ALLSEEN ALLIANCE

AllJoyn™ Software Framework: Enabling the Internet of Everything

Core Training – Part 2

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

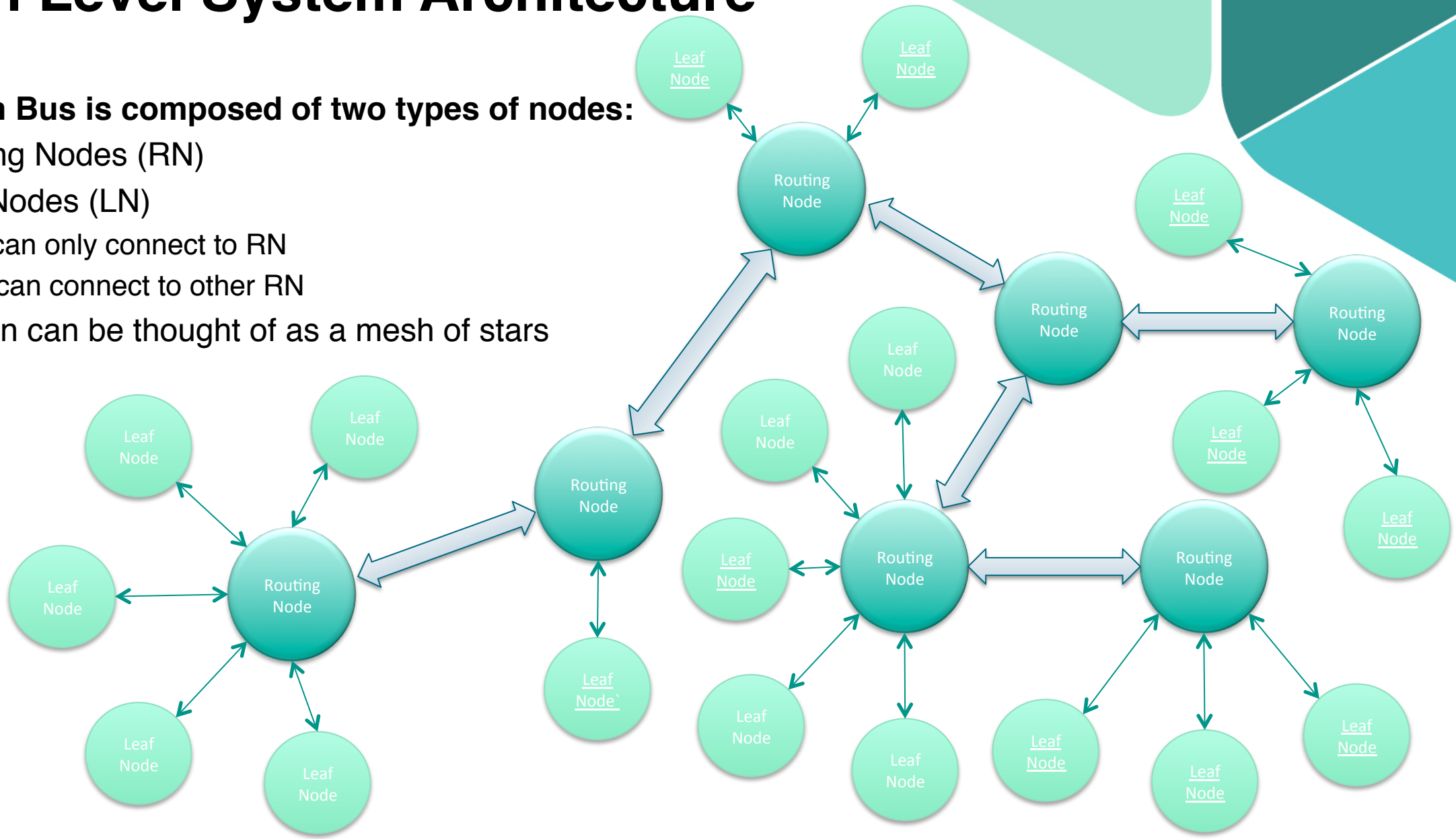
Protocol is network-independent

- 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

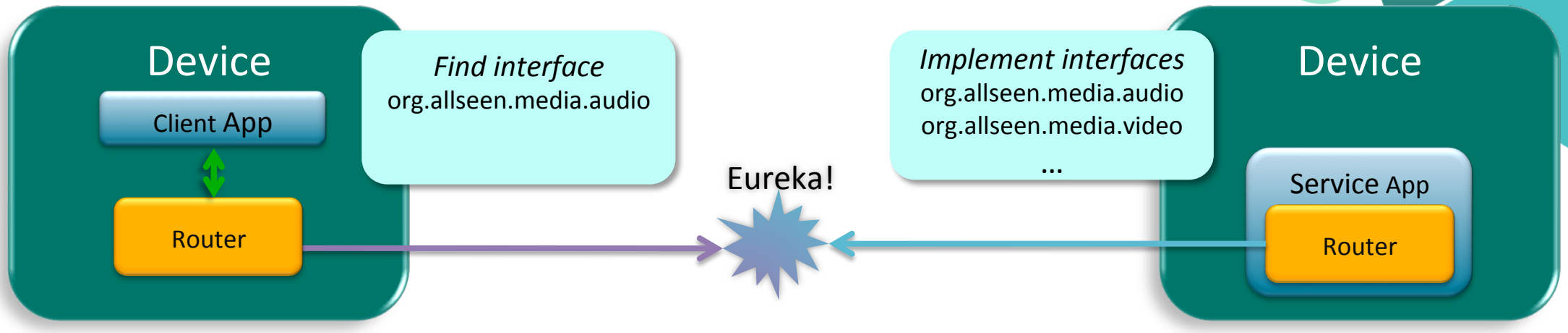
High Level System Architecture

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



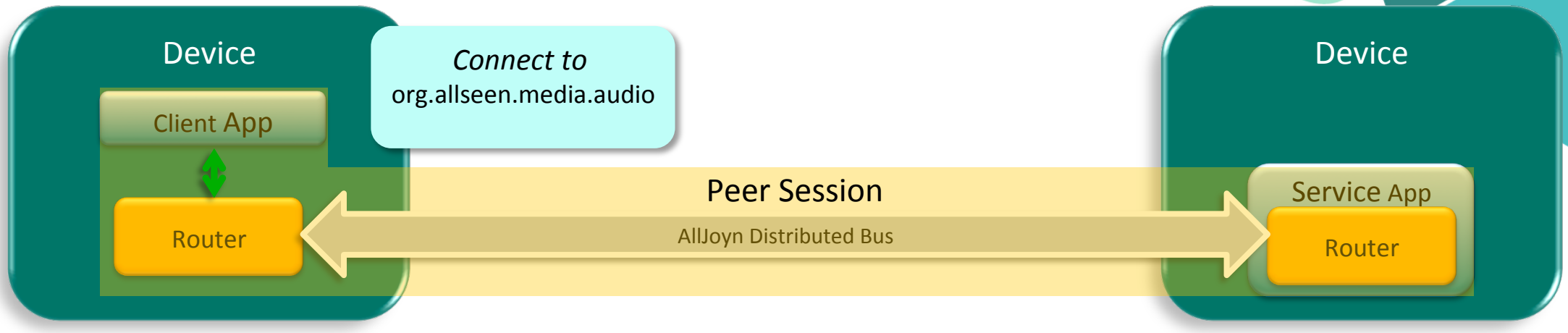
