



JAIN SLEE Tutorial

Introducing JAIN SLEE

Swee Lim
Sun Microsystems

Phelim O'Doherty
Sun Microsystems

David Ferry
Open Cloud

David Page
Open Cloud

Tutorial Objectives

Understand the aims, objectives and benefits of JAIN SLEE

Understand the fundamental concepts of the JAIN SLEE application environment

Identify some considerations related to realizing a JAIN SLEE in core carrier networks

Outline

- **JAIN SLEE in Context**
- Why create JAIN SLEE?
- JAIN SLEE Concepts
- Implementation Considerations

SLEE in Communication Networks

- 2G and 2.5G Networks
 - Service Control Point
 - Service Node (2G to 3G using a Media Gateway)
- 3G Networks
 - Service Switching Control Point
 - Service Switching Point
 - SIP Proxy
 - 3GPP IMS CSCFs
- Convergent Networks
 - Gatekeeper
 - Common Service Delivery Platform
 - Convergent SCP, SSCP, SSP
 - OSA Gateway

Other Technologies and SLEE

- Auto-ID
- Manufacturing
- Real-time health monitoring
- Industrial flow control
- ...

Why are Communications Applications Converging on Java Containers?

- Telco apps moving to component based architectures
- Desire to use Standard, Off-the-shelf container
 - Write-once, run-anywhere
- Container provides important infrastructure services
 - Higher level abstractions for State management, Transactions, Security, Resource pooling, ...
- Focus on core value-add application logic
- Leverage large community of Java developers
- Leverage enterprise development tools, test suites, ...

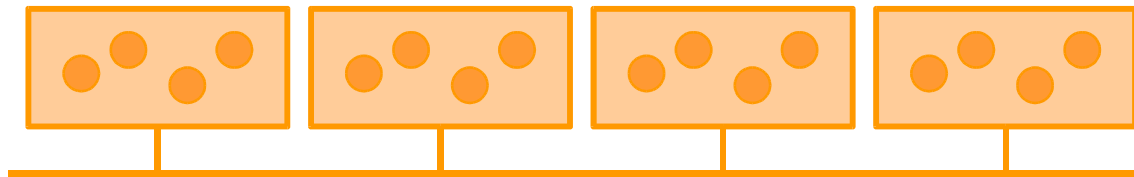
**Time to market and reduced
development cost**

Outline

- JAIN SLEE in Context
- **Why create JAIN SLEE?**
- JAIN SLEE Concepts
- Implementation Considerations

What is JAIN SLEE?

- **SLEE** = **S**ervice **L**ogic **E**xecution **E**nvironment
- Low latency and high throughput application environment for event processing
 - Latency < 100 ms
 - 100's to 1000's of events per second
- Event orientated
- Designed for stringent requirements of core network signaling application
- Designed to allow implementations to support scalability and availability through clustering



JAIN SLEE Benefits

- High performing platform for event driven applications
 - Supports simple and complex applications
 - Applications deal with service logic only
 - System issues handled by container i.e. threading, transactions
- Standard application framework
 - Defined programming model
 - Object Orientated, asynchronous, robust and distributable
- Independent of underlying networks
- Asynchronous support
 - Elaborate event distribution mechanism (with priority)
 - Maps events to method invocations on components
 - Creates component instances in response to initial events

**SLEE reduces cost and improves
time-to-market**

JAIN SLEE Benefits

To Communication Developers

- **Only industry standard Service Logic Execution Environment**
 - Write-once, run anywhere for components
- **Availability and scalability through clustering**
 - Versus traditional primary-secondary
- **Easy to develop robust components**
 - SLEE replicates state and provides transactional semantics
 - Strongly typed component interfaces and profile data
- **Point of integration for multiple protocols and resources**
 - One container, multiple resources, protocols
 - Easy to integrate new technologies

JAIN SLEE Differentiators

To EJB Developers

- Event orientated component model
 - Elaborate event distribution mechanism (with priority)
 - Maps events to method invocations on components
 - Event processing components with strongly typed interfaces
 - Event types received and sends
 - Private state and state shared with other components
 - Component instances have no external and permanent identity
 - Simple and dynamic event subscription model
 - Container manages component lifecycle and Garbage Collection
 - Enables automatic component instance creation and deletion
 - Model is aware of event producer and consumer relationships
 - Important for robustness (avoid dangling component instances)
- Profiles for provisioned data
 - Easy to define, provision, and access profiles

SLEE & J2EE

- SLEE is a component model like EJB, Servlet or JSP, and is most similar to EJB.
- SLEE builds upon concepts in J2EE and other component technologies but is a specialized component model for event driven applications.
- The SLEE can be implemented independent of J2EE and used stand-alone without requiring a J2EE adjunct
 - not dependent on J2EE technologies to outsource critical functions like concurrency control or failure resilience
- SLEE is not a component of J2EE and is not the equivalent of J2EE
- SLEE and J2EE are complementary technologies

SLEE, EJB & JMS

- SLEE has an built-in event model that is part of the component model
 - JMS is external to the EJB component model, it is more like a service
- SLEE allows dynamic creation of activities (like topics and queues) and dynamic association of event sources and event sinks to these activities
 - Binding between JMS queues, topics into the EJB runtime is static, there is no way to dynamically create new queues, topics and dynamically update the event routing mechanism under program control
- SLEE event model facilitates component garbage collection
- JMS can be integrated into SLEE through a Resource Adaptor just as in EJB via a Connector

