Data Management in Application Servers

Dean Jacobs

BEA Systems

Outline

- Clustered Application Servers
- Adding Web Services

Java 2 Enterprise Edition (J2EE)

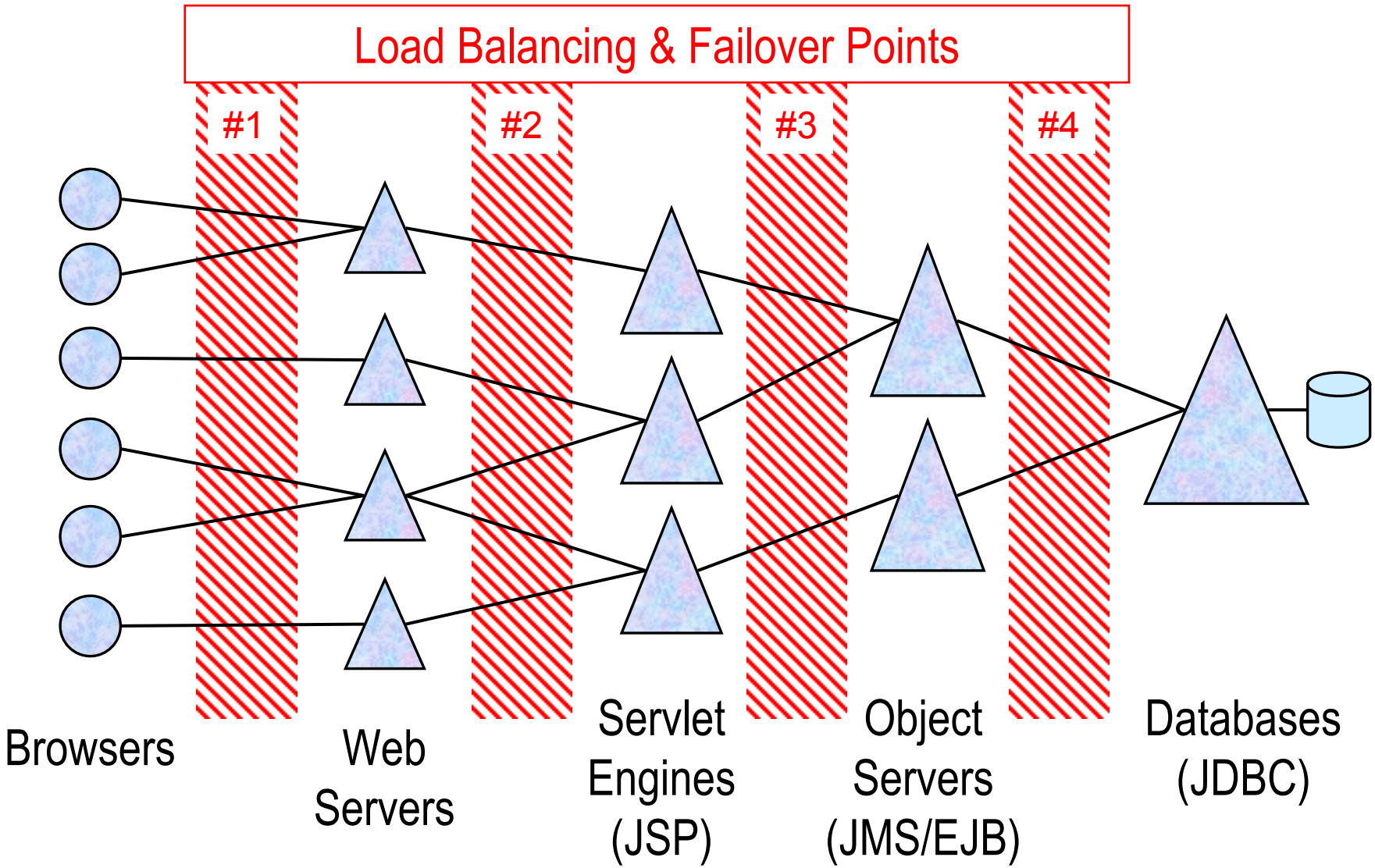
The Application Server platform for Java