Interface descriptions contained in About message

- 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

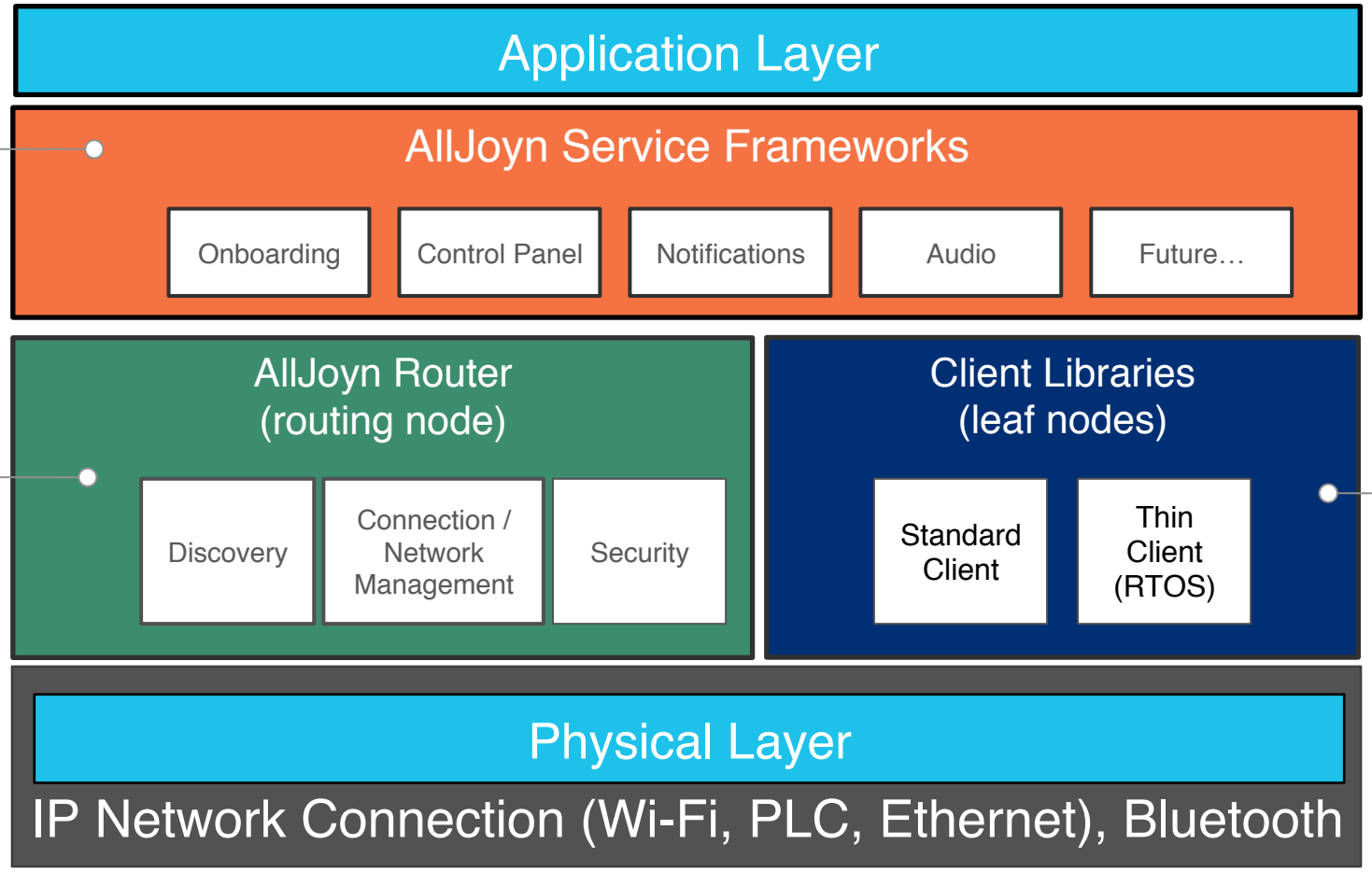


Session creation will cause Routing Nodes to connect and extend the bus

- 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

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

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

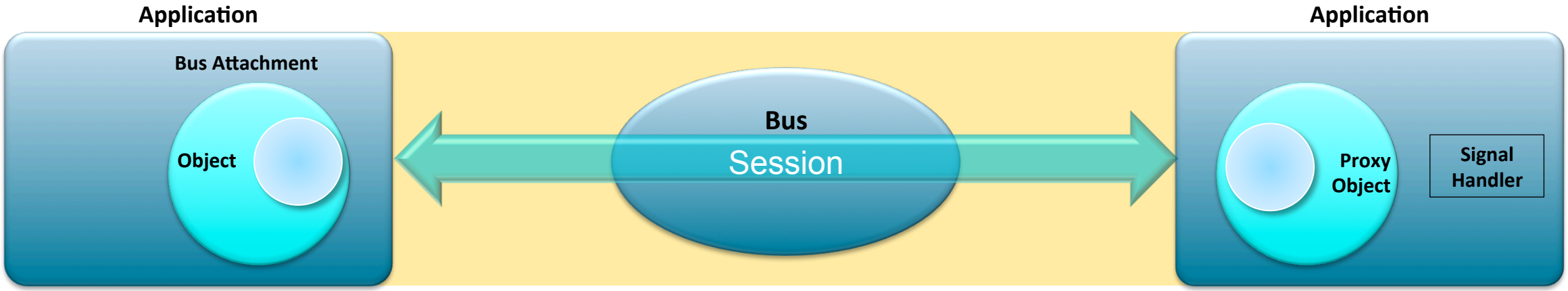
Signal handlers are registered with the Bus Attachment

