ALLSEEN ALLIANCE

AllJoyn™ Software Framework: Enabling the Internet of Everything

Core Training – Part 1

Brian Spencer

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

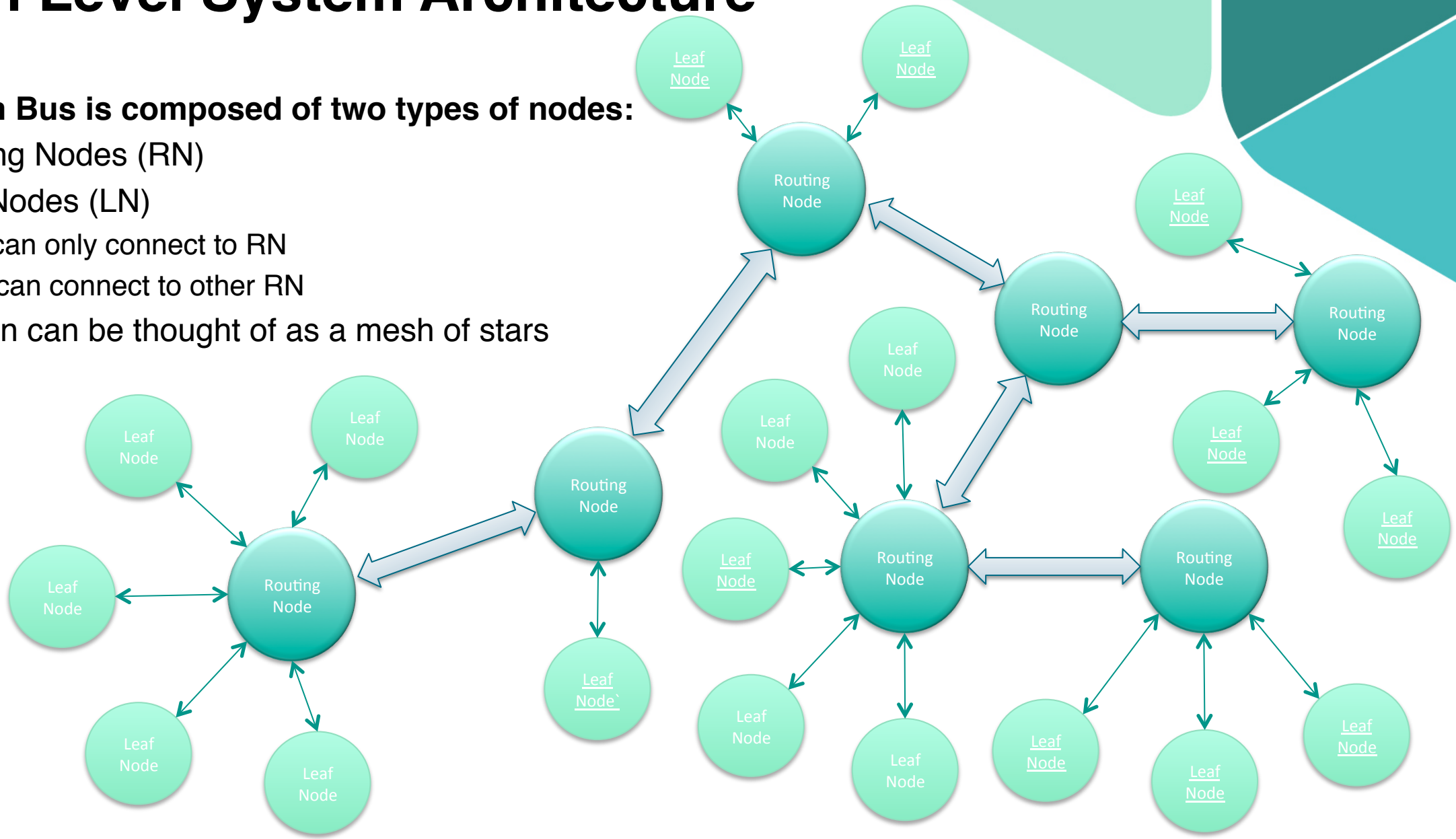
Protocol is network-independent

- 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

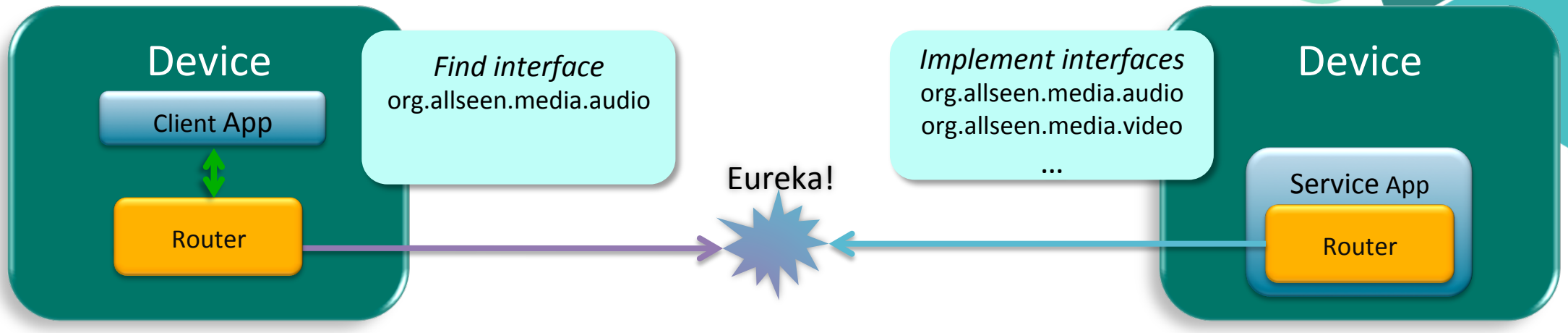
High Level System Architecture

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



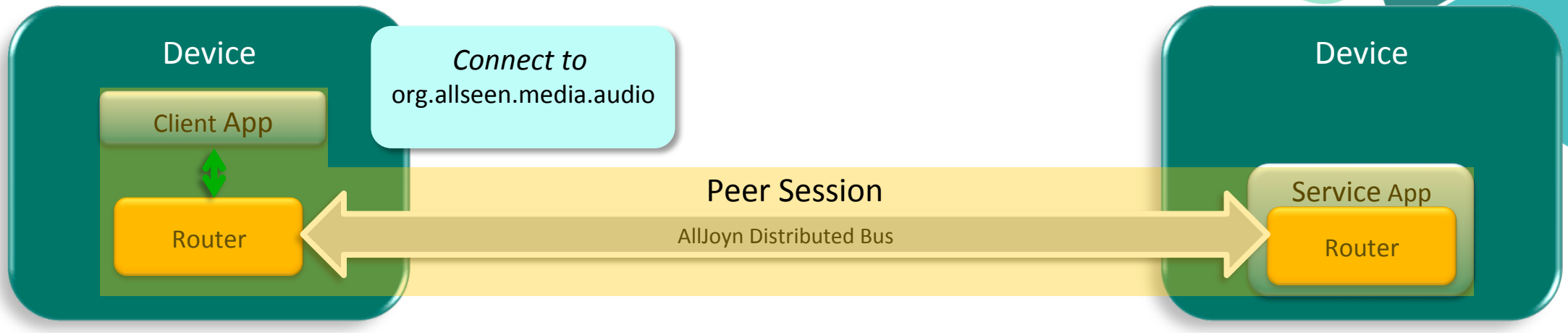
Interface descriptions contained in About message

- 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



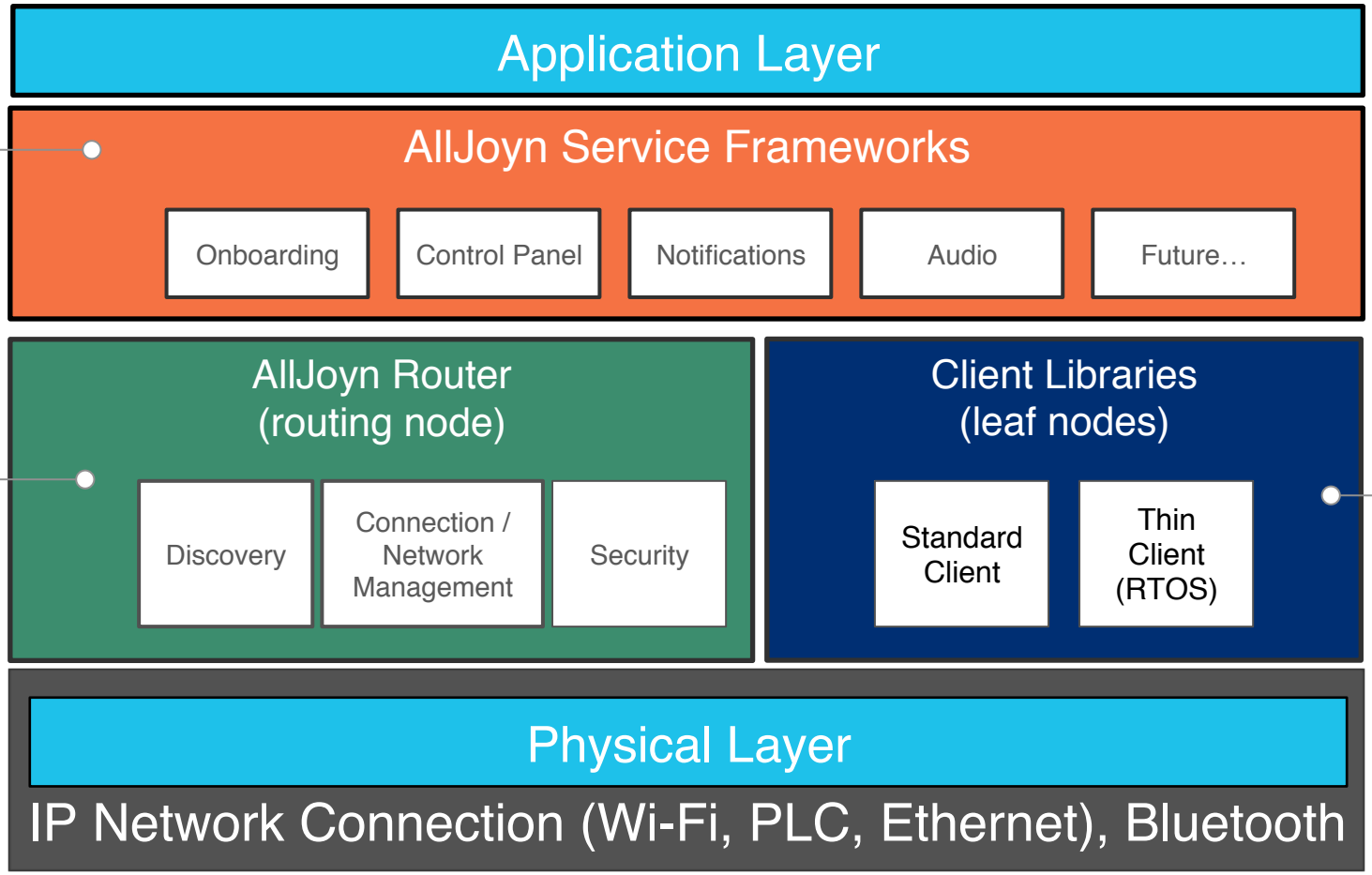
Session creation will cause Routing Nodes to connect and extend the bus

- 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 framework lets devices and applications communicate



AllJoyn Service Frameworks

- Interoperable, cross-platform modules for common IoE functionality

AllJoyn Router

- Fundamental building blocks
- Typically runs on a HLOS
- Router can be bundled with a Standard client application or run stand-alone

AllJoyn Client Libraries

- Developers Interact with Client libraries
- Standard Client can be bundled with Router or stand alone
- Stand alone client libraries require an AllJoyn Router in the network