- Java Servlets / Java Server Pages (JSP)
- Enterprise Java Beans (EJB)
  - Stateless Session, Stateful Session, Entity
- Java Messaging Service (JMS)
- Java Database Connection (JDBC)
- Java Connector Architecture (JCA)
- Java Naming & Directory Interface (JNDI)
- Java Transaction API (JTA)

The BEA WebLogic Server

- All Java, clean-room implementation of the J2EE
- Shipping basic APIs since 1997
- One of the most widely-used Application Servers on the market
  - Over 12,000 customers
- Associated BEA product: TUXEDO
  - Distributed TP Monitor
  - Originally developed at Bell Labs in 1984
  - Influenced the design of WebLogic

WebLogic Cluster Architecture



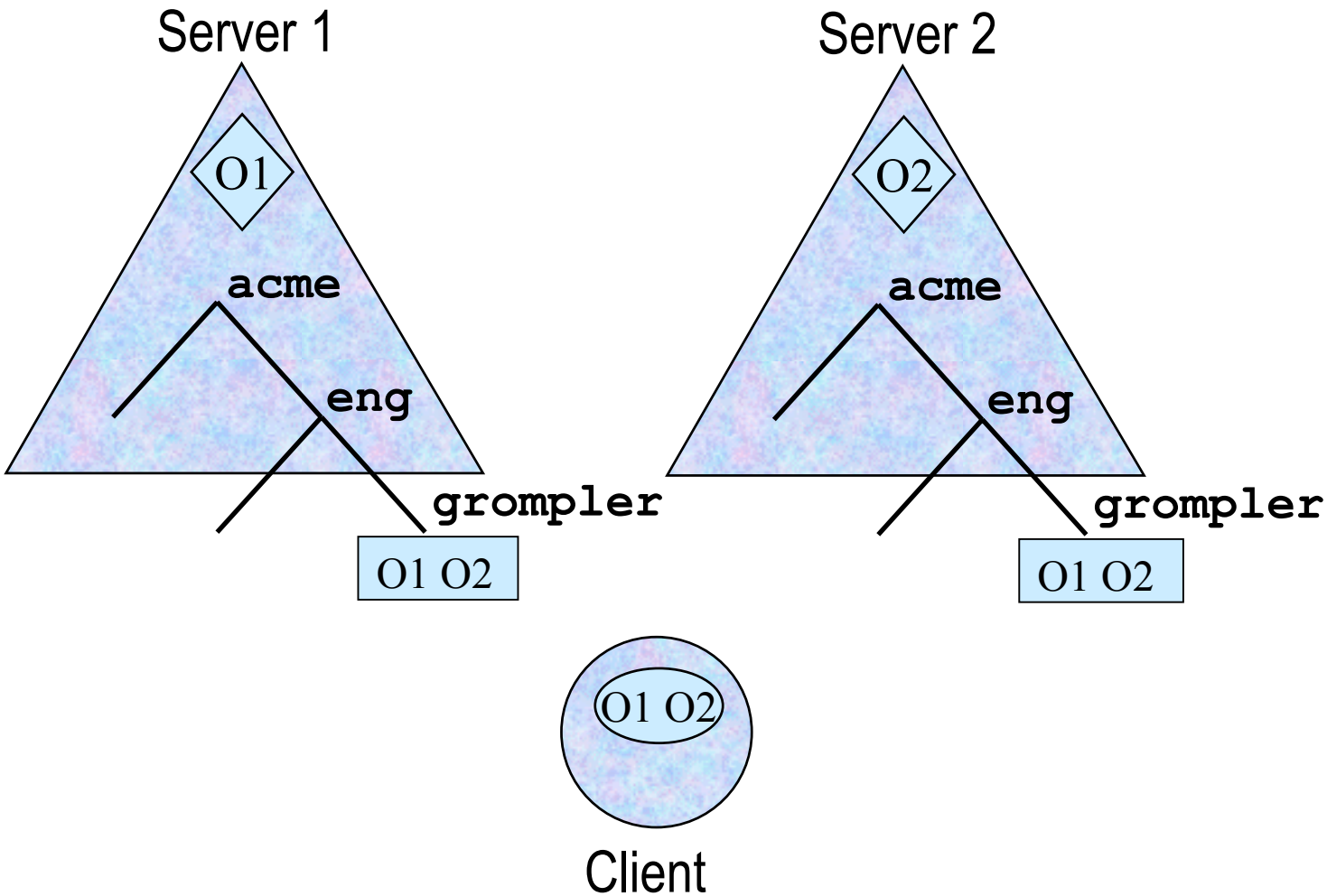
Data Management

- Essential to distribute and maintain data outside of the monolithic backend to improve scalability and performance
- Facilitated by relaxing traditional ACID properties of the data
  - Especially appropriate for reference and transient data
- WebLogic accomplishes this using **five basic service types**, which differ in their treatment of internal state
- The J2EE APIs are implemented in terms of these service types