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

Sessions

– Flow controlled connections between applications

– Can be point-to-point or multipoint



Recap from Core Training 1

Wellknown name

- Describes the service with a hint of what the application is.
  - Eg. “org.allseen.training”

Advertise

- Lets other applications know you’re there
- Publishes the wellknown name to others applications.

Discover

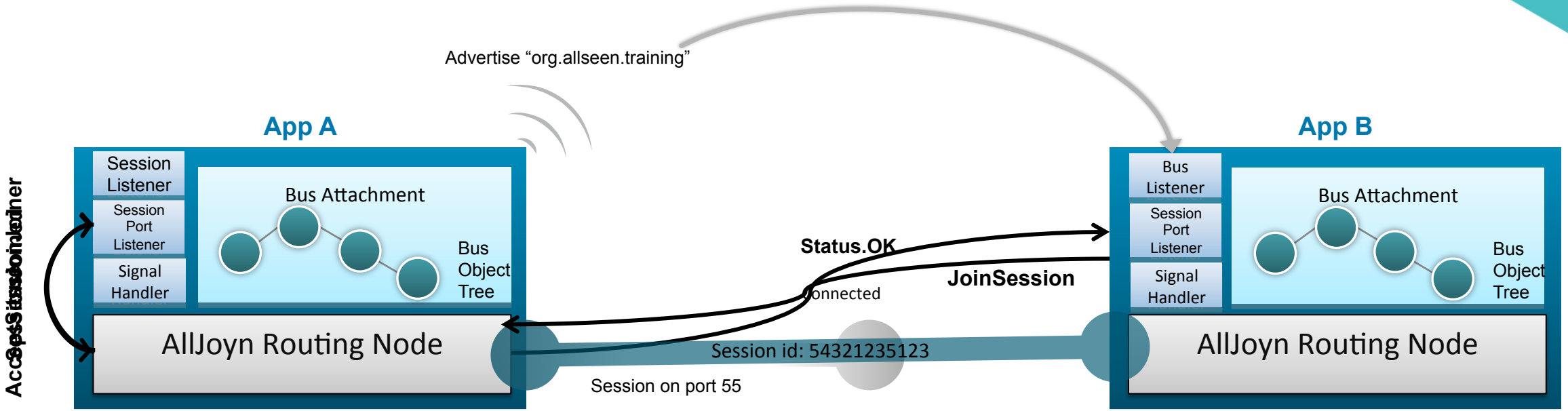
- Lets you find other applications nearby
- Looks for wellknown names that start with a specific prefix

Sessions

- A group of applications that are connected, allowing them to exchange data
- Each session has a unique identifier sessionId that allows for targeted interactions

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

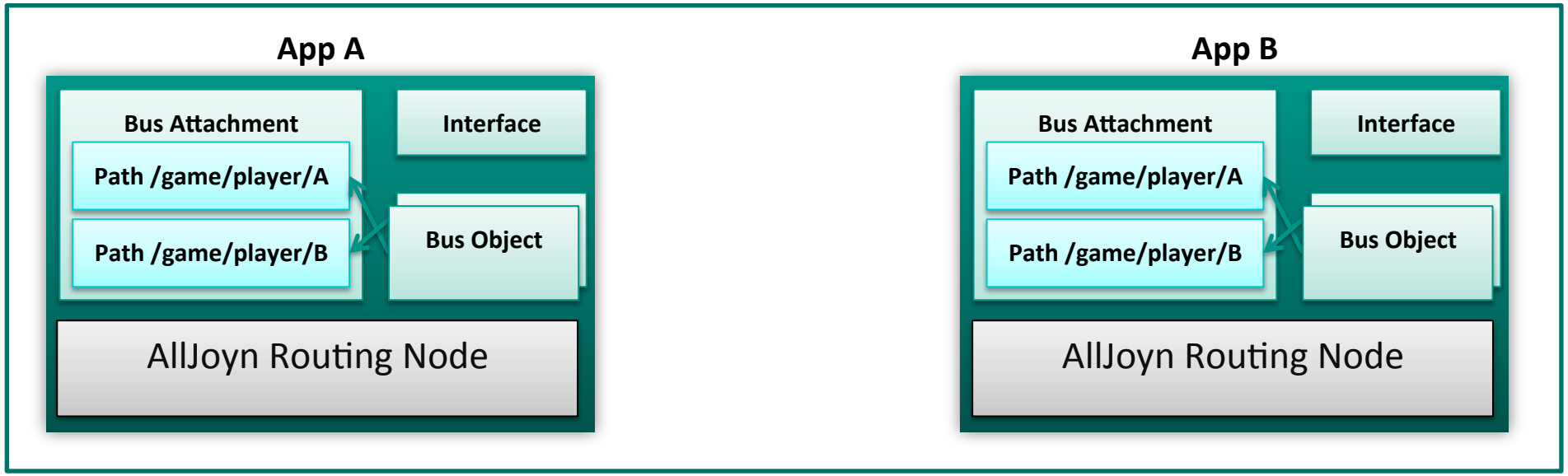
Bus Object

Overview

– 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

AllJoyn Interface

What it is

Defines the interactions that can occur from remote applications

- Think of this as a header file of the distributed system
- Can be introspected by remote Applications for dynamic integration

Collections of the following:

- Bus Method
  - Method call on a remote application
- Signal
  - Sent to all Applications in a session
- Property
  - Variable accessible via get/set
  - May not support both get/set up to implementer of interface

AllJoyn Interface

Data types

All Bus Methods, Signals and BusProperties can contain basic and complex data types

Basic types

– String (s), int (i), unsigned int (u), short (n), byte (y), double/float (d), long (x), and ObjectPath (o)

Arrays

– All data types can be set as an array
– Defined by adding the character (a) before the data type. Eg. Array of strings: (as)

Structs

– Struct is a collection of any combination of data types
– Defined by using the characters ((and) or r). Eg. A contact could be (name, email, number): (ssi) or rssi

Dictionary

– Key-value pair in which the key can be any of the primitive types and the value any data type
– Defined by using the characters ({ and } or e). Eg. A player table for a game could be: {sai} or eai

AllJoyn Interface

Data types

Variant

- AllJoyn provides for the ability of runtime data type detection
- This in tandem with Introspection can enable new innovative experiences
- Variants can be any type
- Defined by the character (v), Eg. A customer record may contain various fields defined in an array (av)

- Upon receipt of a Variant argument the data can be understood by using the typeId field
  - Once the type is understood it can be pulled out of the object to be used

AllJoyn Interface

XML Markup

An interface can be described via an XML markup using the following format:

- Interface tag: <interface name="[name]"> … </interface>

- BusMethod tag: <method name="[name]"> … </method>

- Signal tag: <signal name="[name]"> … </signal>

- Arguments to be used with BusMethods and Signal:

  <arg name="[name]" type="[data type]" direction="[in, out]"/>

- Property tag: <property name="[name]" type="[data type]" access="[read, write, readwrite]"/>

- BusObjects defined with collection of interfaces: <node name="[name]">

*text inside [ ] is strictly for show and would be an actual name, data type, or value defined

AllJoyn Interface

About Feature as XML

```
<node name="/About"
    xmlns:xsi=“http://www.w3.org/2001/XMLSchema-instance”
    xsi:noNamespaceSchemaLocation="http://www.allseenalliance.org/schemas/introspect.xsd">
    <interface name="org.alljoyn.About">
        <property name="Version" type="q" access="read"/>
        <method name="GetAboutData">
            <arg name="languageTag" type="s" direction="in"/>
           <arg name="aboutData" type="a{sv}" direction="out"/>
    </method>
    <method name="GetObjectDescription">
        <arg name="objectDescription" type="a(sas)"
         direction="out"/>
    </method>
    <signal name="Announce">
        <arg name="version" type="q"/>
        <arg name="port" type="q"/>
        <arg name="objectDescription" type="a(sas)"/>
        <arg name="metaData" type="a{sv}"/>
    </signal>
    </interface>
</node>
```

- Taken from About Interface Specification: http://allseenalliance.org/docs-and-downloads/documentation/alljoyn-about-feature-10-interface-specification

AllJoyn Interface

Inline code can be used as well, About Feature in C++

```
InterfaceDescription* p_InterfaceDescription =
      const_cast<InterfaceDescription*>(m_BusAttachment->GetInterface(ABOUT_INTERFACE_NAME));
    if (!p_InterfaceDescription) {
        CHECK_RETURN(m_BusAttachment->CreateInterface(ABOUT_INTERFACE_NAME, p_InterfaceDescription,
false))

        if (!p_InterfaceDescription) {
            return ER_BUS_CANNOT_ADD_INTERFACE;
        }

        CHECK_RETURN(p_InterfaceDescription->AddMethod("GetAboutData", "s", "a{sv}",
"languageTag,aboutData"))
        CHECK_RETURN(p_InterfaceDescription->AddMethod("GetObjectDescription", NULL, "a(oas)",
"Control"))
        CHECK_RETURN(p_InterfaceDescription->AddSignal("Announce", "qqa(oas)a{sv}",
"version,port,objectDescription,aboutData"))
        CHECK_RETURN(p_InterfaceDescription->AddProperty("Version", "q", (uint8_t) PROP_ACCESS_READ))
        p_InterfaceDescription->Activate();
    }

    status = AddInterface(*p_InterfaceDescription);
```

- Code snippet from About Feature: http://git.allseenalliance.org/cgit/core/alljoyn.git/tree/services/about/cpp/src/AboutService.cc

AllJoyn Interface

In Java we define interfaces and annotate them, About Feature in Java

```
@BusInterface (name = AboutTransport.INTERFACE_NAME)
public interface AboutTransport extends BusObject
{
      public static final String INTERFACE_NAME = "org.alljoyn.About";
      public final static String OBJ_PATH       = "/About”;

      @BusProperty(signature="q")
    public short getVersion() throws BusException;

      @BusMethod(signature = "s", replySignature="a{sv}")
    public Map<String, Variant> GetAboutData(String languageTag) throws BusException;


    @BusMethod(replySignature="a(oas)")
    public BusObjectDescription[] GetObjectDescription() throws BusException;


    @BusSignal (signature="qqa(oas)a{sv}")
    public void Announce(short version, short port, BusObjectDescription[] objectDescriptions,
Map<String,Variant> serviceMetadata);

}
```

- Code snippet from About Feature: http://git.allseenalliance.org/cgit/core/alljoyn.git/tree/services/about/java/src/org/alljoyn/about/transport/AboutTransport.java
- Comments removed to fit slide

AllJoyn Interface

In Thin Client Applications written in C we use a table, About Feature in C

```
static const char* const AboutInterface[] = {
    "org.alljoyn.About",
    "@Version>q",
    "?GetAboutData <s >a{sv}",
    "?GetObjectDescription >a(oas)",
    "!Announce >q >q >a(oas) >a{sv}",
    NULL
};
```