Outline

- JAIN SLEE in Context
- Why create JAIN SLEE?
- **JAIN SLEE Concepts**
- Implementation Considerations

Application Characteristics

	Communications	Enterprise
Invocations	Typically asynchronous -Events such as protocol triggers -Event occurrence mapped to method invocation	Typically synchronous -Database, EAI systems -RPC Calls
Event Granularity	Fine-grained events High Frequency	Course-grained events Low Frequency
Components	Light-weight fine-grained objects Short transient lifetimes -Rapid creation, deletion	Heavy weight data access objects Long persistent lifetimes
Data Sources	Multiple data sources -Location, context information -Provisioned data, cached from master copy	Database servers -Definitive master copy Back-end systems
Transactions	Light-weight transactions -For state replication demarcation -Faster completion and more frequent	Database transactions -Slower completion and less frequent
Computation	Compute-intensive -Processing is resource invocations & events	Database access intensive

Application Characteristics

	<i>Communications</i>	<i>Enterprise</i>
Availability	3 to 5 9' s	2 to 3 9' s
Real-time	Soft real-time	
Deployment Distribution	Distributed deployment throughout network	Centralized deployment in small number of data centers
Nodes	1 to 4 CPUs	2 to 32 CPUs
Cluster Sizes	2 to 16 nodes	2 to 4 nodes

**Applications characteristics drive
Container Design!**

Major Subsystems

SLEE Management

JMX, MBeans, SLEE, Service Deployment, Service Mgt, Profile Mgt, ...

SLEE Framework Components

Event Routing, Profile, Facilities, Timers, ...

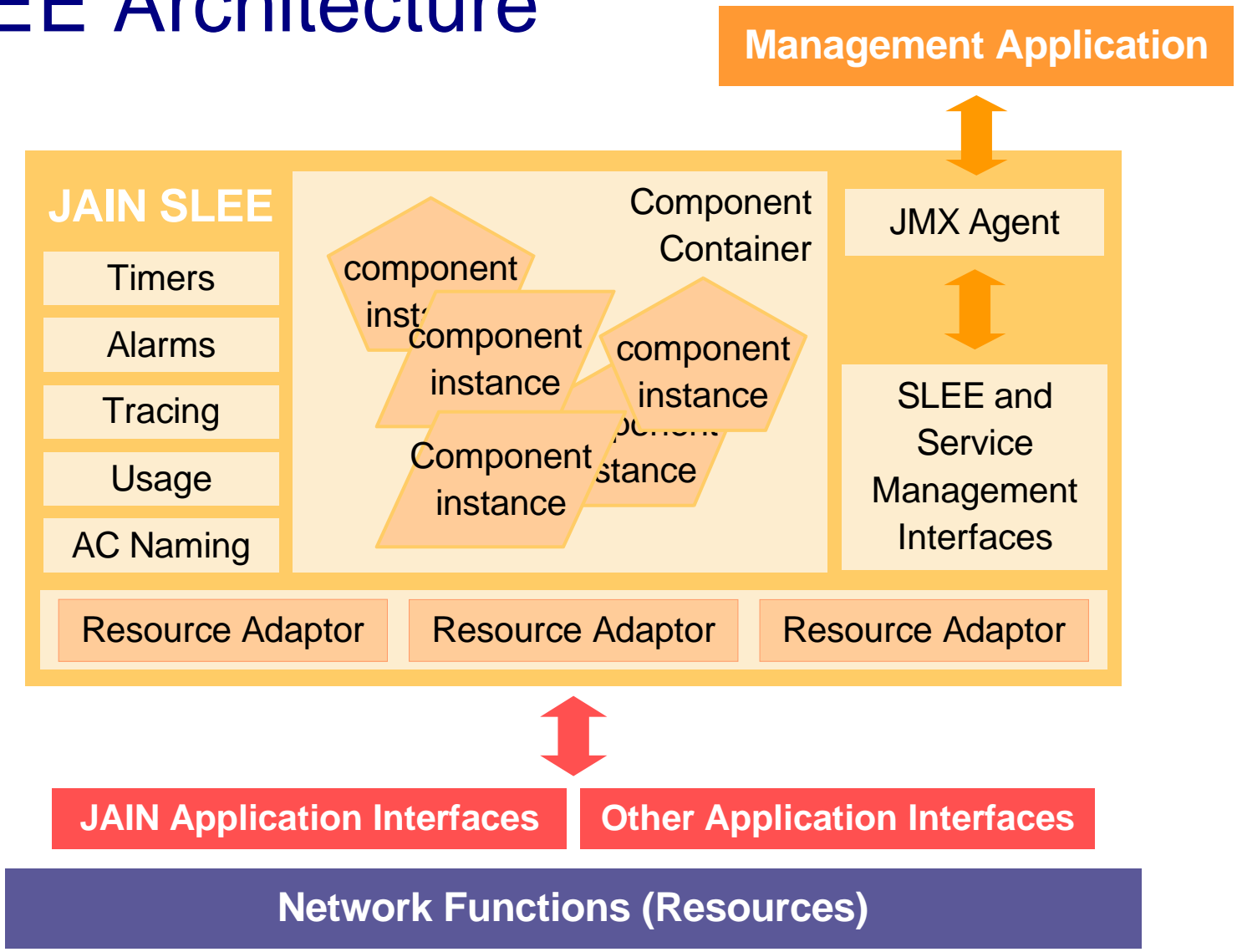
Component Model

Lifecycle, Packaging, Lookup, Events, Invocation Semantics, ...

