Middleware

- Standard middleware components?
- Standard middleware stacks?

Transactions

- Standard Java interfaces:
  - JTA
- Open source implementations:
  - Bitronix BTM
  - Atomikos
  - ...

JTA

- Interfaces:
  - For client operations: e.g.DataSource/Connection
  - For transactions: XAResource
- Transaction manager needs to:
  - Recover connections upon restart
  - Intercept client operations for XID propagation
- Let the TM manage the resources:
  - JNDI and connection pools

JTA

- Explicit transaction demarcation
- Implicit resource registration

```
public class Client {
    public static void main(String[] args) throws Exception {
        Context ctx = new InitialContext();

        UserTransaction txn = (UserTransaction) ctx.lookup("java:comp/UserTransaction");

        txn.begin();
        DataSource ds = (DataSource) ctx.lookup("jdbc/mydb");
        Connection c = ds.getConnection();

        ...

        txn.commit();
    }
}
```

wrapped data source and connection

Messaging

- Standard Java interfaces:
  - JMS
- Implemented in
  - ActiveMQ
  - ...

Message sender

```
public class Client {
      public static void main(String[] args) throws Exception {
            Context ctx = new InitialContext();

            ConnectionFactory factory = (ConnectionFactory) ctx.lookup("jms/MyFactory");
            Queue factory = (Queue) ctx.lookup("jms/MyQueue");
            Connection conn = factory.createConnection();
            Session session = conn.createSession(true, 0);
            MessageProducer sender = session.createProducer(queue);

            TextMessage msg = session.createTextMessage("Hello World!");
            sender.send(msg);
            session.commit();
      }
}
```

Message receiver

```
public class Client {
      public static void main(String[] args) throws Exception {
            Context ctx = new InitialContext();

            ConnectionFactory factory = (ConnectionFactory) ctx.lookup("jms/MyFactory");
            Queue factory = (Queue) ctx.lookup("jms/MyQueue");
           Connection conn = factory.createConnection();
           Session session = conn.createSession(true, 0);
           MessageConsumer receiver = session.createConsumer(queue);
            conn.start();

           TextMessage msg = (TextMessage) receiver.receive();
           session.commit();
      }
}
```

Request/reply pattern

- A temporary queue exists for the duration of a session:

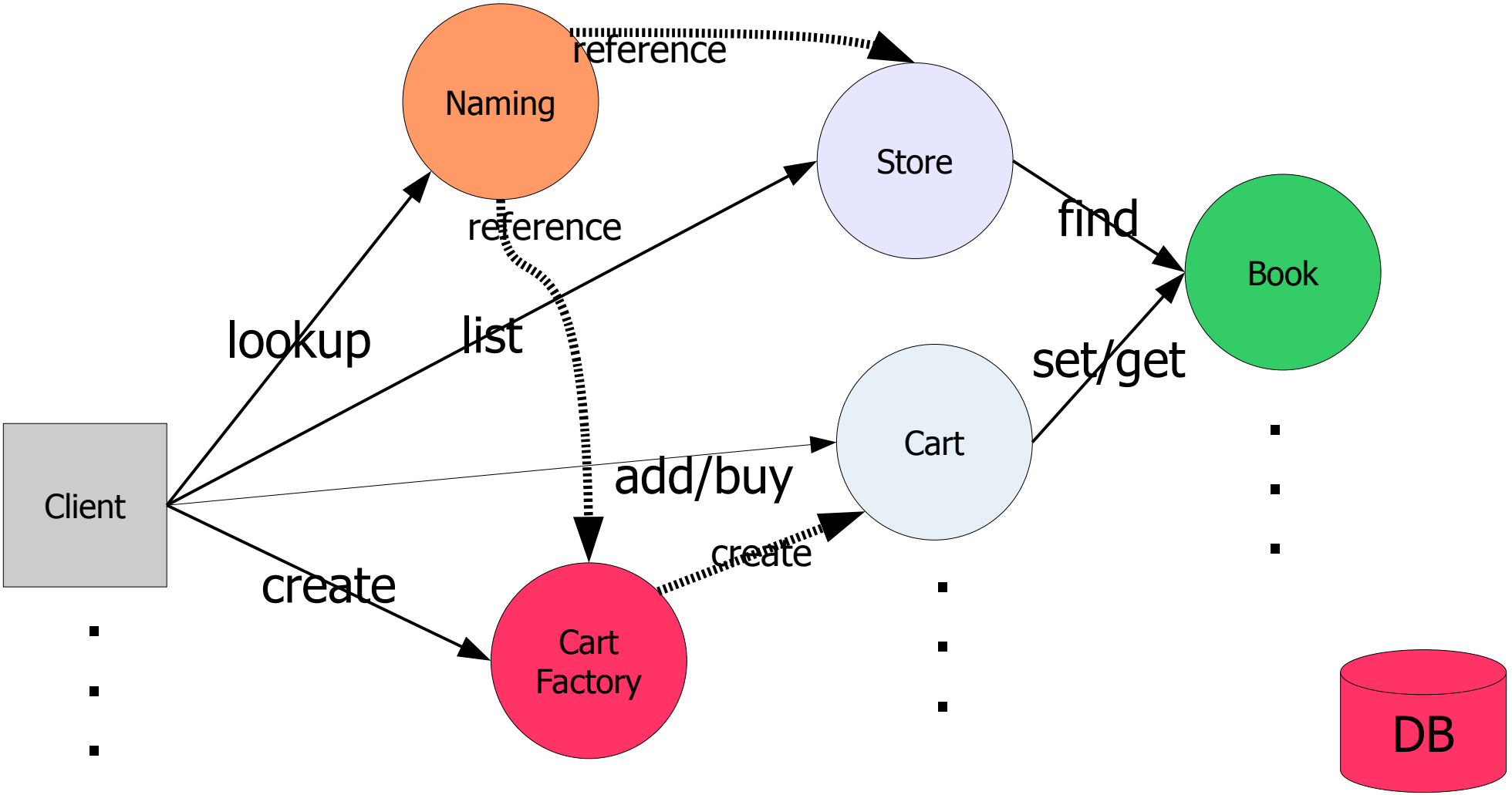
```
Destination r=s.createTemporaryQueue();

TextMessage m = s.createTextMessage();
m.setJMSReplyTo(r);
mp.send(m);
```