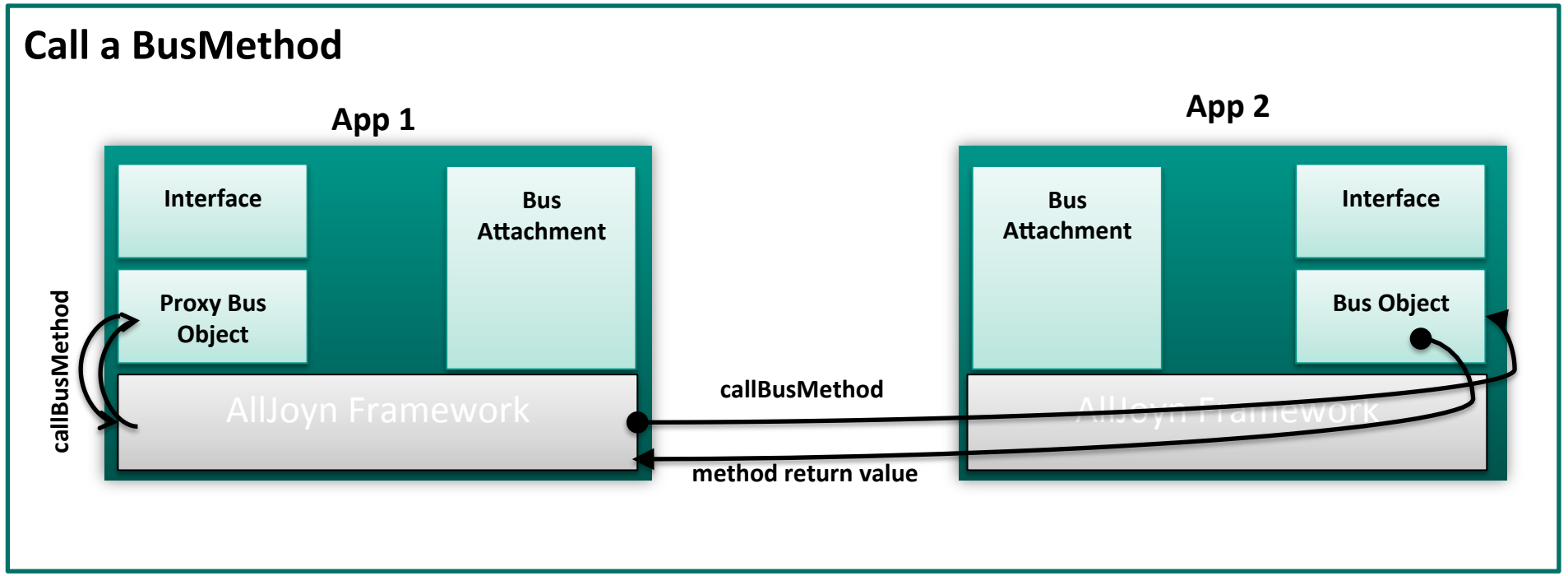
- Code snippet from About Feature: http://git.allseenalliance.org/cgit/core/about_tcl.git/tree/tcl/src/AboutService.c

Bus Method

What is this?

Send data synchronously from one connected application to another and receive a response

- BusMethods are remote method calls that are blocking when executed (synchronous)
- Function as a 1 to 1 interaction
- A session must be established in order to call a Bus Method on another application.
- A Proxy Bus Object is used as a local representation of the object on remote application



Bus Method

Proxy Bus Object creation – About Feature C++

```
ProxyBusObject* proxyBusObj = new ProxyBusObject(*m_BusAttachment, busName,
ABOUT_OBJECT_PATH, sessionId);
    if (!proxyBusObj) {
        return ER_FAIL;
    }

    do {
        CHECK_BREAK(proxyBusObj->AddInterface(*p_InterfaceDescription))

        Message replyMsg(*m_BusAttachment);
        CHECK_BREAK(proxyBusObj->MethodCall(ABOUT_INTERFACE_NAME,
                                    "GetObjectDescription", NULL, 0, replyMsg))

        const ajn::MsgArg * returnArgs = 0;
        size_t numArgs = 0;
        replyMsg->GetArgs(numArgs, returnArgs);
        if (numArgs == 1) {
```

- Code snippet from About Feature: http://git.allseenalliance.org/cgit/core/alljoyn.git/tree/services/about/cpp/src/AboutClient.cc

Bus Method

Proxy Bus Object creation – About Feature Java

```
public BusObjectDescription[] getBusObjectDescriptions() throws BusException
      {
            ProxyBusObject proxyObj = getProxyObject();
            // We make calls to the methods of the AllJoyn object through one of its interfaces.
            AboutTransport aboutTransport =  proxyObj.getInterface(AboutTransport.class);
            BusObjectDescription[] busObjectDescriptions = aboutTransport.GetObjectDescription();
            return busObjectDescriptions;
      }
```

- Code snippet from About Feature: http://git.allseenalliance.org/cgit/core/alljoyn.git/tree/services/about/java/src/org/alljoyn/about/client/AboutClientImpl.java

```
protected ProxyBusObject getProxyObject() throws BusException
      {
            if (!m_isConnected) {
                  throw new BusException("Session is not connected, need to check isConnected, and
reconnect.");
            }

            ProxyBusObject m_proxyBusObj = m_bus.getProxyBusObject( getPeerName(),
                                                                    getObjectPath(),
                                                                    getSessionId(),
                                                                  getObjClassArr());

            return m_proxyBusObj;
      }
```