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

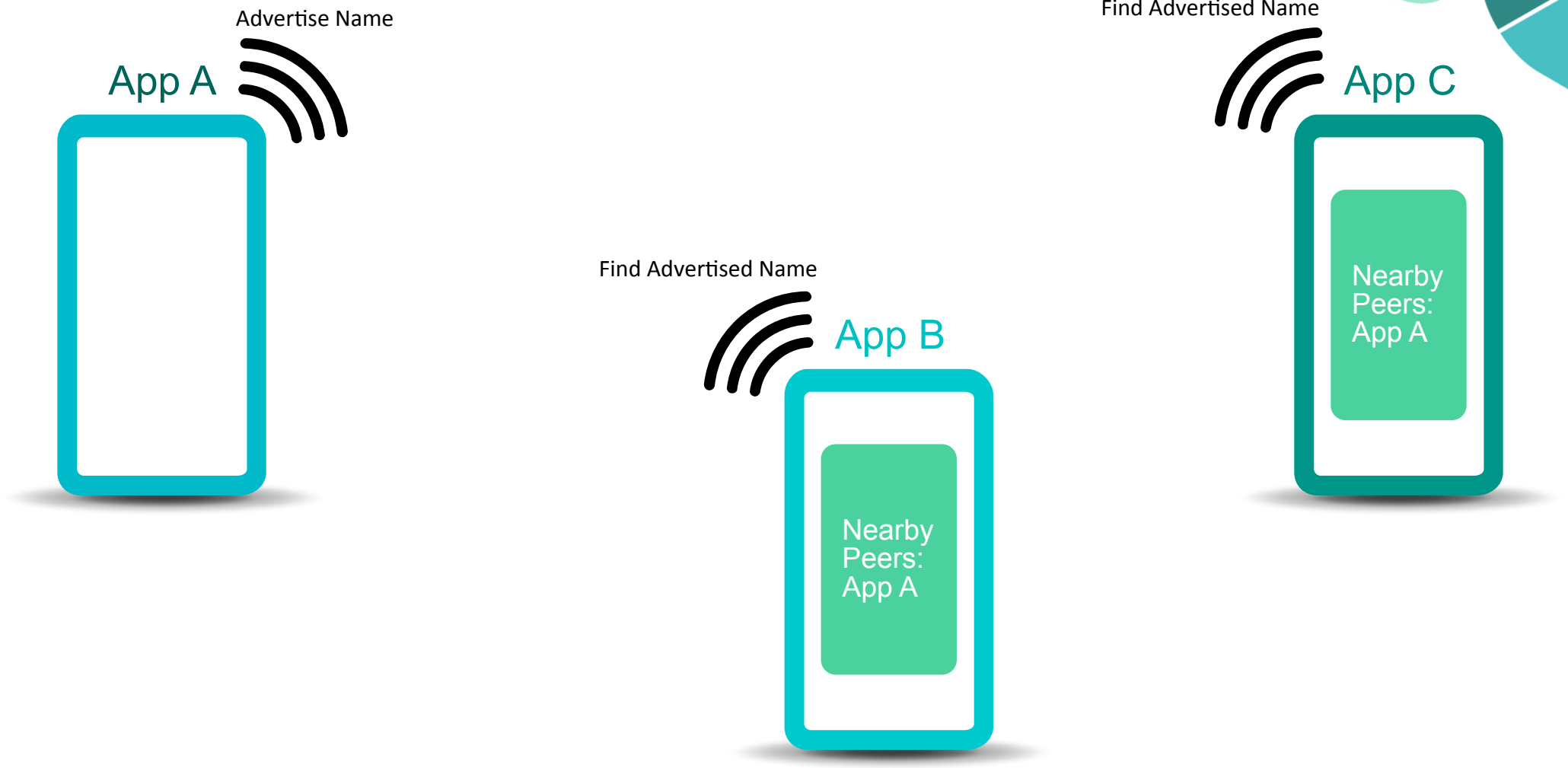
Software Components: Router

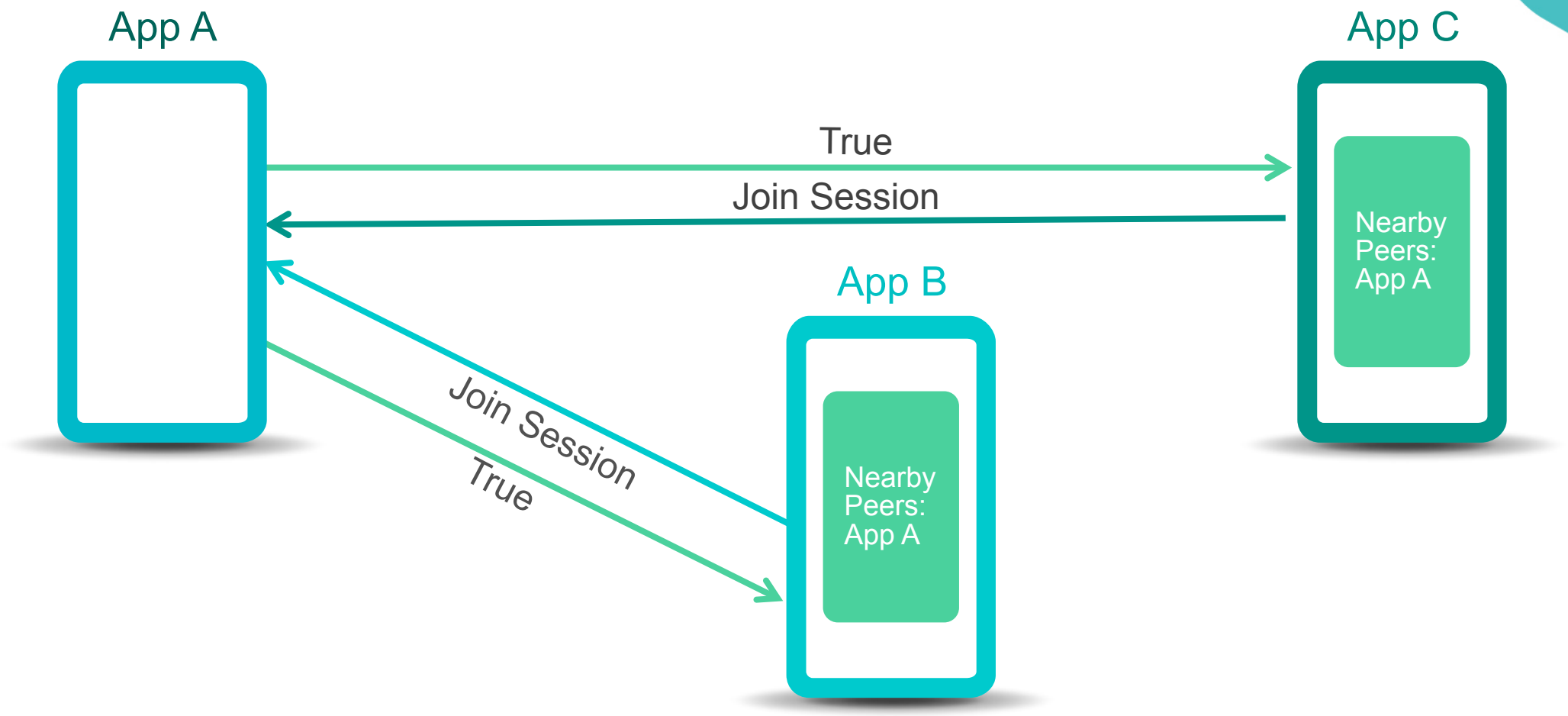
Router

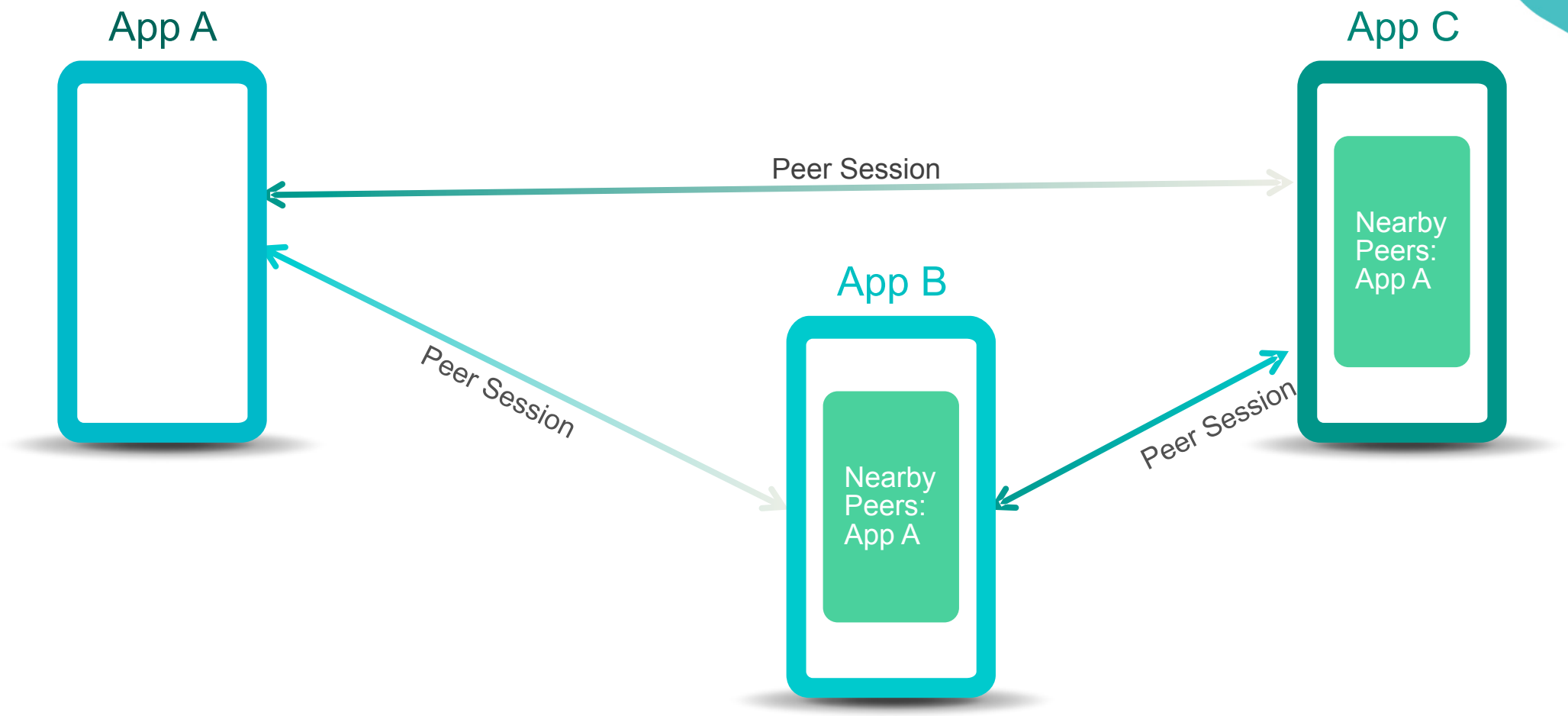
- Any node containing the router is a Routing Node
- The router is built using the Standard Client library, and so must run on an HLOS
- This can be deployed as a standalone daemon/service or integrated with the SC in an application
  - The only platforms on which standalone functionality is currently 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

