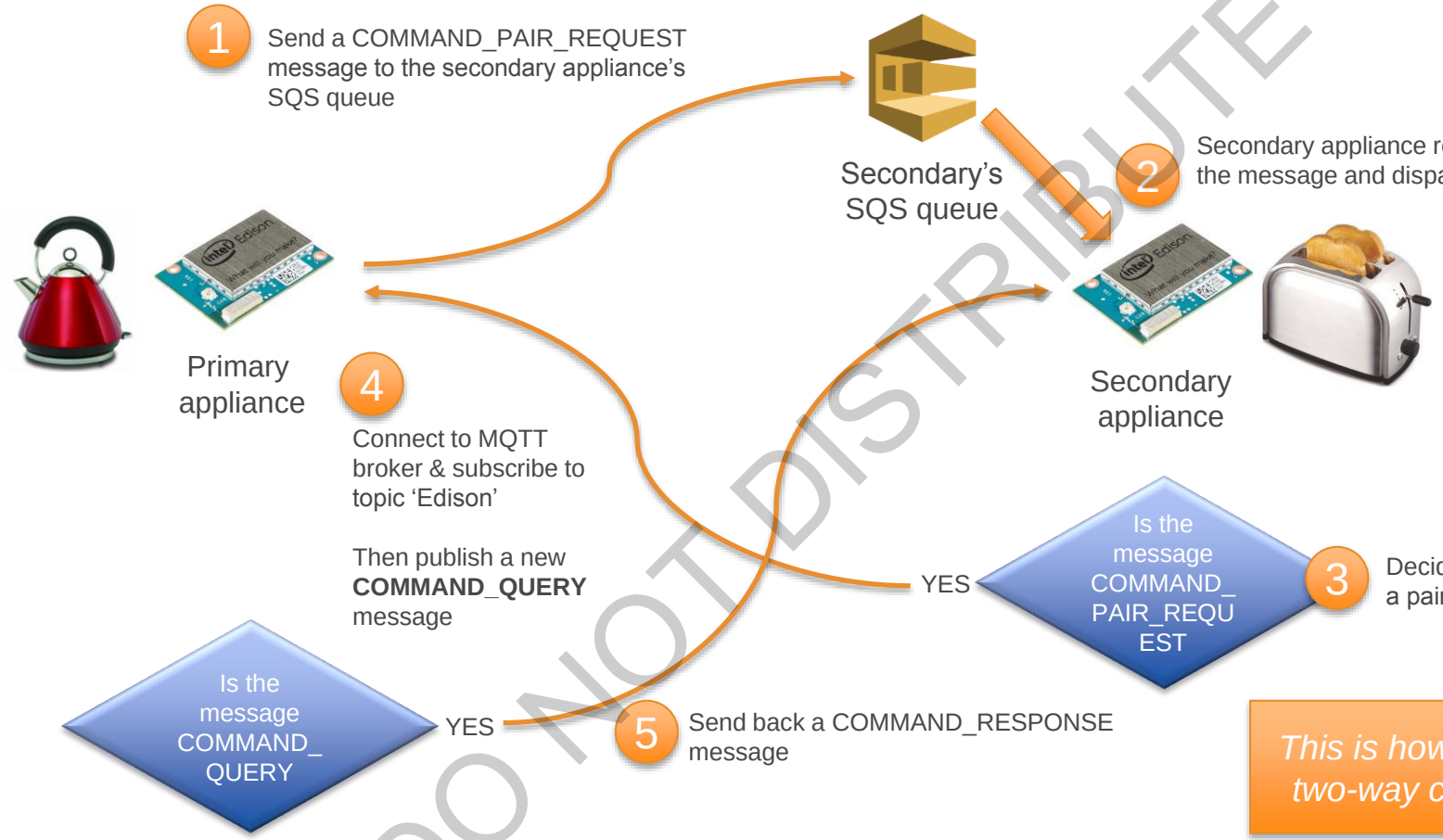
```
var mqtt                    = require('mqtt');
var mqttClient              = mqtt.connect('mqtt://localhost');
```

Note that we listen to our local MQTT broker
for in-bound connections to our appliance
rather than using a central, shared MQTT broker

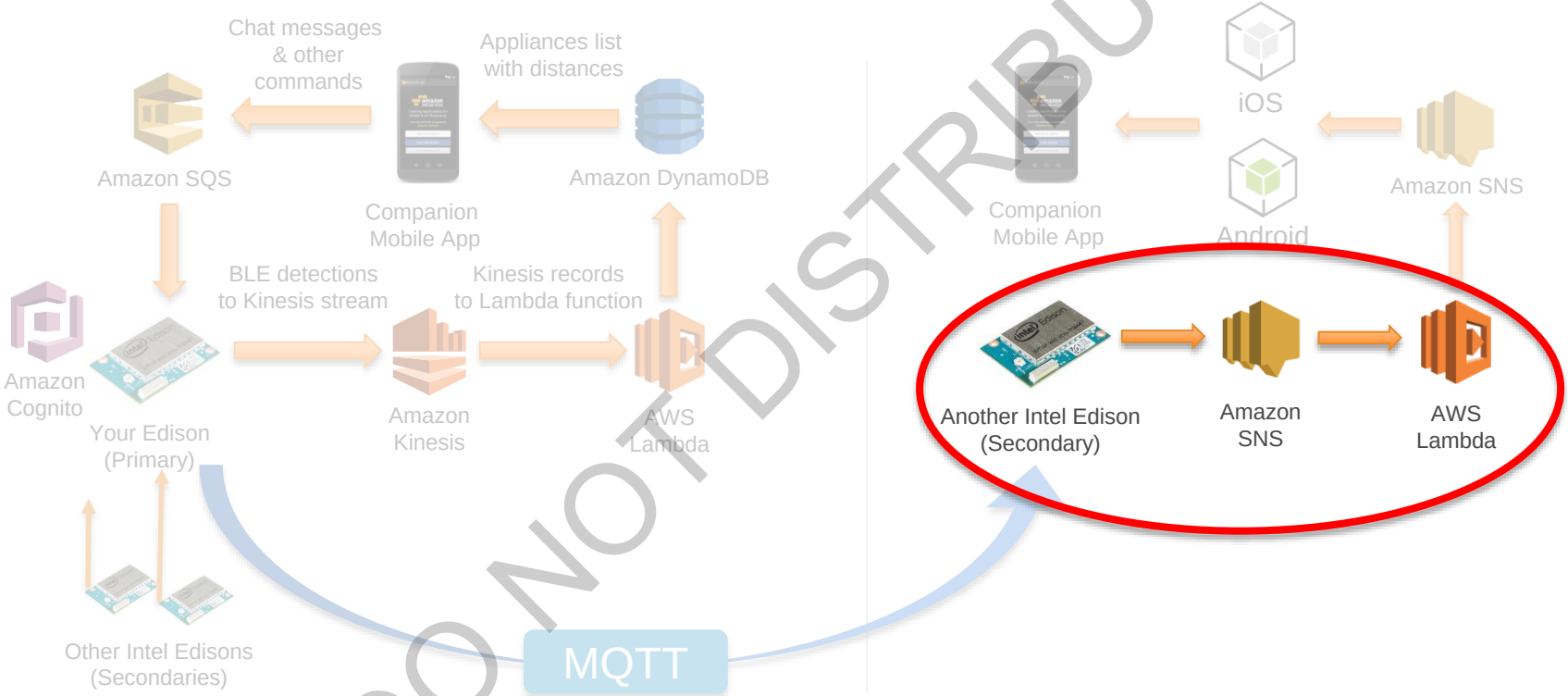
So we need to ignore messages that our appliance initiates!

How our system uses MQTT





System architecture



Amazon SNS

Coordinates and manages the delivery or sending of messages to subscribing endpoints or clients

Amazon SNS

- Versatile

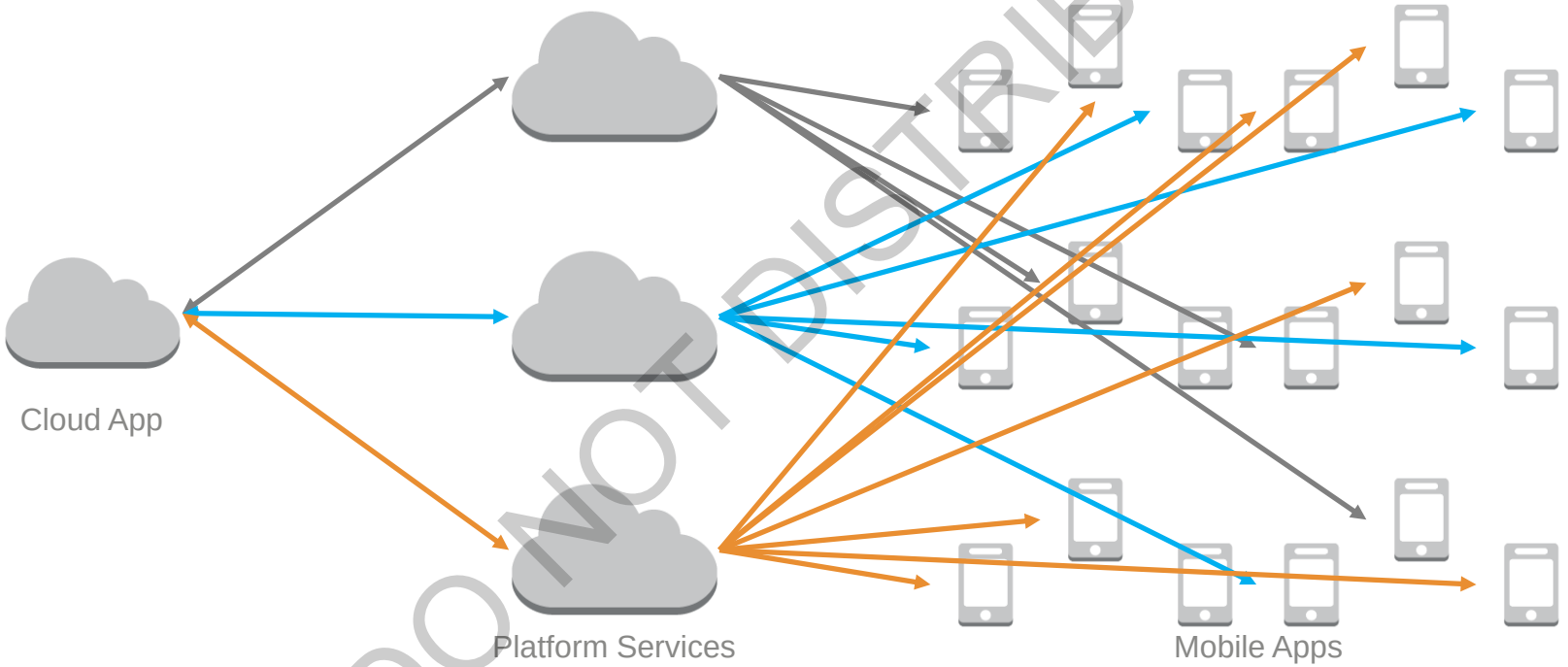
  Send individual notifications or use ‘fan-out’ to send to large numbers of recipients

- Flexible

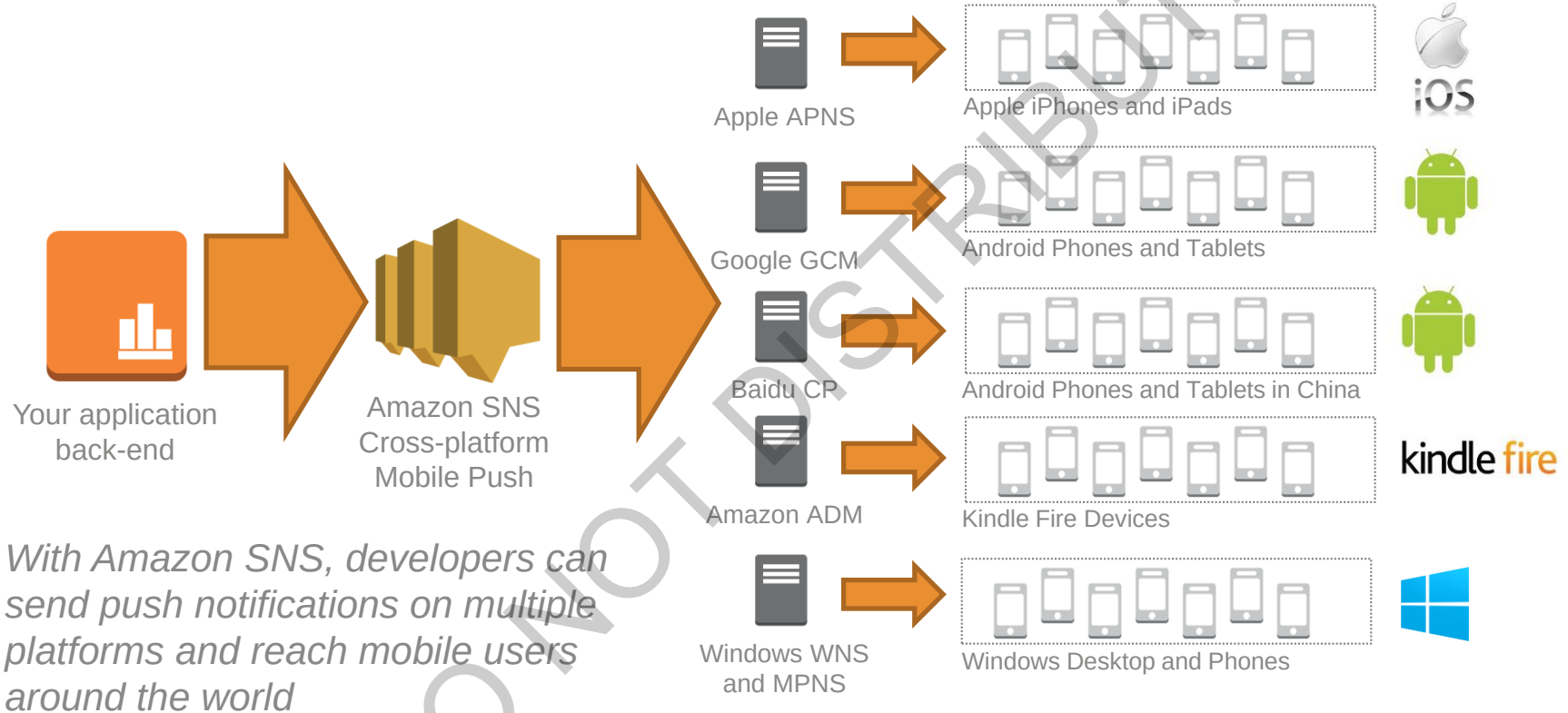
  Send to **mobile devices**, email addresses, HTTP endpoints, Amazon SQS queues or even **AWS Lambda functions**

Amazon SNS mobile push

Each platform works differently, and push gets even more complex as you scale to support millions of devices.

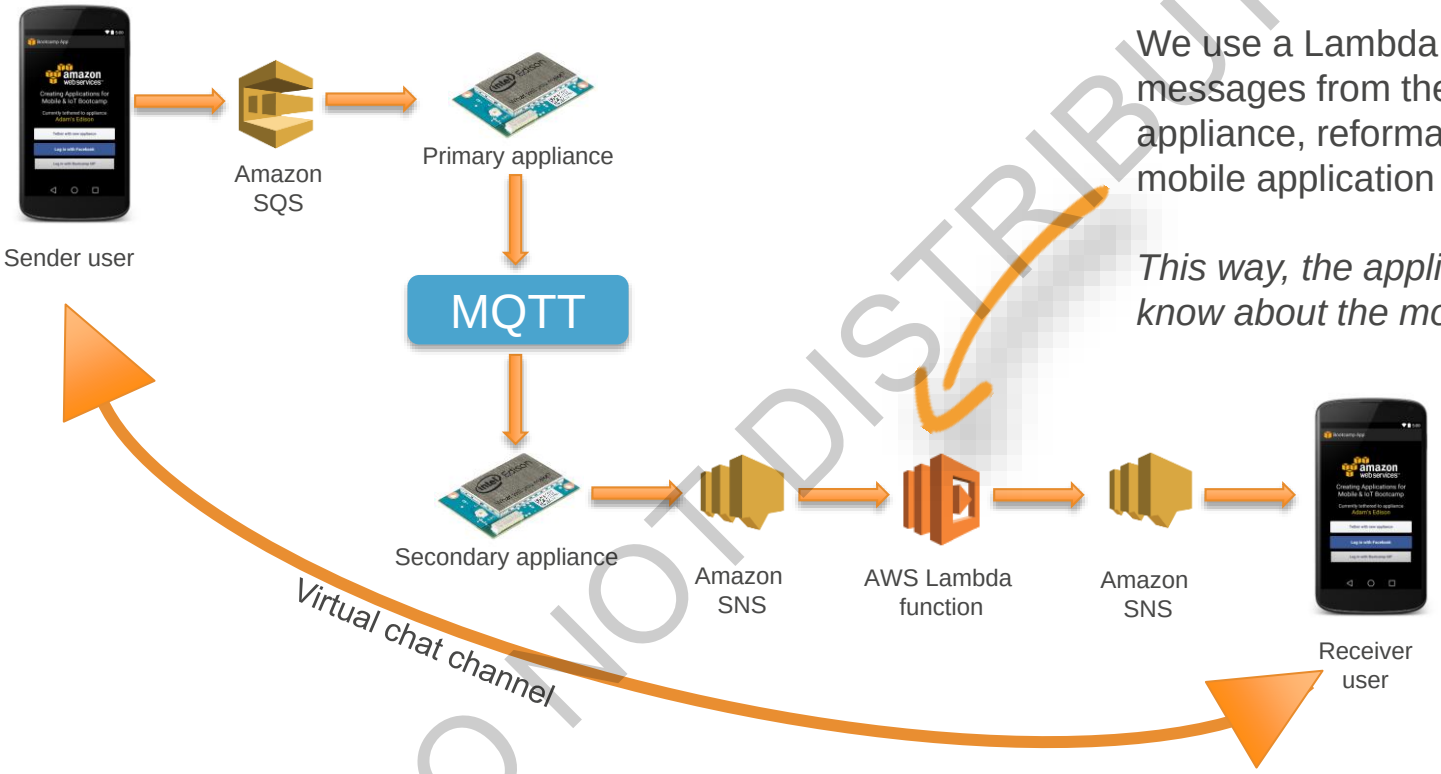


Amazon SNS mobile push



With Amazon SNS, developers can send push notifications on multiple platforms and reach mobile users around the world

