

ABL Data Integration

Today and Tomorrow

David Moloney
Software Architect
Progress Software

PROGRESS
EXCHANGE 2014

Agenda

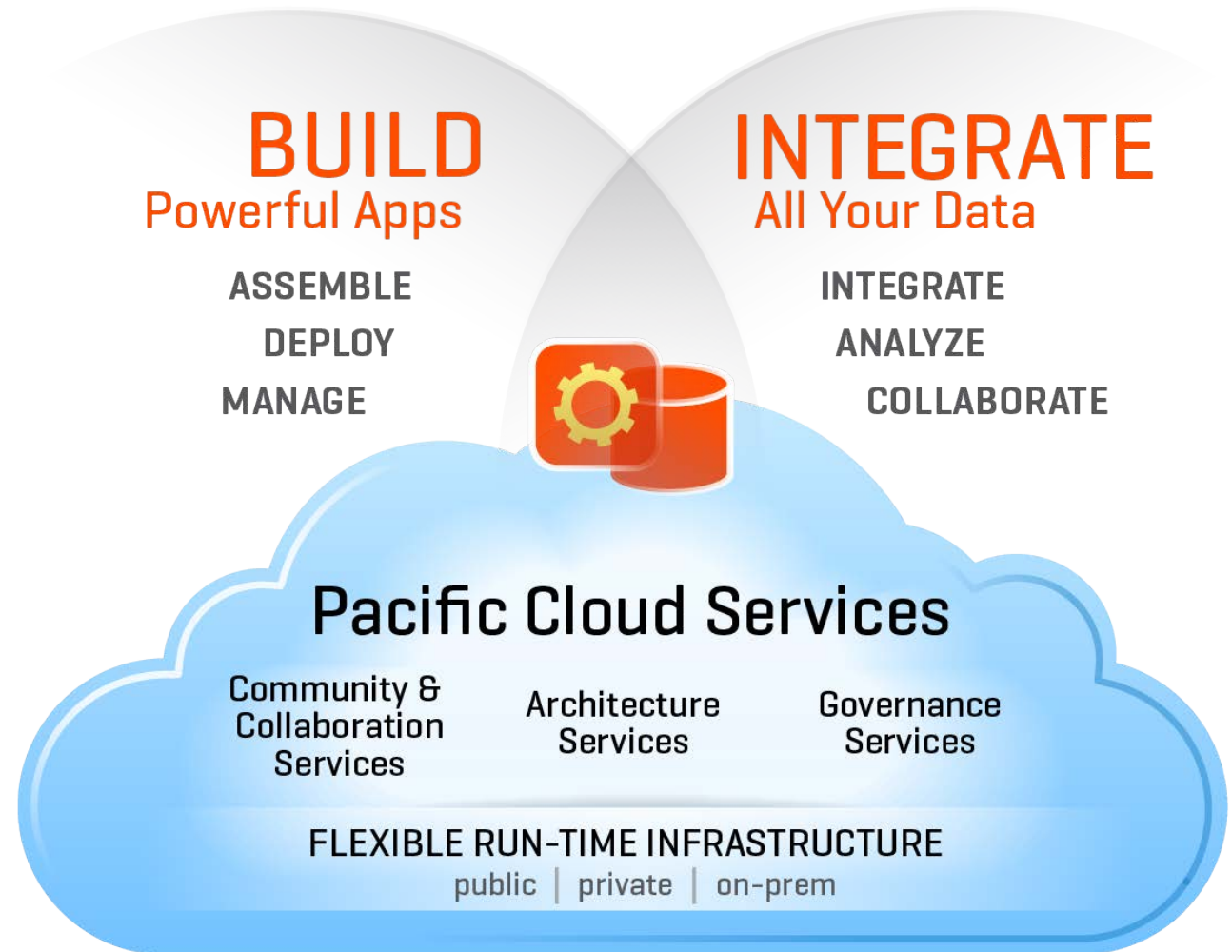
- Challenges of Data Integration
- ABL Data Integration Methodologies
- Loose Versus Tight Coupling at the Point of Integration
- ODBC Bridge
- Future Data Abstractions, Embedded Objects and Tight Integration