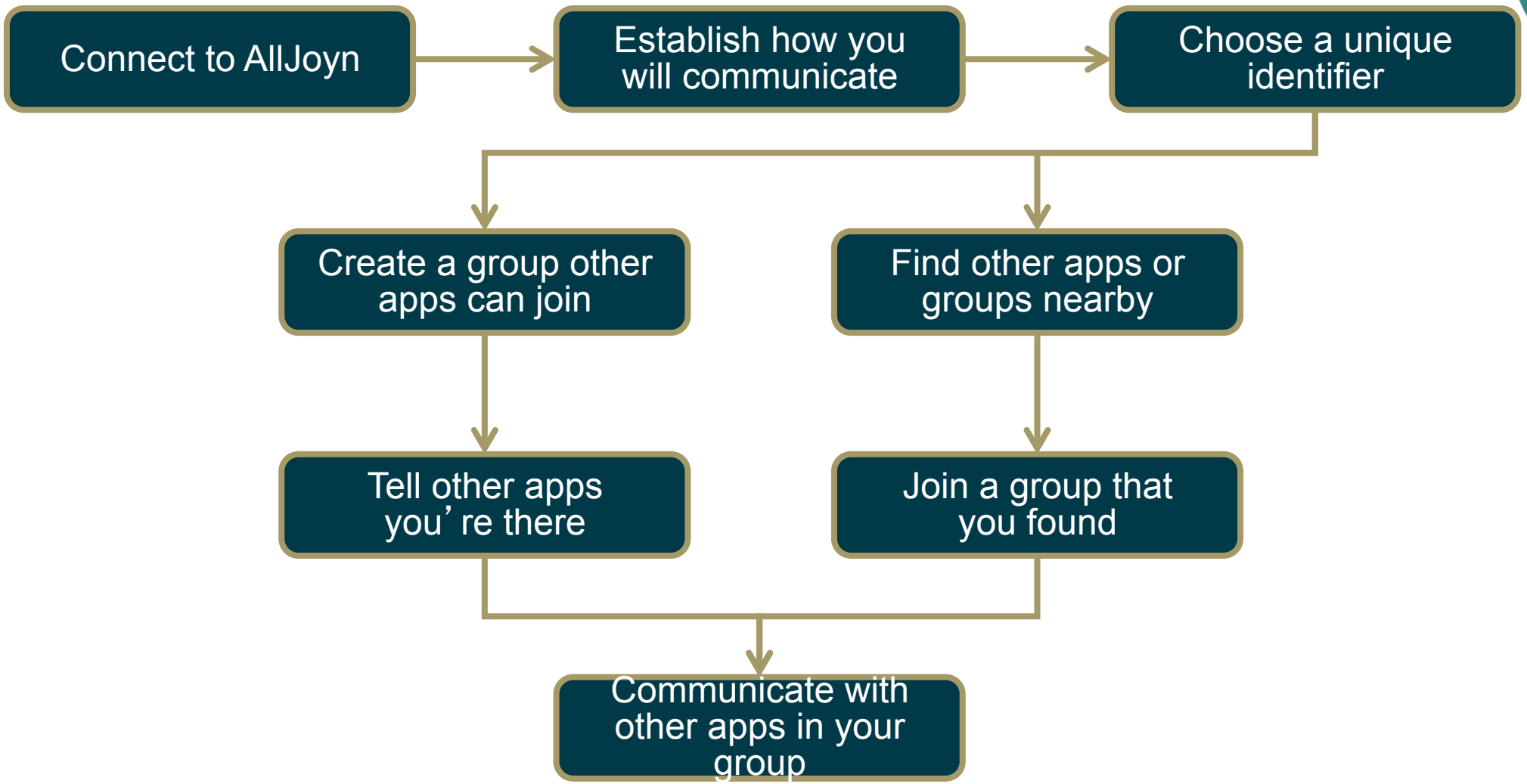
- Routing Nodes do the “heavy lifting” in the Alljoyn system
- Leaf Nodes depend on Routing Nodes to interact with other nodes


Advertise Name
App A
Find Advertised Name
App B
Nearby Peers: App A
Find Advertised Name
App C
Nearby Peers: App A


App A
App C
True
Join Session
Nearby Peers: App A
App B
Join Session
True
Nearby Peers: App A


App A
App C
Peer Session
Nearby Peers: App A
App B
Peer Session
Peer Session
Nearby Peers: App A

AllJoyn application workflow



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

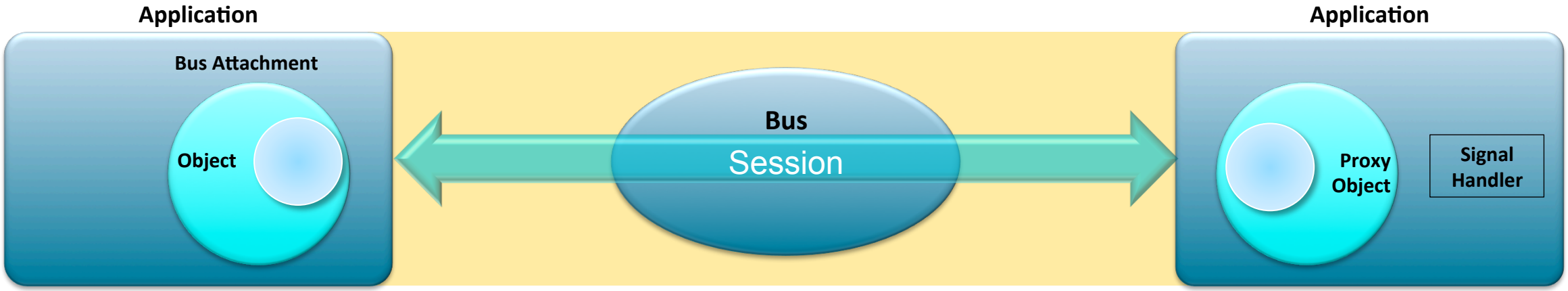
Signal handlers are registered with the Bus Attachment

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

Sessions

– Flow controlled connections between applications

– Can be point-to-point or multipoint



Wellknown Name

Describes the service with a hint of what the application is.

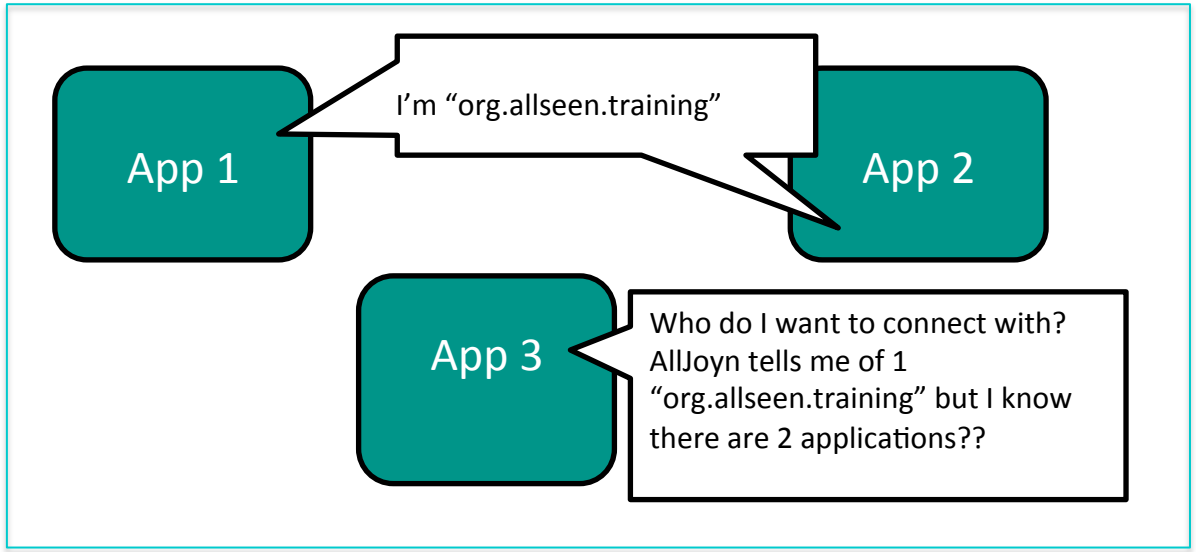
- Used to inform AllJoyn about the specific domains you are interested in
  - A game would only look for the game instances it knows how to connect to.
  - Does not need to know that other applications are in the network that can not be understood.