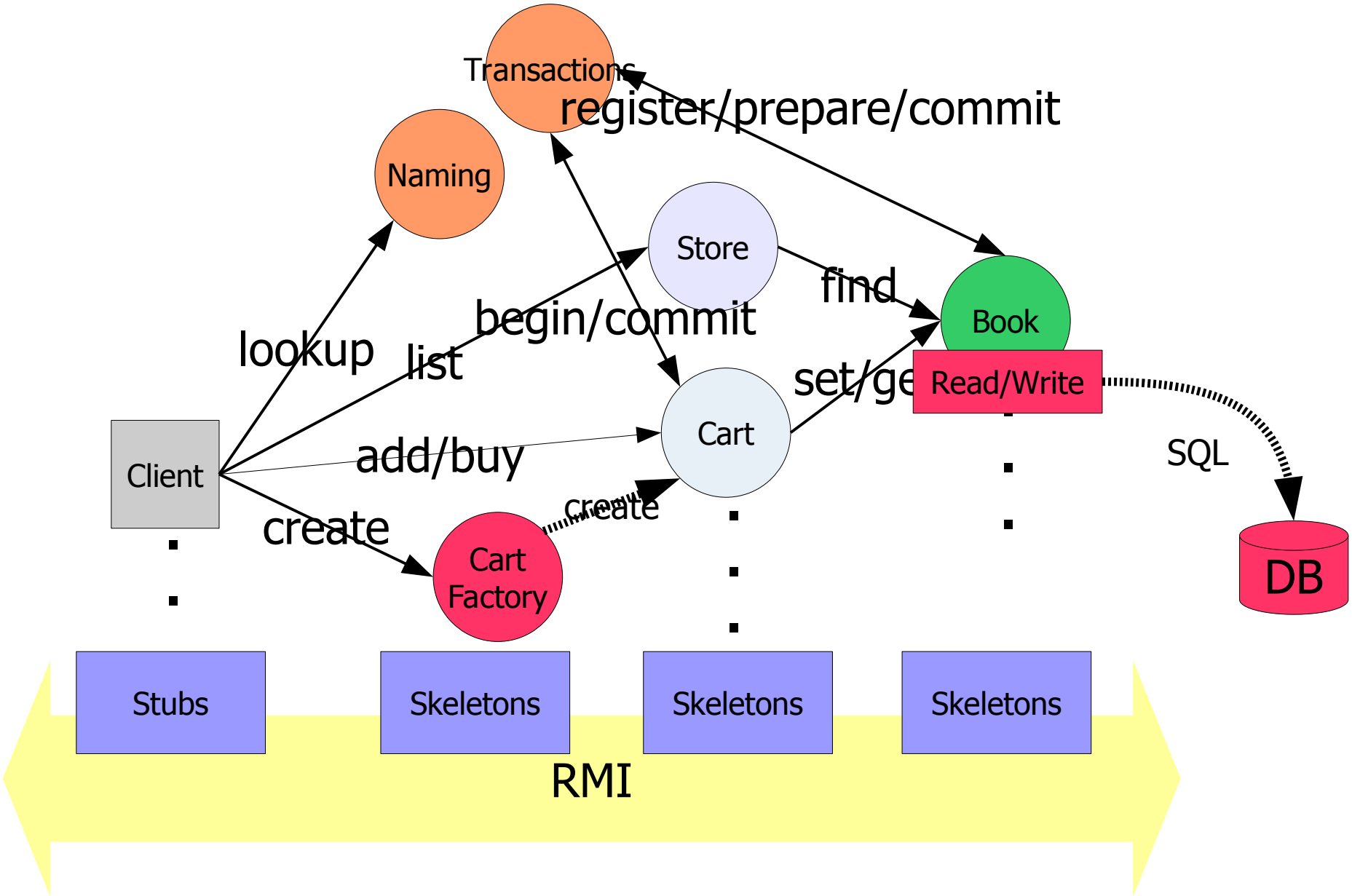
- The reply queue is used with a reference to the request message:

```
MessageProducer mp = s.createProducer(m.getJMSReplyTo());
TextMessage n = s.createTextMessage();
n.setJMSCorrelationID(m.getJMSMessageID());
mp.send(n);
```

Example: Book store



Example: Book store



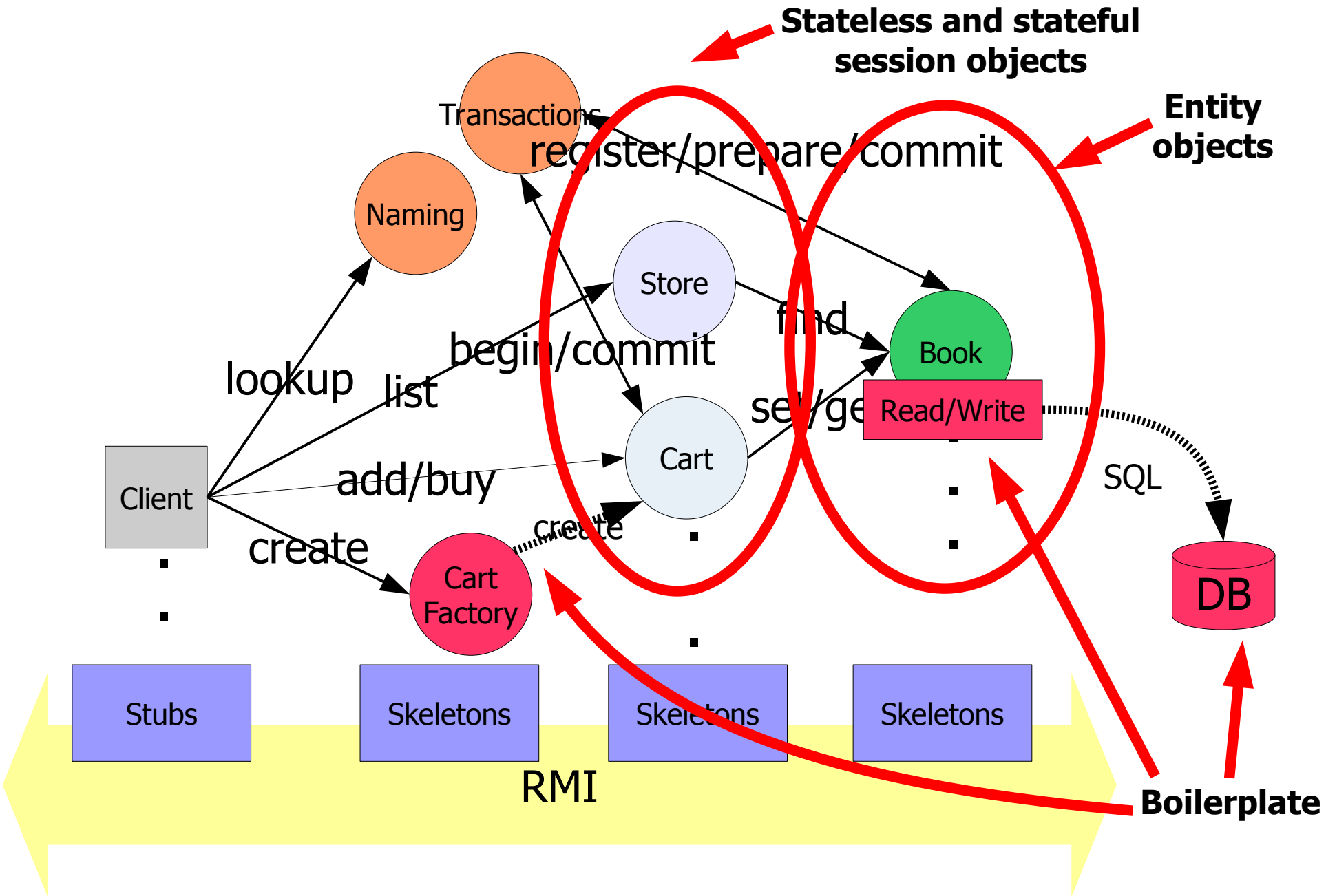
Common patterns

- Stateless objects:
  - Global utility methods
  - Entry point to persistent state
  - Contain business logic
- Session objects:
  - Hold transient state
  - Local to a client session
  - Contain business logic