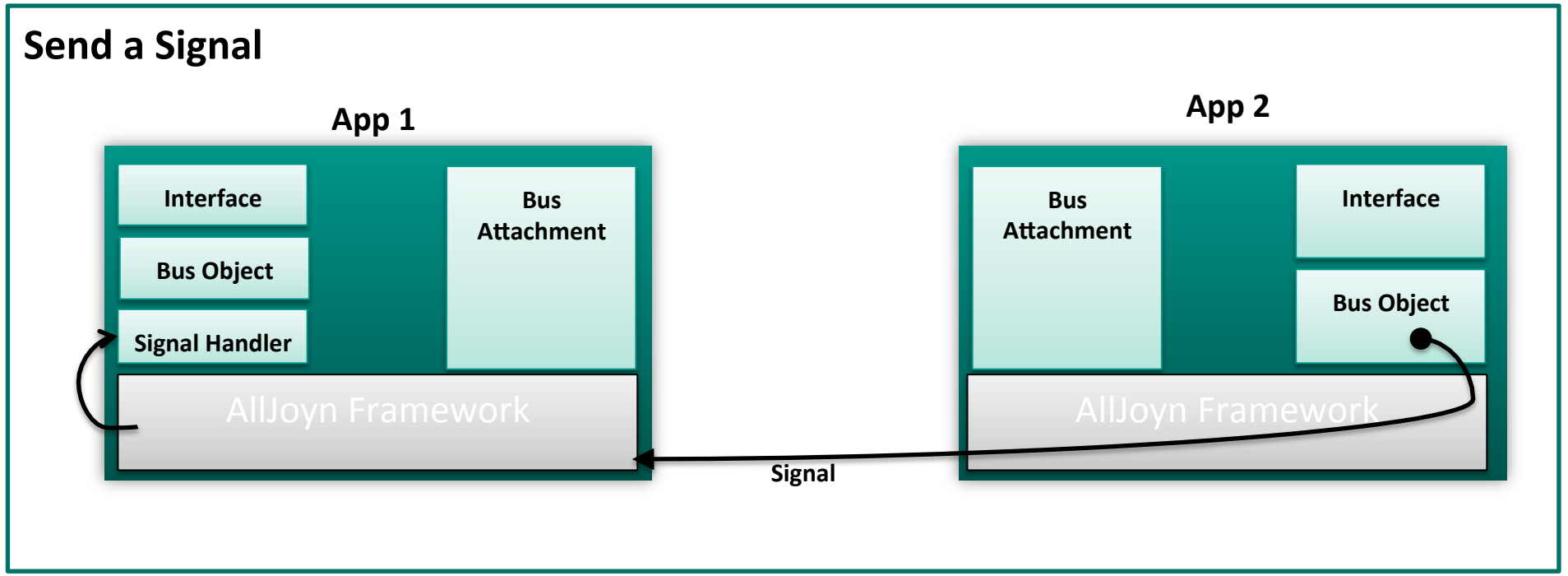
- Code snippet from About Feature: http://git.allseenalliance.org/cgit/core/alljoyn.git/tree/services/about/java/src/org/alljoyn/services/common/ClientBaseImpl.java

Signal

What is this?

Send data asynchronously from one connected application to all applications in a session

- Signals are asynchronous and do not have a reply
- Functions as a broadcast to all Applications
- A session or Routing nodes must be connected in order to transmit signals
- Signals are sent from Bus Objects and received by the registered Signal Handler
- Reliable delivery as long as other Applications have a connection



Signal

Send Signal – About Feature C++

```
MsgArg announceArgs[4];
CHECK_RETURN(announceArgs[0].Set("q", ABOUT_SERVICE_VERSION))
CHECK_RETURN(announceArgs[1].Set("q", m_AnnouncePort))
std::vector<MsgArg> announceObjectsArg(m_AnnounceObjectsMap.size());
int objIndex = 0;
for (std::map<qcc::String, std::vector<qcc::String> >::const_iterator it = m_AnnounceObjectsMap.begin();
     it != m_AnnounceObjectsMap.end(); ++it) {

    qcc::String objectPath = it->first;
    std::vector<const char*> interfacesVector(it->second.size());
    std::vector<qcc::String>::const_iterator interfaceIt;
    int interfaceIndex = 0;

    for (interfaceIt = it->second.begin(); interfaceIt != it->second.end(); ++interfaceIt) {
        interfacesVector[interfaceIndex++] = interfaceIt->c_str();
    }

    CHECK_RETURN(announceObjectsArg[objIndex].Set("(oas)", objectPath.c_str(),
                                                  interfacesVector.size(), interfacesVector.data()))
    objIndex++;
}
CHECK_RETURN(announceArgs[2].Set("a(oas)", objIndex, announceObjectsArg.data()))
CHECK_RETURN(m_PropertyStore->ReadAll(NULL, PropertyStore::ANNOUNCE, announceArgs[3]))
Message msg(*m_BusAttachment);
uint8_t flags = ALLJOYN_FLAG_SESSIONLESS;
status = Signal(NULL, 0, *m_AnnounceSignalMember, announceArgs, 4, (unsigned char) 0, flags);
```

- Code snippet from About Feature: http://git.allseenalliance.org/cgit/core/alljoyn.git/tree/services/about/cpp/src/AboutService.cc

Signal

```
SignalEmitter emitter = new SignalEmitter(m_aboutInterface, SignalEmitter.GlobalBroadcast.Off);
emitter.setSessionlessFlag(true);
emitter.setTimeToLive(0);
m_announcementEmitter = emitter.getInterface(AboutTransport.class);

…
…

BusObjectDescription[] objectDescriptionArray = m_ObjectDescriptions.toArray(new
BusObjectDescription[] {});
Map<String, Object> persistedAnnounceMap      = new HashMap<String, Object>();
try {
     m_propertyStore.readAll("", Filter.ANNOUNCE, persistedAnnounceMap);
} catch (PropertyStoreException pse) {
     throw new AboutServiceException("Failed to read announcable properties from the
PropertyStore, Error: '" + pse.getMessage() + "'");
}
     Map<String, Variant>announceMap = TransportUtil.toVariantMap(persistedAnnounceMap);
     m_announcementEmitter.Announce((short)PROTOCOL_VERSION, m_servicesPort,
objectDescriptionArray, announceMap);
```

