ALLSEEN ALLIANCE

AllJoyn™: enabling the Internet of Everything

Technical Overview

Marcello Lioy

Qualcomm Connected Experiences, Inc.

What is AllJoyn?

An Open Source API Framework For the Internet of Everything

A way devices and applications publish APIs over a network in a standard way

Why APIs?

- Because this is what software developers understand and work with every day

These APIs are the functionality that the “things” on the network expose to other “things”

- E.g. temperature, time of day, etc.…
- Services and/or devices can compose these APIs to provide whatever set of functionality they require
- The APIs are critical to interoperability between devices and services

How do applications know what APIs are available?

- AllJoyn provides application discovery and fine-grained discovery of the APIs supported by applications
- This is accomplished in a platform and radio-link agnostic way

Overview

AllJoyn implements a “distributed software bus”

- The bus provides the “medium” that enables AllJoyn applications to communicate via published APIs
  - Applications may be firmware on microcontrollers, mobile device “apps” or traditional applications on PCs/servers
- Applications publishing APIs are services, while those consuming the APIs are clients
  - An application can be both a service and a client: this is makes AllJoyn a peer-to-peer system
- Communication is via messages that map directly to APIs in high-level programming languages

Bus formation is ad hoc

- Based on discovery of applications/services
- Abstracts link-specific discovery mechanisms

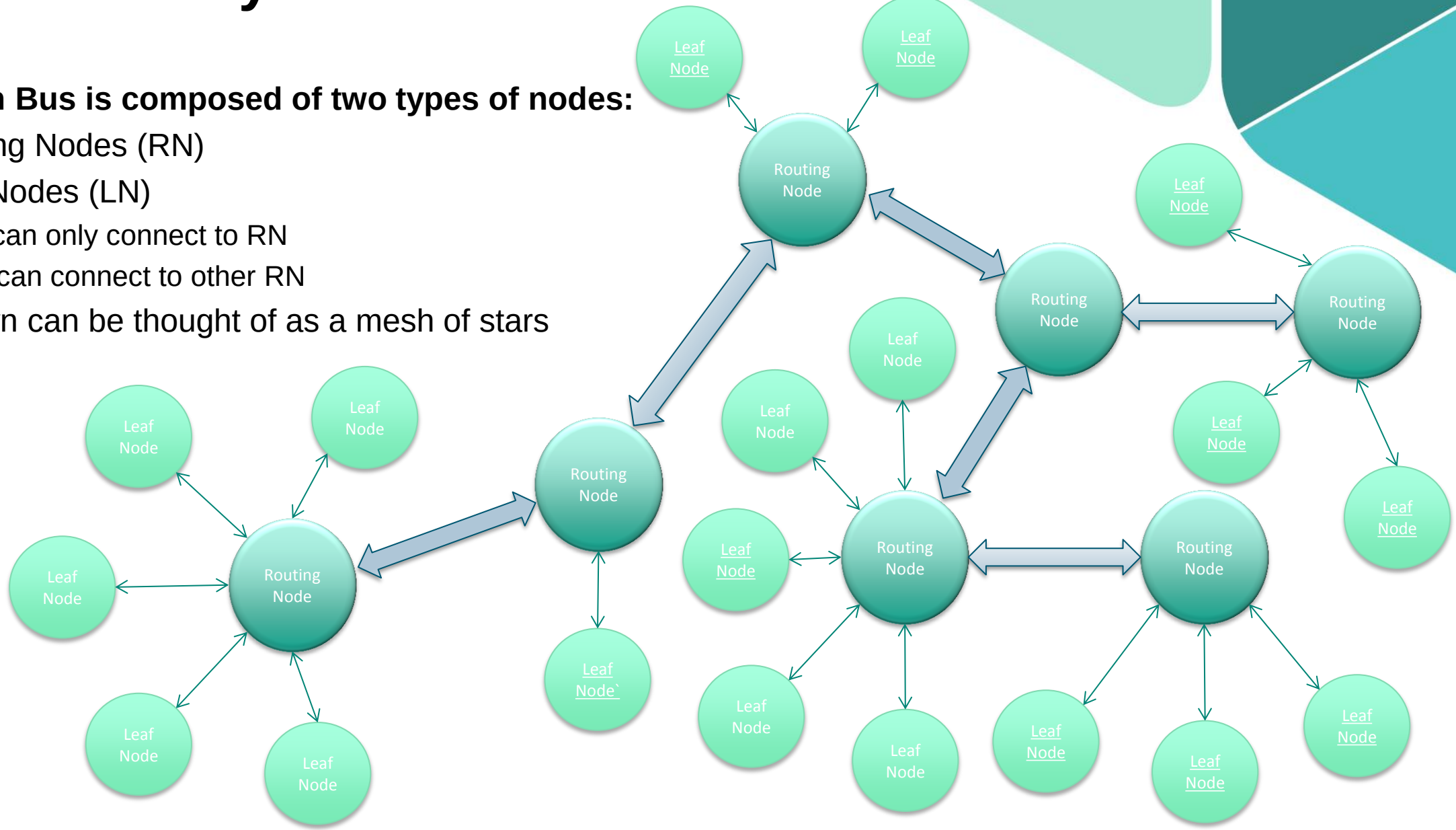
Protocol is network-independent

- Wire protocol is based on the D-Bus wire-protocol with extensions
- Can run over Wi-Fi, Wi-Fi Direct, Ethernet, PLC and Bluetooth
  - Could likely run over others