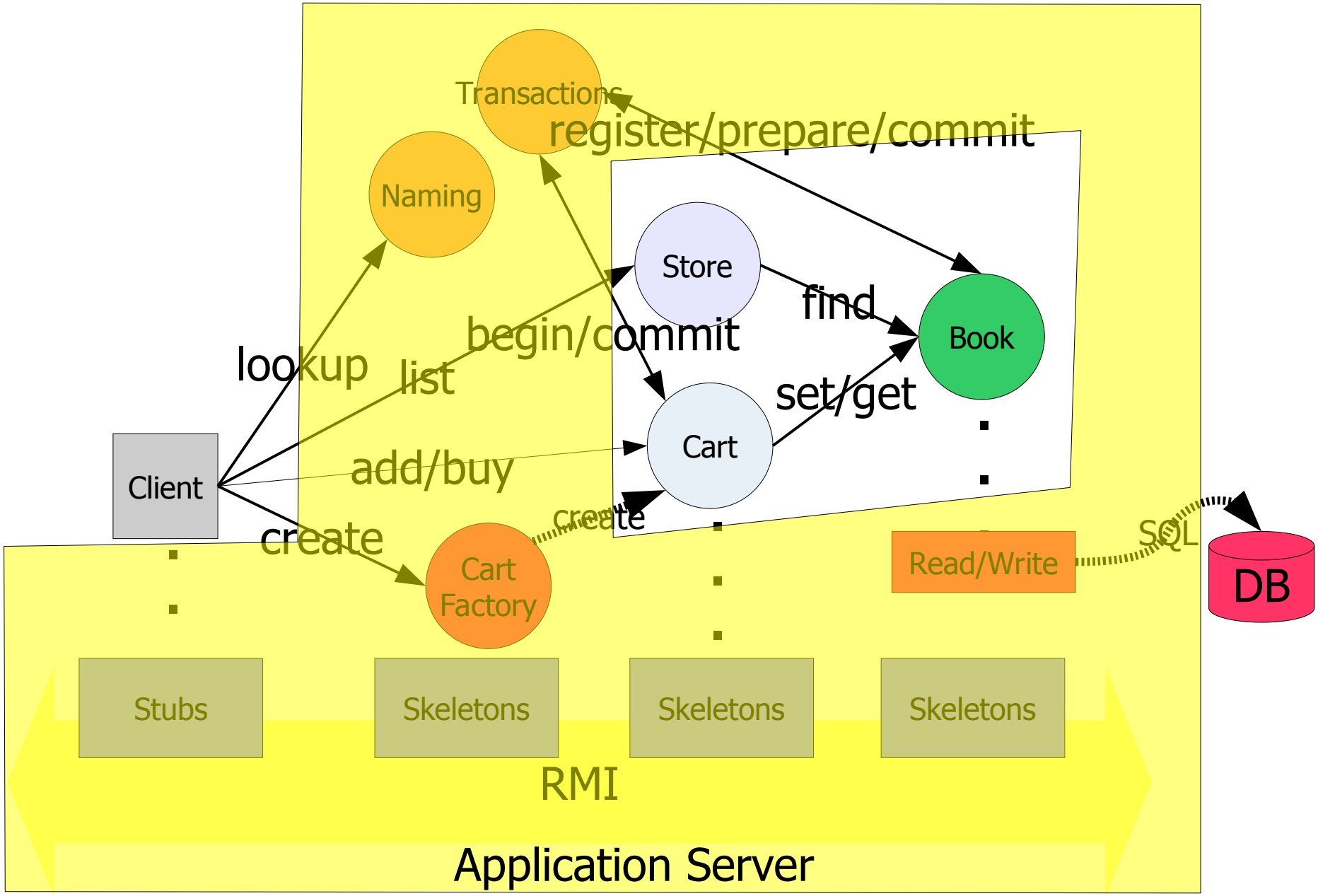
Common patterns

- Entity objects:
  - Collections of attributes, no logic
  - Persistent using a database
  - Shared between session objects
- Infrastructure:
  - DB, factory, naming, transactions, etc...
  - Initialization and configuration
  - Management

Common patterns



Application server



Application server

- The developer provides:
  - Session and entity objects
  - Configuration information
    - Annotations
    - XML
- The application server generates:
  - Configuration defaults
  - Protocol definitions
  - Stubs and skeletons
  - Boilerplate (factories, persistence, etc)

Transaction demarcation (client)

```
public class Client {
    @Resource UserTransaction tran;

    public static void main(String[] args) {
        try {
            tran.begin();

            // multiple invocations on multiple beans

            tran.commit();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Rollback

- Implicit:
  - Always on runtime exceptions
  - On application exceptions, only if:
    @ApplicationException(rollback=true)
- Explicit:

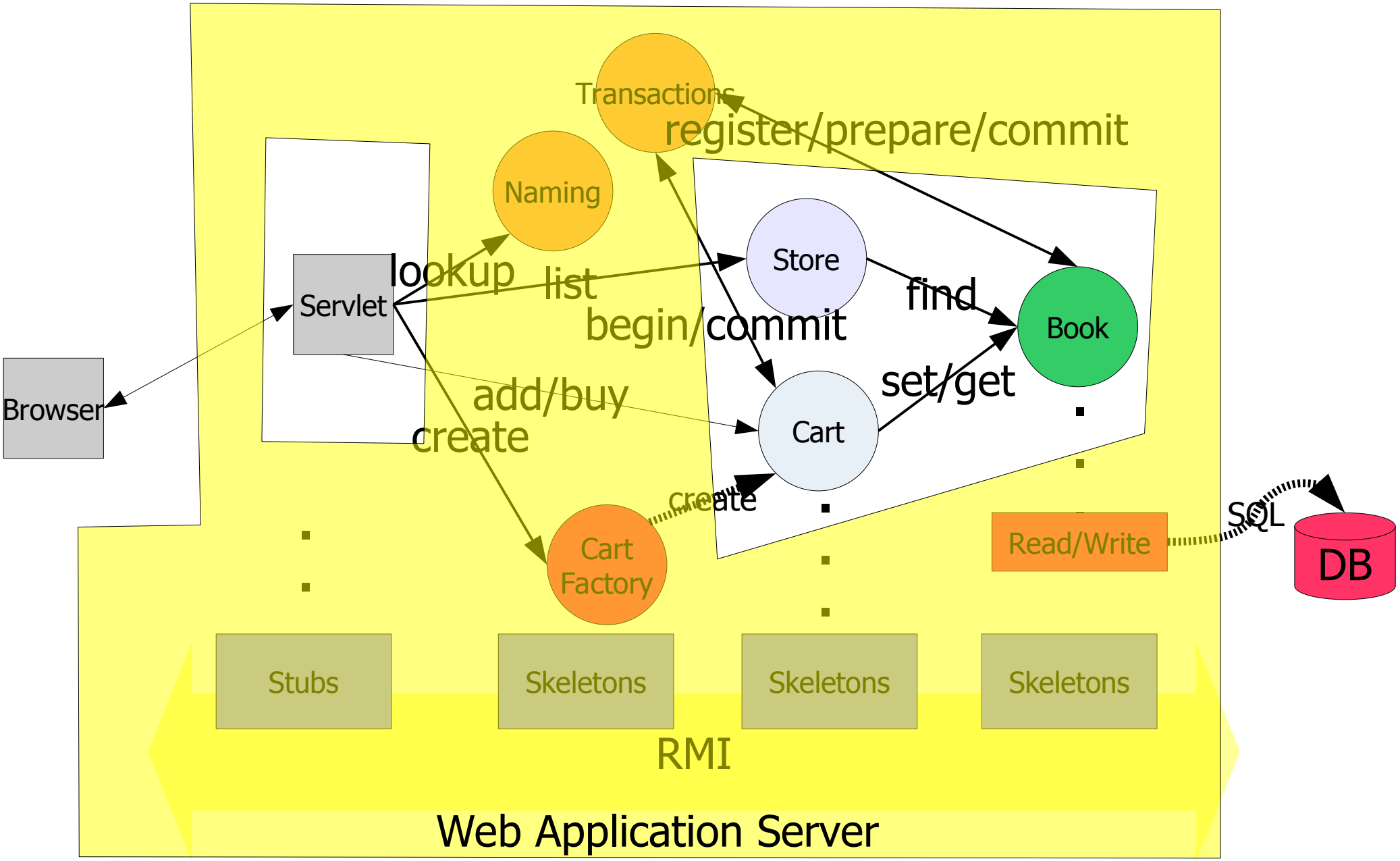
```
public @Stateless class SomeBean implements Some {
      @Resource
      private SessionContext ctx;