How we use Amazon SNS in our system to implement chat



We use a Lambda function to intercept messages from the secondary appliance, reformat and send to the mobile application

This way, the appliance never needs to know about the mobile application

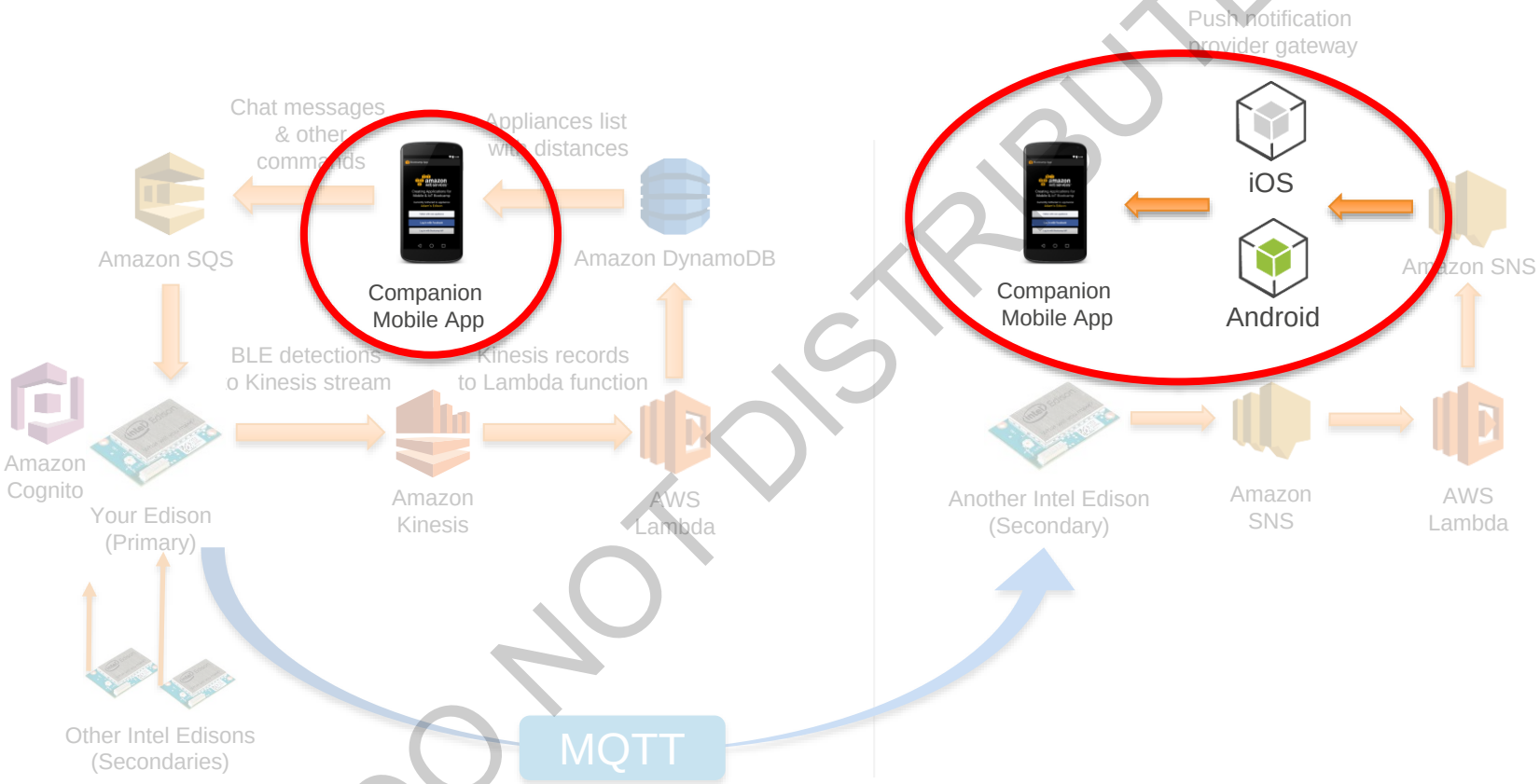
LAB 6

Messaging with Amazon SQS,
Amazon SNS & the MQTT protocol

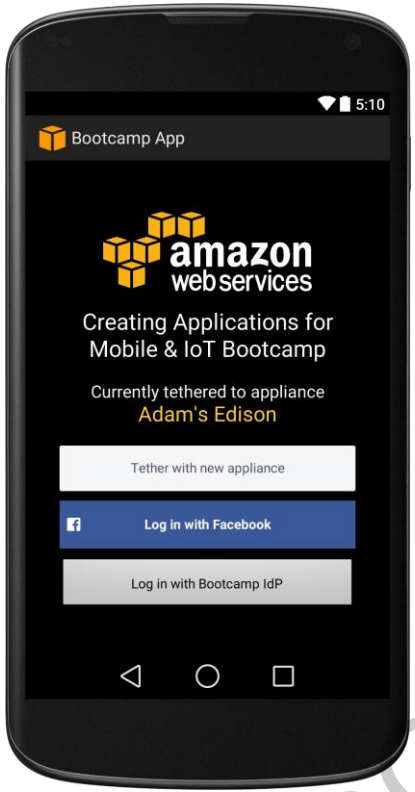
Part 5

Creating the companion Mobile Application

Sign-posting

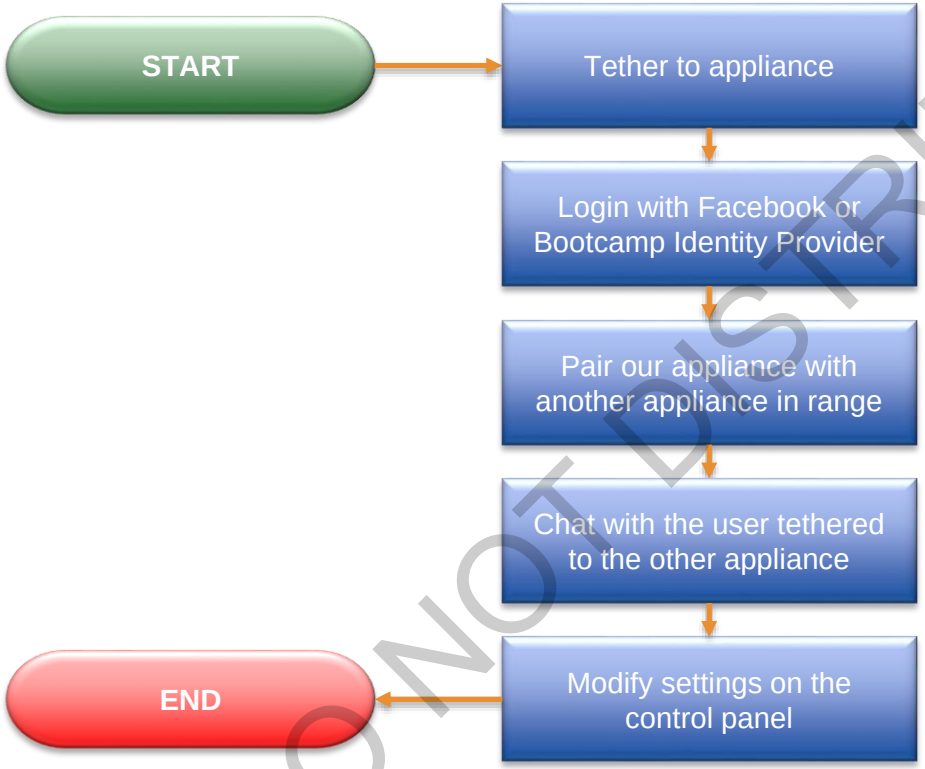


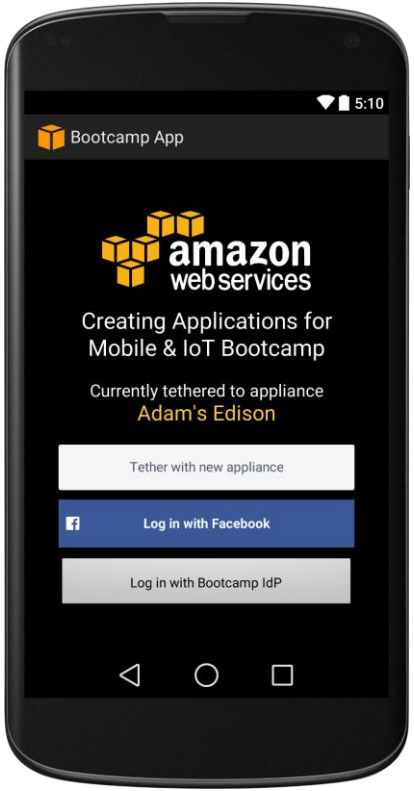
Our Companion Mobile Application



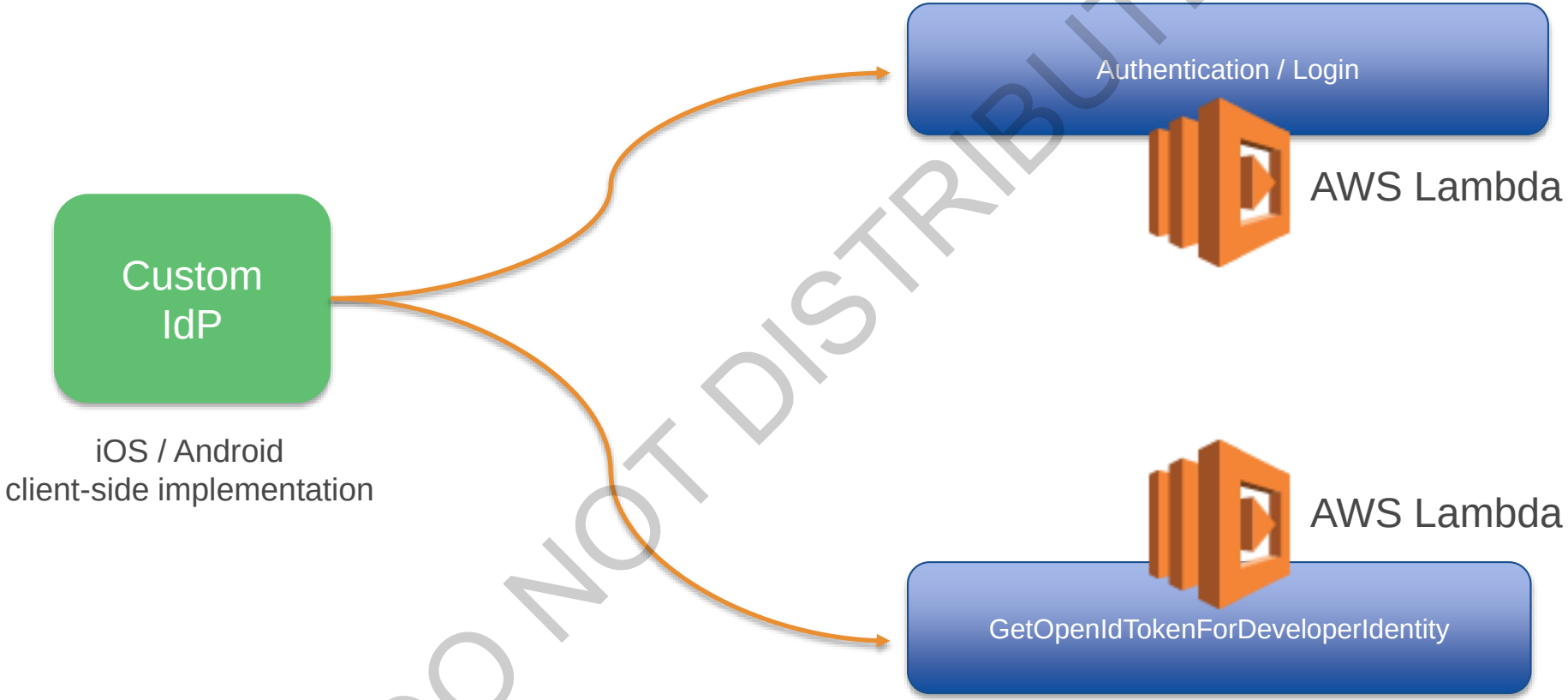
- Remotely controls our ‘appliances’
- Requires authentication
- Dynamically ‘tether’ to any appliance
- Provides control panels for appliances
- Implements Cognito Push Sync
- Demonstration ‘chat’ implementation

Recap: User Workflow

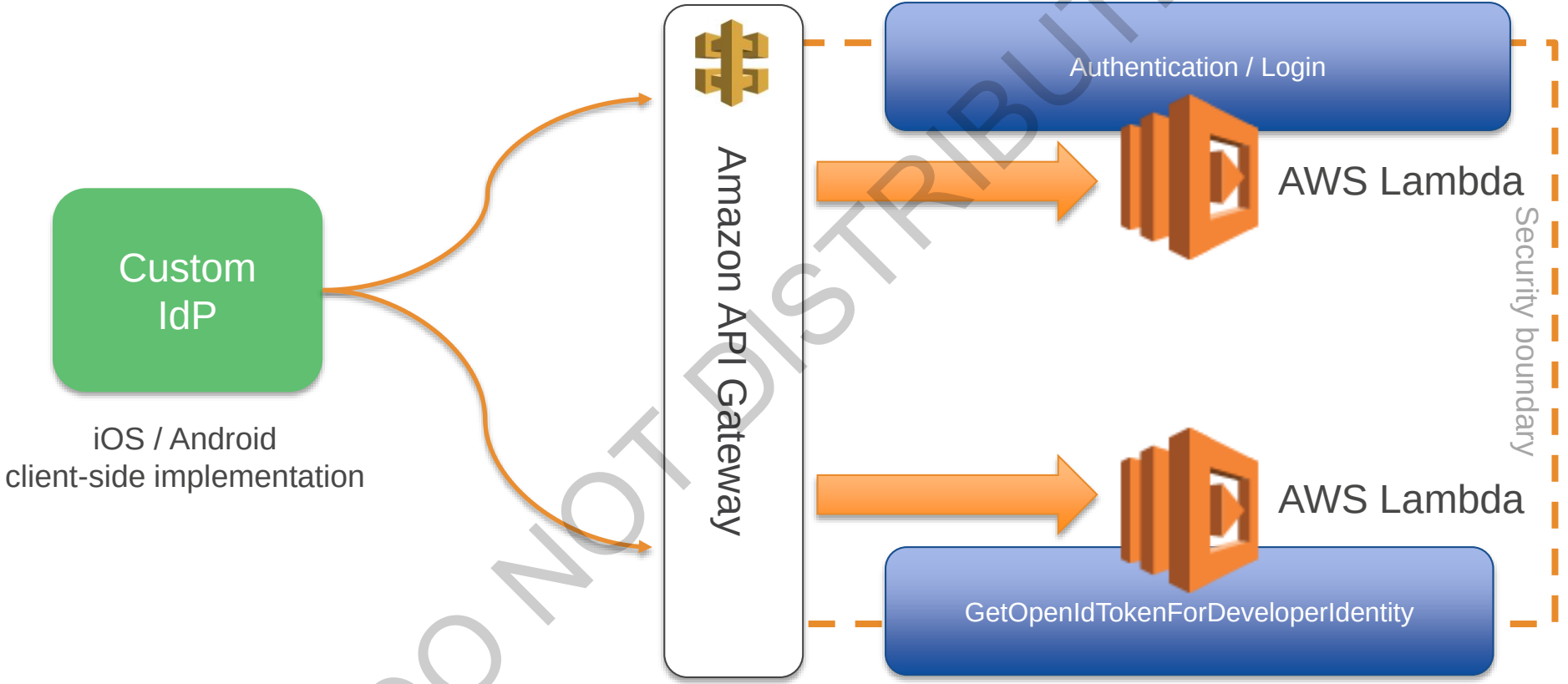




Custom IdP needs 2 calls exposed



We will use Lambda via API Gateway



Amazon API Gateway

Create, publish, maintain, monitor, and secure APIs at any scale

Amazon API Gateway

- **Performance at Any Scale**
  Uses AWS worldwide edge locations for low-latency API request/response

- **Run Your APIs Without Servers**
  Easily expose RESTful endpoints backed by Lambda for server-less APIs for your mobile & web applications

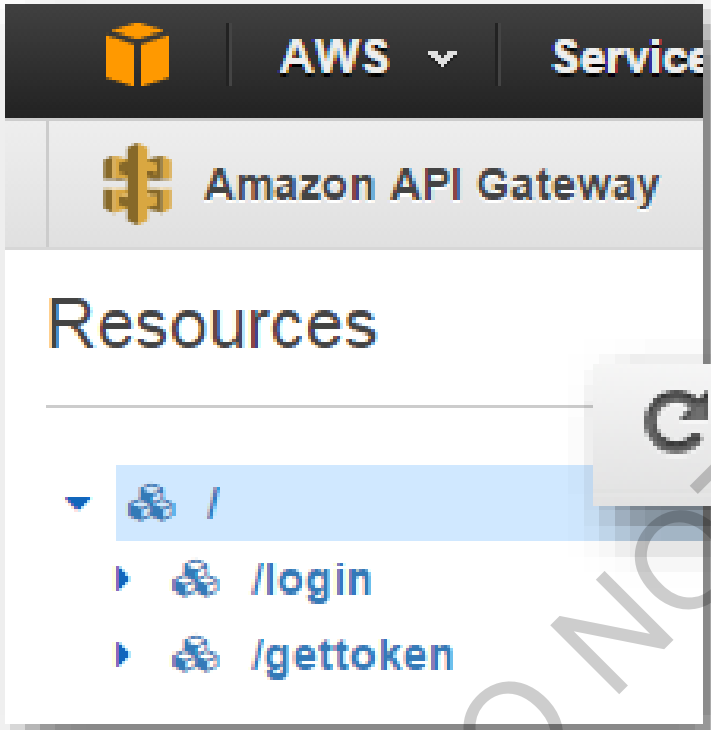
- **Deploy, Generate SDKs & Monitor your API**
  Generate SDKs for iOS/Android & Javascript to streamline API development. Visually monitor calls to your services using Amazon CloudWatch for performance metrics