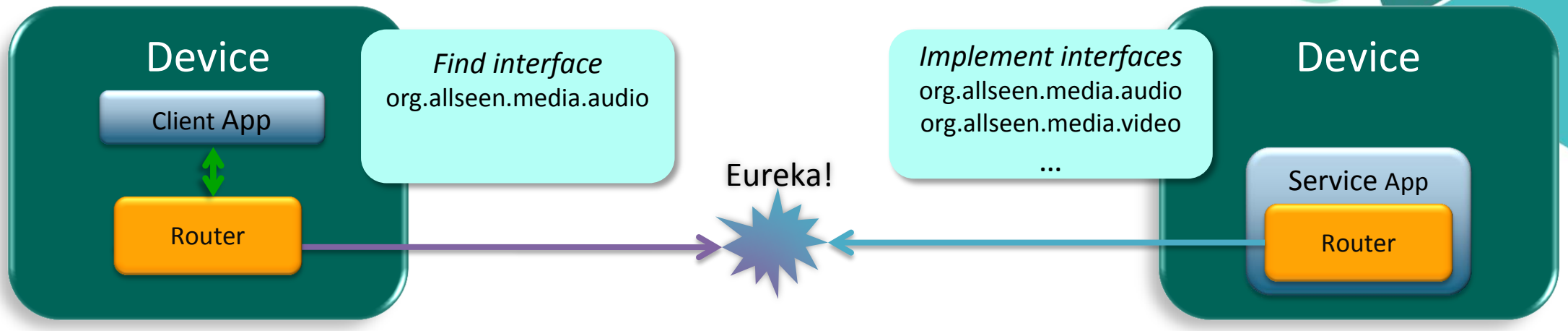
High Level System Architecture

AllJoyn Bus is composed of two types of nodes:

- Routing Nodes (RN)
- Leaf Nodes (LN)
  - LN can only connect to RN
  - RN can connect to other RN
- AllJoyn can be thought of as a mesh of stars



Ad Hoc Bus Formation: Discovery



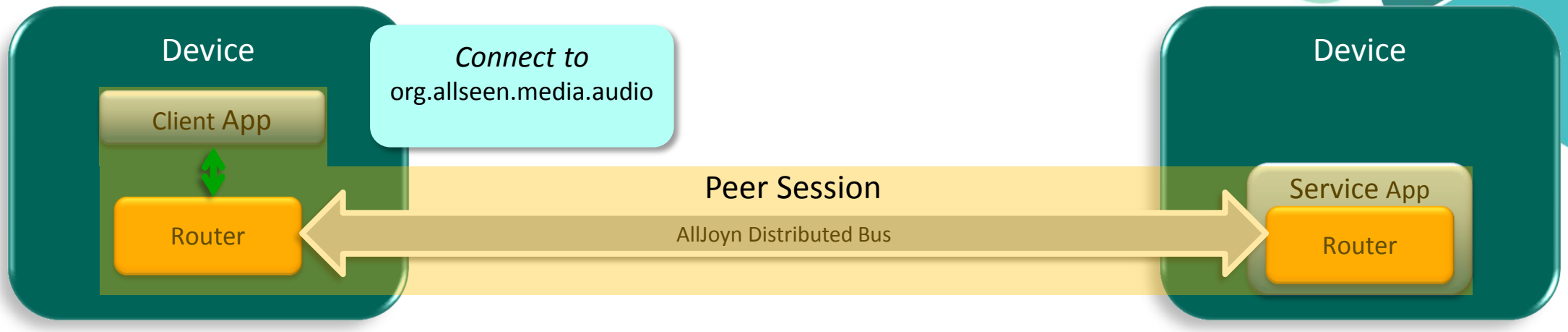
Interface descriptions contained in About message

- Services advertise, and Clients find, About messages
- connect to advertisers supporting desired interfaces

Actual discovery mechanism is transport dependent:

- On Wi-Fi, PLC, Ethernet: lightweight IP multicast protocol
- On Wi-Fi Direct: would use pre-association discovery

Ad Hoc Bus Formation: Session Creation



Session creation will cause Routing Nodes to connect and extend the bus

- Once connected, buses merge and have a single shared namespace
- Peers can discover when other peers join or leave the bus
- Peers can interact via their APIs
- Session reference counting keeps device-to-device connections alive
- Multicast events can be sent to all peers in the session

Software Components

AllJoyn Software Framework: High-level architecture

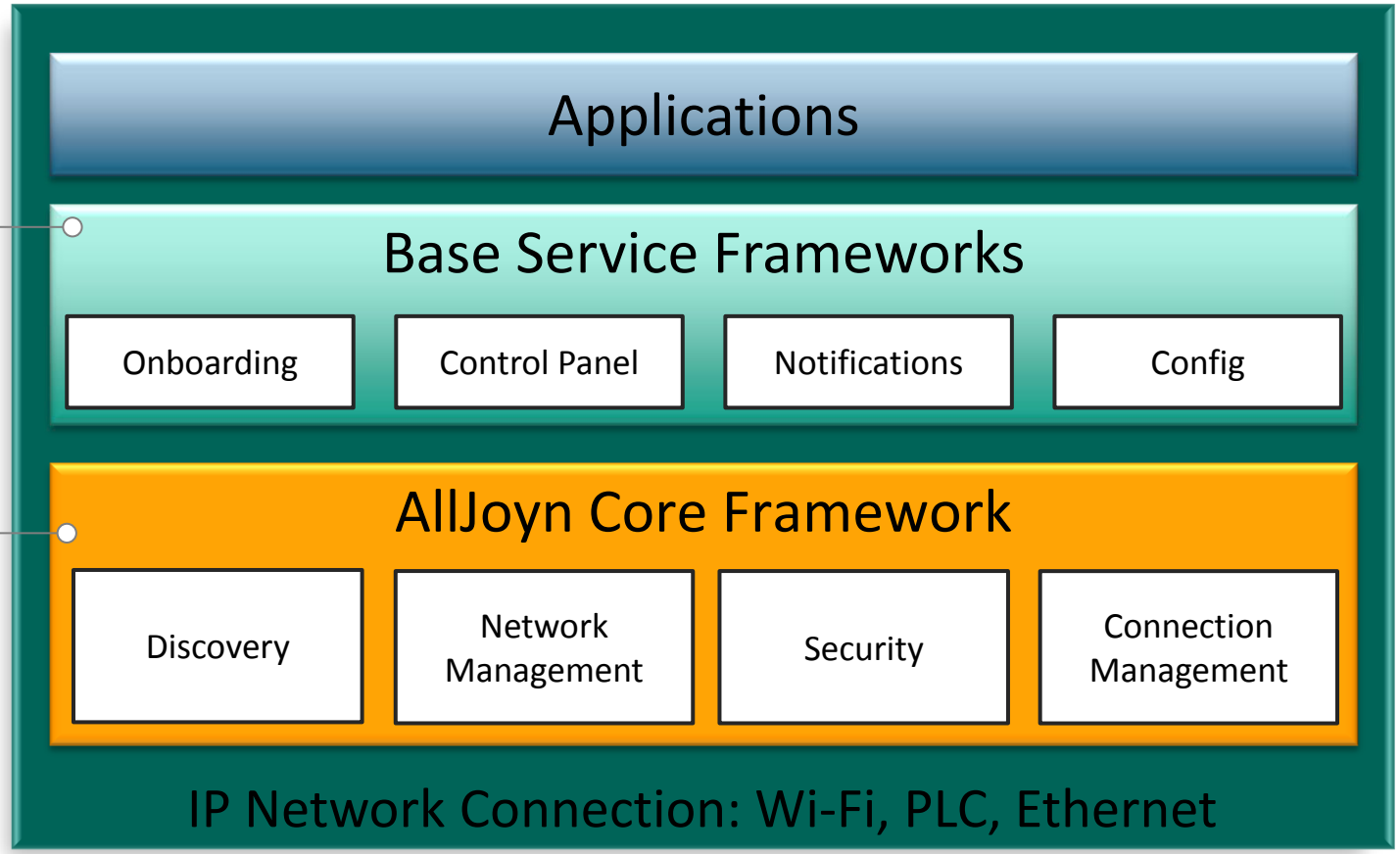
A comprehensive software communication framework