Resource Adaptors and Resource APIs

JCC (Call Control), SIP, INAP, TCAP, SMPP, MM7, ...

SLEE Architecture



Types of Components

- **Service Building Block (SBB)**
 - Contains application and service logic (like an EJB)
 - Event handler, local interface, life cycle, callback method implementations
 - An SBB can be composed from one or more “child” SBBs
- **Service**
 - Specifies information needed to create SBBs to process “initial events”
- **Event Types**
 - Declares an event type
 - Determines how the event is routed and event class of the event
 - Every event has an event type
- **Profile Specification**
 - Specifies the schema of Profile Tables, Profile Table contains Profiles
 - Verification logic for provisioned data in a Profile

SBB and Service

- An **SBB** is a **S**ervice **B**uilding **B**lock
 - A software component
 - A programmatic piece of a Service
- A Service instance can contain one or many SBB instances of different types of SBB
- The same SBB can be included in multiple Services
- A single SBB can only process one event at a time
- Multiple SBBs belonging to the same Service can process events in parallel

SBB Entity

- Instance of an SBB
 - It is a logical entity
 - In the simplest form there is one SBB entity for a service instance
- Represents the persistent per-instance state
 - There exists some managed representation of the entity state that has transactional properties
 - The managed representation could be in process memory, replicated memory, on disk storage etc
 - The per-instance state of an SBB instance is defined by the CMP fields in the SBB abstract class of the SBB component
- Root SBB entity is the root node in an SBB entity tree
 - Instance of a root SBB
 - Instantiated by the SLEE to process its initial event

Resources and Resource Adaptors

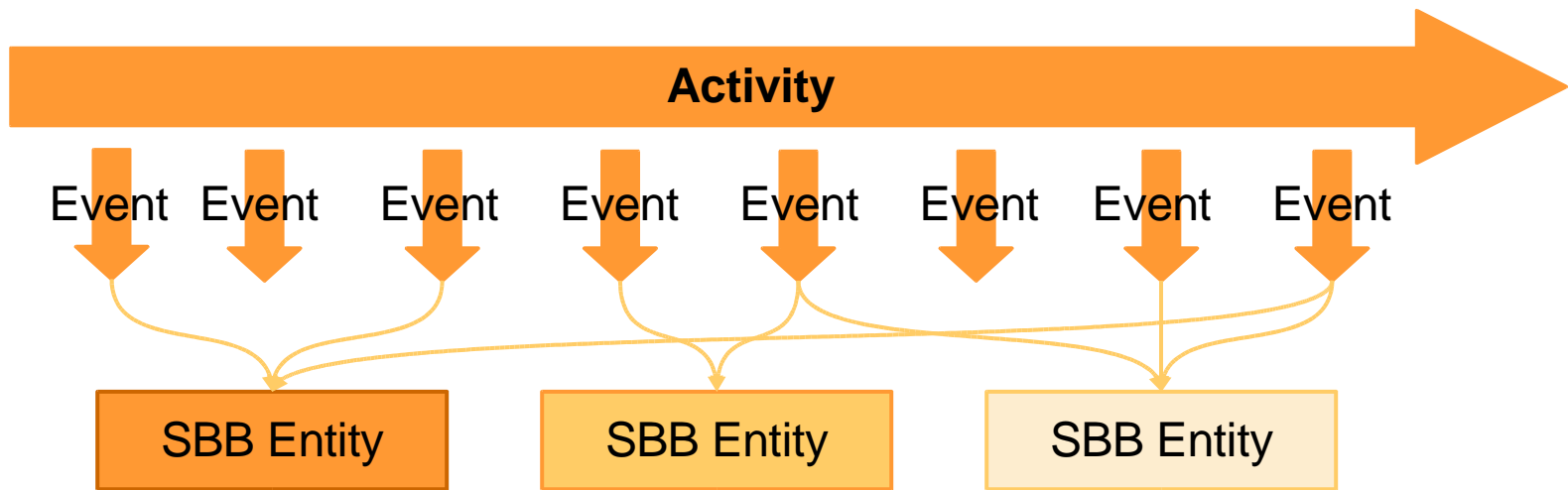
- JAIN SLEE represents an external resource as resource adaptor
 - 'Adapts' the resource for the use of SLEE applications
 - Each resource adaptor has a type
 - e.g. resource adaptor type for JAIN SIP is `javax.sip`
- JAIN SLEE identifies Event by Event types
 - For example JAIN SIP Events are classified as RequestEvents, ResponseEvents and TimeoutEvents, each of these contains numerous 'event types'. The event type of a Request message of type INVITE is `javax.sip.message.Request.INVITE`
- JAIN SLEE represents the flow of events as activities
 - Activities in JAIN SIP are ClientTransactions (locally initiated) and ServerTransactions (remotely initiated)