- Follow reverse domain name naming convention, eg. “org.allseen.training”

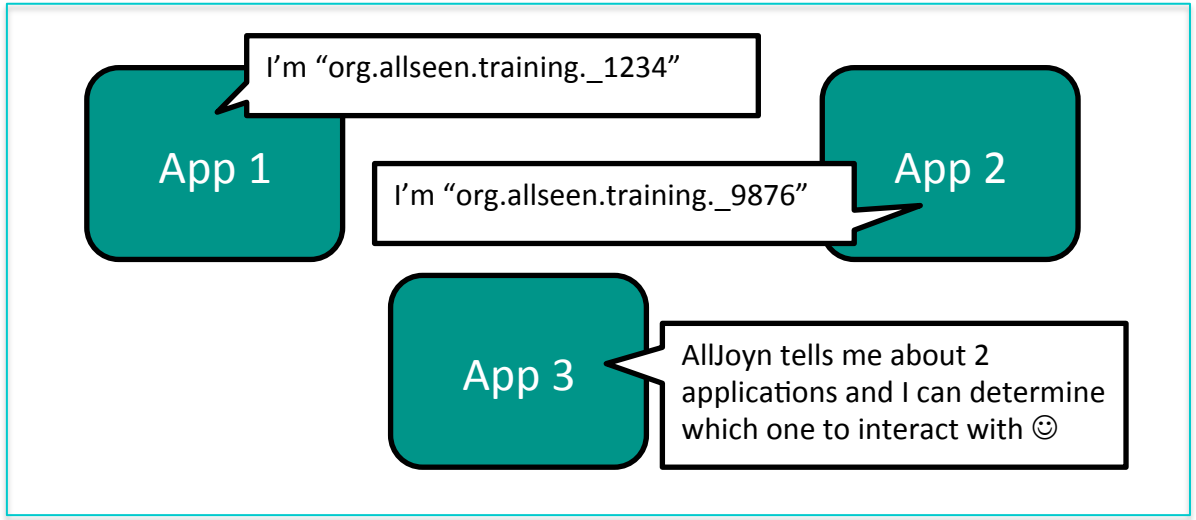
- Can only contain up to 255 characters and follow these rules:
  - Valid only with the characters [a-zA-Z0-9_.]
  - At least one ‘.’ must be present
  - A number [0-9] can not follow a ‘.’

- Every name should be unique
  - Avoids random behavior of who ends up connected
  - Accurately understand how many applications are in proximity

Wellknown Name



Wellknown Name



Wellknown Name

Example names

- **“org.allseen.training” - GOOD**
- **“org.alljoyn.1training” – BAD**
- **“org.alljoyn._1training” – GOOD**
- **“org.alljoyn” – GOOD**
- **“orgalljoyn” – BAD**
- **"org.allseen.training._fc8e88a8a2d0af8ee7cbc843d0c9bb07.Brian" – GOOD**
- **"org.allseen.training._d98c83b8a1e1df86e7524a43a1a8cc7a.Mitch" – GOOD**

– In common practice an end user defined name is appended to be parsed off for display in a UI. For example “Brian” would show up in a UI to indicate that “Brian” is in proximity and can be interacted with. Upon selection, the fully name would be used by AllJoyn to form a connection.

– Can not rely on end user defined names alone, no guarantee of uniqueness. Easy to add the BusAttachment::GetGlobalGuidString() return value for random value to ensure uniqueness.

Wellknown Name

What is this to AllJoyn?

- Maps to the AllJoyn BusAttachment uniqueName.
  - When an application calls BusAttachment::Connect() a uniqueName is assigned
  - Follows the format of a ‘:’ followed by 8 [a-zA-Z0-9] a [.] then a short
    - eg. “:jEbBlp6B.61” “:m7wtb9qZ.2”

- Either the uniqueName or wellknown name can be used in the APIs

- The uniqueName maps to a table that describes how to connect.
  - If over IP Transport than the IP address of the application.

- The uniqueName by itself has no application information and can not be understood without other information
  - Changes each time the connects to the AllJoyn bus

Advertise

"Lets other applications know you're there"

- Publishes the wellknown name to all AllJoyn Routing Nodes
  - Over IP networks this is a UDP broadcast over port 9956
  - The message is named IS-AT
  - Contains the list of wellknown names and uniqueNames, and IP/port information so a connection can be formed in the future
  - Source code lives in the alljoyn_core project inside router/ns

- Typical developer application flow:
  - Create a BusAttachment object
  - Call BusAttachment::RequestName with a generated wellknown name
  - Setup BusObjects
  - Setup listeners and Session
  - Call BusAttachment::AdvertiseName

Advertise

Service level advertising

The About feature allows for publication of the interfaces used in an application

- Instead of a wellknown name, uses a broadcast mechanism in AllJoyn called a Session-less signal

- A Session-less signal is an AllJoyn Signal that does not require explicit joining to receive

- Interface is standardized: “org.alljoyn.About”
  - Contains a set of Signals and BusMethods to express what the application/device is and supports
  - Device information: make, model, manufacturer, etc.
  - Interface list: “org.alljoyn.controlpanel”, “org.allseen.media”, etc.

Advertise

Service level advertising

- Spec for About found here:
  - https://allseenalliance.org/docs-and-downloads/documentation/alljoyn-about-feature-10-interface-specification
- Source code:
  - https://git.allseenalliance.org/cgit/core/alljoyn.git/tree
  - Inside the services/about folder
- Typical developer application flow:
  - Create a BusAttachment object
  - Setup BusObjects
  - Setup listeners and Session
  - Call BusAttachment::AdvertiseName with the uniqueName
  - Setup About feature and call About::Announce

Discovery

What is this

“Lets you find other applications nearby”

- Asks the AllJoyn Routing Node that it is interested in a wellknown name prefix
  - API call to BusAttachment::FindAdvertisedName(prefix)
  - Over IP causes a one time UDP broadcast over port 9956
  - The message is named WHO-HAS
  - Routing Nodes will respond with an IS-AT immediately
  - All Routing Nodes will look at the IS-AT list and execute BusListener::FoundAdvertisedName callbacks for any wellknown name that starts with the
  - Source code lives in the alljoyn_core project inside router/ns

- Typical developer application flow:
  - Create a BusAttachment object
  - Setup BusObjects
  - Setup BusListener callbacks
  - Call BusAttachment::FindAdvertisedName(prefix)

Discovery

BusListener

Base class that an AllJoyn Routing Node will call into

- The really important callbacks are FoundAdvertisedName and LostAdvertisedName
  - Arguments are: name, transport and prefix
  - Name contains the full wellknown name that was discovered/lost. This is used to JoinSession, etc.
  - Transport lets us know over what radio link was used or if it is local.
  - Prefix gives us ability to look for multiple wellknown names and understand what caused the match.