AllJoyn Service Frameworks

- Interoperable, cross-platform building blocks for common IoE functionality
- Audio, onboarding, control, notifications etc.

AllJoyn Core Framework

- Two main components:
  - Client Library and Router
- These provide the core functionality:
- Discovery, connectivity, security
- Language bindings, interoperability
- Dynamic network intelligence



Software Components

AllJoyn has two main architectural components: the Client Library and the Router

Client Library

- This is referred to as a Client Library because all AllJoyn applications are clients of the router
  - This is true regardless of if, in their application context, they are exposing services, or are clients of other services
  - Applications are peers if they implement both client and service functionality
- The Client Library is what software developers interact with: the API set of the AllJoyn SDK
- There are two implementations of the Client Library: the **Standard Client** (SC) and the **Thin Client** (TC)

- The SC is targeted at applications running in HLOS environments
  - The SDK APIs for the SC provide a high level of abstraction, and allow complex multi-threaded applications
  - Native implementation is in C++ and there are a number of language bindings for various platforms available

- The TC is targeted at applications that would reside on deeply embedded devices (i.e. device firmware)
  - Targets a very minimal memory (RAM and ROM) footprint,
  - Implemented in C, with no other language bindings
  - TC depends on a Routing Node running elsewhere, likely off device

Software Components: Router

Router

- Any node containing the router is a Routing Node
- The router is built using the Standard Client library, and so must run on an HLOS
- This can be deployed as a standalone daemon/service or integrated with the SC in an application
  - The only platforms on which standalone functionality is currently supported are Linux-based, such as OpenWRT
- The Router functionality consists of bus management and routing AllJoyn messages
  - Bus management includes
    - Managing the namespace: application addressing over the bus
    - Cross-device communication: discovering services and connecting to the routing node supporting that service on behalf leaf nodes
  - Message routing consists of delivering messages between applications, or to the router itself
    - AllJoyn control signaling also uses AllJoyn messages

- Routing Nodes do the “heavy lifting” in the Alljoyn system
- Leaf Nodes depend on Routing Nodes to interact with other nodes

Software Components: Language Bindings

Standard Client Library supports multiple language bindings

- Java binding available, compatible with Dalvik
- Objective C binding is available for use on iOS and MacOS
- Managed C++, C#, WinJS, and Visual Basic are available for use in Windows 8 style applications
- Unity + C Binding available for use on Android
- NPAPI binding for JavaScript is available

– All of the bindings are currently wrappers around the native C++ implementation

– Same object model for all bindings

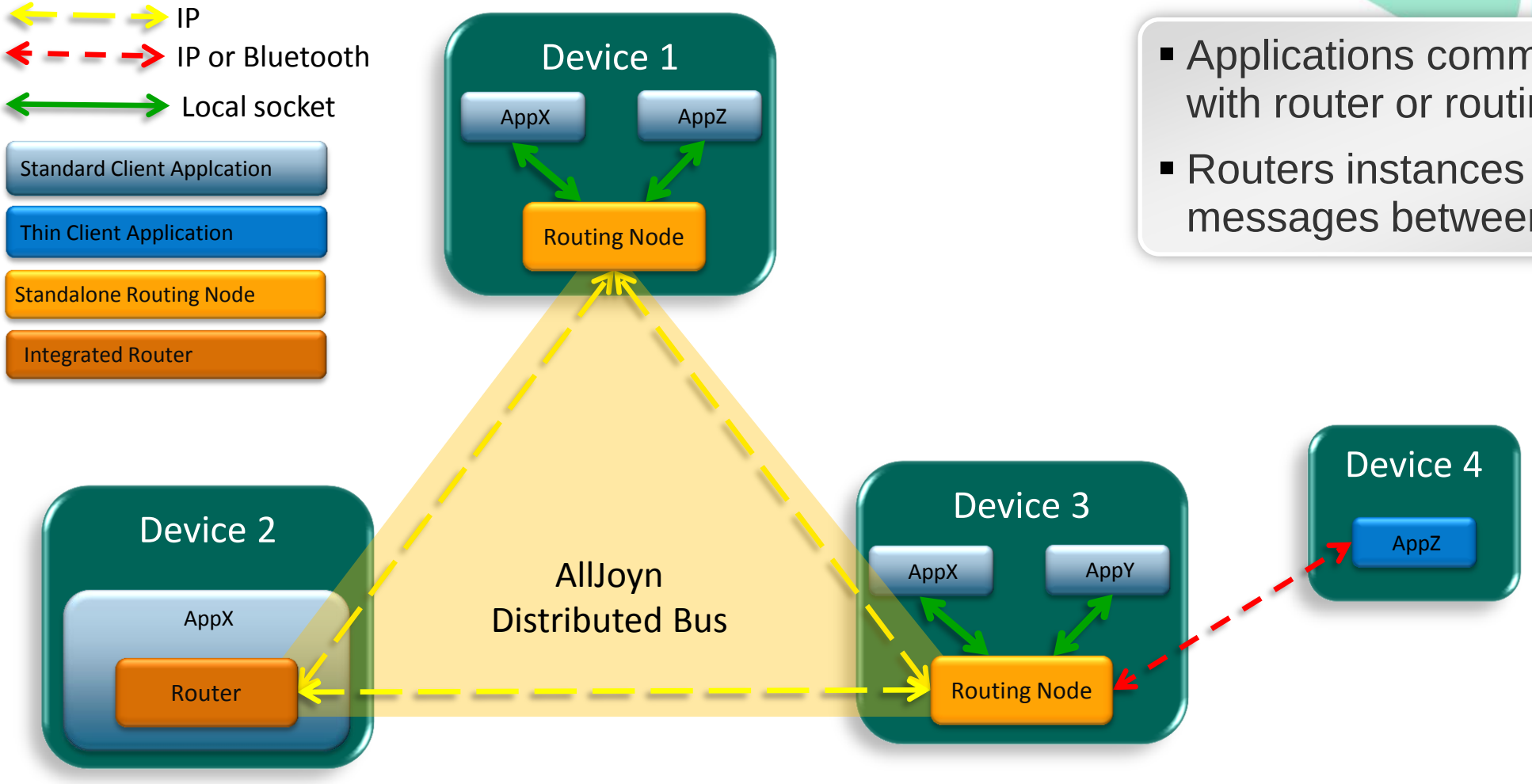
– Seamless interoperability between applications written in different languages