Event Model

- SLEE uses the publish/subscribe model for event distribution
- SBBs attach to and detach from activity contexts for event distribution
- The SLEE architecture does not allow SBBs to register themselves explicitly as listeners of a resource
 - Does not follow JavaBean event model
- In the SLEE model the resource adaptor has outsourced event subscription and arbitration to the SLEE
 - Allows resource adaptors to be consistent in their event model

Activity

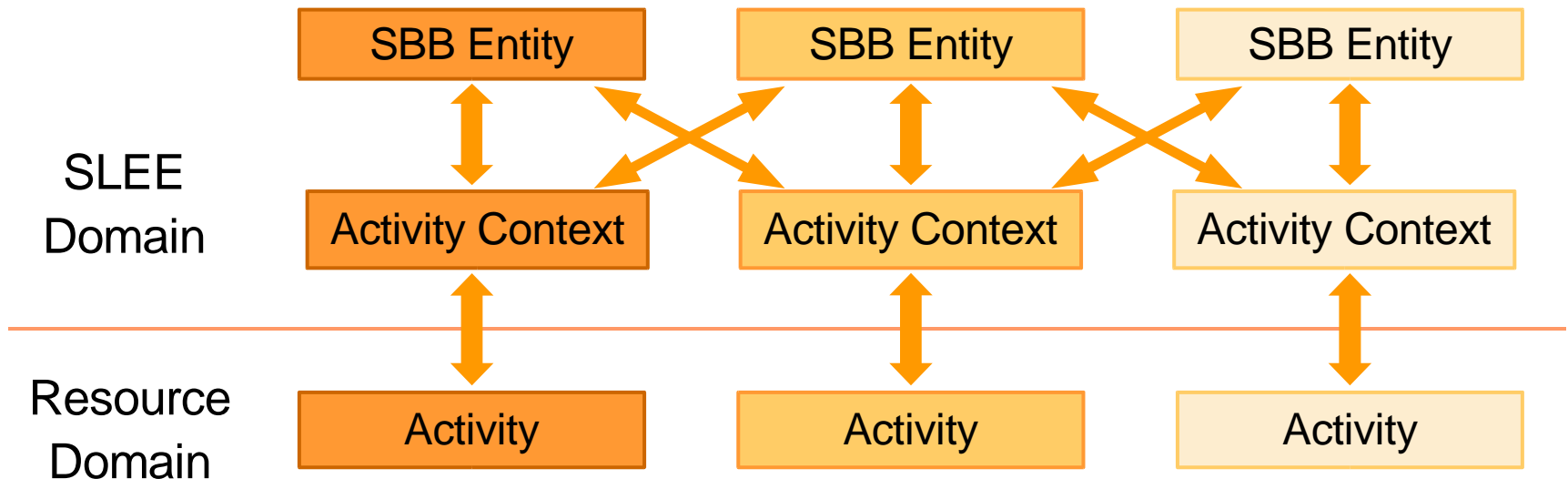
- Abstraction for a related stream of events
- Examples
 - Call object emitting call connected, disconnected, ... events
 - Mobile location report object emitting location update, ... events



Events are **routed** to SBB entity that are interested in them.

Activity Context

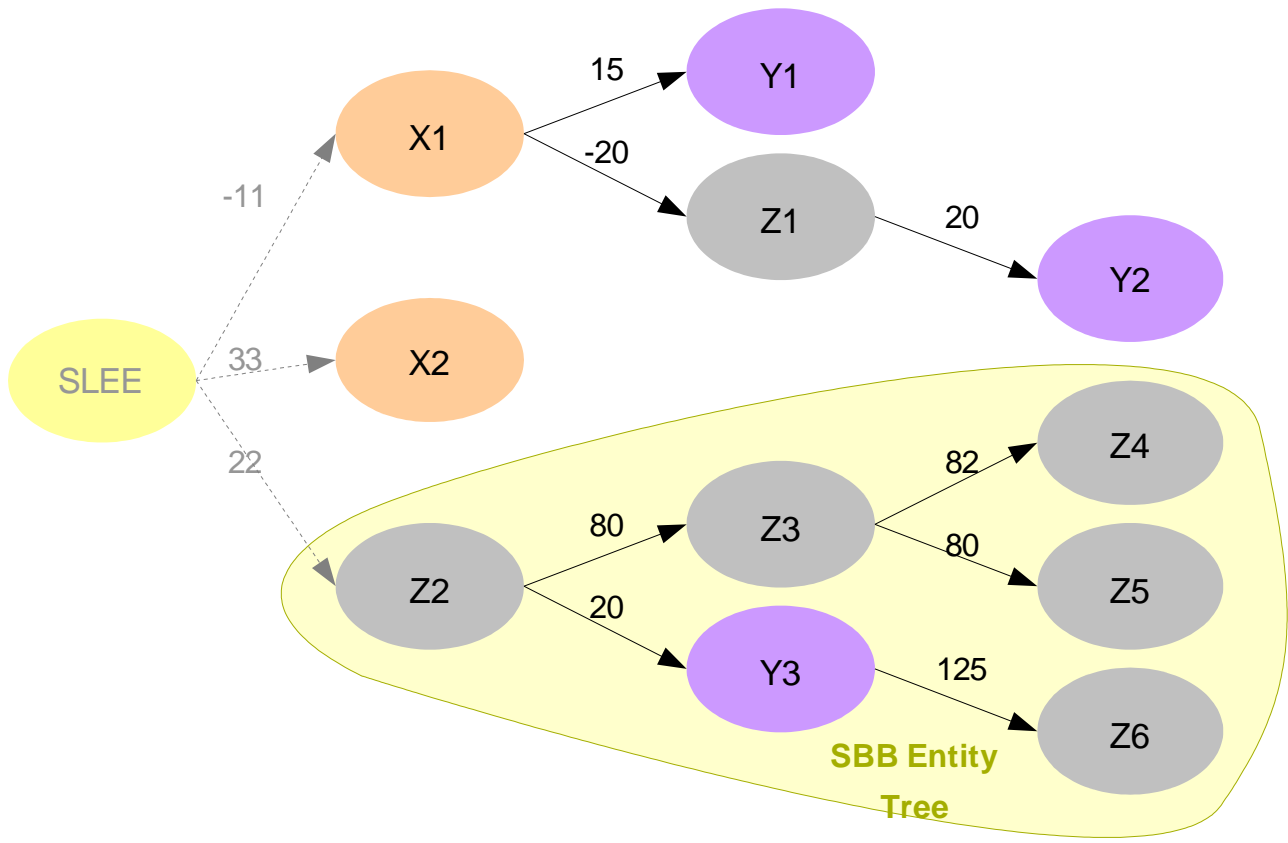
- SLEE domain object to encapsulate Activity defined by resources
- An event channel on which events are fired and delivered on - like a JMS topic
- Holds shared state regarding the Activity
- Many-to-many attachment relationships
- SBBs only receive events from attached Activities