      public void m() {
            if (error)
                  ctx.setRollbackOnly();
      }
}
```

Distributed Objects vs Application Server

- With distributed objects:
  - All objects are equal
  - Can be used for objects that do not fit the patterns (callbacks, concurrency)
  - Developer must provide the boilerplate
- An application server:
  - Treats different objects differently
  - Can be used only when the application fits the patterns
  - Avoids boilerplate and defers configuration
  - Bundles a number of services together

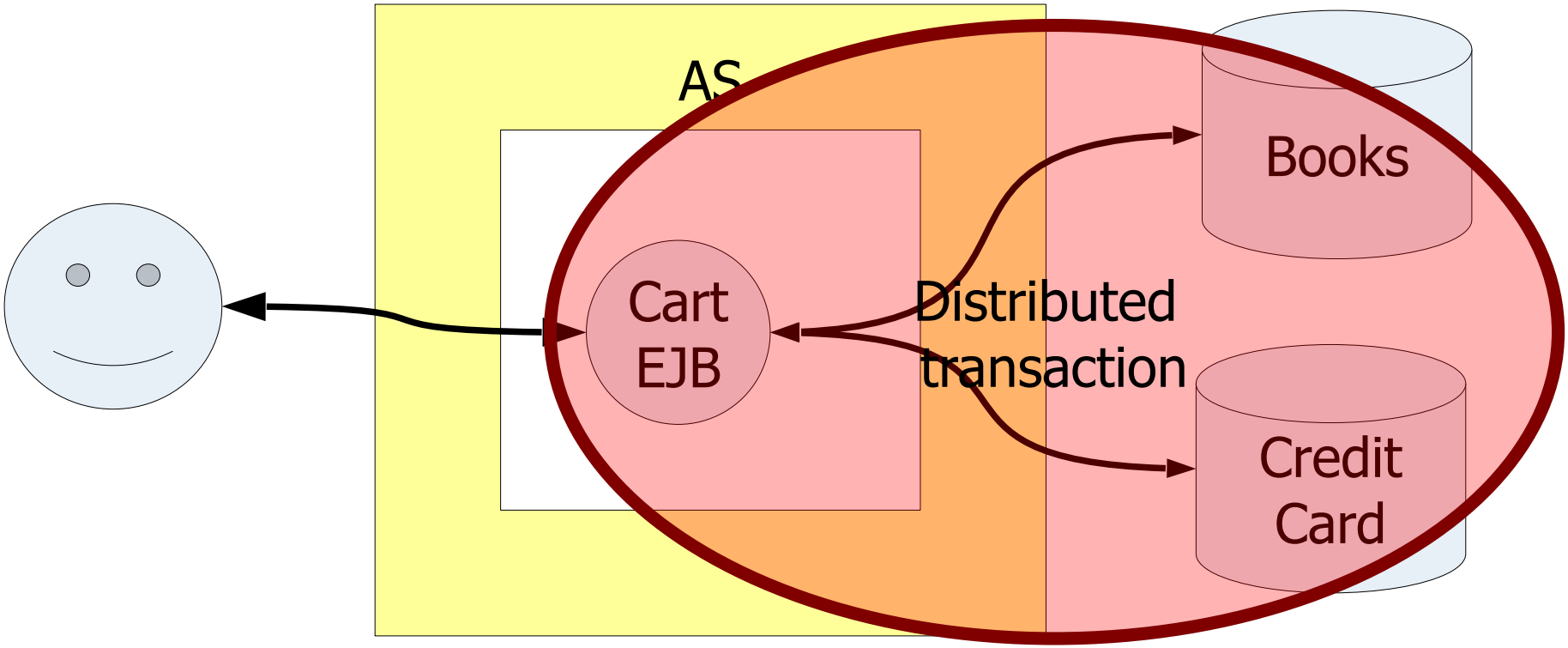