Thin Client Library only supports C

– Most appropriate language for embedded RTOS development

– Other language wrappers around the C implementation being investigated

Components deployed in AllJoyn Distributed Bus



- Applications communicate with router or routing node
- Routers instances deliver messages between devices

AllJoyn SDK Concepts

Exposing Functionality

AllJoyn applications expose their functionality via APIs implemented in objects

- Most applications will expose only a single object
- Object hierarchies are supported if required by application model
  - AllJoyn will create parent objects automatically if application object is not the root

Objects implement one or more interfaces

- These are the APIs that can be discovered using About

Interfaces are composed of members, which fall into three categories

- Methods – classic OO object interaction
- Signals – asynchronous event notification
  - Can be broadcast, multicast, or point-to-point
  - Can also send a Sessionless Signal: a broadcast signal that doesn't require an application session to be delivered
- Properties – data members
  - These are accessed by built-in get/set methods

All of this information can be introspected remotely

AllJoyn SDK Concepts

Connecting-to and Using the Bus

Bus Attachment is required to interact with/over the bus

– They are the application's presence on the AllJoyn Bus

– They provide the root (/) of the object hierarchy

Objects are published to local Bus Attachment

– Object path names look like file paths

- e.g. /Games/chess

Proxy Objects

– Local representation of a remote object

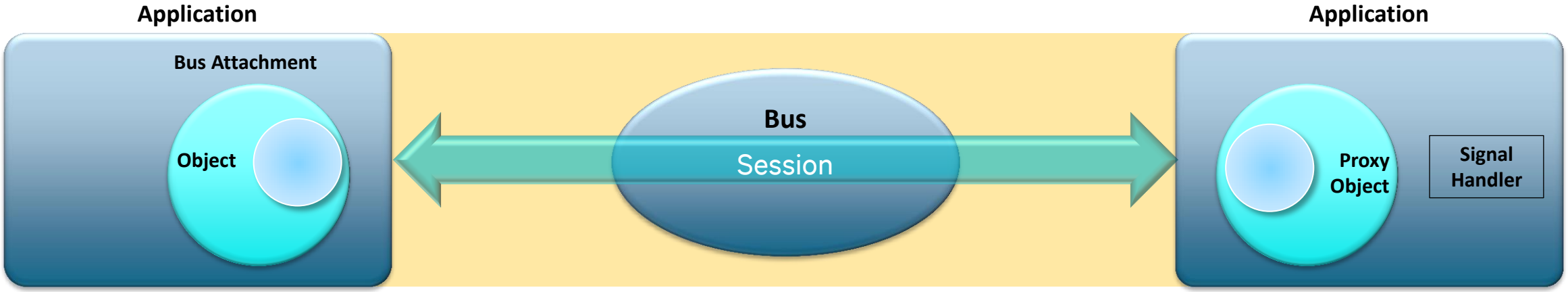
Signal handlers are registered with the Bus Attachment

– Registered by applications to take action when a signal is received

Sessions

– Flow controlled connections between applications

– Can be point-to-point or multipoint



AllJoyn SDK Concepts

Method Calls and Signals

Method Calls are made by using Proxy Objects

– This results in a method call message being sent to the object the proxy object represents

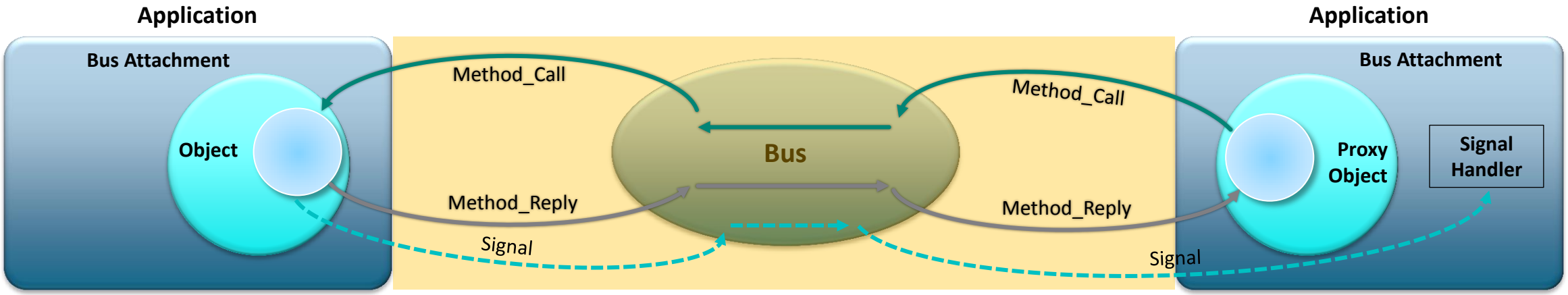
– Method handler is invoked to execute method and generate the reply

– The method reply is sent by the object back to the proxy object

Signals are emitted by Objects and consumed by Signal Handlers

– Sessionless signals work the same way, but do not require the application to create a session

- They are broadcast and are delivered to any app interested in receiving sessionless signals
- Only a single instance of a signal will be sent, i.e. if the same signal is emitted multiple times, only the last will be delivered
  - Useful for sending isochronous data, such as state updates