Amazon API Gateway - concepts

- Resources

  A resource is a logical entity that can be accessed in an API via its **resource path**

Amazon API Gateway - concepts



This is the **resource path**

login is a resource in the **my.api.com** API

Amazon API Gateway - concepts

- **Resources**
  A resource is a logical entity that can be accessed in an API via its **resource path**

- **Methods**
  A method is identified by the combination of a **resource path** and an HTTP verb, such as **GET**, **POST**, and **DELETE**

- **Models**
  A model defines the **format**, also known as the **schema** or **shape**, of some data inbound or outbound from the API

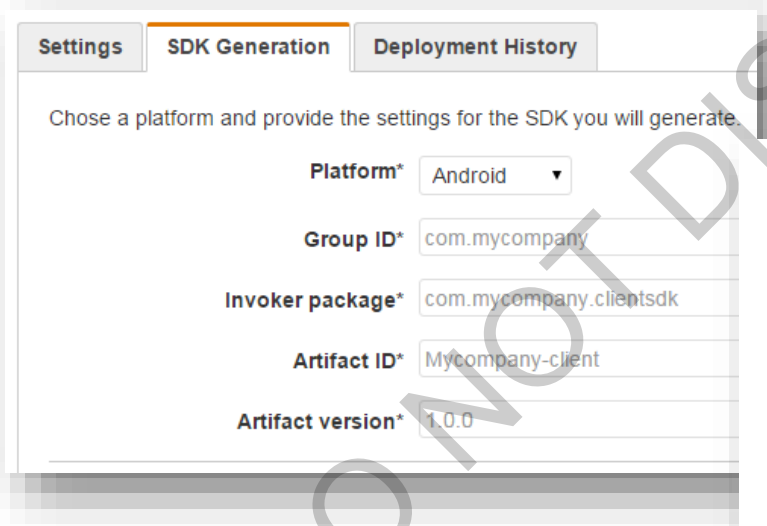
Amazon API Gateway – Generating SDKs

- Generate SDKs from your API definition

  API Gateway creates classes based on the models you define for your methods

iOS

Android




```
/**
 *
 *
 * @param body
 * @return LoginResponseModel
 */
@Operation(path = "/login", method = "POST")
LoginResponseModel loginPost(
        LoginRequestModel body
);
```

Custom IdP – *Login* Lambda function

```
//
// Authenticate this username/password - you would call out
// password database for this - this is only for demonstrat
//
if ( event.username === "username" && event.password === "password")
{
    context.succeed({ succeeded: true, message: "OK" });
    console.log("Authentication succeeded");
}
else
{
    //
    // FAILED to authenticate!
    //
    context.succeed({ succeeded: false, message: "BootcampIdP has failed
}
```

INTERFACE

```
IN:  username
IN:  password
OUT: succeeded
OUT: message
```

NOTE: This code is not production ready! There is no authentication in use – we have skipped this in this Bootcamp, to make it more clear for instructional purposes, but you must add encryption when you use this in production.

Custom IdP – *GetToken* Lambda function

NOTE: This code is not production ready! There is no authentication in use – we have skipped this in this Bootcamp, to make it more clear for instructional purposes, but you must add encryption when you use this in production.

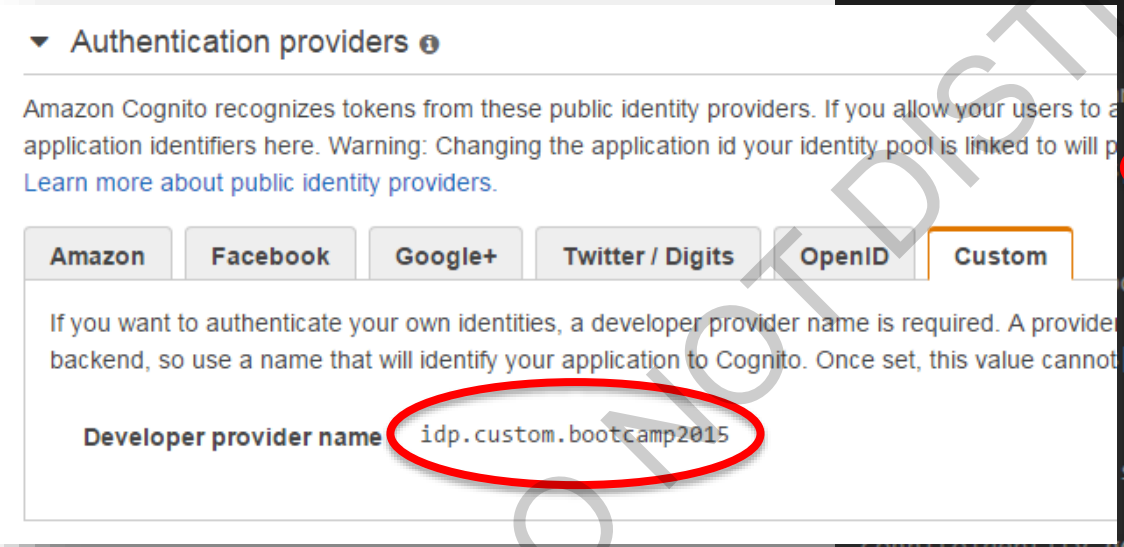
```
INTERFACE
    IN:  bootcampAuth
    IN:  facebookAuth
    IN:  cognitoId
    OUT: errorMessage
    OUT: errorType
    OUT: identityId
    OUT: token
```

getOpenIdTokenForDeveloperIdentity

AWS API call to generate the OpenID token from the supplied IdentityPoolId & provided logins

```
var cognitoidentity = new AWS.CognitoIdentity();
var params =
{
    IdentityId      : event.cognitoId,
    IdentityPoolId  : COGNITO_IDENTITY_POOL_ID,
    Logins          : {}
};

if ( event.bootcampAuth !== null && event.bootcampAuth !== "")
{
    params.Logins["idp.custom.bootcamp2015"] = event.bootcampAuth;
}

if ( event.facebookAuth !== null && event.facebookAuth !== "")
{
    params.Logins["graph.facebook.com"] = event.facebookAuth;
}

console.log(JSON.stringify(params));

cognitoidentity.getOpenIdTokenForDeveloperIdentity(
    params,
    function (err, data)
    {
```

Custom IdP – *GetToken* Lambda function




```
var cognitoidentity = new AWS.CognitoIdentity();
var params =
{
    IdentityId      : event.cognitoId,
    IdentityPoolId  : COGNITO_IDENTITY_POOL_ID,
    Logins          : {}
}

mpAuth !== null && event.bootcampAuth !== "")

"idp.custom.bootcamp2015"] = event.bootcampAuth;

okAuth !== null && event.facebookAuth !== "")

["graph.facebook.com"] = event.facebookAuth;

stringify(params));

cognitoidentity.getOpenIdTokenForDeveloperIdentity(
    params,
    function (err, data)
    {
```

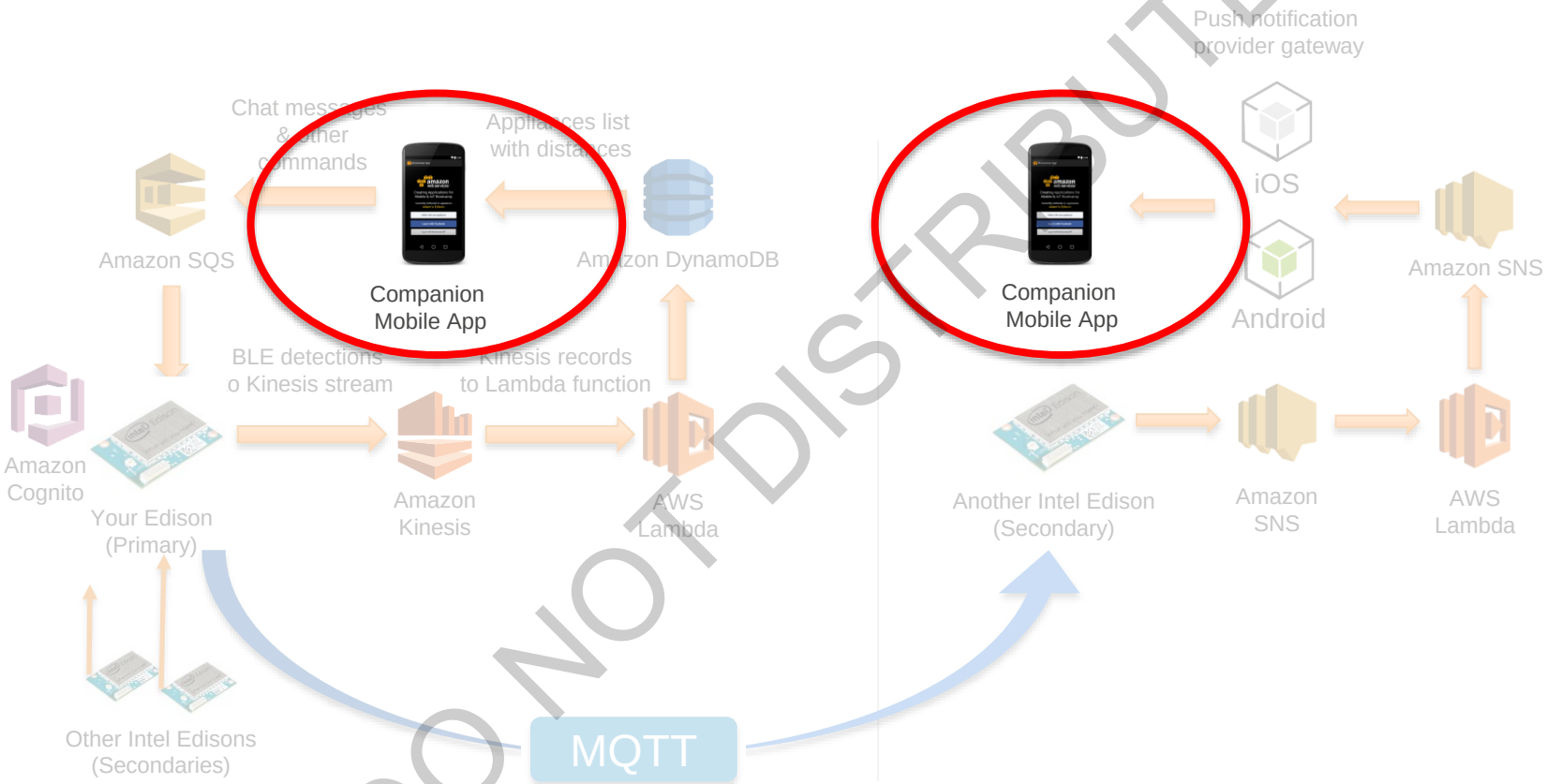
Lab 7 – Developer Authentication Lambda function

```
////////////////////////////////////////////////////////////
//
// Configuration
//
////////////////////////////////////////////////////////////

var COGNITO_IDENTITY_POOL_ID = "ENTER COGNITO IDENTITY POOL ID HERE";
```

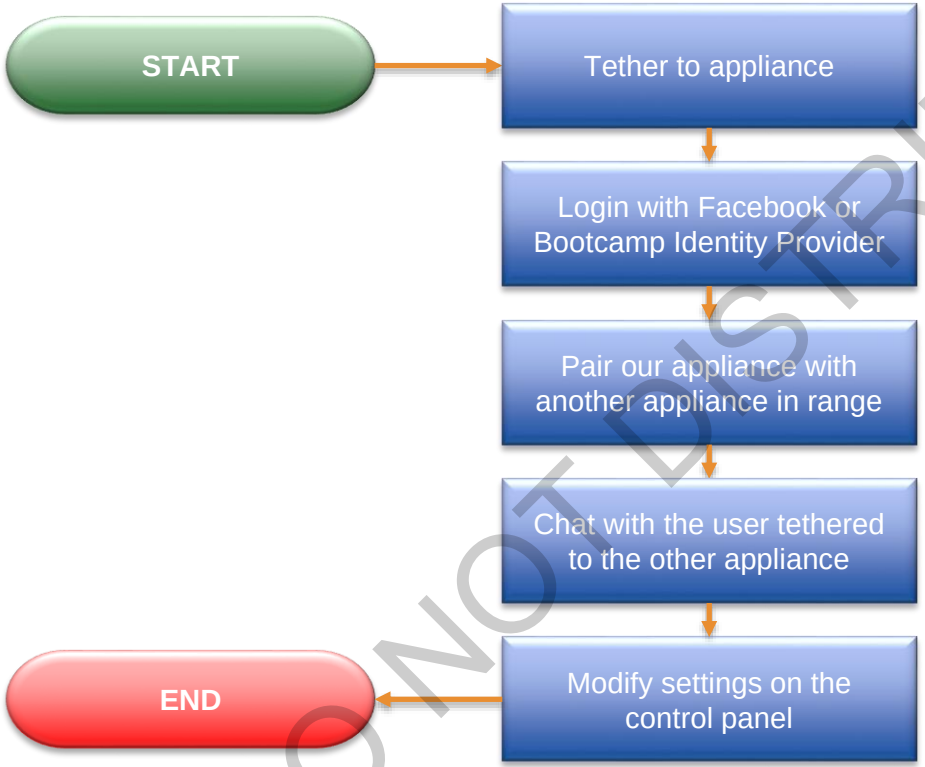
LAB 7

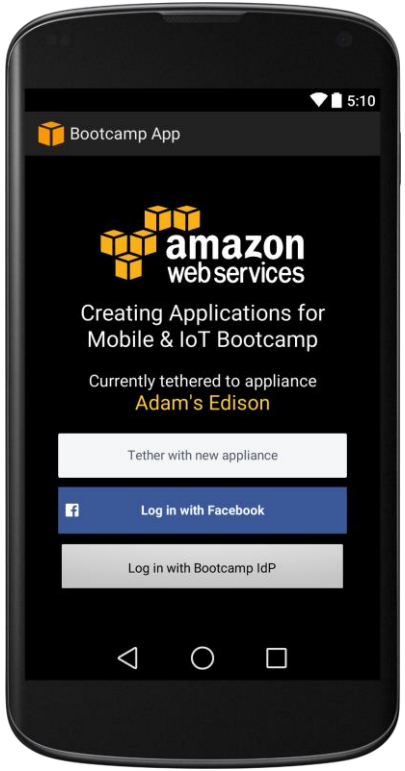
Implement Developer Authentication for our Mobile app, using Amazon API Gateway and AWS Lambda functions