Web application server



Example: Book store revisited

- Store operations:
  - List books
- Shopping cart operations:
  - Add to cart
  - Buy
- Front-end to legacy applications:
  - Credit card processing application
  - Shipping application
  - Re-stocking application
  - ...

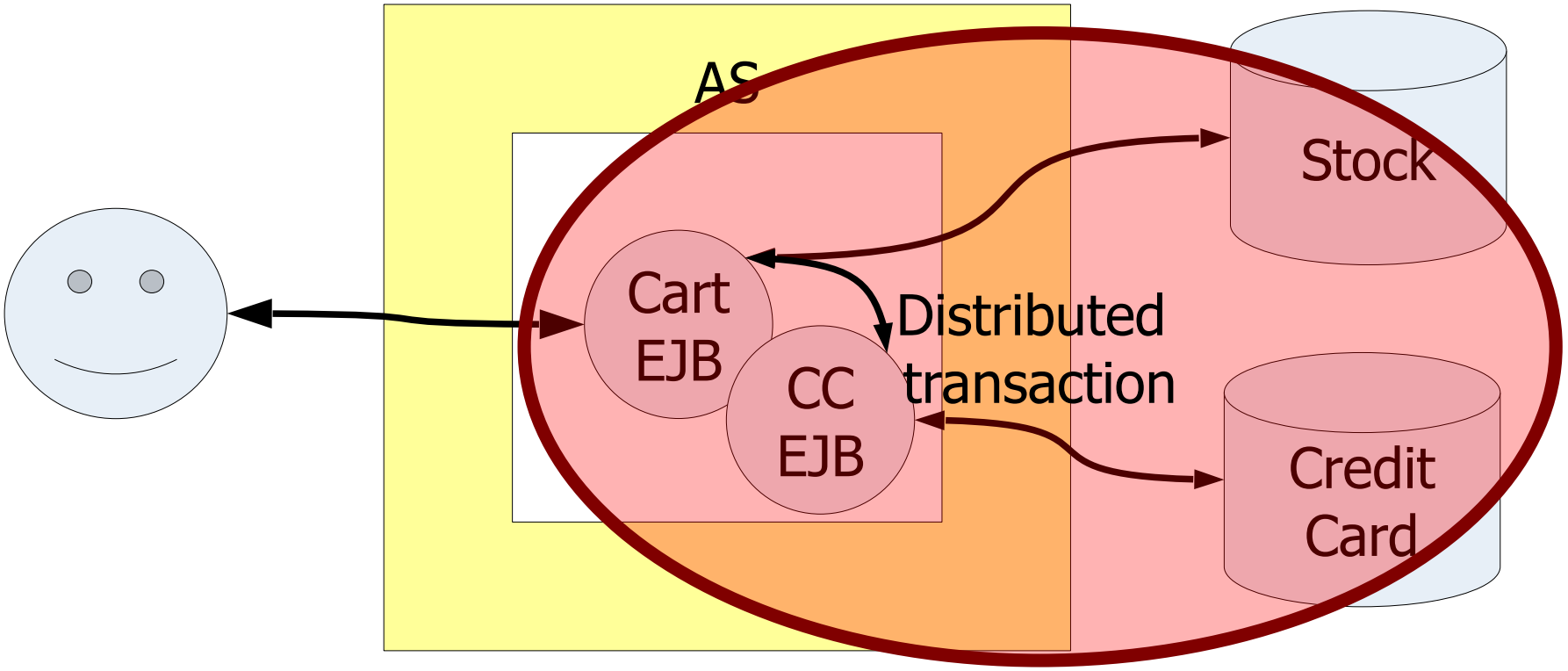
Legacy applications in a DBMS



Legacy applications in a DBMS

- Assumes:
  - Legacy business logic in the DBMS
  - Connection to the DBMS is possible
  - Applications always on-line
- No encapsulation!

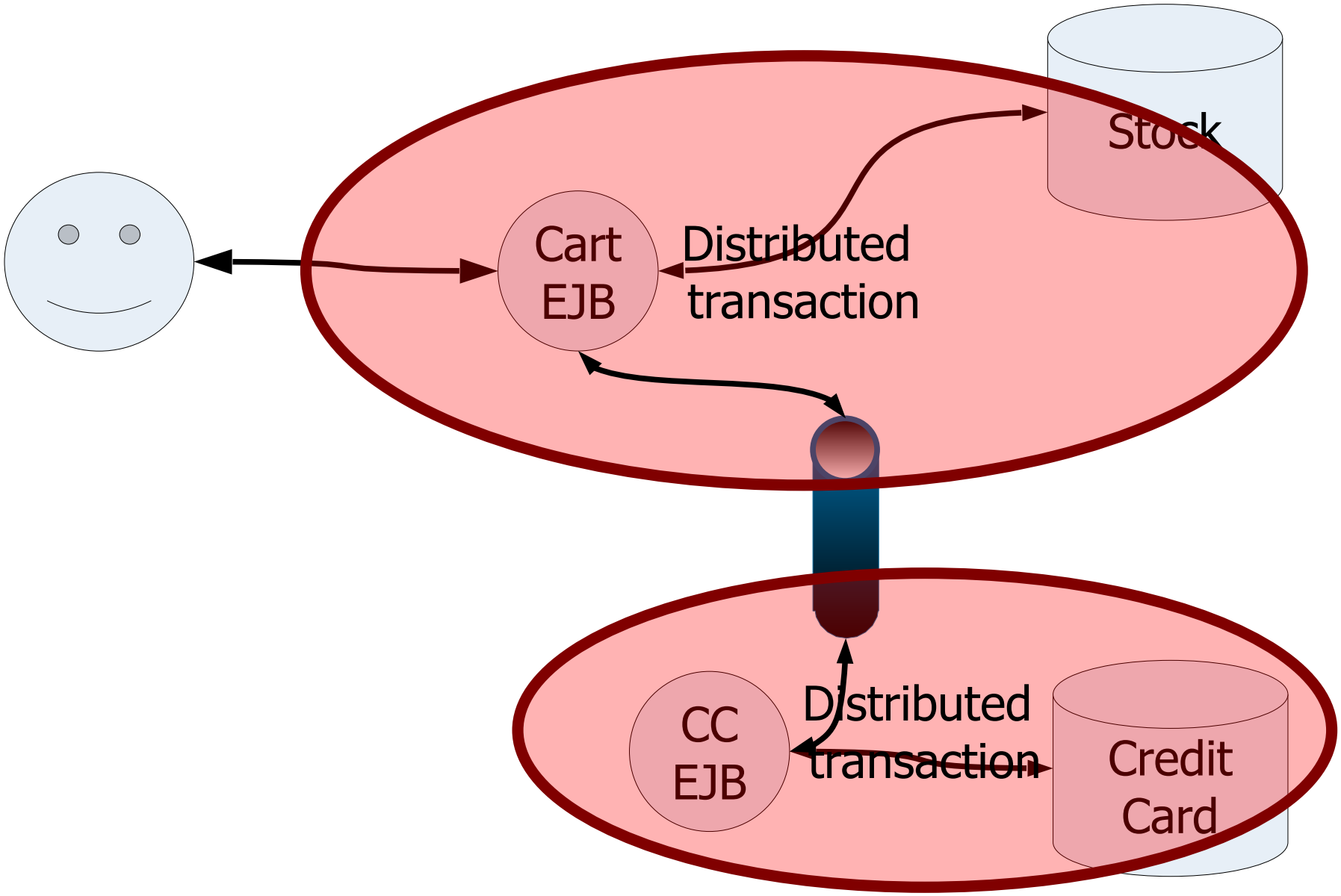
Legacy applications in the AS