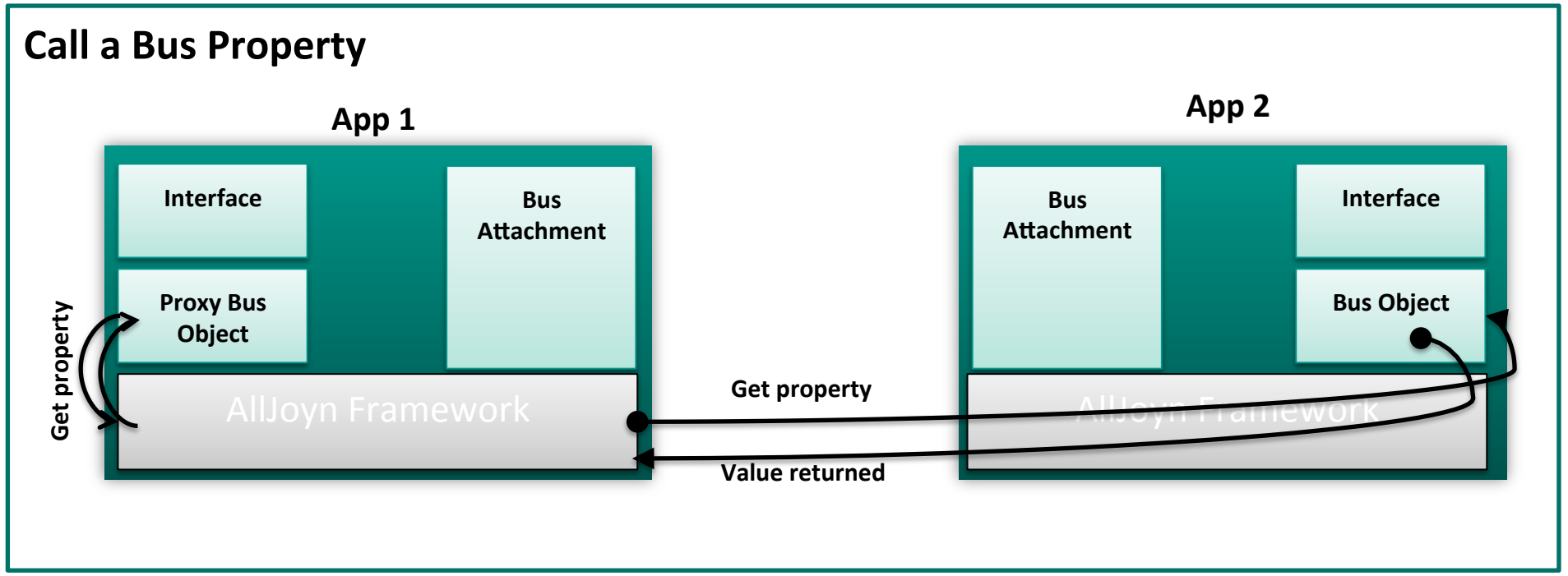
- Code snippet from About Feature: http://git.allseenalliance.org/cgit/core/alljoyn.git/tree/services/about/java/src/org/alljoyn/about/AboutServiceImpl.java

Bus Property

What is this?

Similar to a BusMethod except sole purpose is for get/set methods

- Conceptually a variable that can be read and written
- When defined the AllJoyn framework will perform access checks, Eg. read variable can not be written to
- Functions and called the same way as a Bus Method



Bus Property

Service side special Get method – About Feature C++

On the Service Side we receive a Get request upon access

```
QStatus AboutService::Get(const char*ifcName, const char*propName, MsgArg& val)
{
    QCC_DbgTrace(("AboutService::%s", __FUNCTION__));
    QStatus status = ER_BUS_NO_SUCH_PROPERTY;
    if (0 == strcmp(ifcName, ABOUT_INTERFACE_NAME)) {
        if (0 == strcmp("Version", propName)) {
            status = val.Set("q", ABOUT_SERVICE_VERSION);
        }
    }
    return status;
}
```

- Code snippet from About Feature: http://git.allseenalliance.org/cgit/core/alljoyn.git/tree/services/about/cpp/src/AboutService.cc

Bus Property

Make request to Get or Set – About Feature C++

```
 const InterfaceDescription* p_InterfaceDescription = m_BusAttachment-
>GetInterface(ABOUT_INTERFACE_NAME);
    if (!p_InterfaceDescription) {
        return ER_FAIL;
    }
    ProxyBusObject* proxyBusObj = new ProxyBusObject(*m_BusAttachment, busName,
ABOUT_OBJECT_PATH, sessionId);
    if (!proxyBusObj) {
        return ER_FAIL;
    }
    MsgArg arg;
    if (ER_OK == proxyBusObj->AddInterface(*p_InterfaceDescription)) {
        status = proxyBusObj->GetProperty(ABOUT_INTERFACE_NAME, "Version", arg);
        if (ER_OK == status) {
            version = arg.v_variant.val->v_int16;
        }
    }
```

- Code snippet from About Feature: http://git.allseenalliance.org/cgit/core/alljoyn.git/tree/services/about/cpp/src/AboutClient.cc

Bus Property

Service side special Get method – About Feature Java

For Java we define in the interface our get and set methods then annotate

```
@BusInterface (name = AboutTransport.INTERFACE_NAME)
public interface AboutTransport extends BusObject
{
    public static final String INTERFACE_NAME = "org.alljoyn.About";
    public final static String OBJ_PATH       = "/About”;

    @BusProperty(signature="q")
   public short getVersion() throws BusException;
```

- Code snippet from About Feature: http://git.allseenalliance.org/cgit/core/alljoyn.git/tree/services/about/java/src/org/alljoyn/about/transport/AboutTransport.java

Bus Method

Make request to Get or Set – About Feature Java

```
public  short getVersion() throws BusException
     {
          ProxyBusObject proxyObj = getProxyObject();
          // We make calls to the methods of the AllJoyn object through one of its interfaces.
          AboutTransport aboutTransport =  proxyObj.getInterface(AboutTransport.class);
          return aboutTransport.getVersion();
     }
```