Sign-posting

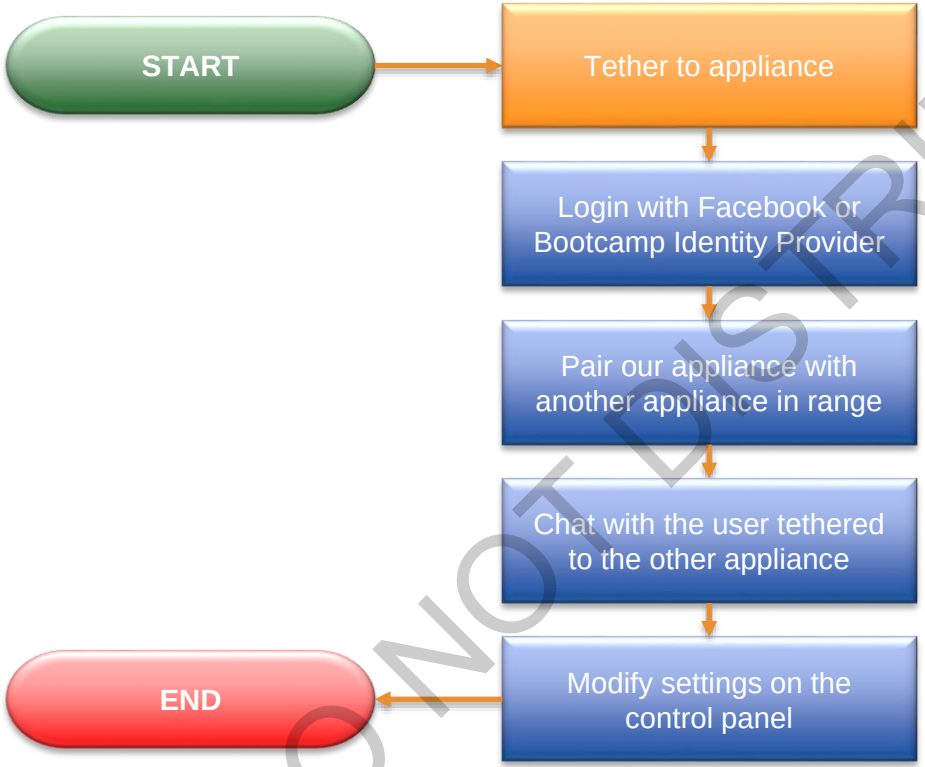


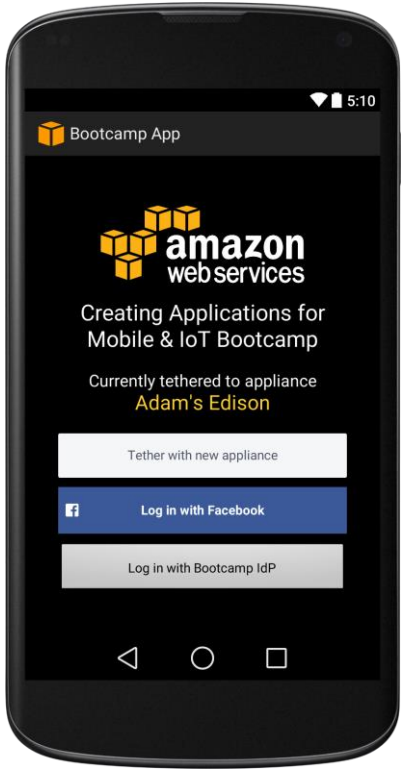
Recap: User Workflow





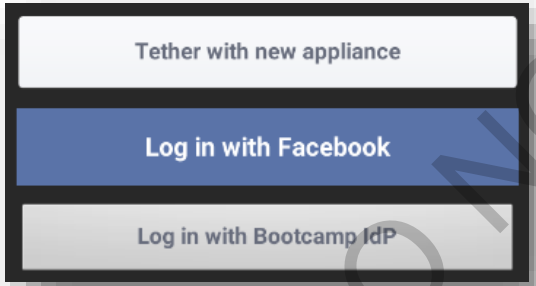
Recap: User Workflow



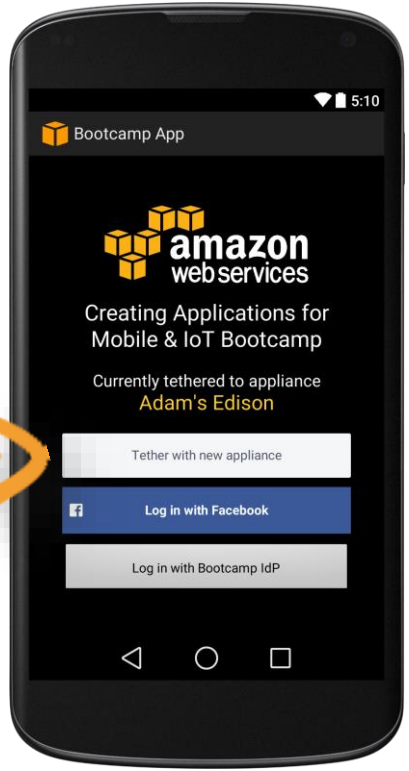


The ‘login’ panel – step 1: tethering

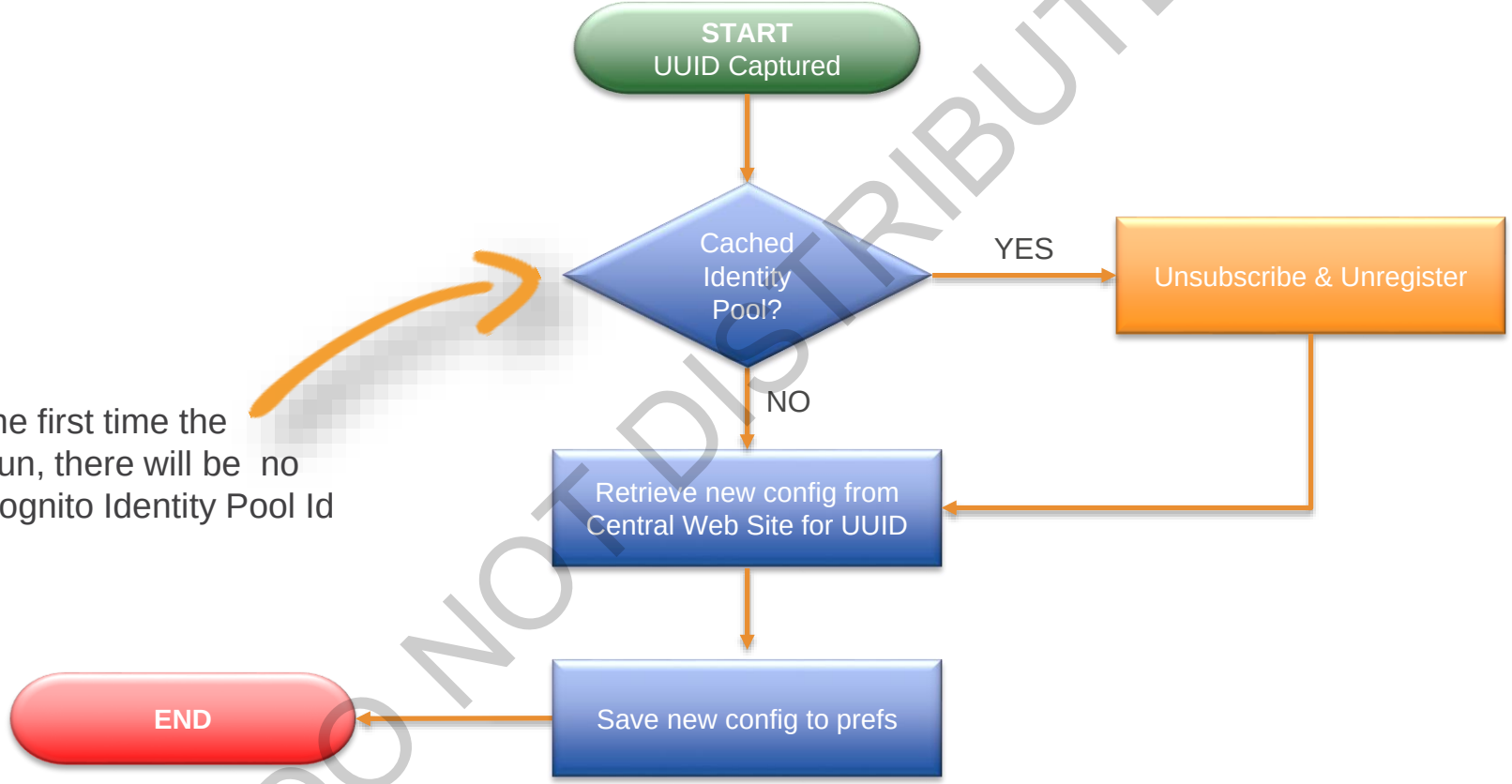
‘Tethering’ associates the mobile application with a specific IoT device identified by a UUID, downloading the configuration from the Central web site so the mobile app knows about the Amazon SQS queue, Amazon Cognito Identity Pool and other configuration items



Step 1 – Tether appliance
The mobile application needs to be tethered to a specific IoT device before you can log in and interact with the appliance

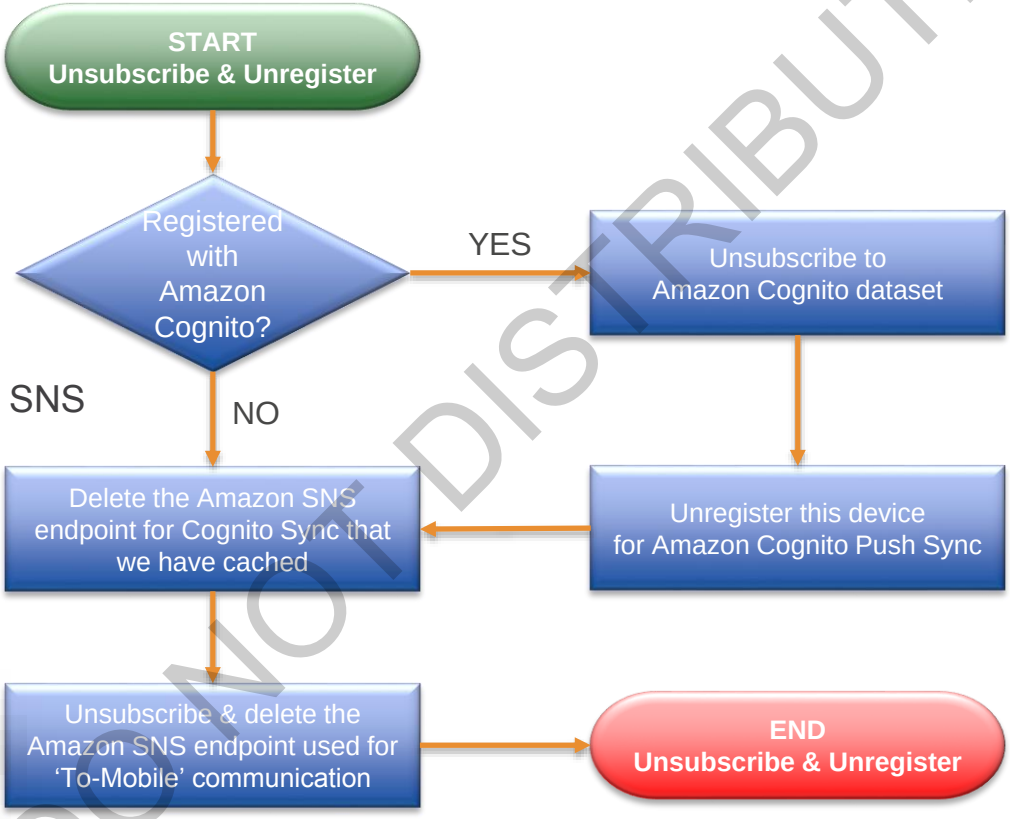


Tether workflow



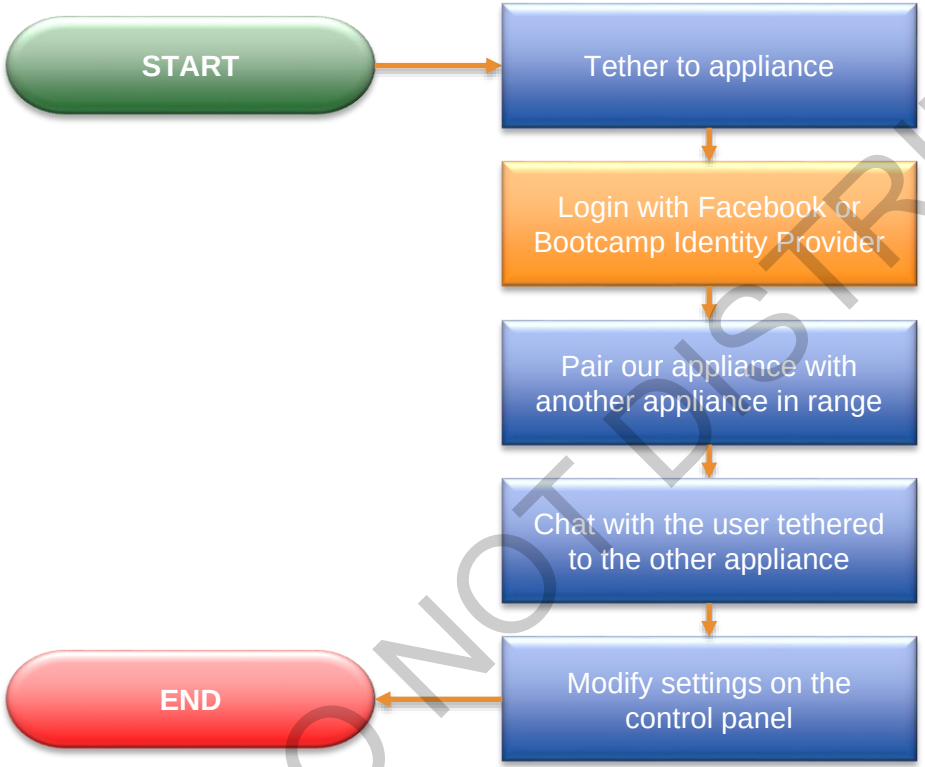
If this is the first time the app has run, there will be no cached Cognito Identity Pool Id

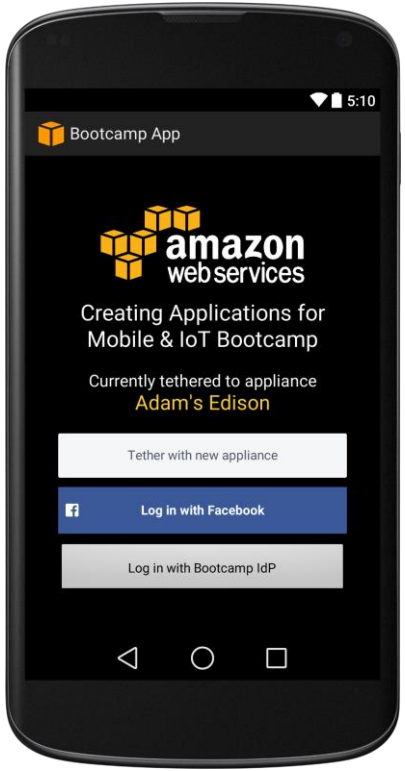
Tether workflow – Unsubscribe & Unregister



We 'clean up' all Amazon SNS endpoints and subscriptions when we change tethering

Recap: User Workflow

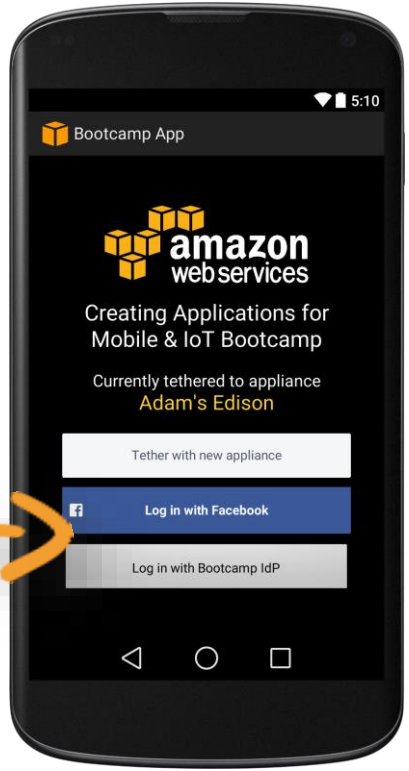




The ‘login’ panel – step 2: authentication

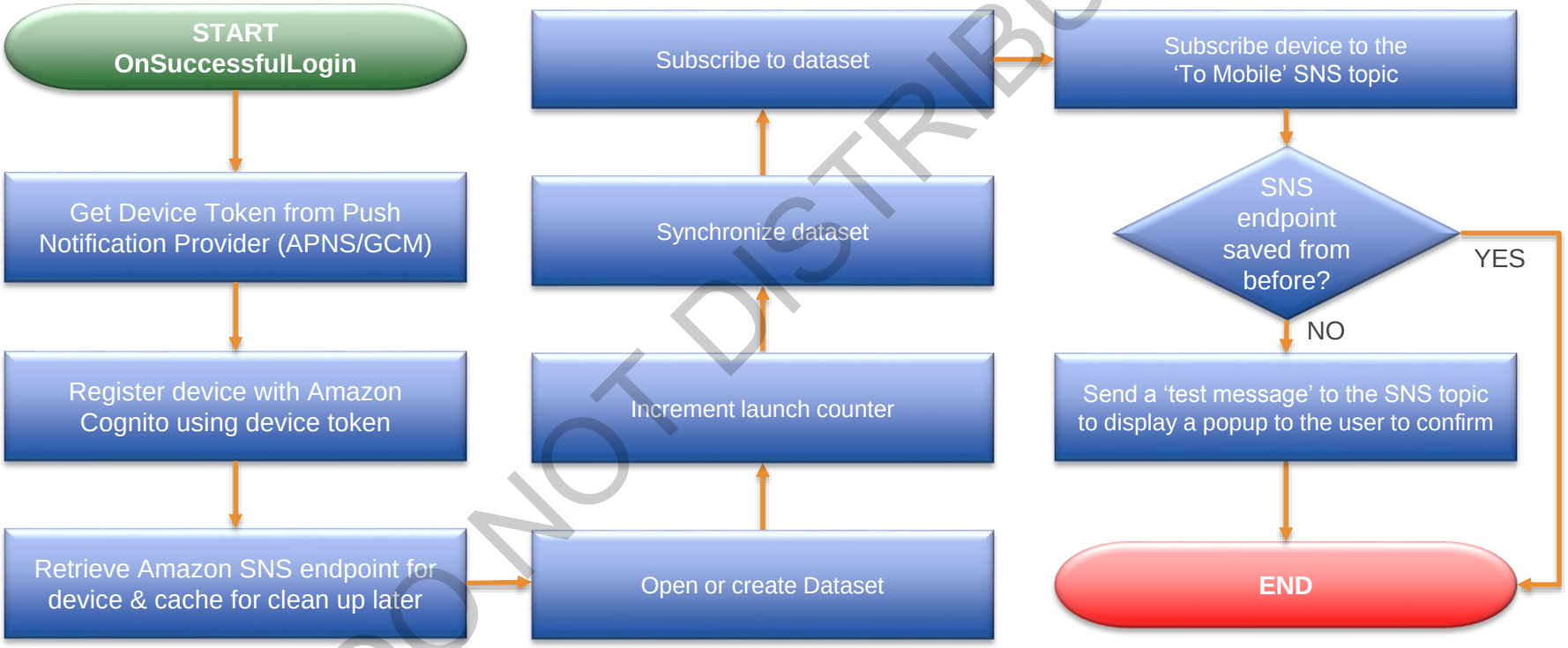
Once tethered, the user can choose to authenticate using Facebook or the Bootcamp Identity Provider (IdP)

You must provide ‘*username*’ for the user name, and ‘*password*’ for the password to successfully authenticate