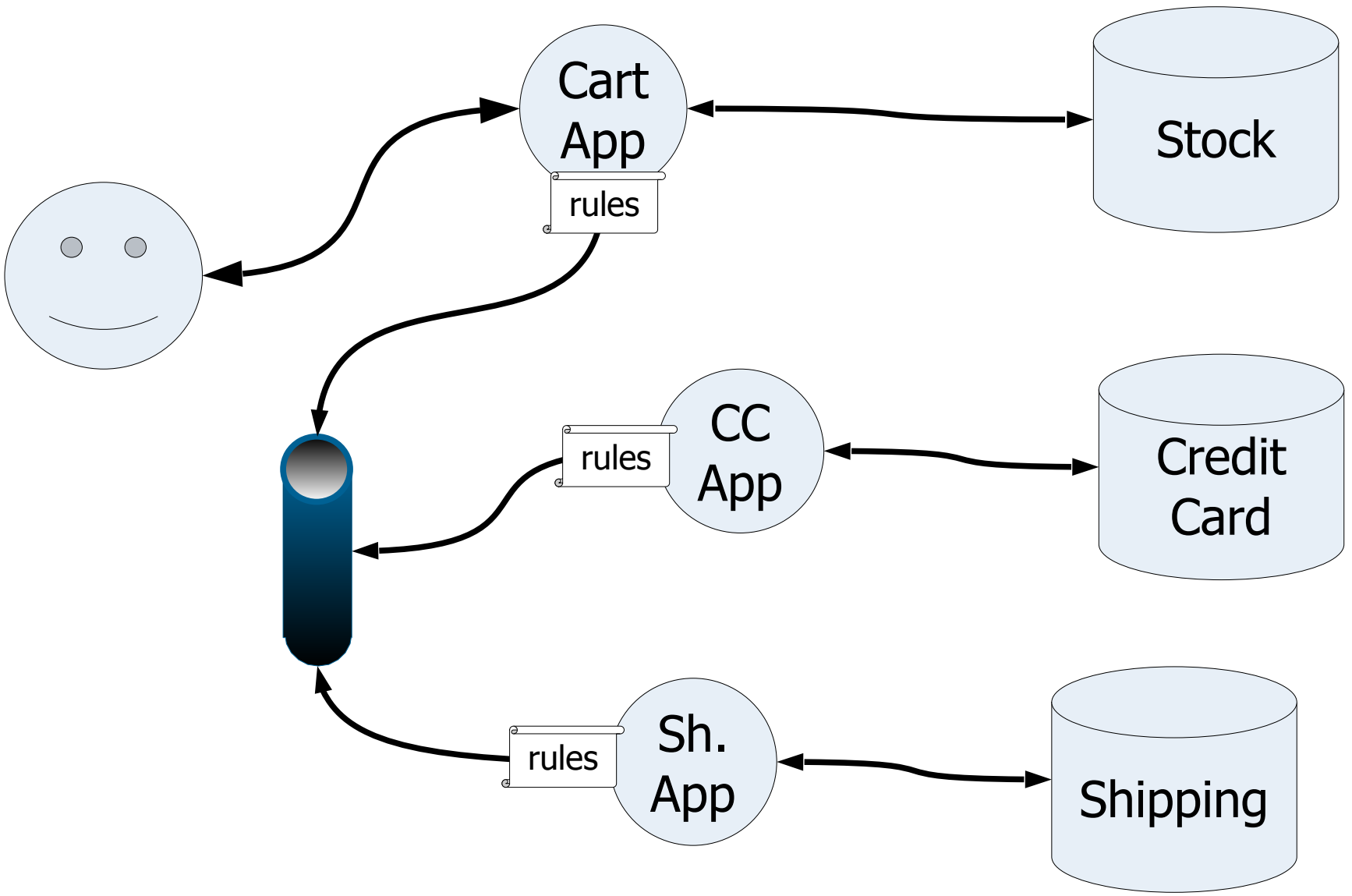
Legacy applications in the AS

- Assumes:
  - Legacy applications in the same middleware platform
  - Applications always on-line
- No heterogeneity!
- Reconsider distributed transactions:
  - Large distributed transactions are inefficient
  - Many applications don't need to / must not cause a rollback
  - Some steps will happen asynchronously
  - Some steps need human intervention

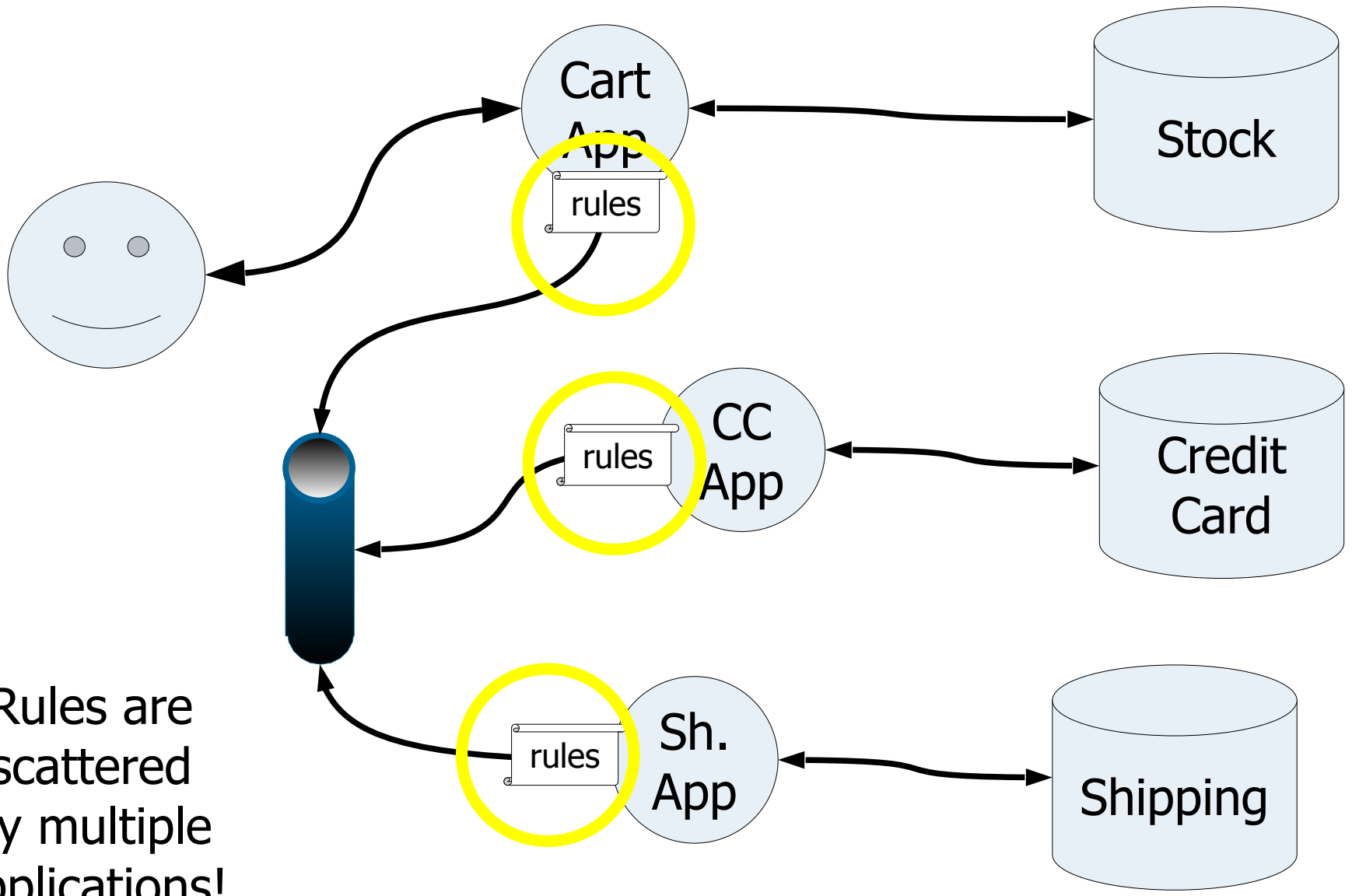
Integration by message queues



Integration by messaging

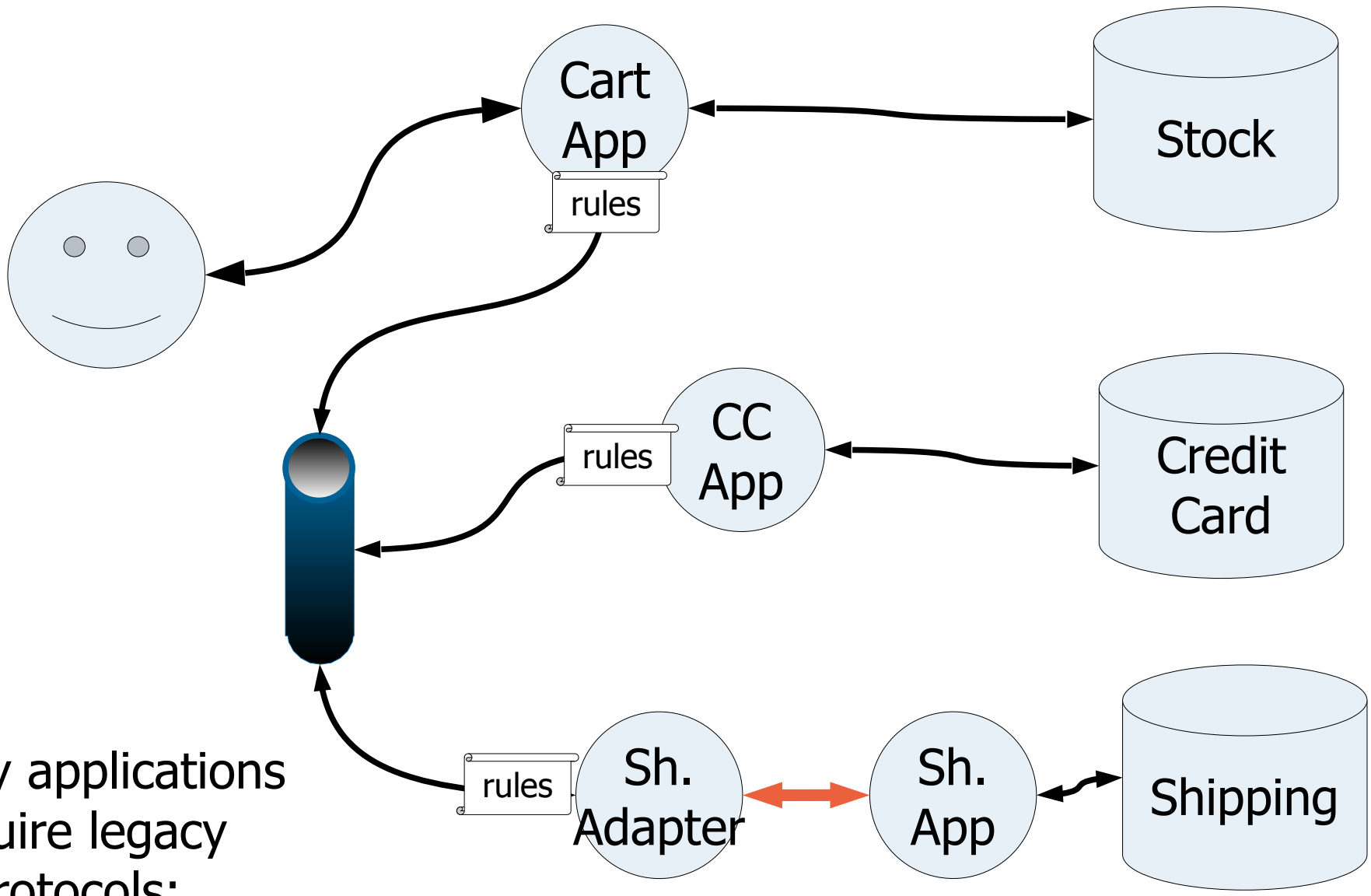