Service Types

	State in memory	Persistent	Consistent Reads	APIs
1 Stateless	No			EJB/JMS/JDBC/JCA factories, JSP SSS, EJB Stateless/Entity
2 Conversational	Yes	No		JSP SSS, EJB Stateful
3 Cached	Yes	Some	No	JSP fragments, EJB Entity
4 Optimistic	Yes	Yes	Some	EJB Entity
5 Singleton	Yes	Yes	Yes	JMS destinations, JTA Tx Managers, Admin Server

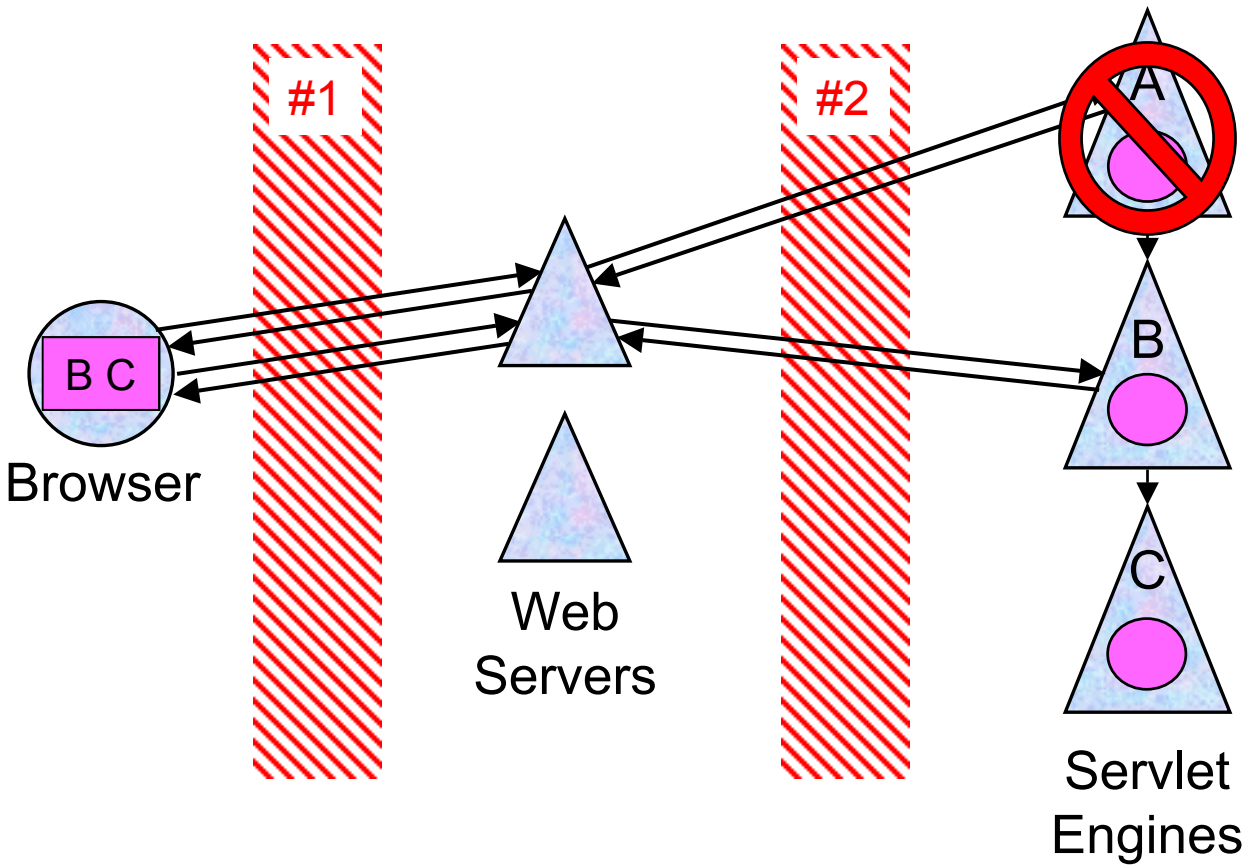