AllJoyn SDK Concepts

Object Model

As stated earlier, the Object model is similar for all language bindings

- Create a **`BusAttachment`** to connect to the bus
- Create **`BusInterface`**(s) that will be implemented by **`BusObjects`**
- Create and register **`BusObject`**(s) with the **`BusAttachment`**
- **`BusObjects`** handle remote method calls and emit Signals
- Applications call methods on remote objects using **`ProxyBusObjects`**
- Applications register Signal handlers with **`BusAttachment`**

Security Concepts

Design of Security Framework

Authentication and encryption is designed to be app-to-app

- Trust relationship established between the applications*
  - There is no trust relationship established at the device level
- The Router is not involved other than to route the messages
- Device pairing not required unless the transport requires it

Security is enabled per interface

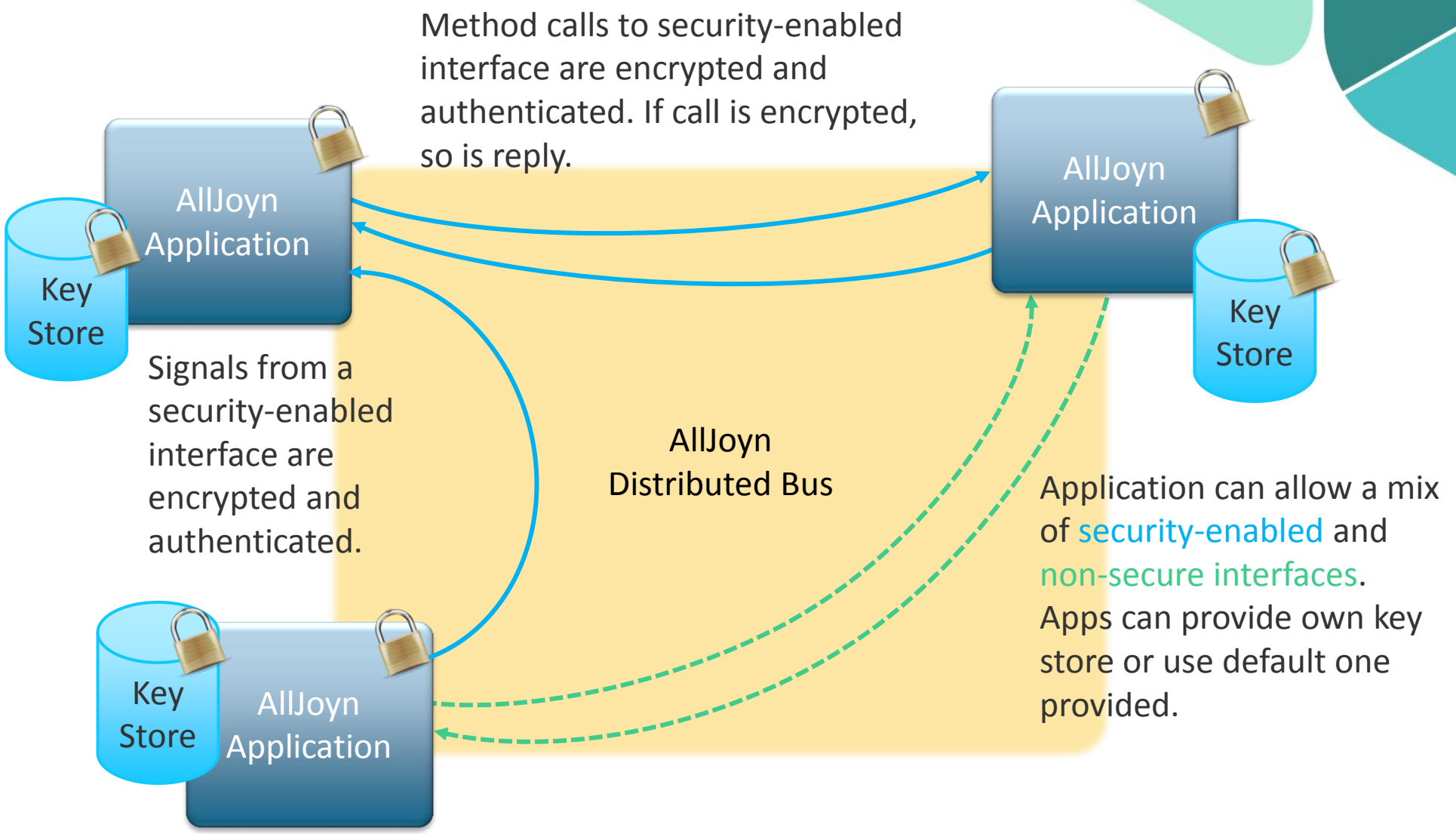
- Authentication and key exchange initiated on demand
- Applications can have a mixture of secured interfaces and open interfaces

Security-enabled interface

- Authentication is required to make method calls
- Authentication required to receive signals
- All messages are encrypted
  - Regardless of authentication mechanism, encryption always uses same AES-128 CCM algorithm

*** Applications could be firmware of embedded device**

Security Model – Authentication and Encryption



Built-In Authentication Mechanisms

Four mechanisms supported

– Mechanism negotiated using SASL protocol

– Mechanism chosen by application developer

– Most mechanisms adapted from TLS protocols per RFC 5256

Simple Pin-code pairing

– Specifically for Thin Client applications that lack resources to support big-number arithmetic

Pin-code SRP

– Pairing with a single-use password using Secure Remote Password protocol

Logon SRP

– User name and password using Secure Remote Password protocol

– Password required every time peers connect

Certificate-based

– RSA public key authentication and X.509 certificate-based

– Trust relationship lasts while certificate valid

Base Services

AllJoyn Base Services for fundamental use cases

- Standard AllJoyn building blocks for generically useful services
  - Onboarding
  - Notifications
  - Control Panel
  - Config
- Accelerate application and device development