Activity & Activity Context

- An Activity object is a resource object.
 - Represents some resource system that encapsulates a flow of events.
- A single Activity can only process one event at a time.
- An ActivityContext is the SLEE representation of a resources Activity.
- SBBs receive events on an ActivityContext.
- An ActivityContextInterface is a SBBs view of the ActivityContext.
 - SBBs may read and write state in an ActivityContext.
- Each ActivityContext has a 1-2-1 relationship with an Activity object.
- Each SBB defines a Java interface that represents their view onto the state stored in the ActivityContext.
 - This provides type safety and reduces bugs as a contract is defined between components, activity contexts and the state being modified.
- The appropriate ActivityContext is gained by passing an Activity to the ActivityContextInterfaceFactory.

SBB Entity Trees



SBB entity tree shows parent child relations among instantiated SBB entities

SLEE is logical parent of all *root SBB entities*

Root SBB entities are instances of SLEE instantiated *root SBBs*

	SBB entity
	parent to child relation
-20	actual priority

- Event delivery order (by priority, parent then child)
- Cascading removal of SBB entity sub-tree

SBB Entities & SBB Objects

SBB Entity

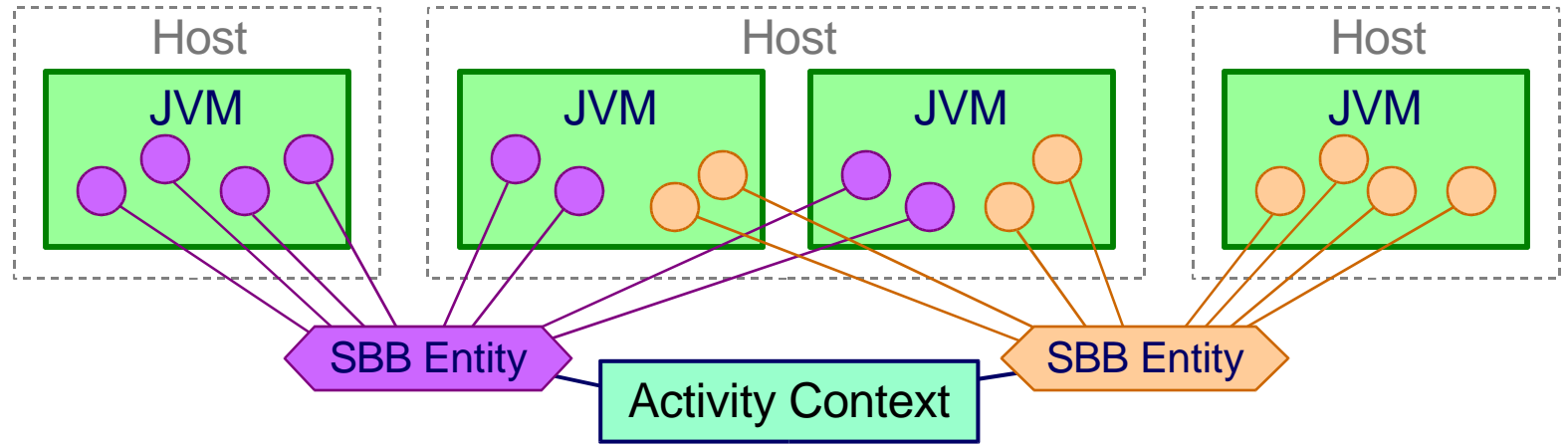


- Instance of **SBB**
- A logical entity
- **Represents persistent state** of SBB (as defined by CMP fields)
- **Maintains relations** with other entities (attached Activity Contexts, child SBB entities, ...)