Limitations of EAI by messaging



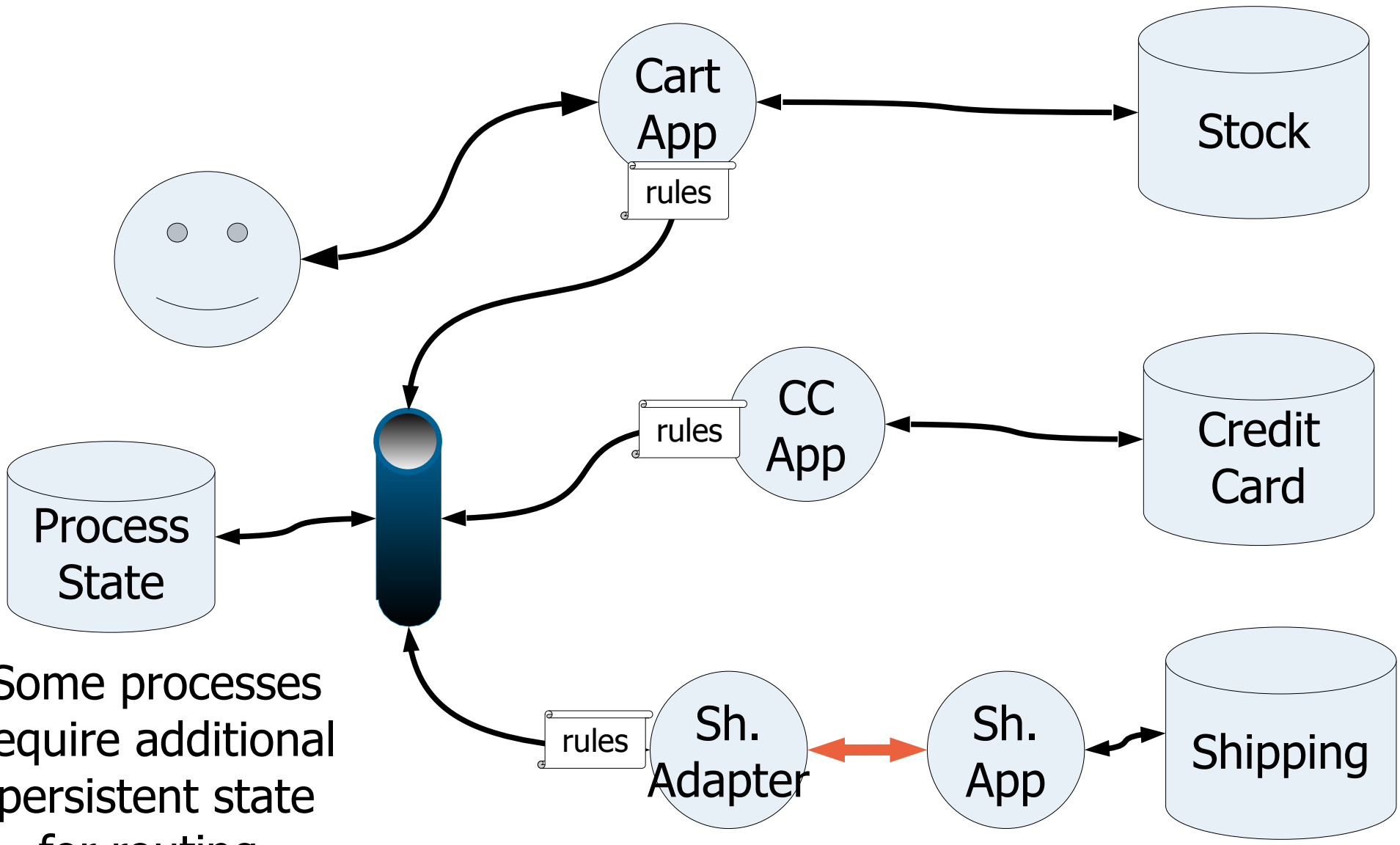
Rules are scattered by multiple applications!

Limitations of EAI by messaging



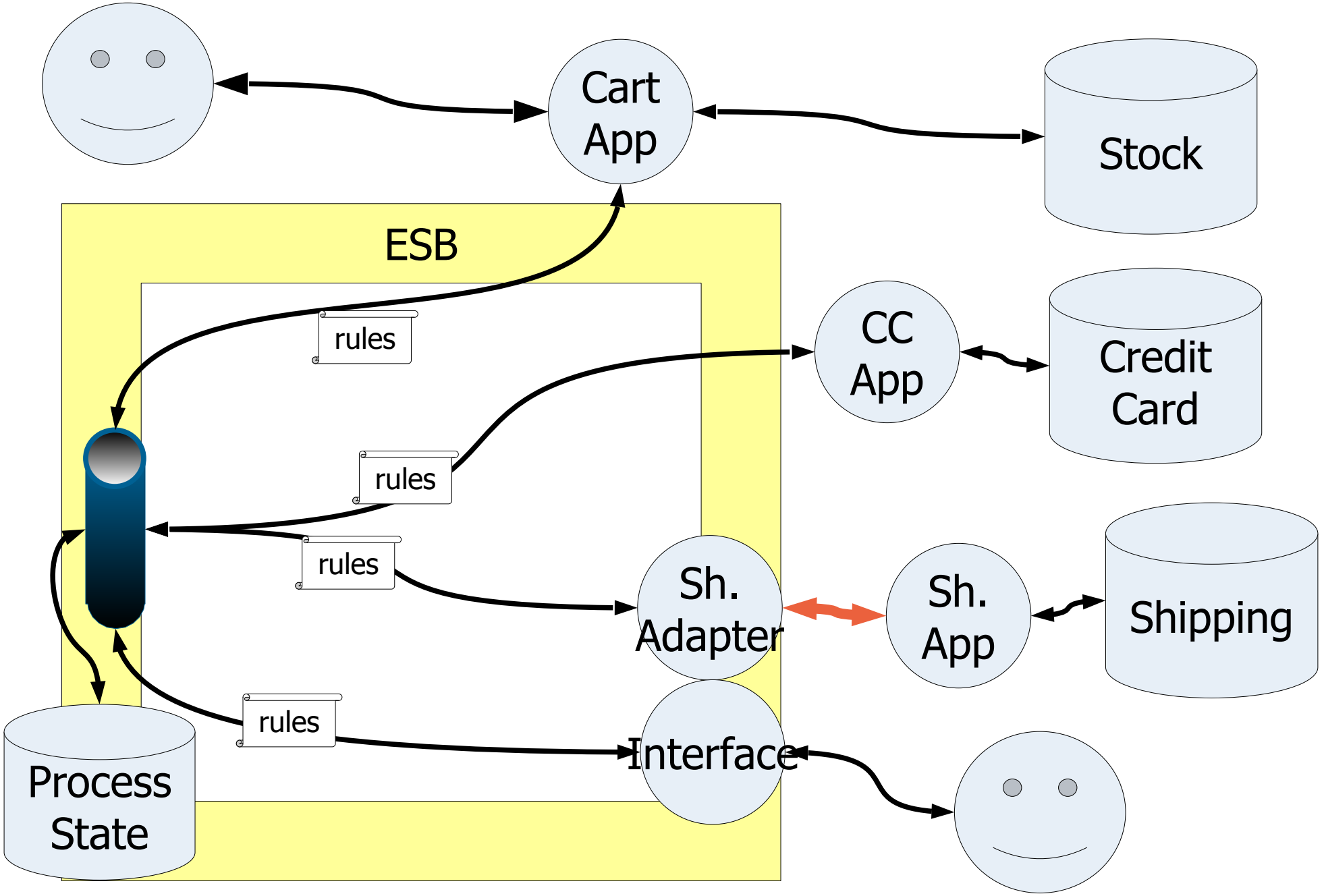
Legacy applications require legacy protocols:
FTP, files, mail, ...

Limitations of EAI by messaging



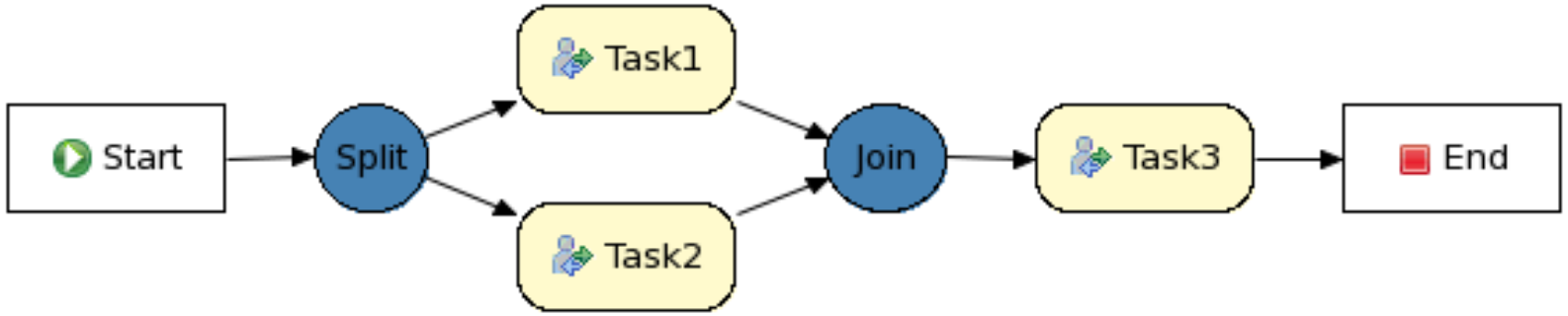
Some processes require additional persistent state for routing.

Enterprise Service Bus



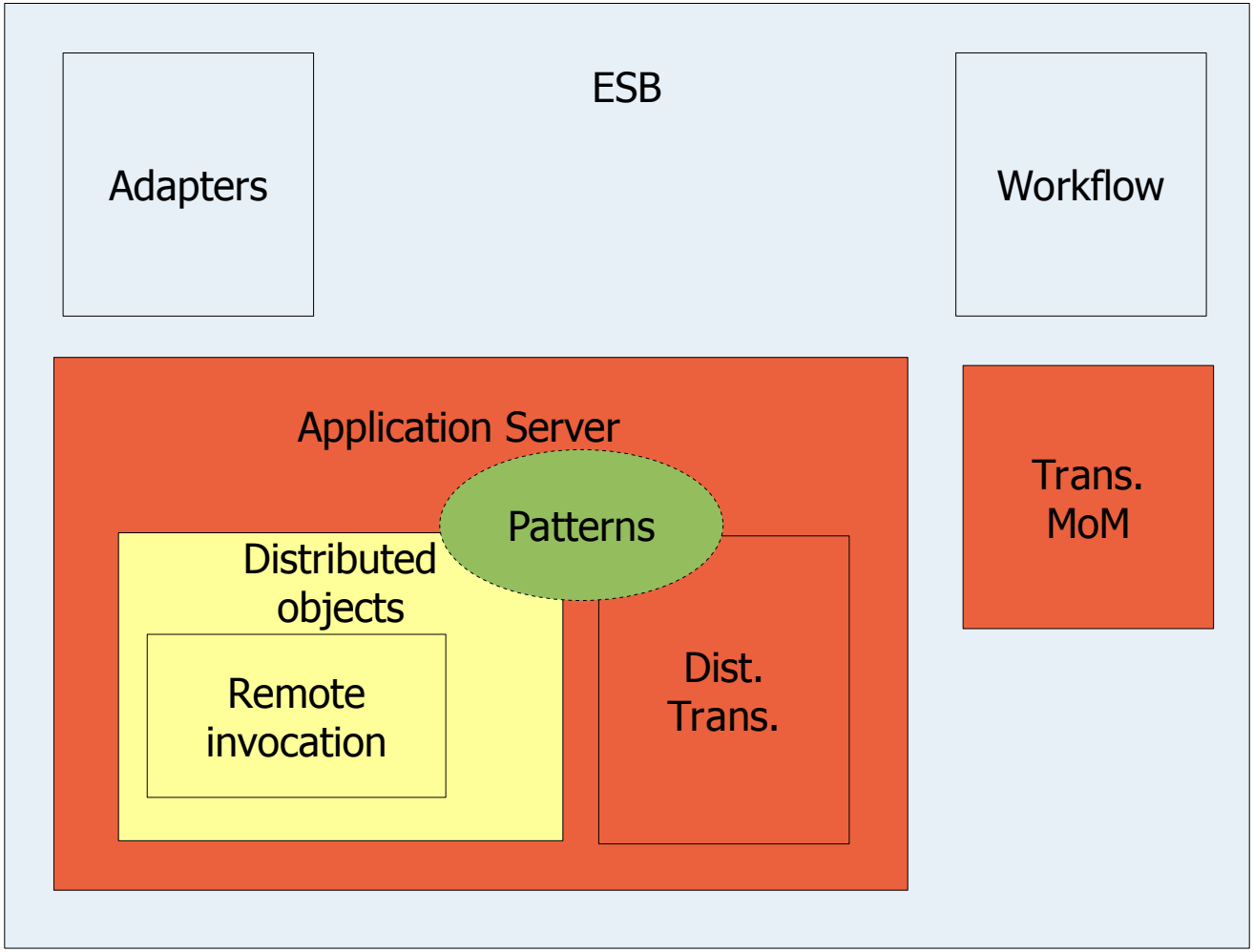
Enterprise Service Bus

- Combines messaging with common adapters
- Provides common routing applications:
  - split/join/...
- Directly supports workflow / business processes:
  - Message routing rules compiled from high-level business process descriptions:



Conclusions

- Bottom-up view of middleware stack:



Key enablers

- Reflection:
  - Inferring IDL
  - Inferring database schema
- Run-time code generation:
  - Stubs and skeletons
  - Persistence
- Transactions:
  - Composition and concurrency make deadlocks unavoidable
  - Rollback required

Conclusions

- Transactions in middleware provide:
  - Reliability
  - Programming by composition
  - Simple declarative abstractions