Onboarding Service

Onboarding

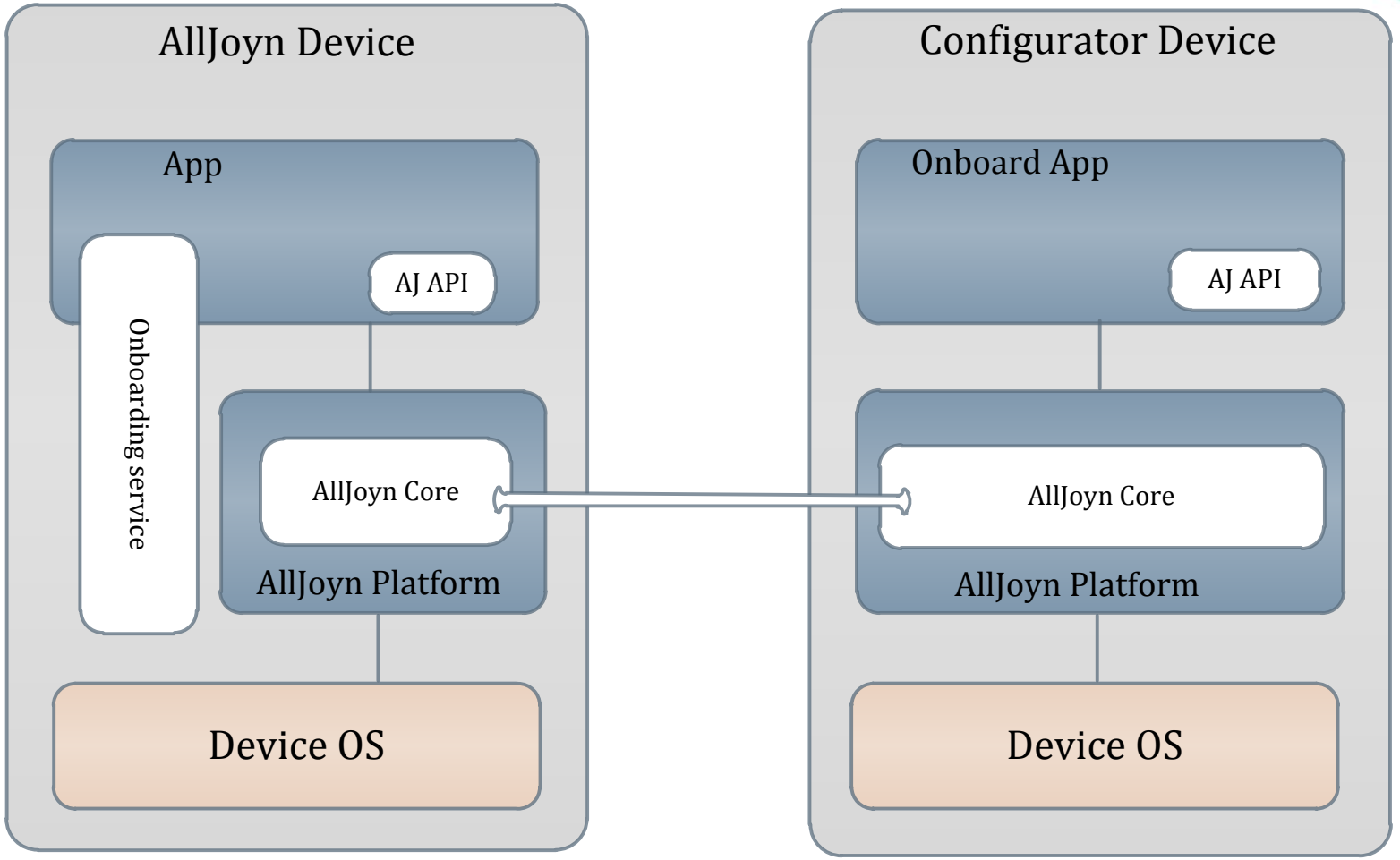
Provides a standard way to get devices onto a WiFi Network

- Onboarder is an application running on a smart device
- Onboardee is the device to be added to the WiFi network

Basic flow

- Onboarder discovers device that needs to be onboarded
- Connects to it, and provides configuration information
- Onboardee verifies it can connect
  - Informs onboarder that it has been successful or not

Onboarding Service



Notification Service Framework

AllJoyn Notifications Service Framework

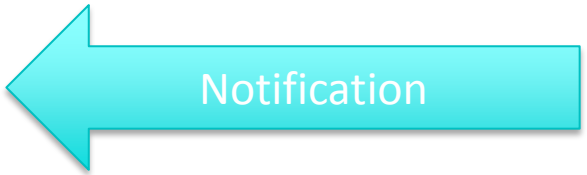
Simple, standardized interface for sending and receiving human-readable messages

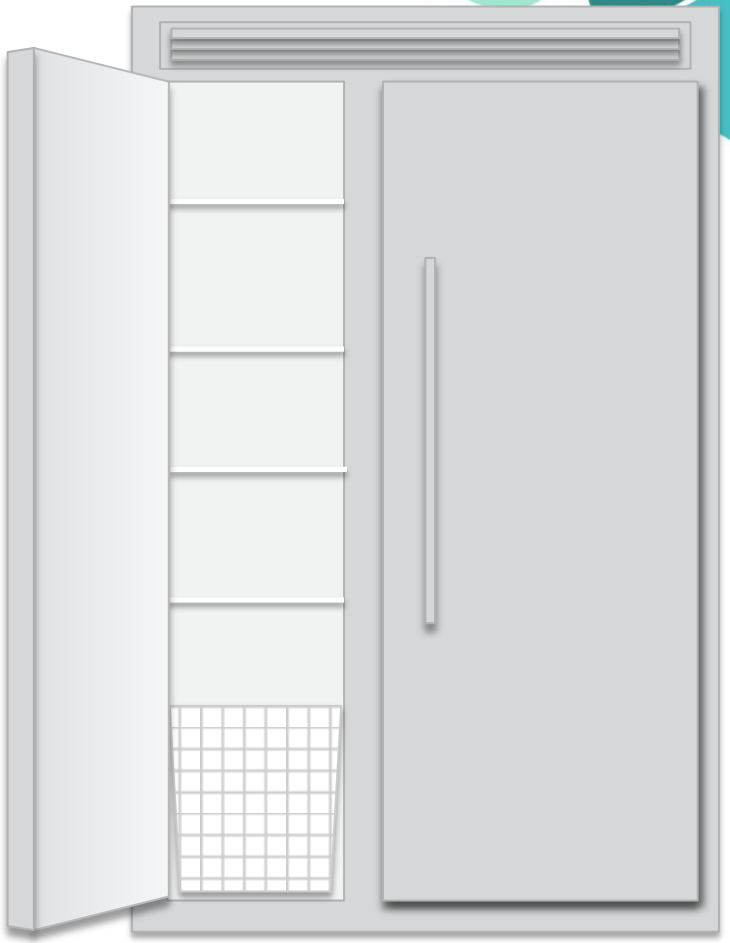
- Contents are text
  - Possible to reference image, audio, video: application can fetch media using reference
- Works across devices, operating systems and connection types
  - WiFi, Ethernet, PLC, etc…
- Producer (sender) can assign priority to notification
- Consumer (receiver) can configure preferences on types of notifications it receives