Challenges of Data Integration

PROGRESS
EXCHANGE 2014

The Challenges of Data Integration

- **Form:** Consumed, hosted, (un)structured, Integrity, Volume, Velocity, Variability
- **Approach:** Technology/cost, scalability, efficiency, throughput, availability, data analysis
- **Business Climate:** Customer/partner demands, competition, mergers, divestments



The Challenges of Data Integration

Data  Application

Transaction-Oriented OpenEdge Applications Are About Data

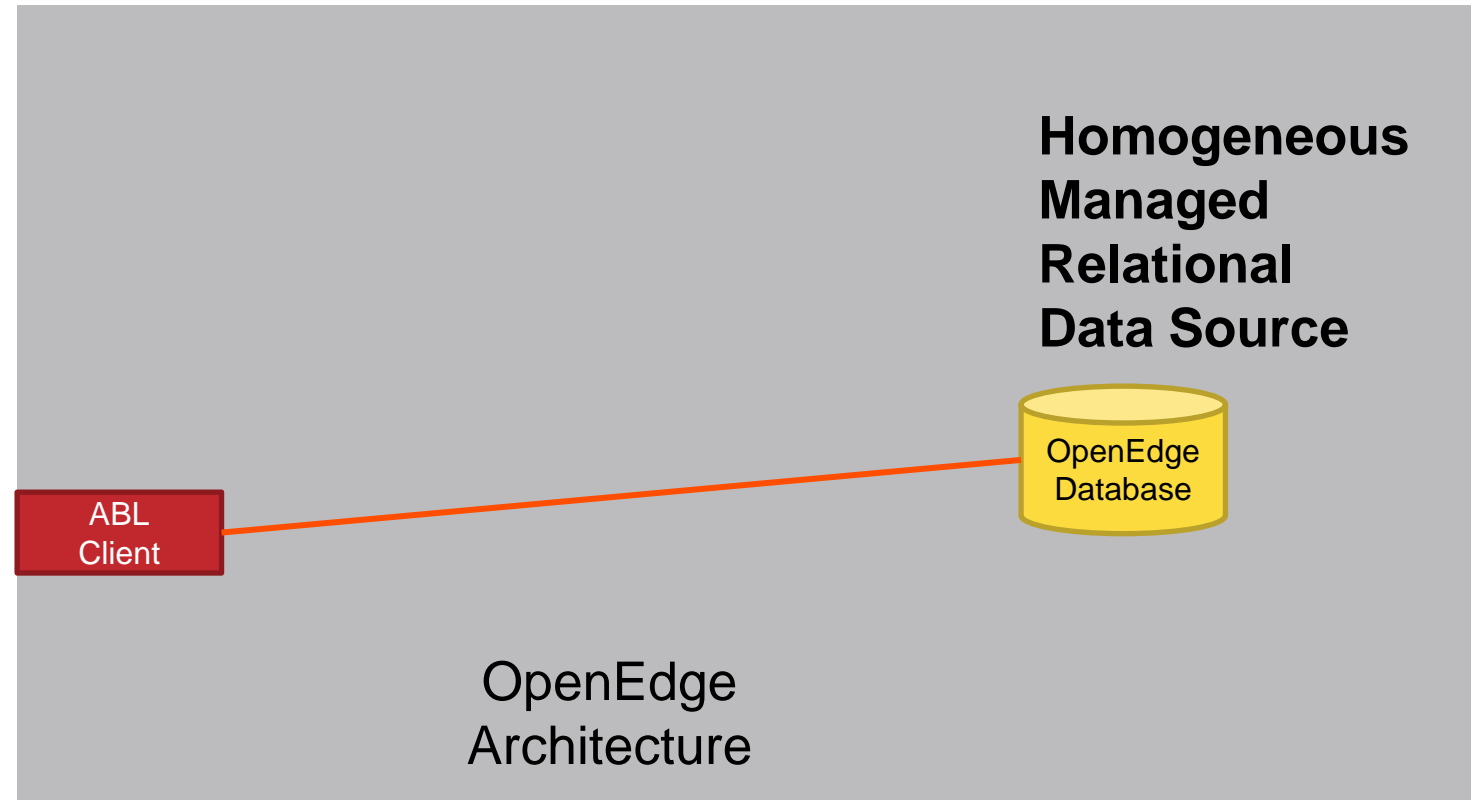
Discovering, cleansing, monitoring,
transforming, aggregating and delivering
data from disparate sources to where it is
needed, when it is needed