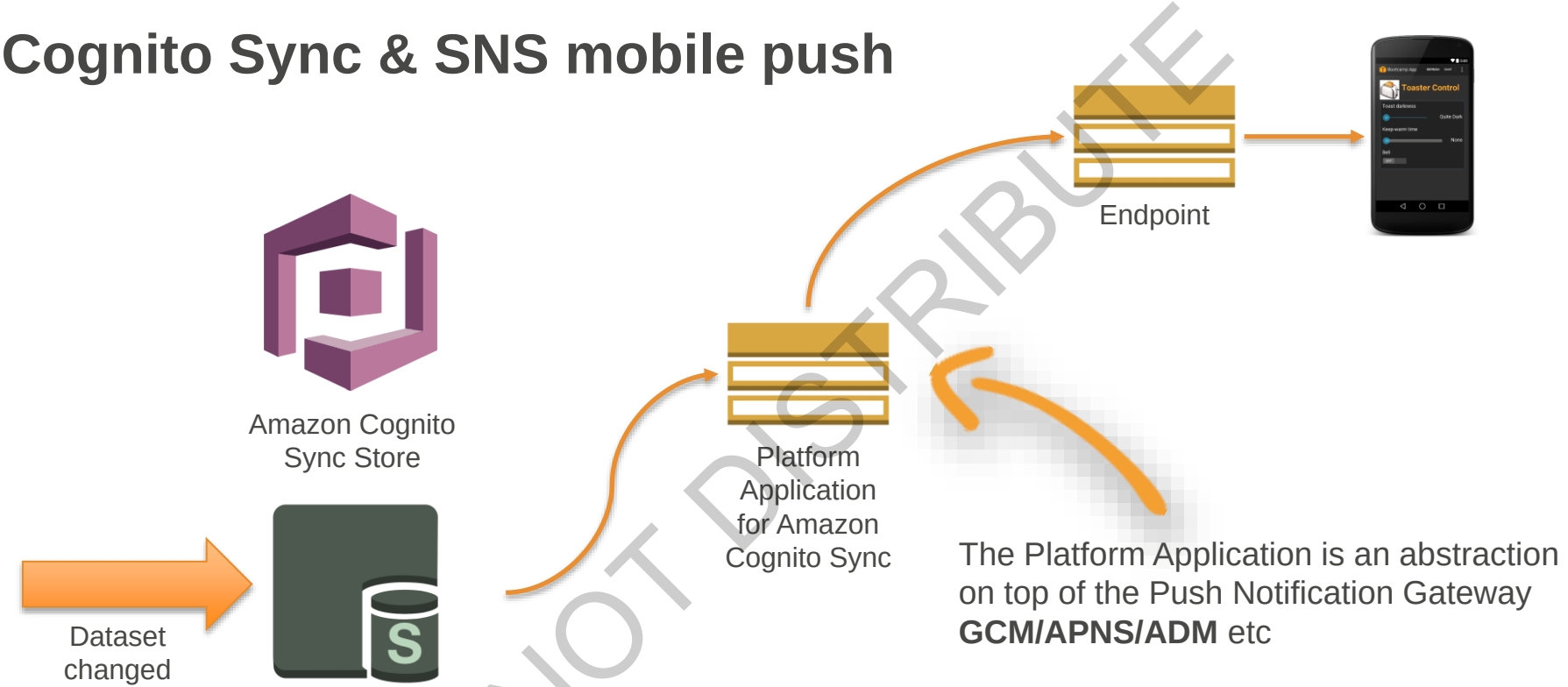


Step 2 – Authenticate
Once authenticated, the app behaves the same regardless of whether the User authenticated with Facebook or the Bootcamp Identity Provider

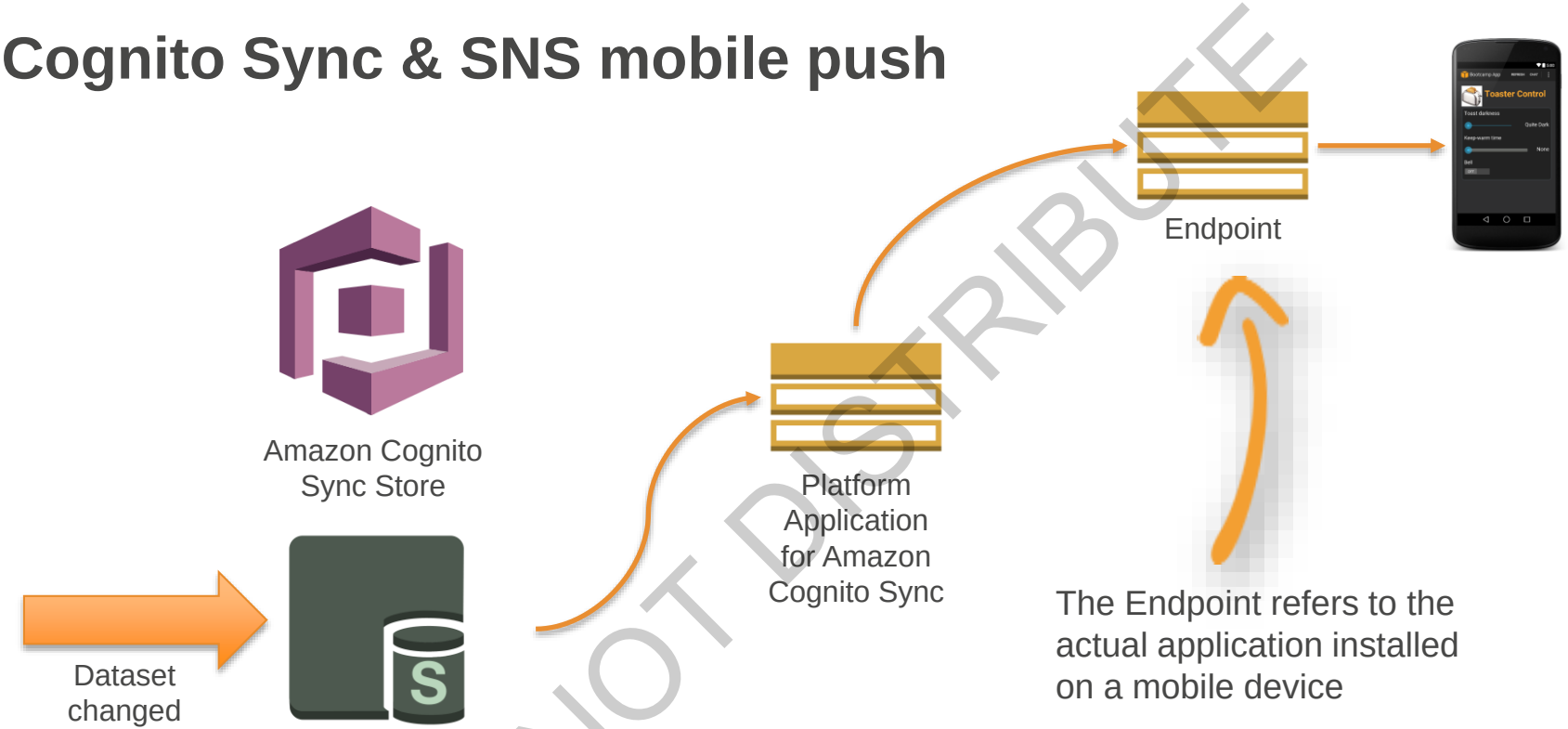
Login workflow – post authentication



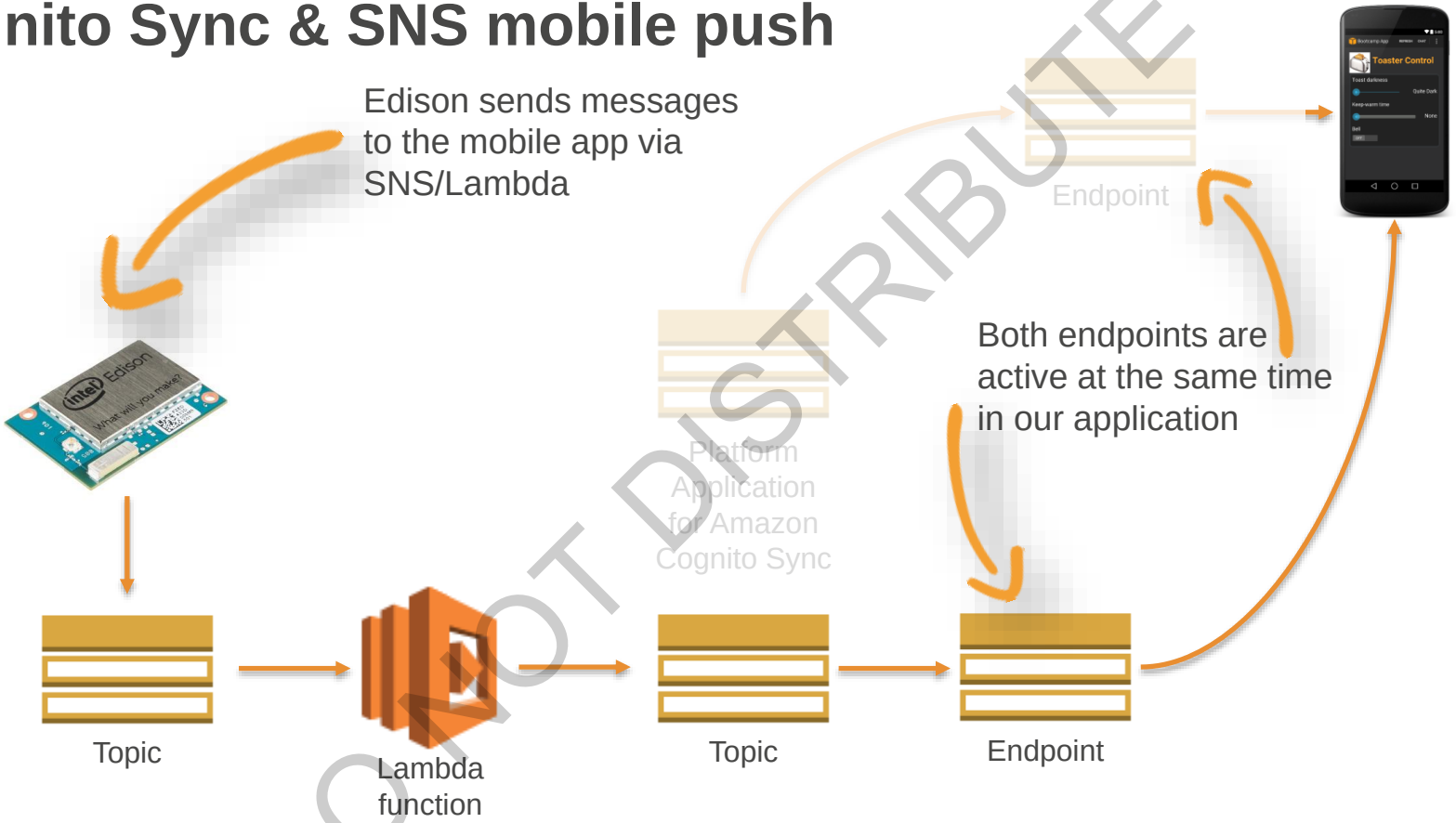
Cognito Sync & SNS mobile push



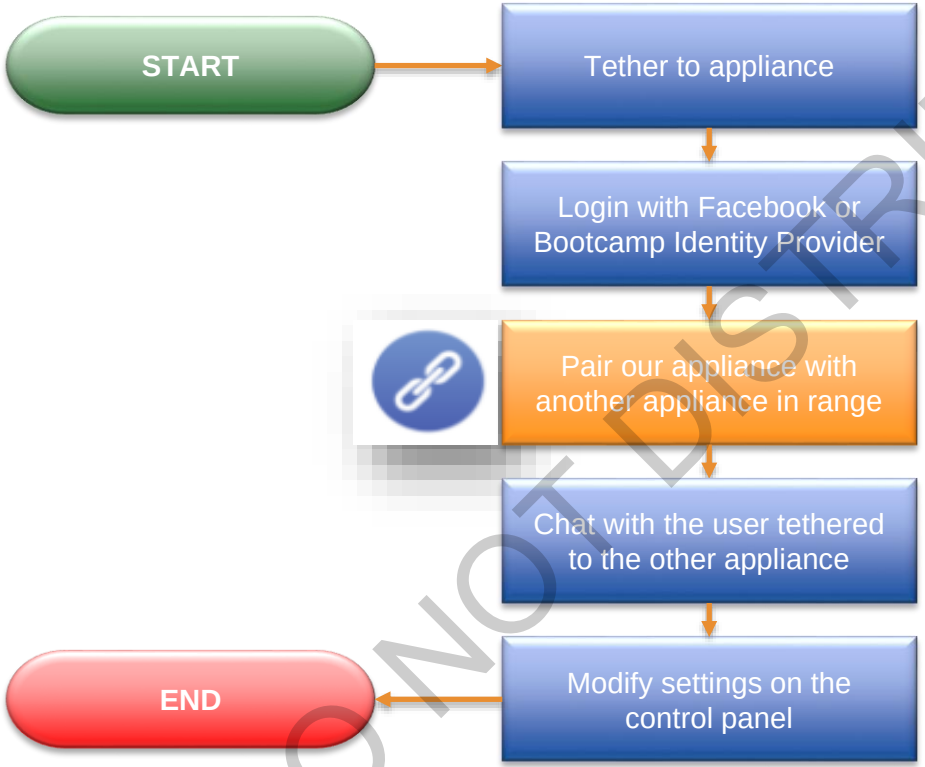
Cognito Sync & SNS mobile push



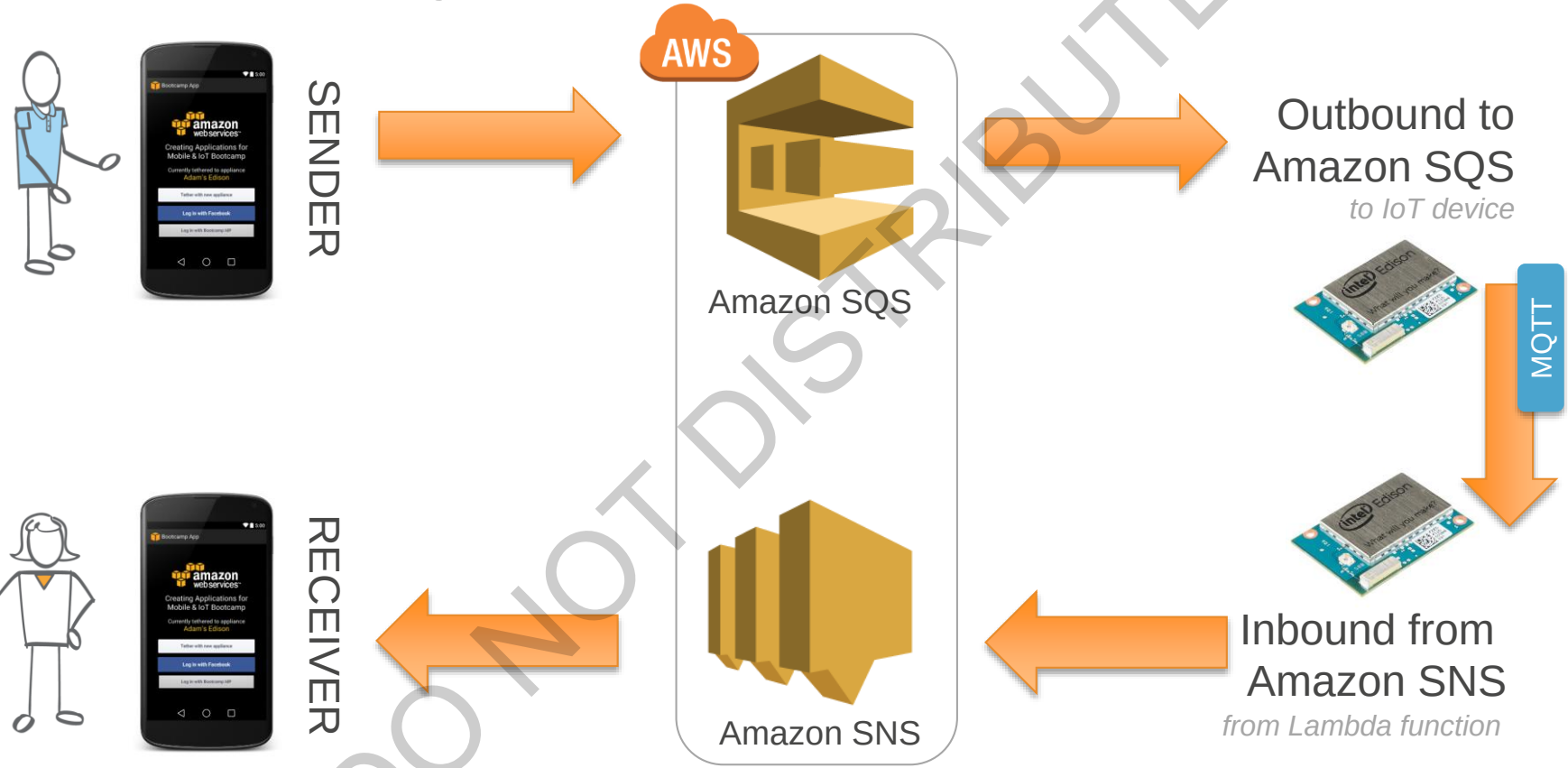
Cognito Sync & SNS mobile push



Recap: User Workflow



Data-flow is easy to remember!