Stateless Services



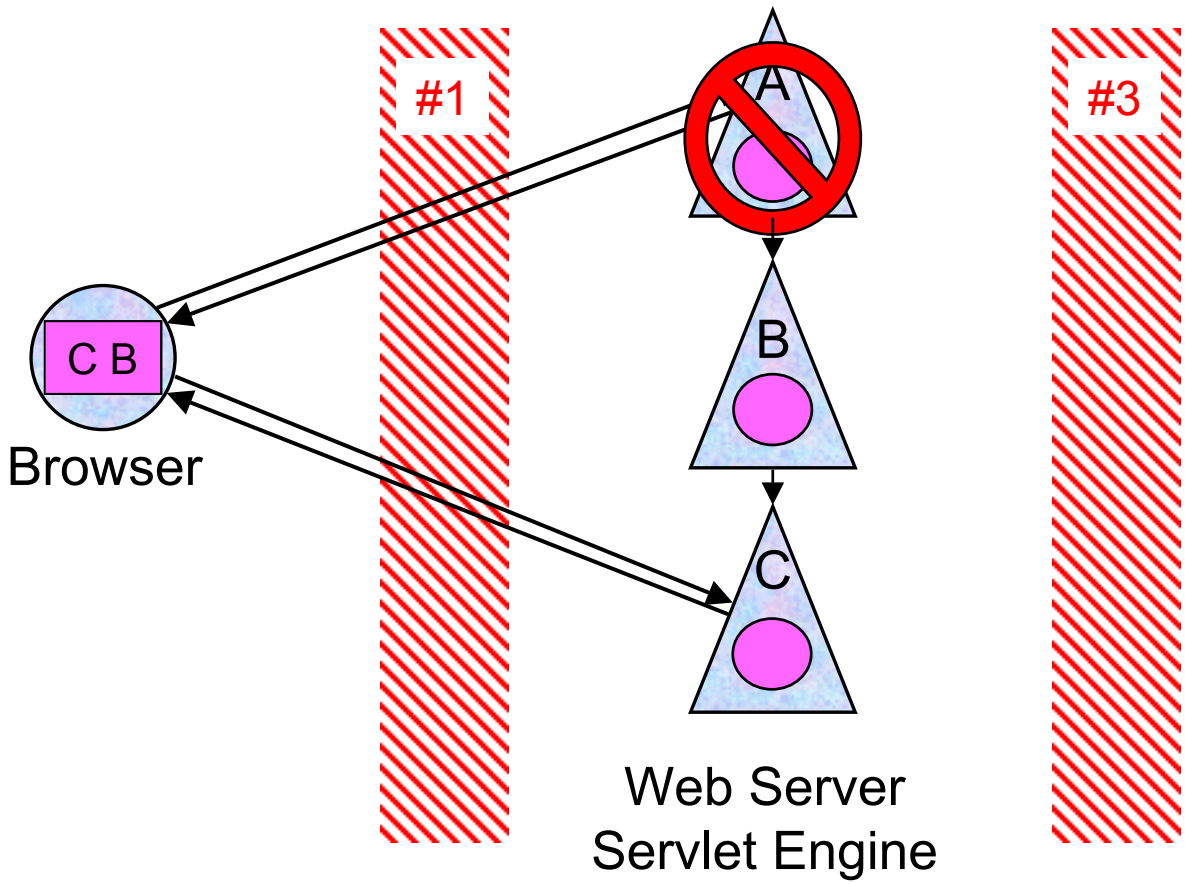
Service Types

	State in memory	Persistent	Consistent Reads	APIs
1 Stateless	No			EJB/JMS/JDBC/JCA factories, JSP SSS, EJB Stateless/Entity
2 Conversational	Yes	No		JSP SSS, EJB Stateful
3 Cached	Yes	Some	No	JSP fragments, EJB Entity
4 Optimistic	Yes	Yes	Some	EJB Entity
5 Singleton	Yes	Yes	Yes	JMS destinations, JTA Tx Managers, Admin Server