ABL Data Integration Methodologies

PROGRESS
EXCHANGE 2014

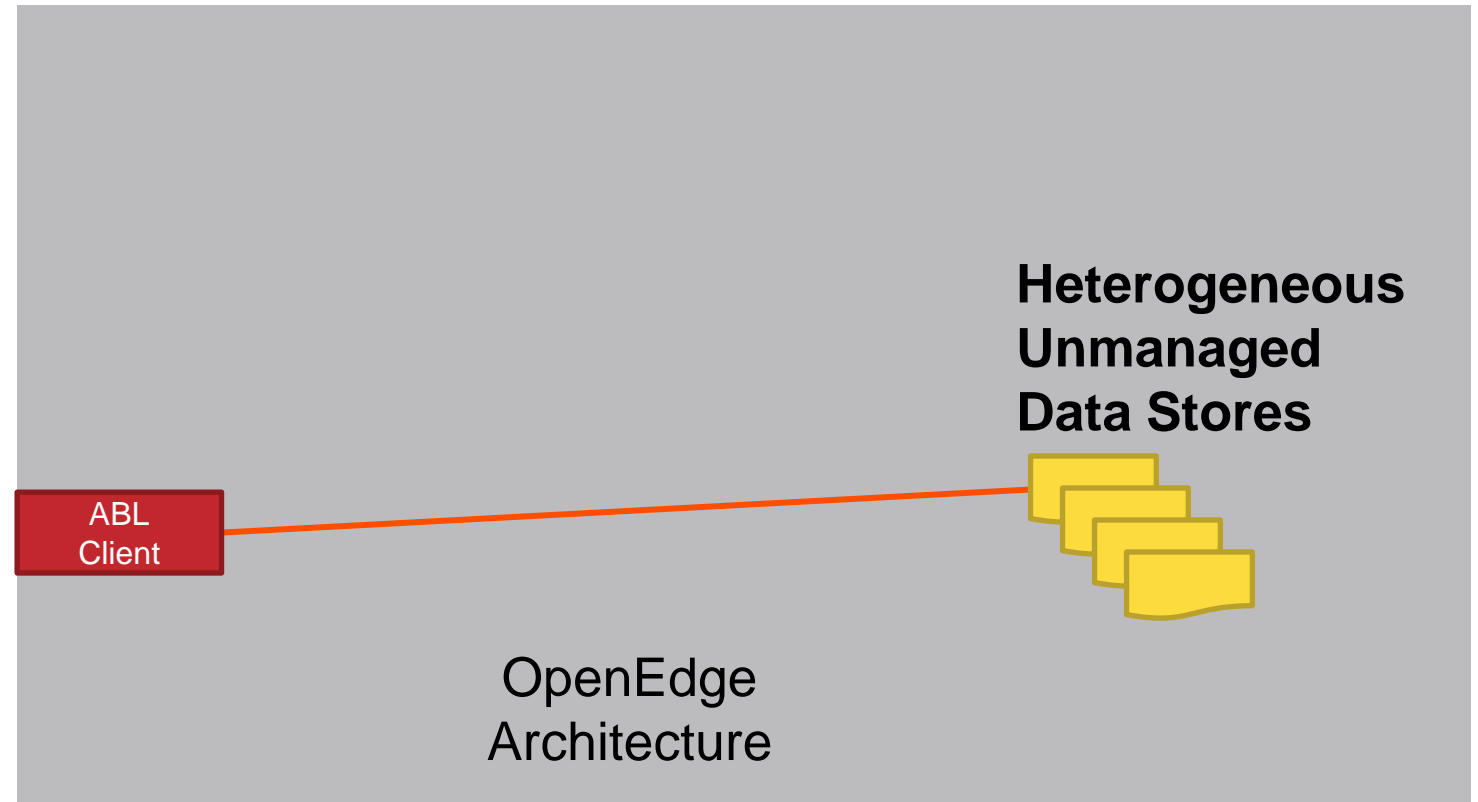
Traditional ABL Language Binding: Relational Data Access