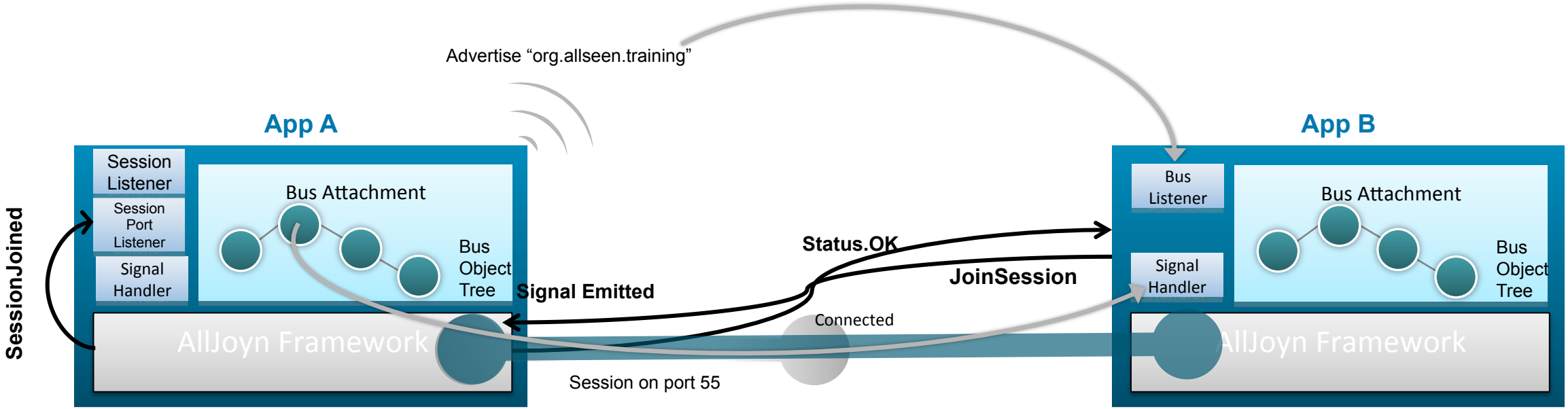
- Code snippet from About Feature: http://git.allseenalliance.org/cgit/core/alljoyn.git/tree/services/about/java/src/org/alljoyn/about/client/AboutClientImpl.java

```
protected ProxyBusObject getProxyObject() throws BusException
     {
          if (!m_isConnected) {
               throw new BusException("Session is not connected, need to check isConnected,
and reconnect.");
          }

          ProxyBusObject m_proxyBusObj = m_bus.getProxyBusObject( getPeerName(),
                                                                  getObjectPath(),
                                                                  getSessionId(),
                                                                  getObjClassArr());

          return m_proxyBusObj;
     }
```

- Code snippet from About Feature: http://git.allseenalliance.org/cgit/core/alljoyn.git/tree/services/about/java/src/org/alljoyn/services/common/ClientBaseImpl.java

Sessionless Signal

Sends a Signal without needing to be in a Session



Sessionless Signal

Sends a Signal without needing to be in a Session

Sessionless Signal

Sends a Signal without needing to be in a Session

Signal has flag to specify if Sessionless when sent

- About Feature and Notification Service Framework uses sessionless signals
- Avoid worry about which application advertises/joins and which discovers/binds
- Only to be used for small data transmissions that fit into a single signal
  - Not for file transfer or image transfer
- Not intended for high level traffic
  - Each signal overwrites the previous signal if not yet delivered

Sessionless Signal

Send Signal – About Feature C++

```
MsgArg announceArgs[4];
CHECK_RETURN(announceArgs[0].Set("q", ABOUT_SERVICE_VERSION))
CHECK_RETURN(announceArgs[1].Set("q", m_AnnouncePort))
std::vector<MsgArg> announceObjectsArg(m_AnnounceObjectsMap.size());
int objIndex = 0;
for (std::map<qcc::String, std::vector<qcc::String> >::const_iterator it = m_AnnounceObjectsMap.begin();
     it != m_AnnounceObjectsMap.end(); ++it) {

    qcc::String objectPath = it->first;
    std::vector<const char*> interfacesVector(it->second.size());
    std::vector<qcc::String>::const_iterator interfaceIt;
    int interfaceIndex = 0;

    for (interfaceIt = it->second.begin(); interfaceIt != it->second.end(); ++interfaceIt) {
        interfacesVector[interfaceIndex++] = interfaceIt->c_str();
    }

    CHECK_RETURN(announceObjectsArg[objIndex].Set("(oas)", objectPath.c_str(),
                                                  interfacesVector.size(), interfacesVector.data()))
    objIndex++;
}
CHECK_RETURN(announceArgs[2].Set("a(oas)", objIndex, announceObjectsArg.data()))
CHECK_RETURN(m_PropertyStore->ReadAll(NULL, PropertyStore::ANNOUNCE, announceArgs[3]))
Message msg(*m_BusAttachment);
 uint8_t flags = ALLJOYN_FLAG_SESSIONLESS;
status = Signal(NULL, 0, *m_AnnounceSignalMember, announceArgs, 4, (unsigned char) 0, flags);
```

- Code snippet from About Feature: http://git.allseenalliance.org/cgit/core/alljoyn.git/tree/services/about/cpp/src/AboutService.cc

Sessionless Signal

```
SignalEmitter emitter = new SignalEmitter(m_aboutInterface, SignalEmitter.GlobalBroadcast.Off);
emitter.setSessionlessFlag(true);
emitter.setTimeToLive(0);
m_announcementEmitter = emitter.getInterface(AboutTransport.class);

…
…

BusObjectDescription[] objectDescriptionArray = m_ObjectDescriptions.toArray(new
BusObjectDescription[] {});
Map<String, Object> persistedAnnounceMap      = new HashMap<String, Object>();
try {
     m_propertyStore.readAll("", Filter.ANNOUNCE, persistedAnnounceMap);
} catch (PropertyStoreException pse) {
     throw new AboutServiceException("Failed to read announcable properties from the
PropertyStore, Error: '" + pse.getMessage() + "'");
}
     Map<String, Variant>announceMap = TransportUtil.toVariantMap(persistedAnnounceMap);
     m_announcementEmitter.Announce((short)PROTOCOL_VERSION, m_servicesPort,
objectDescriptionArray, announceMap);
```

- Code snippet from About Feature: http://git.allseenalliance.org/cgit/core/alljoyn.git/tree/services/about/java/src/org/alljoyn/about/AboutServiceImpl.java

Recap

Application flow

Connect to AllJoyn

- Create a BusAttachment
- Call connect and setup listeners
- Implement BusObjects which implement AllJoyn Interfaces

Advertise and/or Discover

- Lets you find other applications nearby
- Looks for wellknown names that start with a specific prefix

Sessions

- A group of applications that are connected, allowing them to exchange data
- Each session has a unique identifier sessionId that allows for targeted interactions

Interact with Bus Objects on other Applications

- Fixed interaction by containing the same interfaces in Applications
- Dynamically discover and create via Introspection

Source Code license information

Source code contained in this presentation is licensed by the AllSeen Alliance under the ISC open source license:


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!