Session State Replication



Session State Replication



Service Types

	State in memory	Persistent	Consistent Reads	APIs
1 Stateless	No			EJB/JMS/JDBC/JCA factories, JSP SSS, EJB Stateless/Entity
2 Conversational	Yes	No		JSP SSS, EJB Stateful
3 Cached	Yes	Some	No	JSP fragments, EJB Entity
4 Optimistic	Yes	Yes	Some	EJB Entity
5 Singleton	Yes	Yes	Yes	JMS destinations, JTA Tx Managers, Admin Server

Caching Strategies

- Flush at regular intervals (TTL)
  - Best when data is **frequently** updated
- Flush after an update completes
  - Best when data is **infrequently** updated
  - Implemented using multicast
  - Manual flush API to allow notification of “backdoor” updates
- Pre-load and refresh (Not currently supported)
  - Interesting to persist the data remotely
  - Refresh at regular intervals (data warehouse) or after update (data replication)
  - Facilitates querying through the cache
  - Worthwhile primarily for data that will be hit many times per refresh

Transactional Reads

- Authoritative copy of the data in a database
- The service maintains a copy in memory between invocations
- Reads by any client use the in-memory copy
- The challenge: Maintain consistency with the database given updates that go through other instances or the “backdoor”
- Possible solutions
  - ✗ Distributed concurrency control
  - ✗ Centralized lock manager
  - ✓ Use the database
  - ✓ Partition so exactly one copy of each data item

Service Types

	State in memory	Persistent	Consistent Reads	APIs
1 Stateless	No			EJB/JMS/JDBC/JCA factories, JSP SSS, EJB Stateless/Entity
2 Conversational	Yes	No		JSP SSS, EJB Stateful
3 Cached	Yes	Some	No	JSP fragments, EJB Entity
4 Optimistic	Yes	Yes	Some	EJB Entity
5 Singleton	Yes	Yes	Yes	JMS destinations, JTA Tx Managers, Admin Server

Optimistic Services

- Use the database to implement **optimistic concurrency control** for transactions with writes
  - No protection for read-only transactions
- Upon commit, compare the before-and-after values of fields that were read and throw a concurrency exception if they don't match
  - Can be done fairly efficiently using UPDATE-WHERE
  - Does not require modification of the database schema
- To minimize the likelihood of such exceptions, flush after updates occur (but not within the transaction)
- Does not ensure serializability in that, for example, an increment and decrement of the same field will look like it was not modified
  - This is a feature in that it allows safe but non-serializable transactions

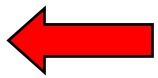
Service Types

	State in memory	Persistent	Consistent Reads	APIs
1 Stateless	No			EJB/JMS/JDBC/JCA factories, JSP SSS, EJB Stateless/Entity
2 Conversational	Yes	No		JSP SSS, EJB Stateful
3 Cached	Yes	Some	No	JSP fragments, EJB Entity
4 Optimistic	Yes	Yes	Some	EJB Entity
5 Singleton	Yes	Yes	Yes	JMS destinations, JTA Tx Managers, Admin Server

Availability of Singleton Services

- Harden individual servers
  - Multiple execute queues / thread pools
  - Deny rather than degrade service
  - Redundant networks
- Restart failed and ailing servers
  - Lifecycle and health monitoring APIs
  - Under control of a nanny daemon or HA framework
- Upgrade without interruption
  - Rolling upgrade of servers
  - Hot redeploy of applications
- Migrate singleton services
  - Manually from the Administrative console
  - Automatically using distributed agreement to avoid split-brain syndrome

Outline

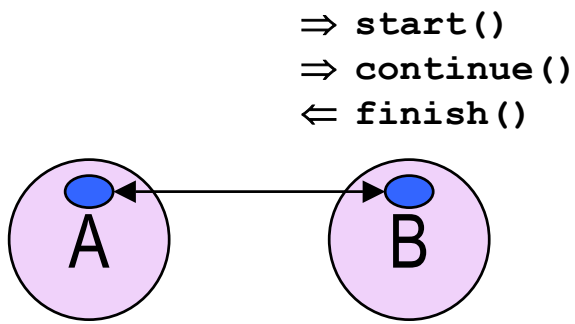
- Clustered Application Servers
- Adding Web Services 