Pairing

Pairing

SQS Queue

Bootcamp2015-Edison-Command

- All command traffic from mobile app is sent to this queue
- Pairing requests, chat messages, chat acknowledgements
- Edison code reads queue with a ‘long poll’ and dispatches messages

Pairing

SQS Queue

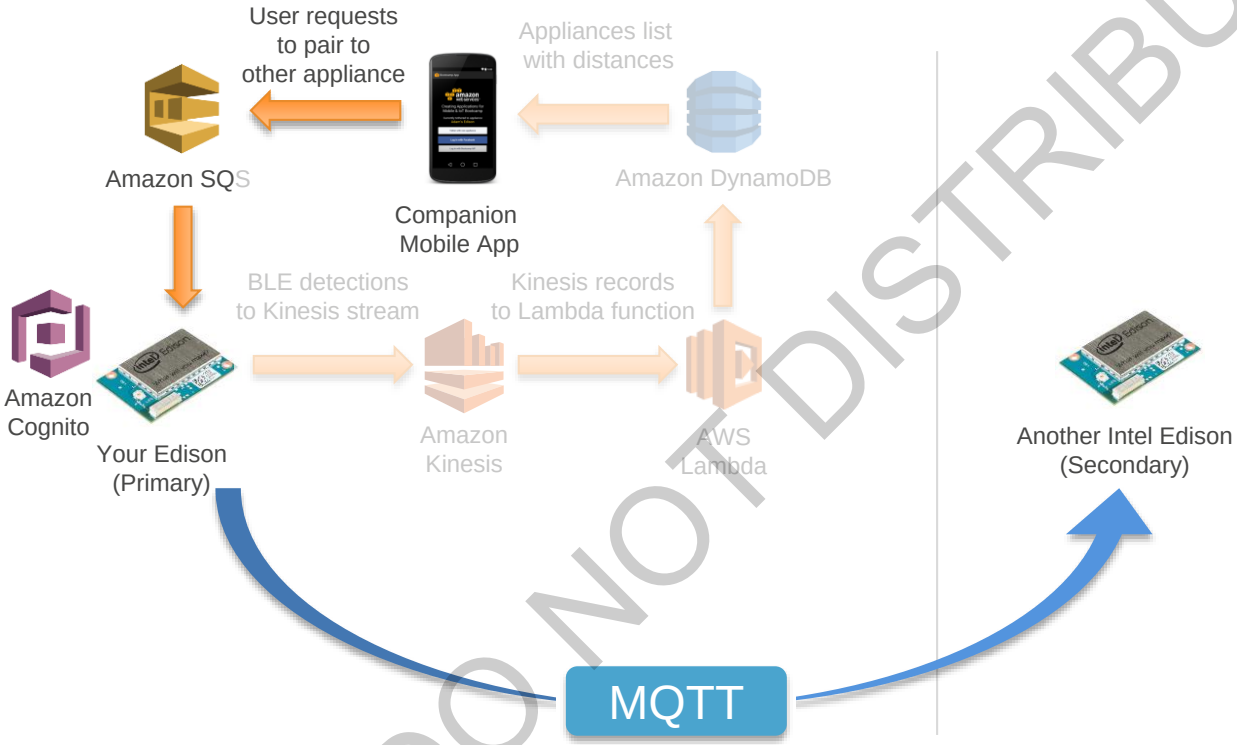
Bootcamp2015-Edison-Command

- Command vocabulary:

  COMMAND_PAIR_REQUEST
  COMMAND_SEND_MESSAGE
  COMMAND_SEND_MESSAGE_ACK

- For a Pair Request, the payload sent to the Edison contains the ***IP Address*** of the target

Pairing



Pairing

MQTT

MQTT exchange

- Edison that initiated the Pairing request connects using MQTT to the IP Address of target
- Each Edison runs an MQTT broker (called RSMB, part of the Yocto distribution)
- On connect, the initiator Edison sends an ‘announcement’ message which causes a COMMAND_PAIR_REQUEST message to be sent from the target Edison to SNS topic

MQTT

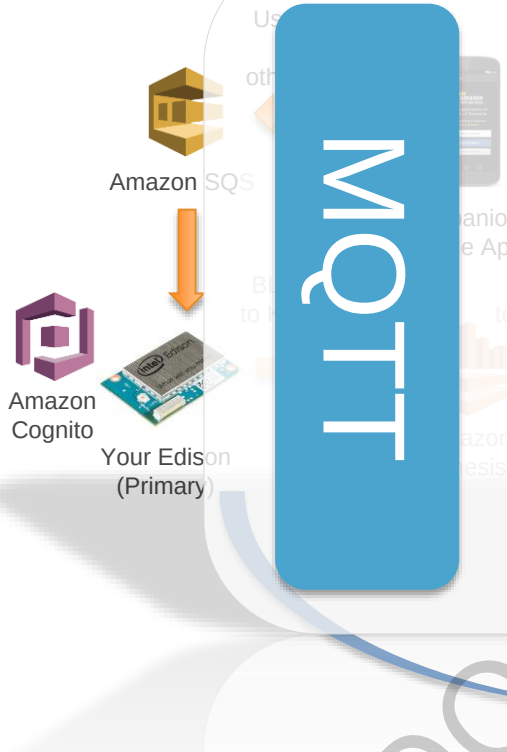
Pairing

MQTT exchange

- On connect, the other Edison performs an ‘auto-pair’ ***by connecting in the opposite direction*** back to the original Edison, as if the user of the mobile app requested a pairing
- We do this so we can establish a ***two-way communication channel***

MQTT

Pairing

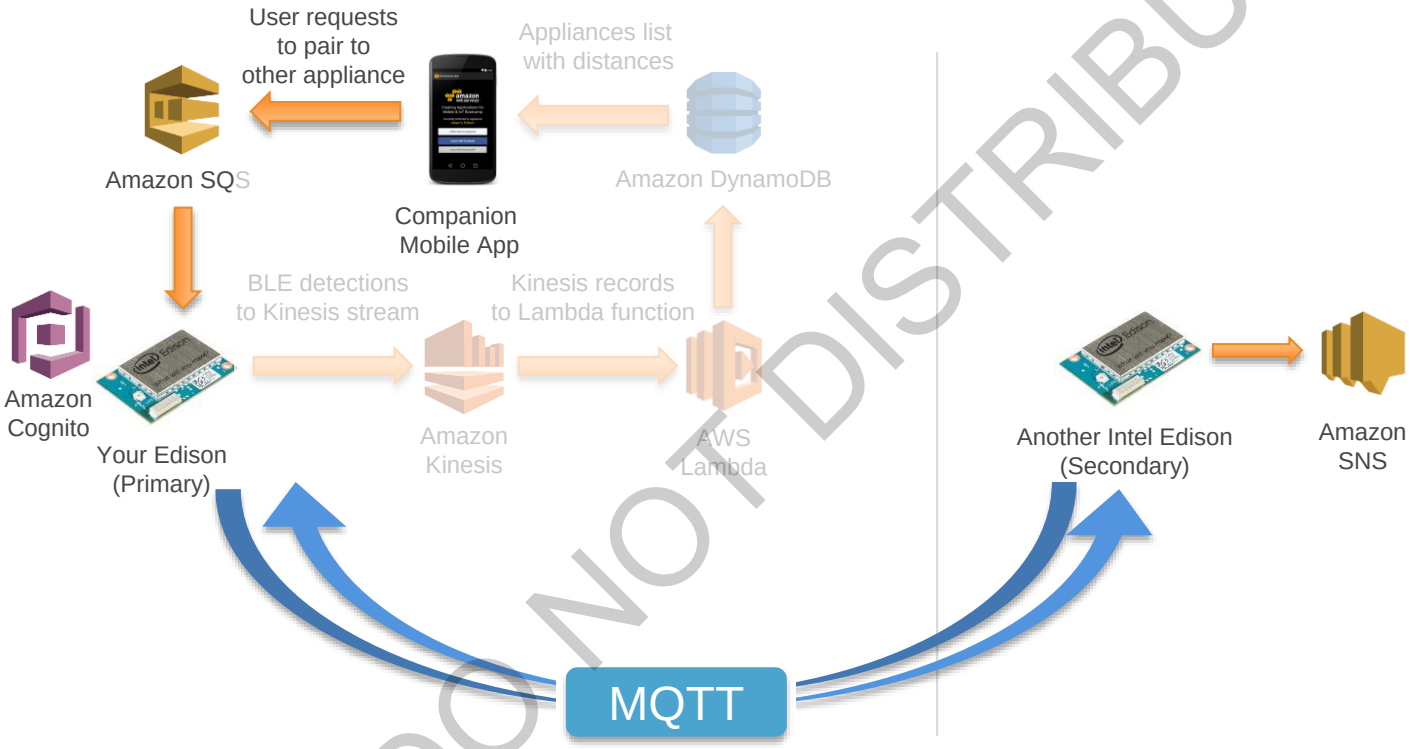


MQTT exchange

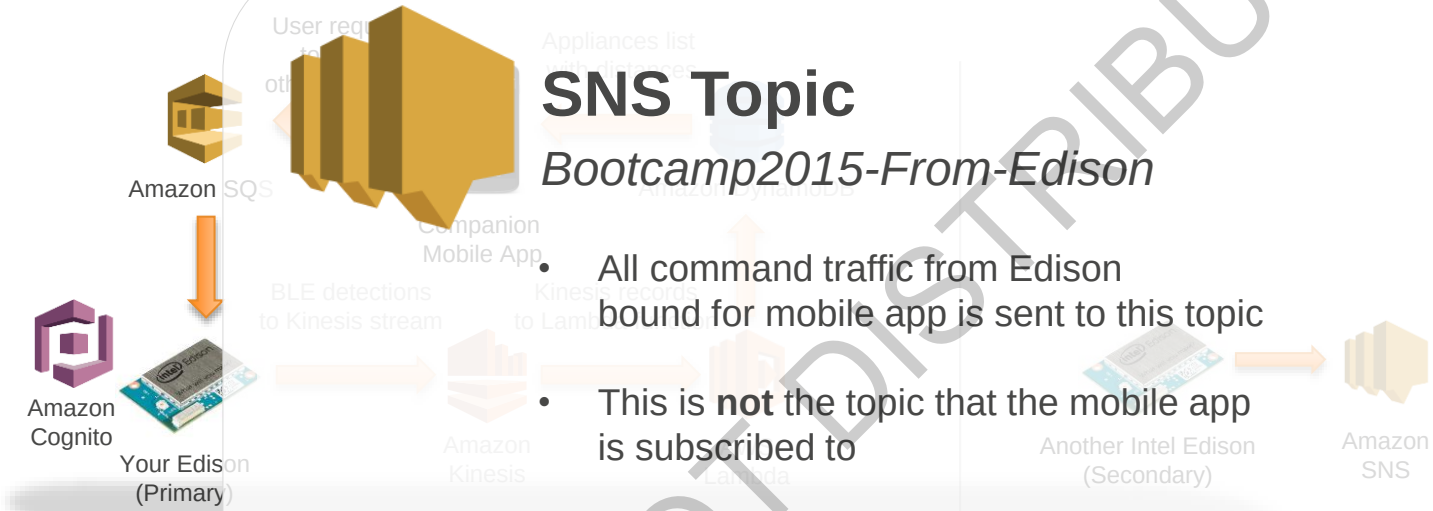
- The Pairing initiator also sends a COMMAND_PEER_LIST_UPDATE message to the SNS topic (the ‘from Edison’ topic)

  This causes the Lambda function to update the Amazon DynamoDB table, and mark the target peer appliances as ‘paired’

Pairing



Pairing

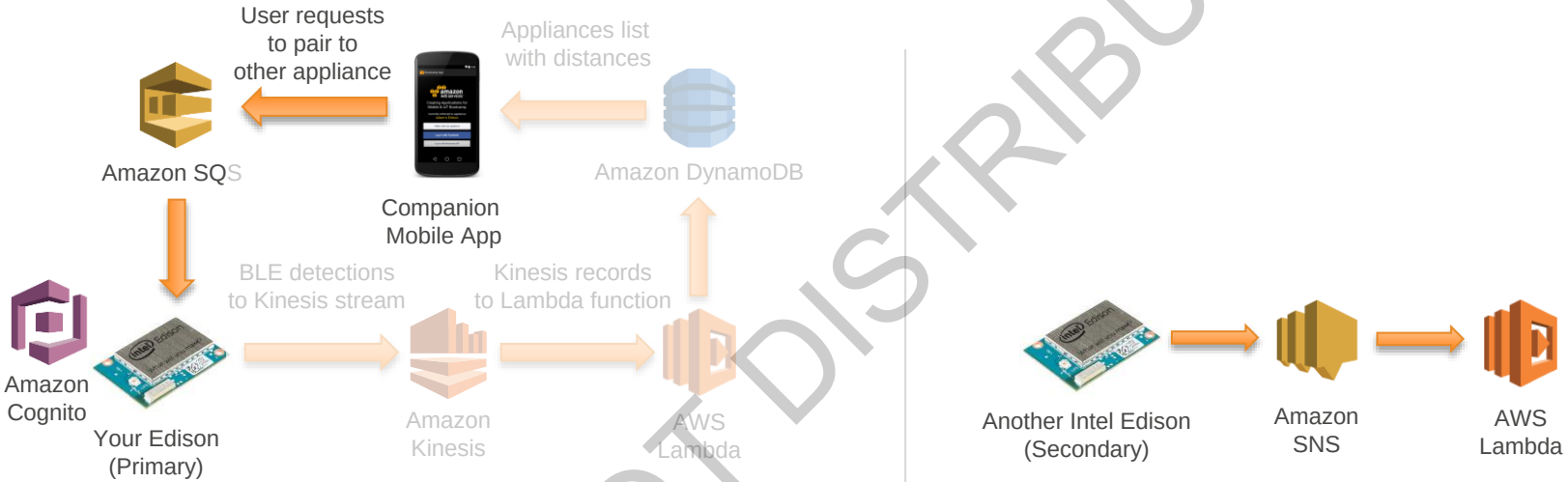


SNS Topic

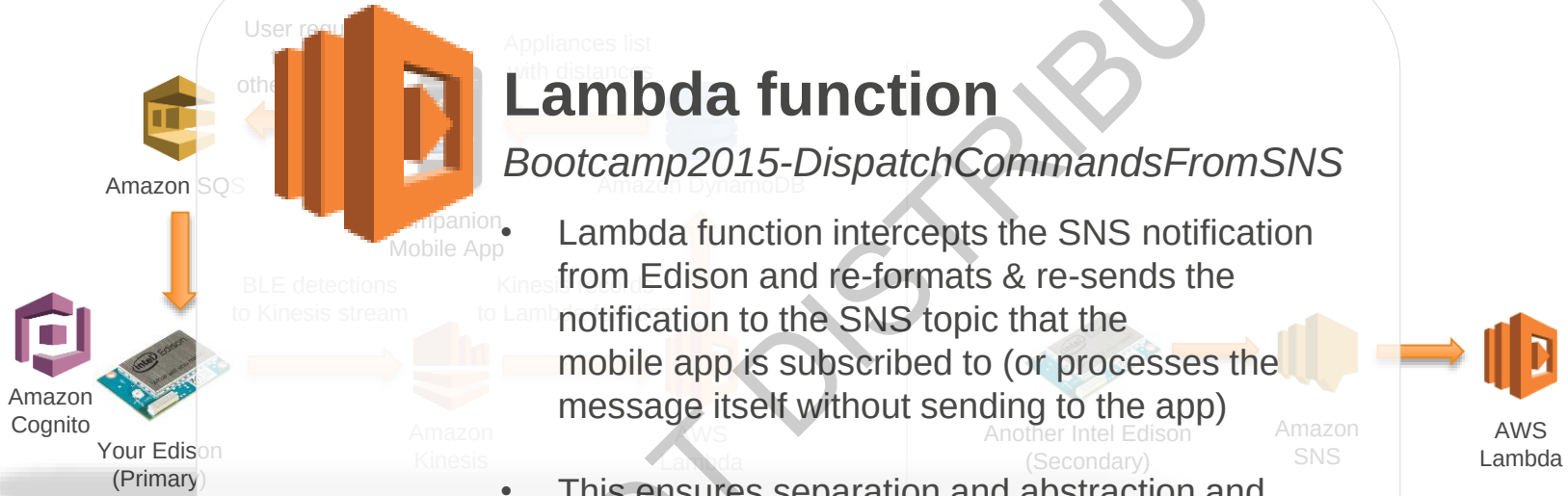
Bootcamp2015-From-Edison

- All command traffic from Edison bound for mobile app is sent to this topic
- This is **not** the topic that the mobile app is subscribed to