- FIND
- DELETE
- CAN-FIND
- IMPORT
- COPY-BUFFER
- FOR EACH
- CREATE
- AVAILABLE
- EXPORT
- DEFINE QUERY/DATASET
- UPDATE



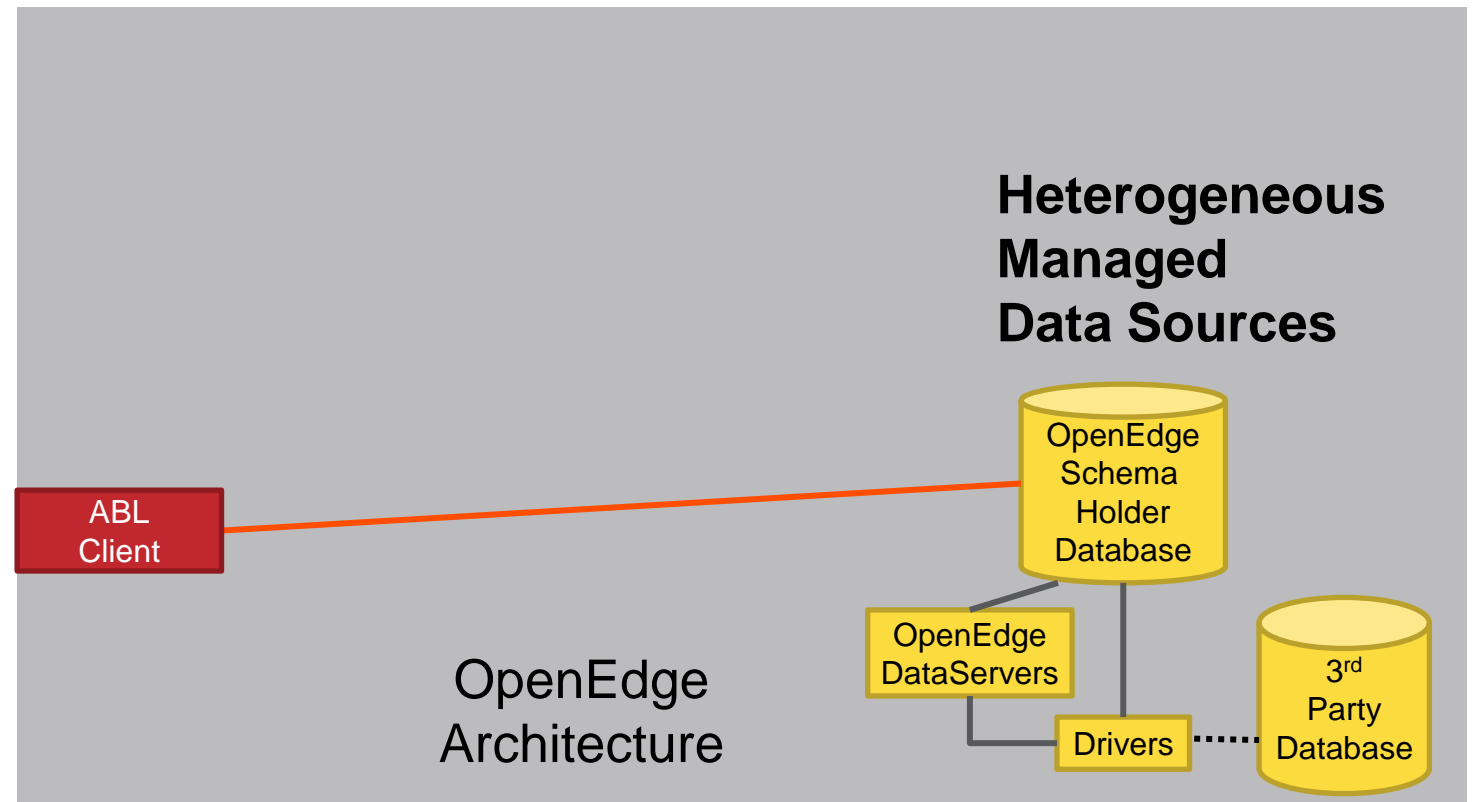
Traditional ABL Language Binding: File Distribution & Streaming