WSDL

- Web Services Description Language
- Payload/transport-neutral but demands support for SOAP/HTTP
- Specifies a set of service signatures of various types
  - **One-way** Receive a message
  - **Request-response** Receive a message and send a correlated message
  - **Solicit-response** Send a message and receive a correlated message
  - **Notification** Send a message
- Implies an interesting programming model
  - Can be implemented in terms of the J2EE, but demands at least special packaging and optimizations

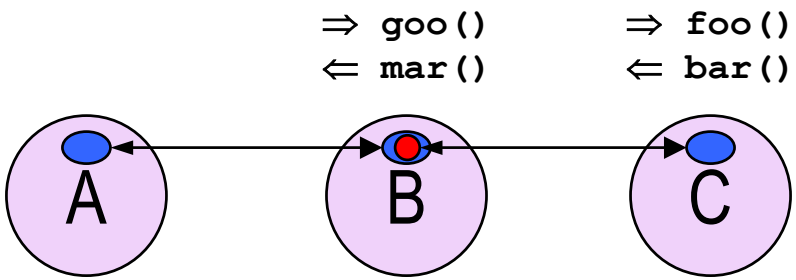
Web Services Programming Model

- Unified model for synchronous RPC and asynchronous messaging
  - Queuing should occur **under the covers** for one-way messages
  - A typed messaging model, like MS queued components and unlike JMS
- Peer-to-peer conversational services
  - An object on each side representing the state of the conversation
  - Either side can initiate communication regarding the conversation
  - Conversations have finite but potentially long lifetimes
  - One local transaction per service invocation



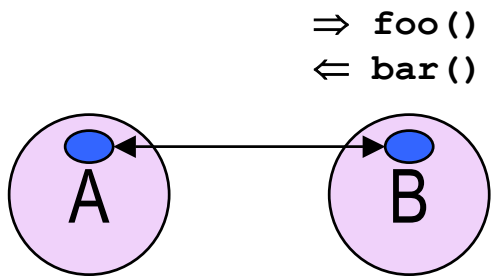
Dependent Conversations

- Callbacks realized using the listener pattern
  - B registers with C as a bar listener
- A mechanism for isolating the interfaces must be provided
  - B should not expose bar to A
  - B may want to also register with D as a bar listener
- Implement as a dependent sub-object
- A conversation may have several simultaneous users



Asymmetry Between Peers

- The **initiator** A plays a different role than the **responder** B
- All services that may be invoked within the conversation are described in the WSDL of the responder
- Incremental step beyond client/server where the client can be asynchronously contacted
  - In keeping with the realities of managing distributed computations
- If both are servers, B may invoke a service described in A's WSDL, but then it is a different conversation and the roles are reversed



Queuing

- An option for both in-bound and out-bound messages
- One input/output queue pair co-located with each deployed instance of a Web Service in a cluster
- Messages stored in serialized format (XML rather than Java) to facilitate loose coupling
- Participate in future Web Services protocols for reliable messaging
- Decision of whether or not to store messages durably should be tied to the properties of the conversation