Examples

- Refrigerator could send a notification that freezer door has been left open for more than 2 minutes
  - This could be rendered on any producer: mobile device, TV, set top box, etc…
- Washing machine can send a notification when wash cycle is complete

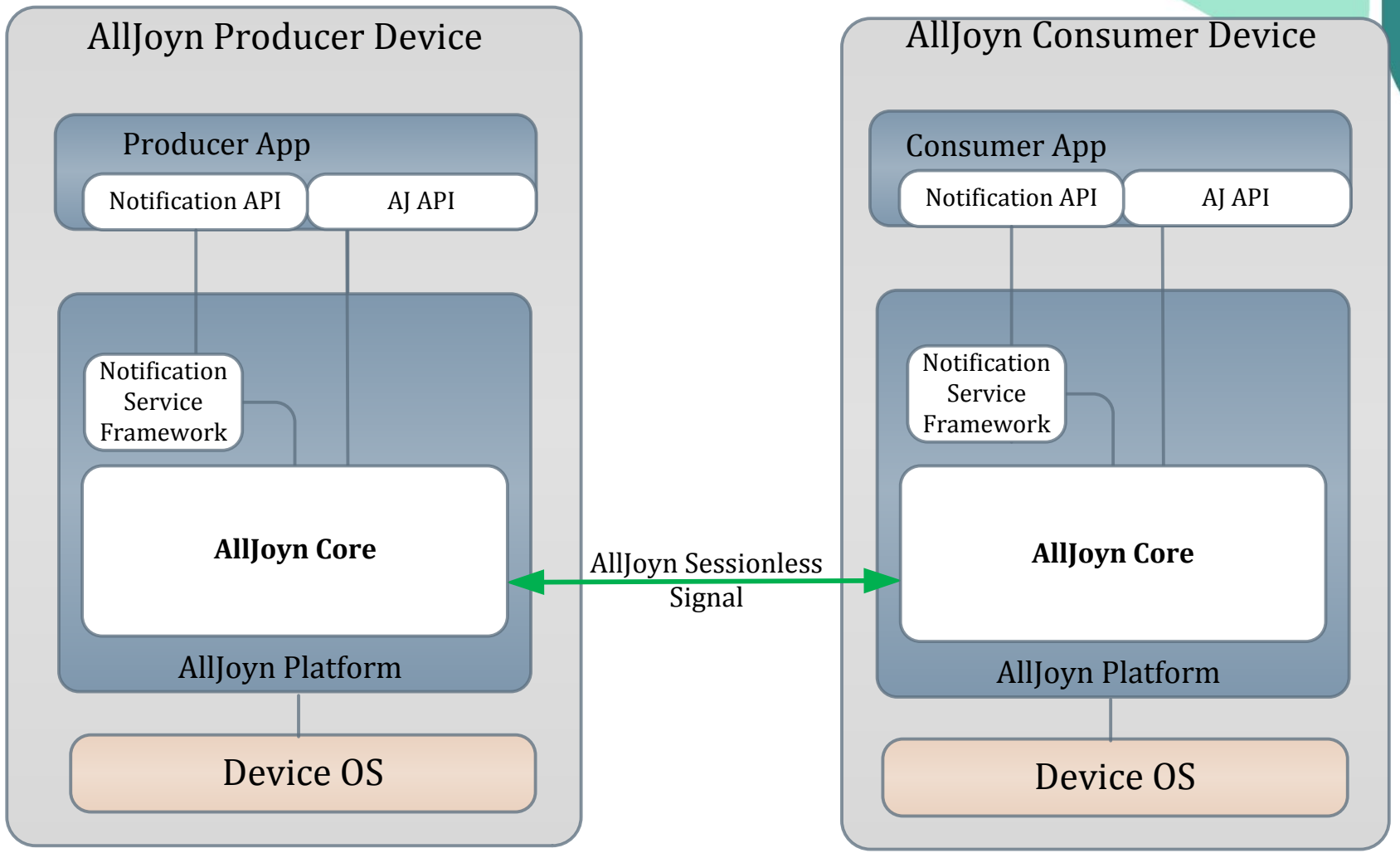
Notification Service Framework Example





- Freezer door left open for > 5 minutes
- Refrigerator emits notification
- TV renders it
  - Could also be rendered on a mobile device

Notification Service



Control Panel Service Framework

Control Panel Service Framework

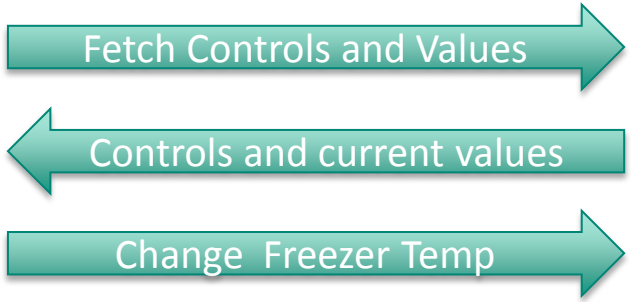
Infrastructure for exposing user interfaces for devices remotely