Pairing



Pairing

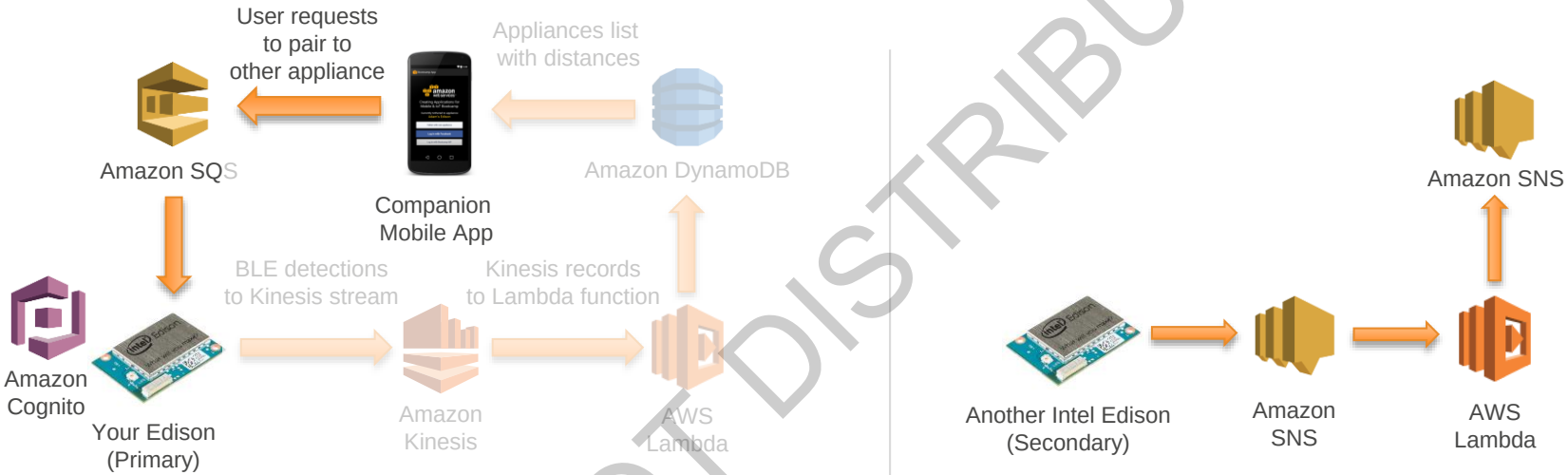


Lambda function

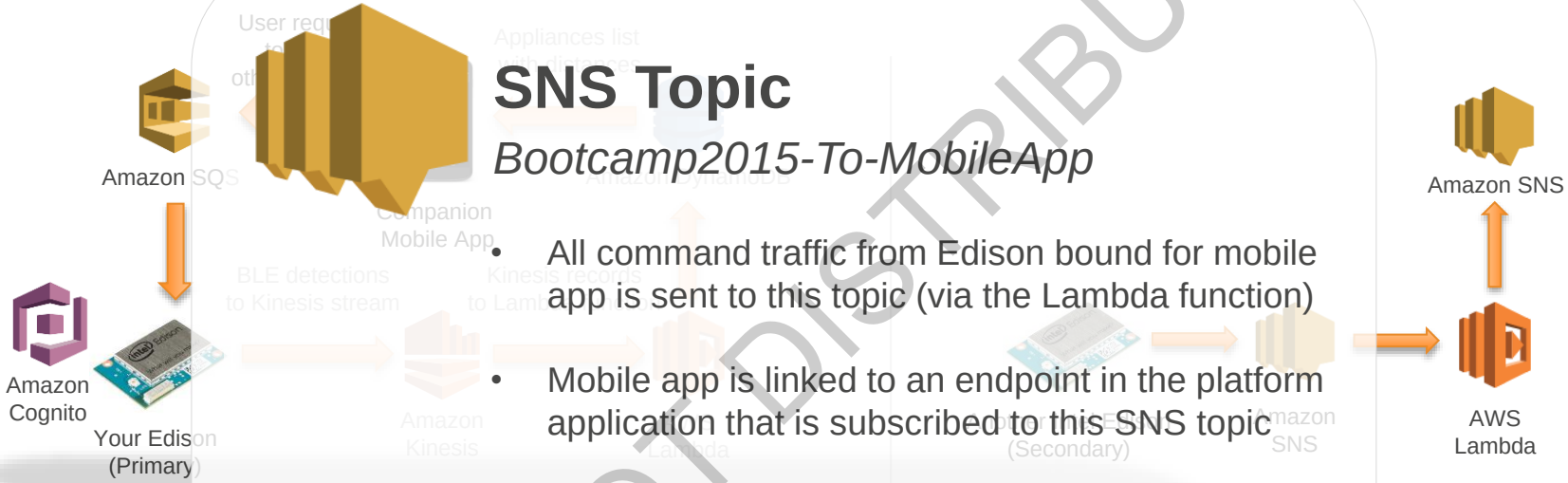
Bootcamp2015-DispatchCommandsFromSNS

- Lambda function intercepts the SNS notification from Edison and re-formats & re-sends the notification to the SNS topic that the mobile app is subscribed to (or processes the message itself without sending to the app)
- This ensures separation and abstraction and allows us to perform other functions on behalf of the Edison (if required)

Pairing



Pairing

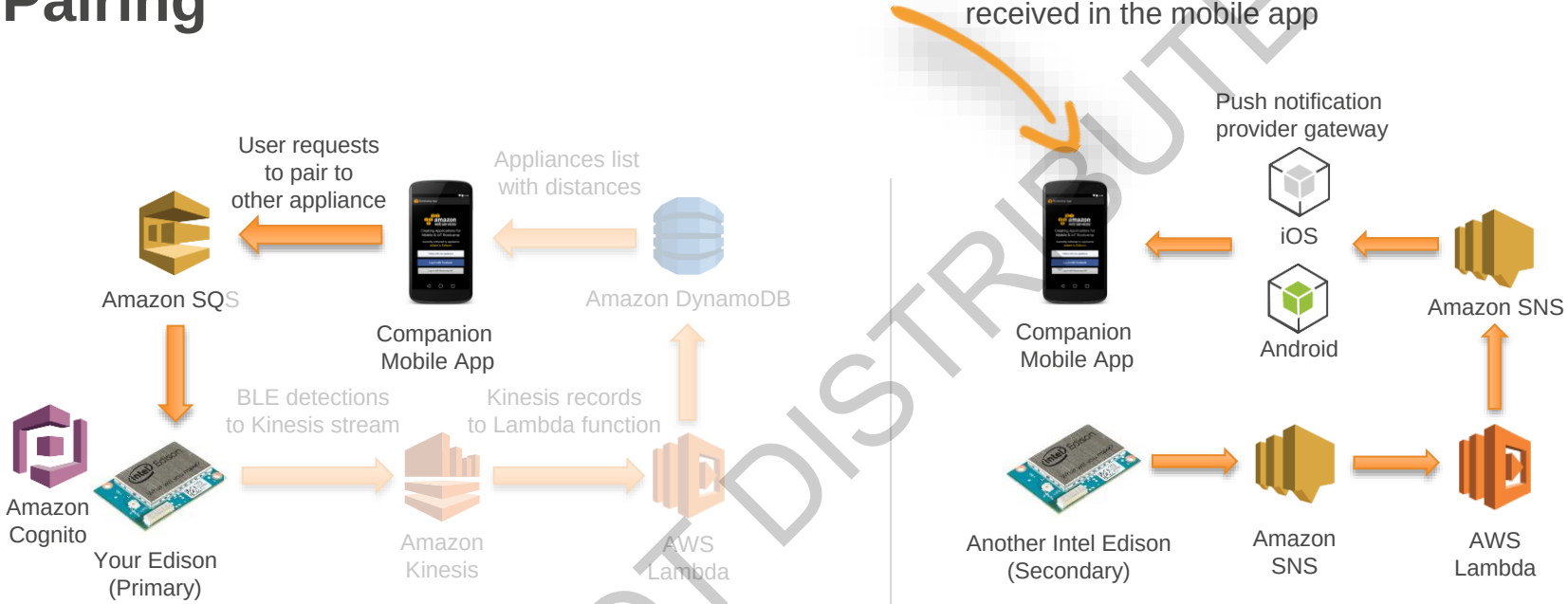


SNS Topic

Bootcamp2015-To-MobileApp

- All command traffic from Edison bound for mobile app is sent to this topic (via the Lambda function)
- Mobile app is linked to an endpoint in the platform application that is subscribed to this SNS topic
- Notifications sent to this topic will be sent as push notifications to the mobile app

Pairing

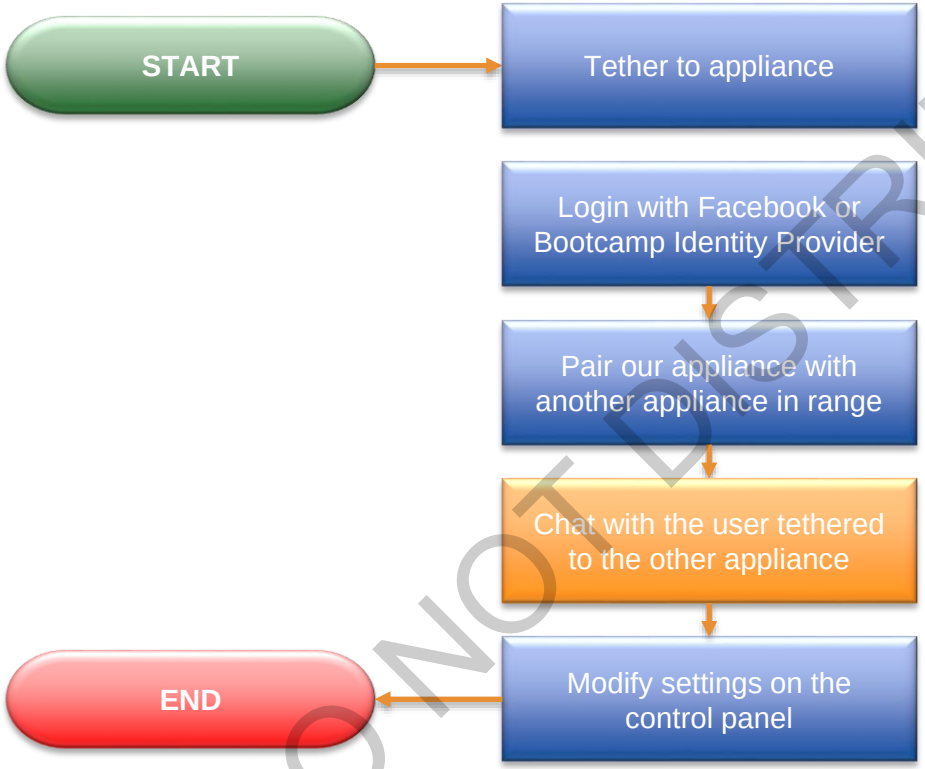


Pairing

Mobile app pairing response

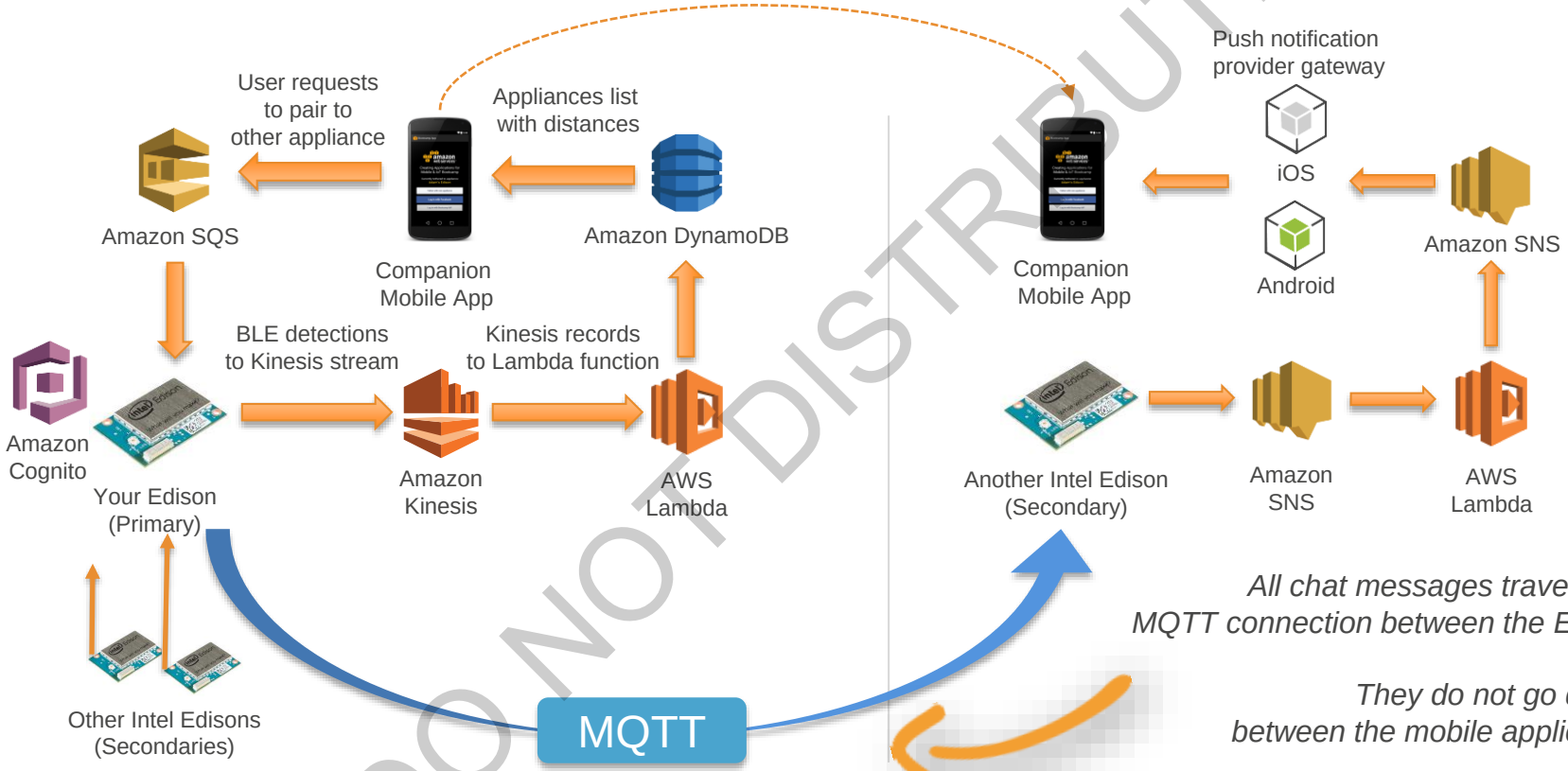
- Mobile app receives a CLOUD_MESSAGE_TYPE_PAIR_COMPLETE message from Lambda via Amazon SNS mobile push
- Shows a notification to the user that pairing has completed
- The pairing was ‘automatic’ as far as the target user’s mobile app is concerned

Recap: User Workflow



Chat

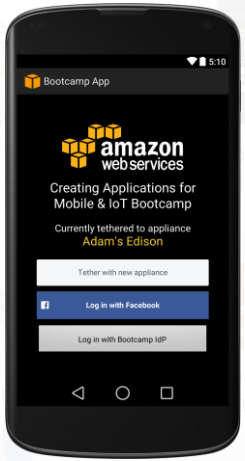
Virtual chat channel via Intel Edison devices



All chat messages traverse the MQTT connection between the Edisons.

They do not go directly between the mobile applications!

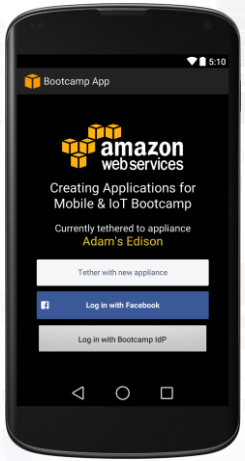
Chat



Chat flow – same as Pairing!

- Data flow is the same as with Pairing request we just reviewed – via SQS, MQTT and SNS
- Mobile app sends via SQS a CLOUD_MESSAGE_TYPE_USER_MESSAGE with user payload & *messageId*

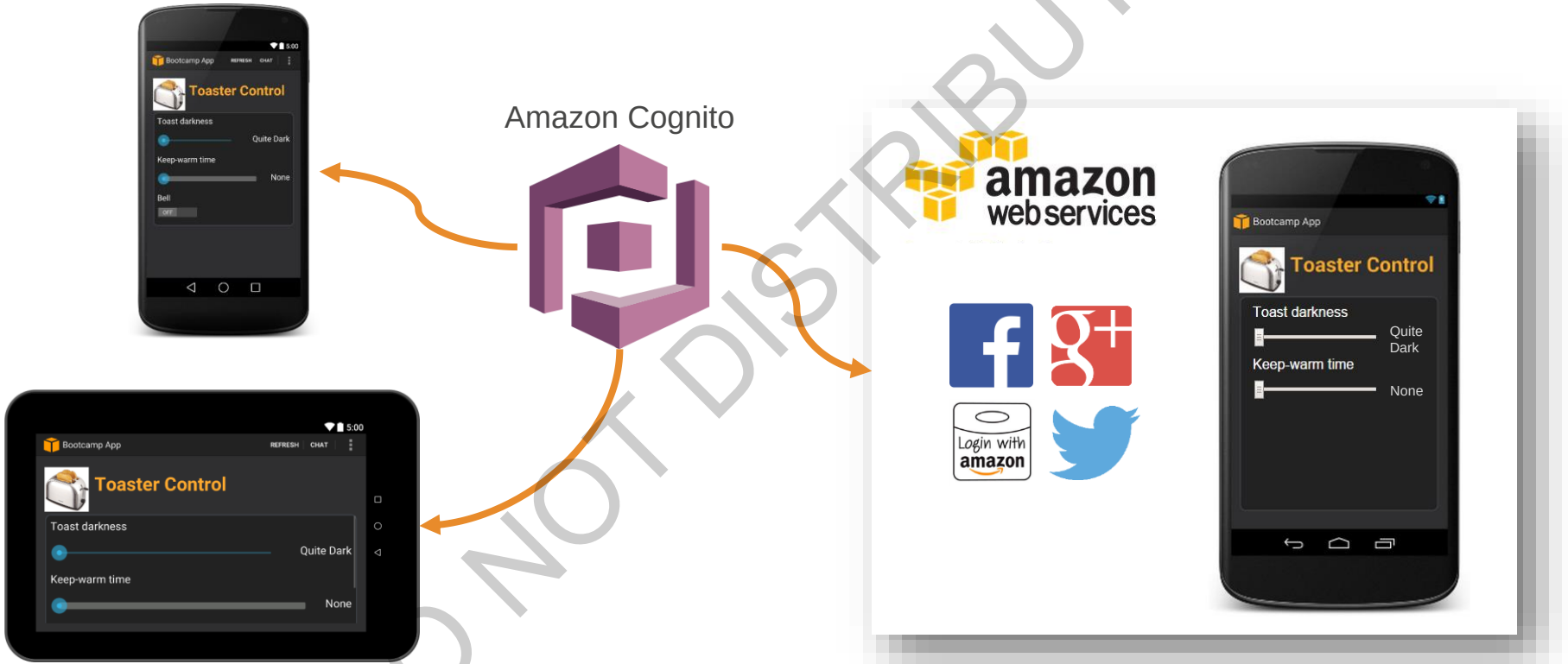
Chat



Chat flow – same as Pairing!