Conversations

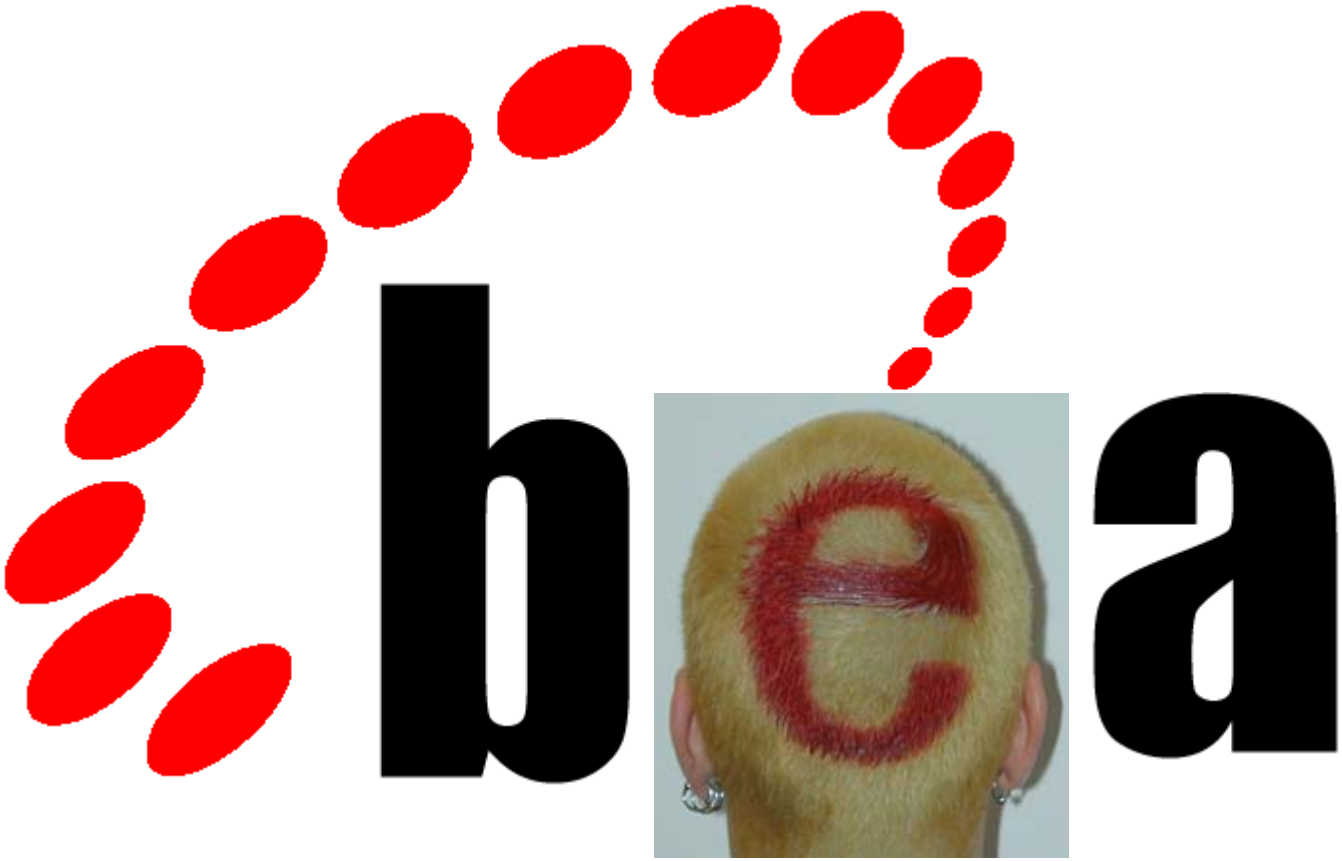
- Durable conversations
  - Expected to survive backend server failures
  - Also applies to messages that have been queued for/by them
  - Requires ACID transactions, which limits performance and scalability
- Non-durable conversations
  - Kept in memory along with any messages that have been queued for/by them
  - May be paged out to free memory, but writes are not required to hit the disk and the data is not expected to survive server failures
  - Co-locating the conversation with its messages provides a nice unit of failure, in that both are wiped out together
  - Appropriate when reliability is provided by the application and for shopping-cart and read-only applications

Implementing Non-durable Conversations

- The hard part is tracking them down within a cluster
  - A peer-to-peer version of servlet session state
- A basic design goal is to not use singleton services, since they reduce performance and scalability which partially defeats the optimization
- A better approach is to rely on **session affinity** when it is available and **routing** based on conversation IDs (CIDs) when it is not

Finding Conversations

- Session Affinity
  - Widely-applicable because conversation identity can be based on anything, even application data
  - Efficient because it eliminates the need for extra hops inside the cluster
  - For HTTP, can be achieved using DNS or a hardware load balancer
  - Regardless, there will be circumstances where session affinity fails, particularly for long-lived conversations
- CID-Based Routing
  - Accomplished by embedding server identity in the CID
  - Should occur before queuing
  - BEA is attempting to standardize protocols that support “Dual CIDs”, where each side gets to specify a CID that the other side is expected to use
  - Even if that is successful, there is a pattern – a sequence messages sent without waiting for any response – where CID-based routing is not possible



www.bea.com