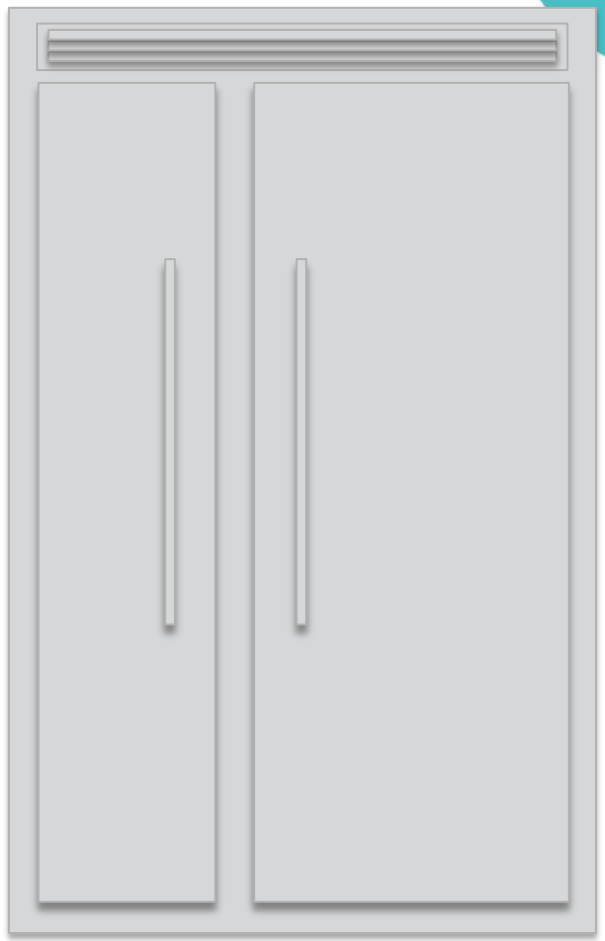
- Model is there is a controller and a controllee
- Controlee exposes it's UI using the framework,
- Controller renders the UI and sends control commands back to controllee base on UI input
- Defined such that any control application using the framework can render a controllee exposed UI
  - That is a generic app can control any type of devices that exposes controls using this framework

Examples

- After receiving a notification that the oven has been on Broil for 5 minutes a user could bring up the ovens control panel and change the setting to "bake at 250" to keep the food warm
- A user could check the current values of a refrigerator (including current temperature) and modify the settings to make things hotter or colder as needed.

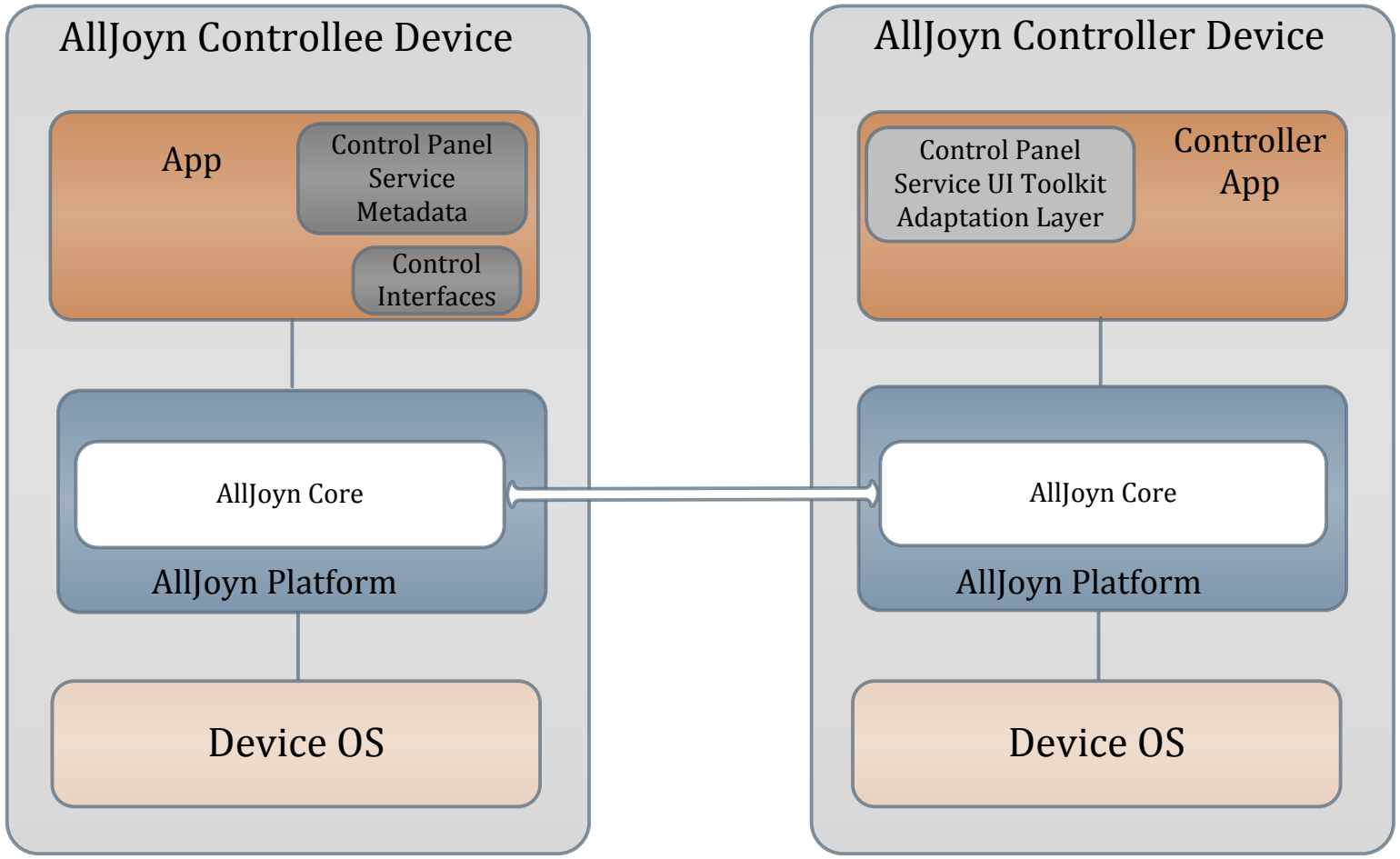
Control Panel Service Framework Example





- After detecting refrigerator
- Mobile device fetches the controls and values
- On receipt it renders them on the display
- Freezer temperature changed on mobile
- Change in the temperature is reported to refrigerator

Control Panel Service High Level Architecture




質問は？
Questions?
问题？
Ερωτήσεις?
Questions?
Fragen?
Вопросы?
Spørsmål?
Questions?
Domande?
Questions?
Vragen?
질문이 있습니까?
Cwestiynau?
Spørgsmål?
Questions?
FRÅGOR?
¿Preguntas?
Questions?
Kysymyksiä?

Thank You!