SBB Object



- An **SBB class** instance
- A Java Object
- **Caches persistent state** of an SBB entity
- Has a life cycle (may be pooled)
- May cache different SBB entities through its life time

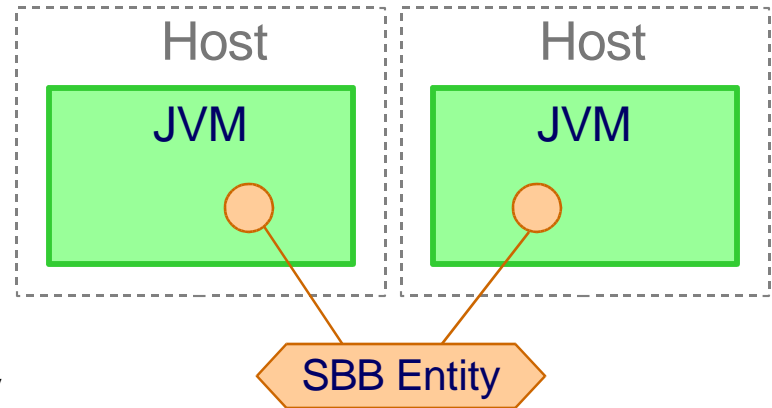


Transactions

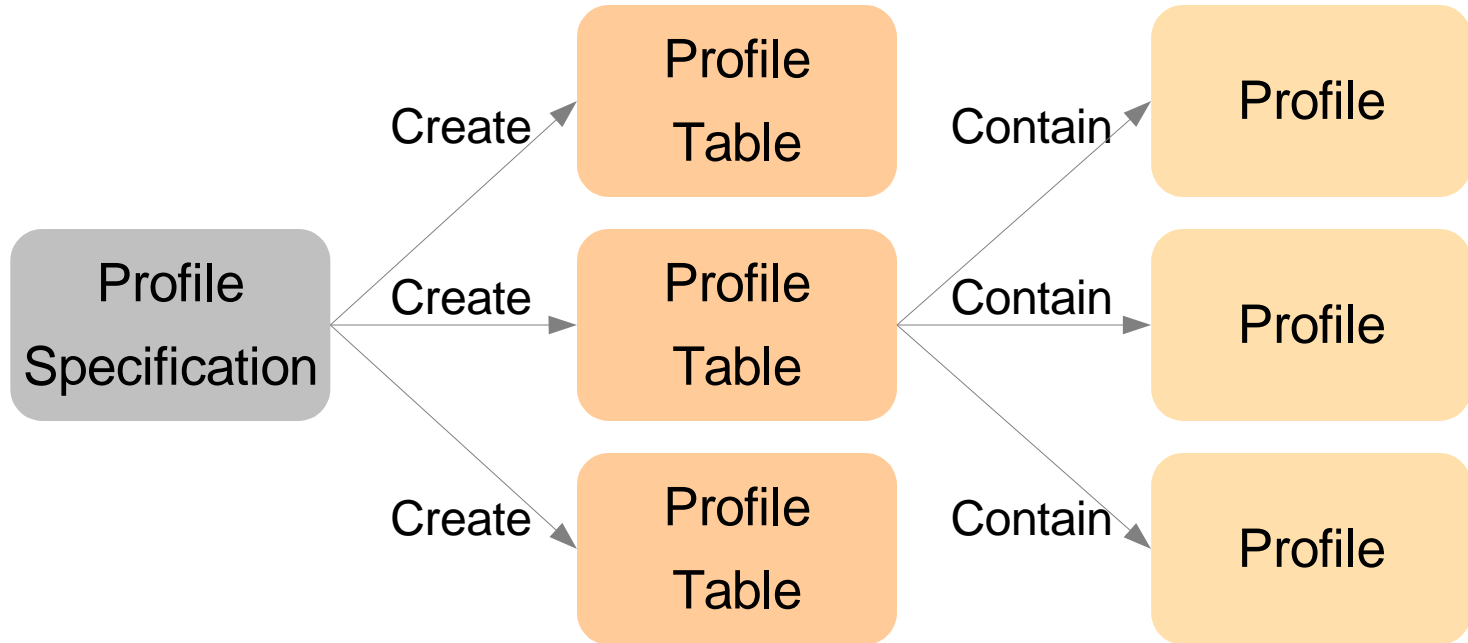
- Simplify the job of the SBB developer
 - SLEE uses transaction for ease of application development through a well-understood concurrency control and failure model
- The key SLEE benefits derived from transactions are
 - isolation for concurrency control
 - consistency when the JVM crashes in the middle of SBB code
- Transactions are a well defined and proven model
 - The infrastructure for implementing transactions (either for AC or SBB), will be reused to support transactions for each entity
 - The complexity of supporting transactions beyond a single entity is the transaction co-ordination
- Another use of transactions is demarcation
 - to replicate state to other nodes (for memory based redundancy)
 - to stable store (for disk based redundancy)

What happens if there are failures?