- When another mobile app receives the user message, it responds back with a CLOUD_MESSAGE_TYPE_USER_MESSAGE_ACK for that *messageId* so the other mobile app can show the message as 'received'
- Responses work in the same way, just in the opposite direction

Lab 8 – Implementing the control panels

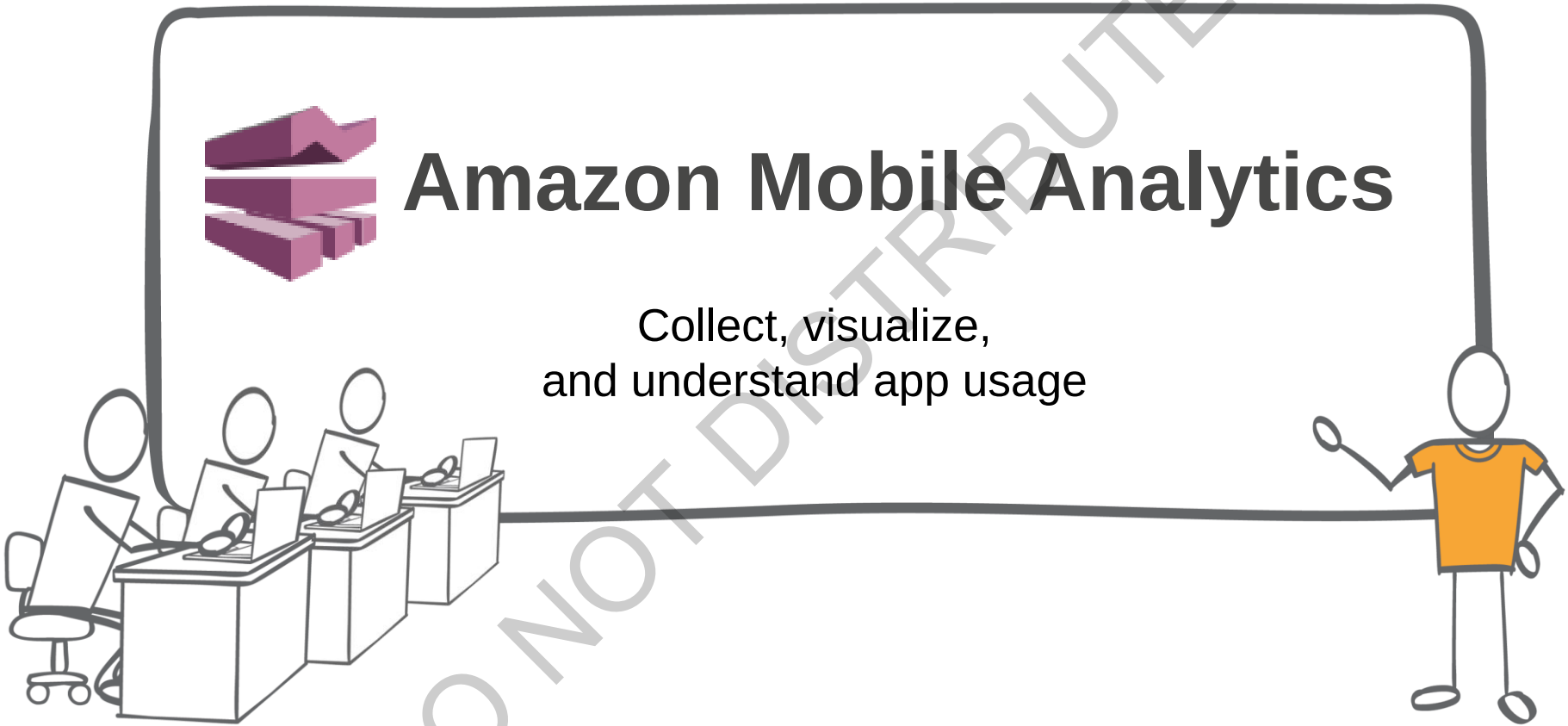


LAB 8

Implementing ‘Control panels’ in the mobile app – Amazon Cognito Sync

Bonus Round: LAB 9

Amazon Mobile Analytics platform

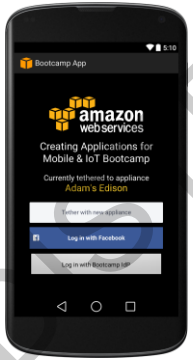


Amazon Mobile Analytics

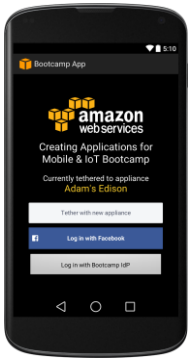
Collect, visualize,
and understand app usage

Amazon Mobile Analytics

Engagement & Monetization

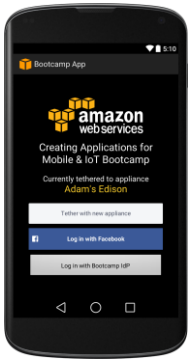


Retention



Behavior

Amazon Mobile Analytics



Engagement + Monetization

Active Users

Sessions

In-app Revenue

Lifetime Value (LTV)

Retention

Post-install Retention Funnel

Behavior

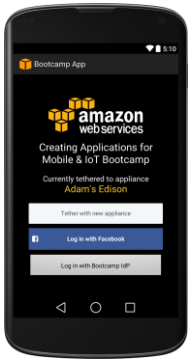
Custom Events

Split By
Platform

API
Submit Events

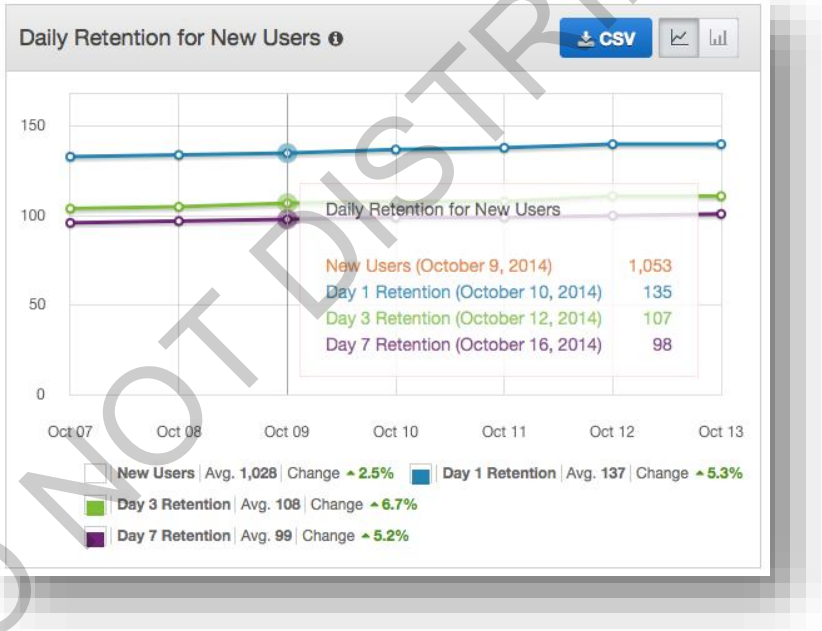
Export
CSV

Amazon Mobile Analytics

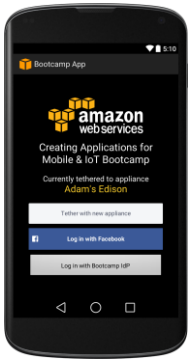


Retention

Post-install Retention Funnel

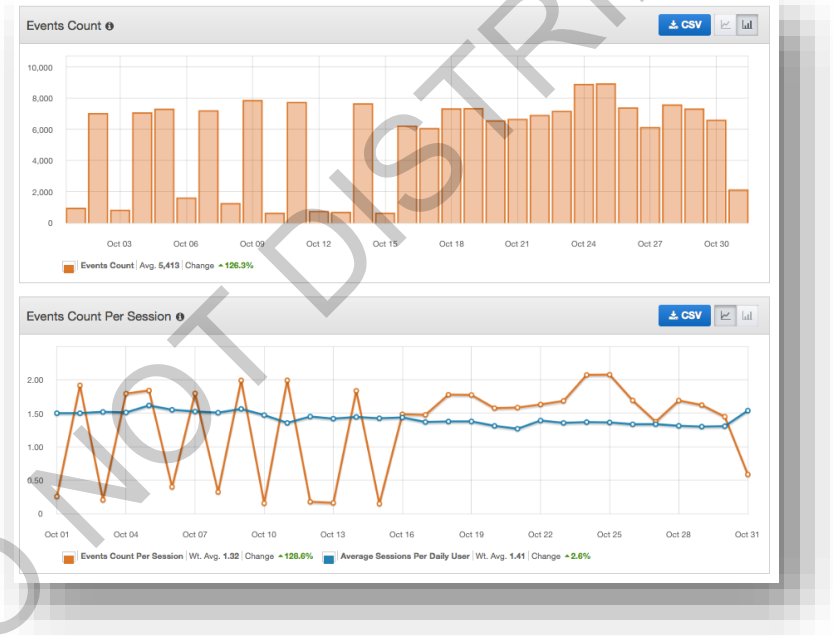


Amazon Mobile Analytics

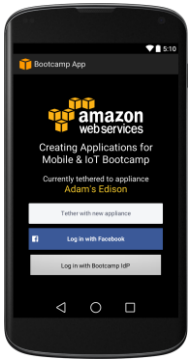


Behaviour

Custom events



Amazon Mobile Analytics



Fast
Event data is processed in < 60 minutes

Affordable
Free 100 MM Events per Month
$1 per MM Events beyond the free tier

Private
You own your data. We do not use your data to report on, or share with 3rd parties

Lab 9: How we use Amazon Mobile Analytics

- Custom events

  We send various metrics from attributes of the Control Panels to Amazon Mobile Analytics

  - Toaster – keep-warm time
  - Kettle –boil time

- In-app purchases

  We simulate in-app purchases and send the detail to Amazon Mobile Analytics so we can plot the average revenue per user

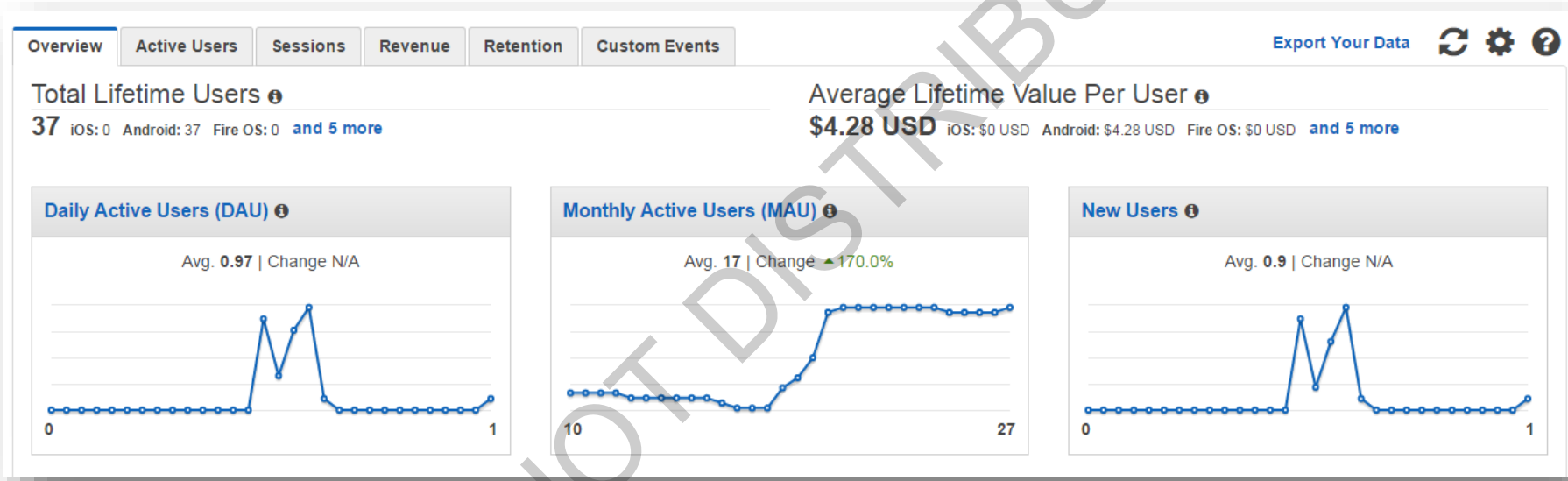
Lab 9: How we use Amazon Mobile Analytics

```
@Override
protected void RecordAnalytics()
{
    super.RecordAnalytics();

    //
    // We record the temperature and duration so we can look at some averages in the console
    //
    AnalyticsEvent syncEvent = Cognito.getAnalytics().getEventClient().createEvent(getPanelNameForAnalytics())
            .withMetric("MaxTemp", (double)CalculateTemperatureSliderValueFromProgress(this.seekbar_temperature.getProgress()

    Cognito.getAnalytics().getEventClient().recordEvent(syncEvent);

    syncEvent = Cognito.getAnalytics().getEventClient().createEvent(getPanelNameForAnalytics())
            .withMetric("BoilTime", (double)CalculateBoilTimeSliderValueFromProgress(this.seekbar_boiltime.getProgress()) );

    Cognito.getAnalytics().getEventClient().recordEvent(syncEvent);
}
```

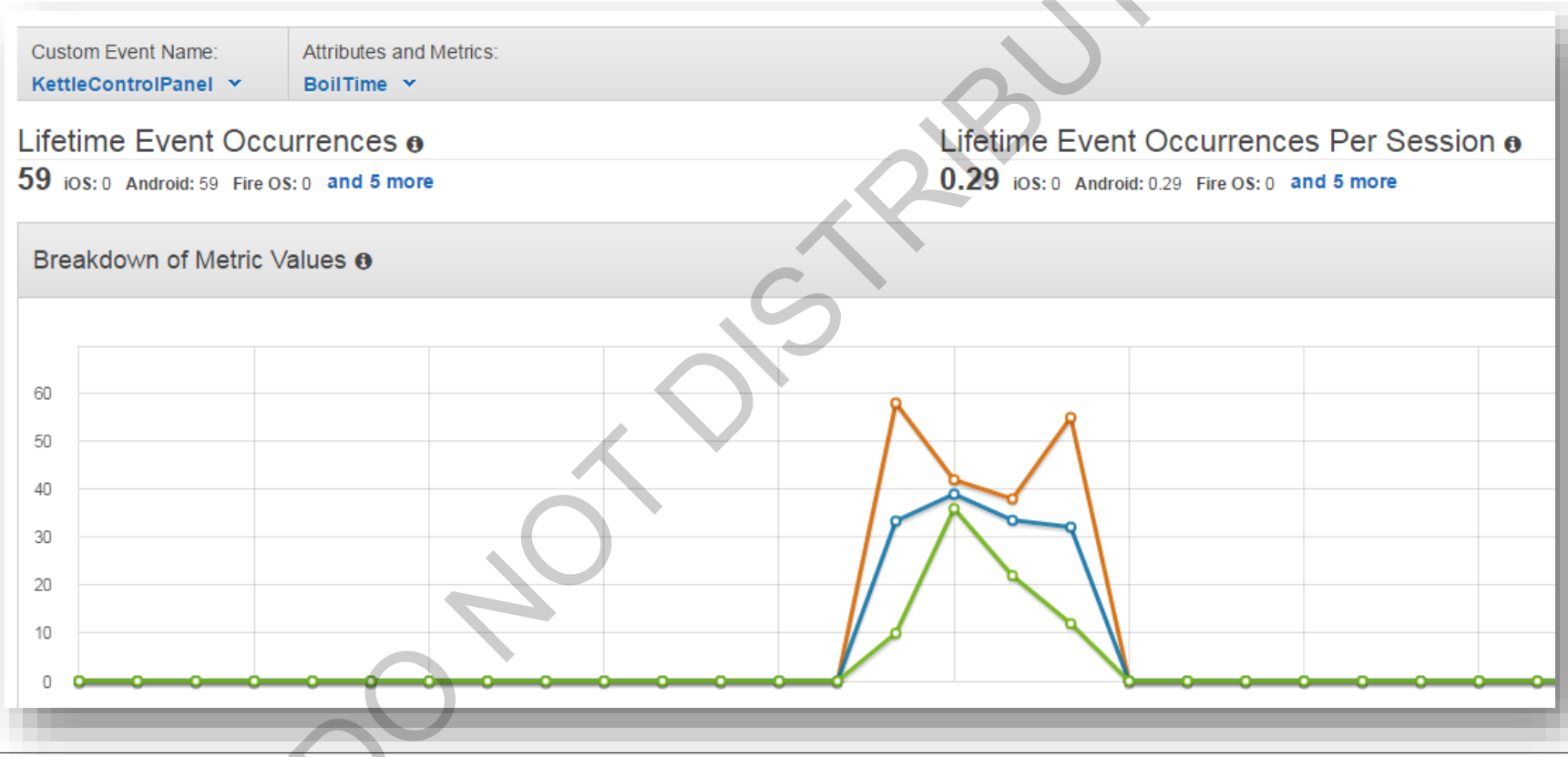
Lab 9: How we use Amazon Mobile Analytics

```
protected void ChargeForPanel(String sku, String price)
{
    // get the event client from your MobileAnalyticsManager instance
    EventClient eventClient = Cognito.getAnalytics().getEventClient();

    // create a builder that can record purchase events for Google Play IAP
    GooglePlayMonetizationEventBuilder builder = GooglePlayMonetizationEventBuilder.create(eventClient);

    // build the monetization event with the product id, formatted item price,
    // transaction id, and quantity
    // product id and formatted item price can be obtained from the JSONObject
    // returned by a product sku request to Google's In App Billing API
    // transaction id is obtained from the Purchase object that is returned upon the
    // completion of a purchase
    // Google IAP currently only supports a quantity of 1
    AnalyticsEvent purchaseEvent = builder.withProductId(sku).withFormattedItemPrice(price)
            .withTransactionId( new Date().getTime() + "" ).withQuantity(1.0).build();

    // record the monetization event in mobile analytics
    eventClient.recordEvent(purchaseEvent);

    Toast.makeText(BaseControlPanel.this,
            "You have been charged $" + price + " to use this panel",
            Toast.LENGTH_LONG
    ).show();
```

Lab 9: How we use Amazon Mobile Analytics



Lab 9: How we use Amazon Mobile Analytics



LAB 9

Amazon Mobile Analytics platform