- FoundAdvertisedName(name, transport, prefix)
  - Until a JoinSession is called, this only hints that a device is in proximity, not a guarantee
  - Will execute when an IS-AT arrives that causes an entry into the internal AllJoyn routing table

- LostAdvertisedName(name, transport, prefix)
  - Indicates that there is no longer a way to reach the application
  - Will execute when an IS-AT does not arrive in the 120 second window
  - Causes an entry in the internal AllJoyn routing table to be removed

Discovery

Service level discovery

The About feature allows for discovery of the interfaces used in an application

- Instead of a wellknown name prefix, look for applications that support a known interface

- Interface is standardized: “org.alljoyn.About”
  - Contains a set of Signals and BusMethods to understand what the sender application side is
  - Device info: make, model, manufacturer, etc.
  - Interfaces: “org.alljoyn.controlpanel”, “org.allseen.media”, etc.

- Currently no callback in the About feature informs of “Lost”
  - Would use a BusListener or try to JoinSession to verify in proximity

Discovery

Service level discovery

- Spec for About found here:
  - https://allseenalliance.org/docs-and-downloads/documentation/alljoyn-about-feature-10-interface-specification
- Source code:
  - https://git.allseenalliance.org/cgit/core/alljoyn.git/tree
  - Inside the services/about folder
- Typical developer application flow:
  - Create a BusAttachment object
  - Setup BusObjects
  - Setup listeners and Session
  - Setup About feature and register AboutListener
    - Take action when listener executed

Session

What is this conceptually?

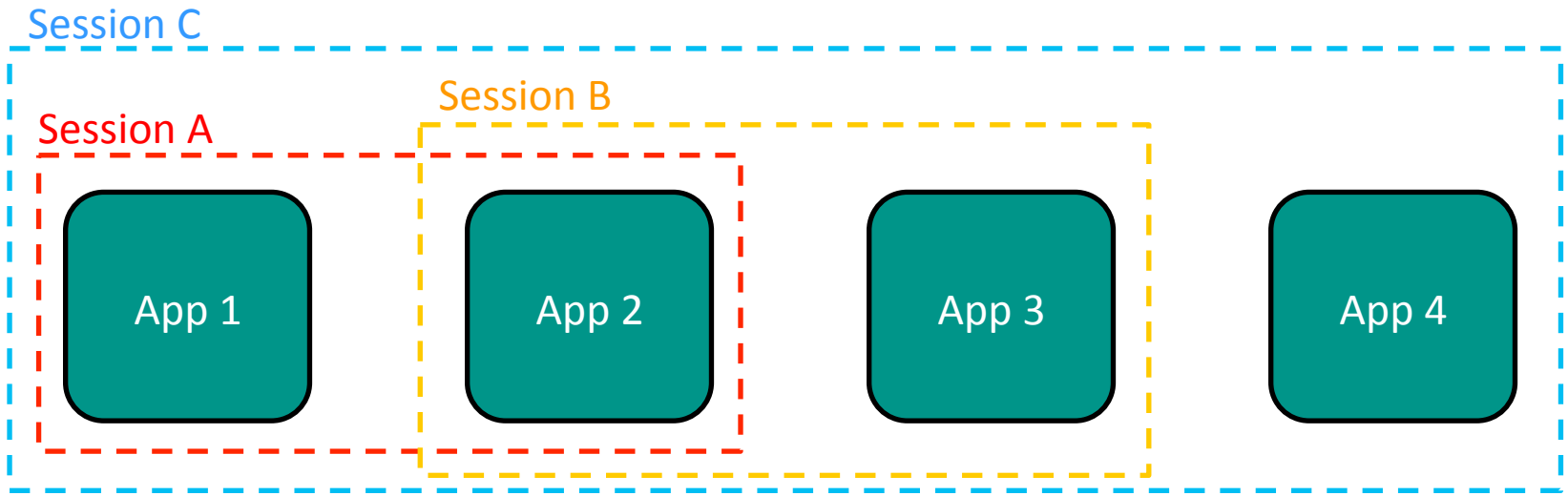
A group of applications that are connected, allowing them to exchange data

– Any application can create a session (BusAttachment::BindSession)

– Advertisement is the process of letting others know you can be joined

– Can be 1:1 or 1:many

– Allows for signals to travel to only those with the same session ID.



Session

Create a session

- BindSessionPort(port value, session options, sessionPortListener)
  - Port value
    - ***NOT an IP port***
    - Allows for developer to encapsulate their application traffic
  - Session options
    - Specific settings that must be matched by the other side
    - Can set specific network transport value
    - 1:1 or 1:many (isMultipoint)
  - SessionPortListener
    - Callback class to be informed about devices joining
    - Contains method to grant/deny a connection

- Typical developer application flow:
  - Create a BusAttachment object
  - Setup BusObjects
  - Call BusAttachment::BindSession
  - Advertise a wellknown name or use About feature

Session

SessionPortListener

Base class that an AllJoyn Routing Node will call into upon connection requests

- Bool AcceptSessionJoiner(session port, joiner, session options)
  - Returning True will allow the joiner side to complete the connection
  - False will cause an ER_ALLJOYN_JOINSESSION_REPLY_REJECTED on the joiner's BusAttachment::JoinSession
  - The callback parameters are to be used to validate an acceptance for a connection.
  - Method must not block -> timeouts will occur

- SessionJoined(session port, session ID, joiner)
  - Executes after a return of True from AcceptSessionJoiner
  - Informs that the connection request is now completed
  - Allows a lookup for the connected applications to determine the address
  - Joiner lets us know the uniqueName who is now connected
  - Traditionally a SessionListener is created to monitor the session if not already set