- State replication
 - Can access persistent state on another node
- Transactional semantics
 - Atomicity
 - All updates complete successfully or none completes
 - Isolation
 - Similar to some serial execution order
 - Automatic
 - No transaction API - (JTA not required)
 - Many implementations of transactional semantics possible
- Exception handling callbacks
 - Unchecked exceptions
 - Transaction rollbacks



Profile Concepts

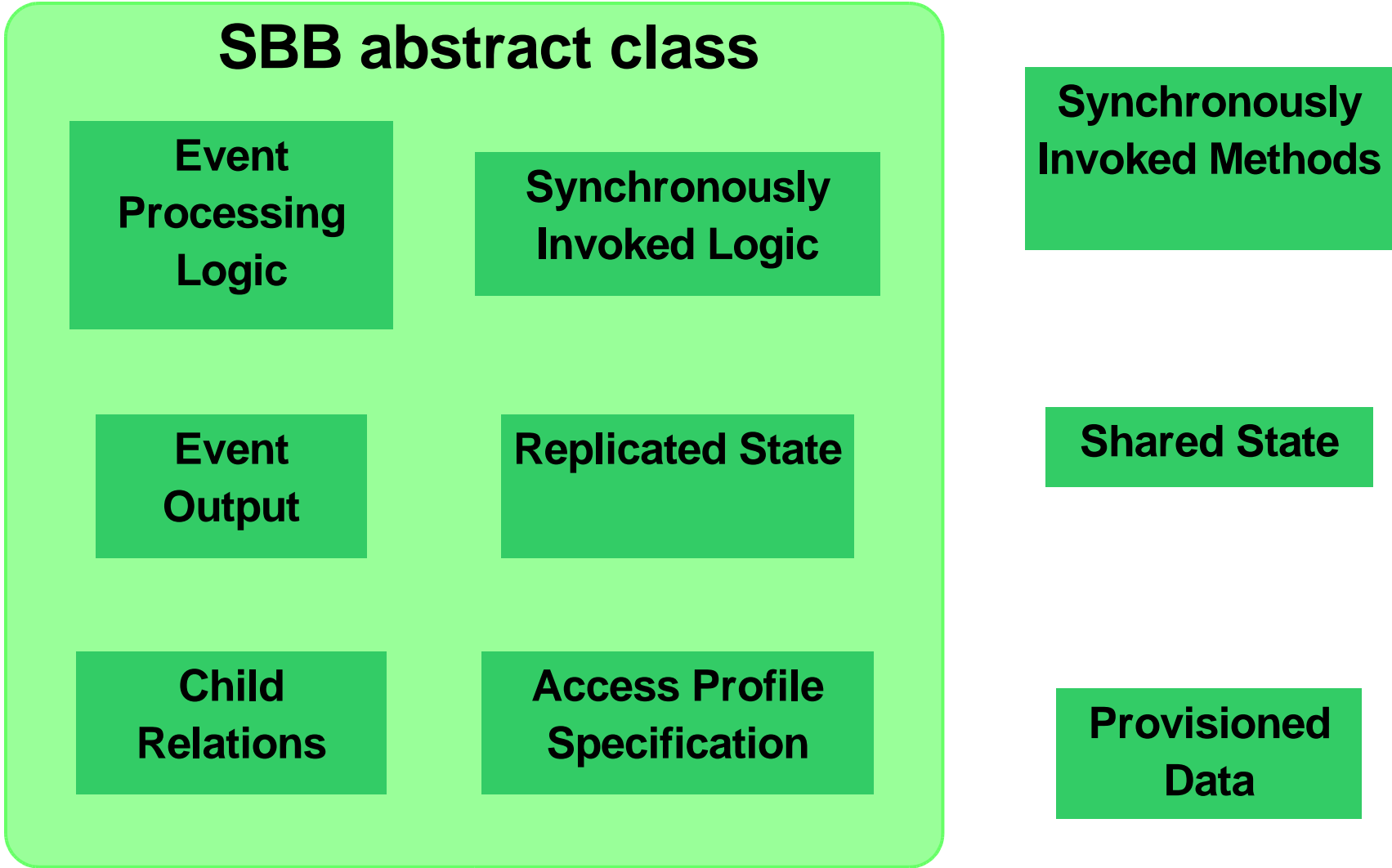


- Profile Specification defines
 - Schema (i.e. attributes) of Profile Tables (and Profiles in Profile Tables)
 - Interfaces and classes of the Profile Specification
- Profile Tables are created from Profile Specification
- Profile Tables contain Profiles

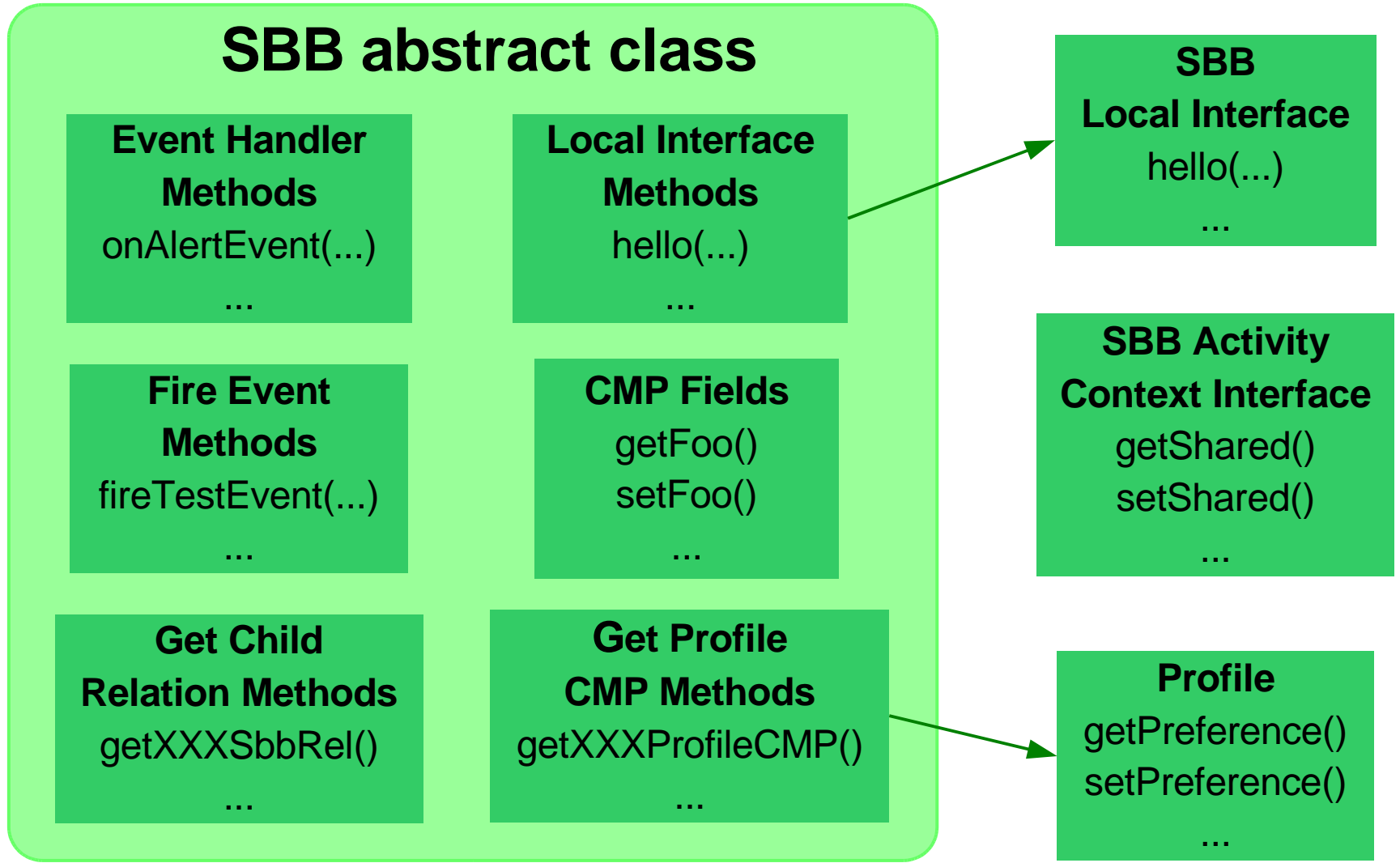
Developing an SBB

- The JAIN SLEE specification defines the Sbb Interface
 - Base SBB interface defined by the SLEE specification
- SBB developer implements SBB abstract class by implementing SBB interface
 - Each SBB created provides an abstract class that implements the SBB interface
 - SBB abstract class contains the developer provided code for the life cycle callbacks, event handler methods, and abstract fire event methods
- The SLEE deployment tool generates the SBB concrete class by implementing the SBB abstract class
 - The SBB concrete class is generated by the SLEE when the SBB is deployed into the container
 - The SBB developer doesn't implement the SBB concrete class
 - The deployment tool implements the SBB concrete class by extending the SBB developer provided SBB abstract class

Service Building Block (SBB)



Service Building Block (SBB)



Outline

- JAIN SLEE in Context
- Why create JAIN SLEE?
- JAIN SLEE Concepts
- **Implementation Considerations**

SLEE Performance Requirements

- Low latency
 - The delay for a packet of data to get from one designated point to another
 - Call setup time < 500 ms, average latency between 50...100 ms per transaction
 - Should be achievable in a cluster of 2 to 4 nodes each with 2 to 4 CPUs with availability and redundancy enabled
- High throughput
 - The ability to handle millions of Busy Hour Call Attempts (Operator specific)
 - Translates to approx 2500 transactions per second WITH state updates that are transacted and replicated to survive single node failures
- High availability, 99.999% expected
 - Less than 6 minutes downtime per year (planned and unplanned)
 - Different notion of availability
 - Small partial failures (< 5%) count against availability
- Graceful handling of load spikes above maximum system capacity

Low Latency Requirements

- What is required to set up a call in 500 ms?
- Traverse 2 ... 5 network nodes or application servers
- Minimum 2 protocol messages events per node (application dependant)
- 1 transaction per event
- Approx 4 ... 10 events to process within 500 ms (excluding propagation delay, some processing overlapped with propagation)
- 50 ... 125 ms per transaction (for simple transactions with redundancy for single failure tolerance)
- 95th percentile round trip time is 200 ms

High Throughput Requirements

- What does 1,000,000 BHCA (Busy Hour Call Attempts) require?
- 2 ... 10 protocol messages or events per call (application dependant)
- 1 transaction per event
- Approx 280 call attempts per second
- 560 ... 2800 transactions per second with transactional state updates and replication

Design Implications

- Multi-site distributed system
- Main memory storage of application and infrastructure state
- Memory replication
- Overload management

Conclusion

JAIN SLEE is a generic application environment designed specifically for high performing event-driven applications

<http://www.jainslee.org>

JAIN SLEE: JSR 22

<http://jcp.org/en/jsr/detail?id=22>