Session

Join a session

– JoinSession(wellknown or uniqueName, session port, BusListener, reference session id, session options)

- Wellknown or uniqueName
  - Determined through discovery
  - Because the wellknown name maps to the uniqueName by the AllJoyn Routing Node either can be used
- Session port
  - ***NOT an IP port***
  - Must match in most cases, check may occur on remote Application
- BusListener
  - Callback class to be informed on session state
- Session id
  - Will be filled in upon a successful join
- Session opts
  - Preferred connect options, can be modified by AllJoyn Routing Node

– Typical developer application flow:

- Create a BusAttachment object
- Setup BusObjects
- Start discovery process
- Upon finding interested services call BusAttachment::JoinSession

Session

SessionListener

Base class that an AllJoyn Routing Node will call informing session status

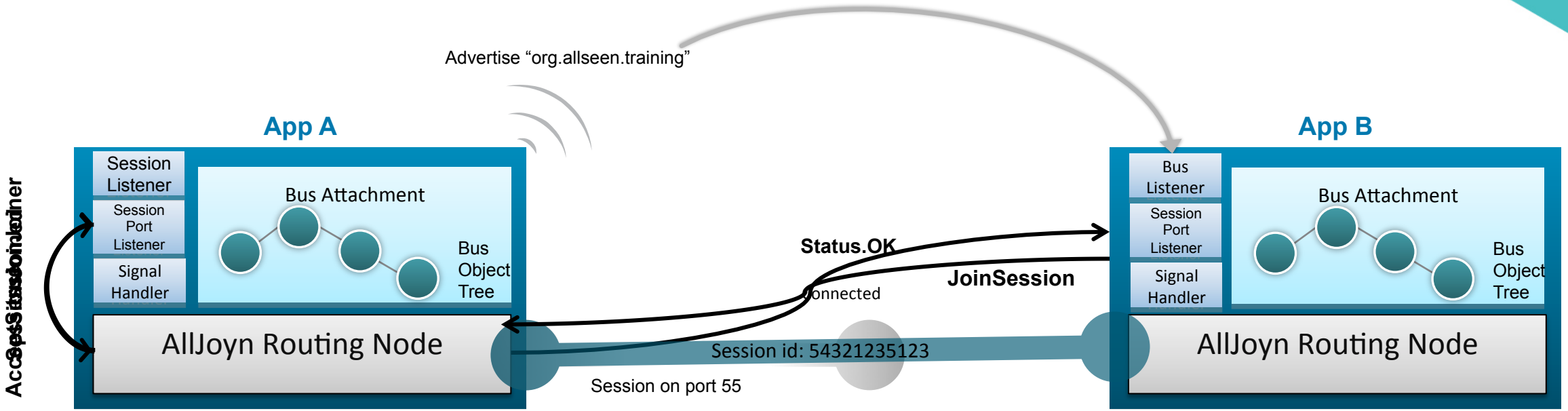
- SessionLost
  - Executes when the Routing Node detects there are no other applications connected to a session ID
  - Reliable way to determine connection status -> within 40 seconds
  - It is possible to receive a LostAdvertisedName for an application, but not SessionLost
  - This is why Advertisements are hints and not absolute information

- SessionMemberAdded
  - For use with multi-point session
  - Will execute upon a new application joining a session
  - Since not the host no direct knowledge, AllJoyn Routing node can provide this to the application
  - Can now setup BusMethod calls to newly added application

- SessionMemberRemoved
  - For use with multi-point session
  - Will execute upon application leaving session, but session still established

Session

Sequence of events



Bus Object

Introduction

Base class that all applications use to expose functionality

Bus Objects implement AllJoyn Interfaces

- AllJoyn Interfaces are the definition of the service
- Make up the API of interaction

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

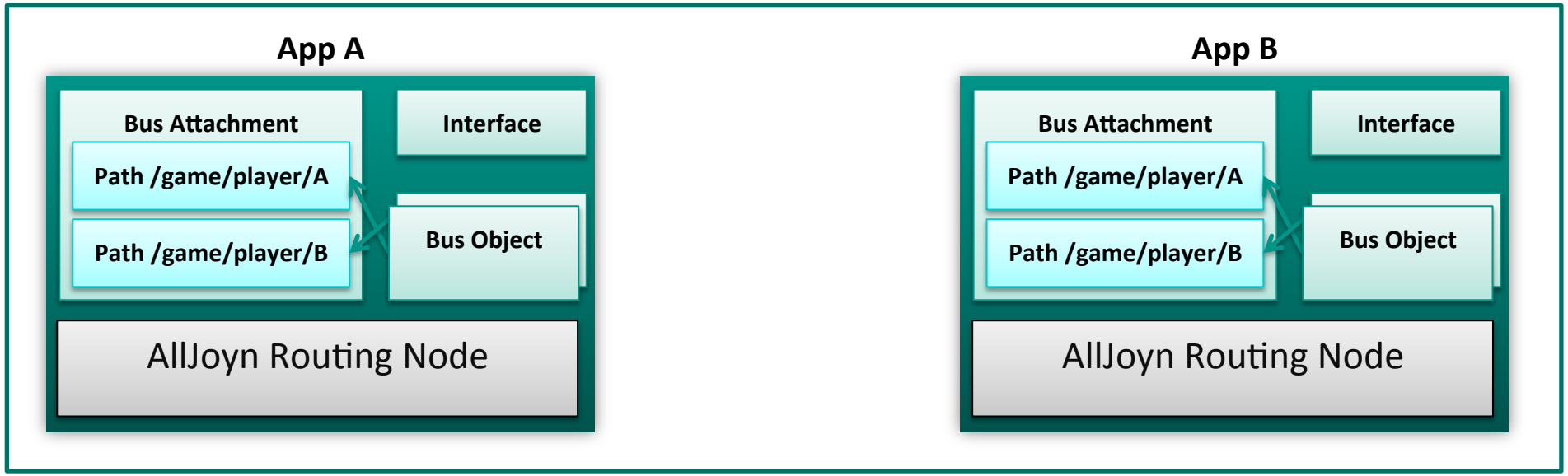
Bus Object

Introduction

– Bus Objects Implement 1 or more of the Interfaces

– The Bus Object box represents 2 instances of the same Bus Object

– OOP style of programming allowing for Objects to be of the same type but semantically different



Each Bus Object is identical except that they exist on a different path


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

Thank You!