- IMPORT / EXPORT
- INPUT FROM / OUTPUT TO
- PUT
- INPUT THROUGH
- OUTPUT THROUGH



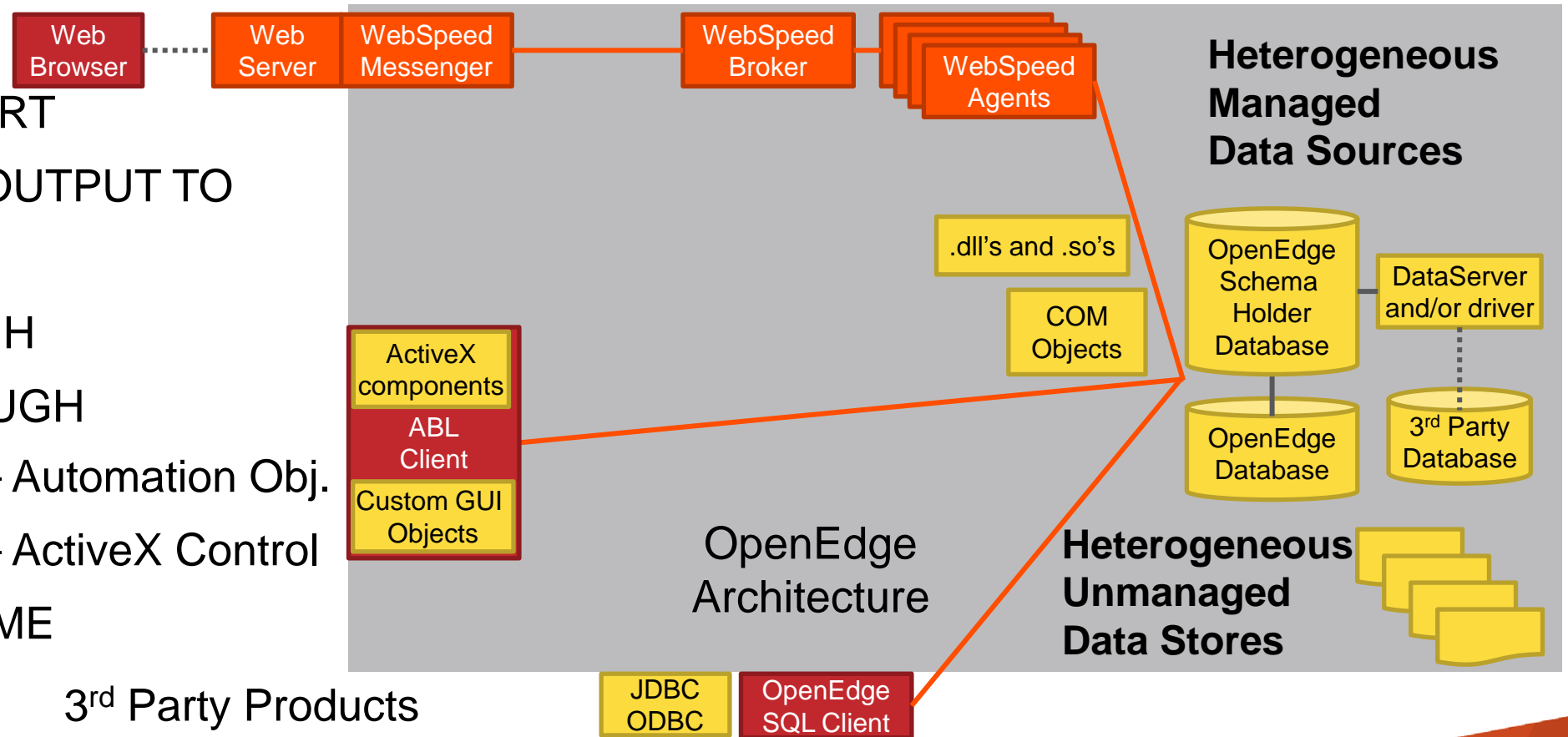
Traditional ABL Language Binding: Relational Data Access via “Gateways”

- FIND
- DELETE
- CAN-FIND
- IMPORT
- COPY-BUFFER
- FOR EACH
- CREATE
- AVAILABLE
- EXPORT
- DEFINE QUERY/DATASET
- UPDATE



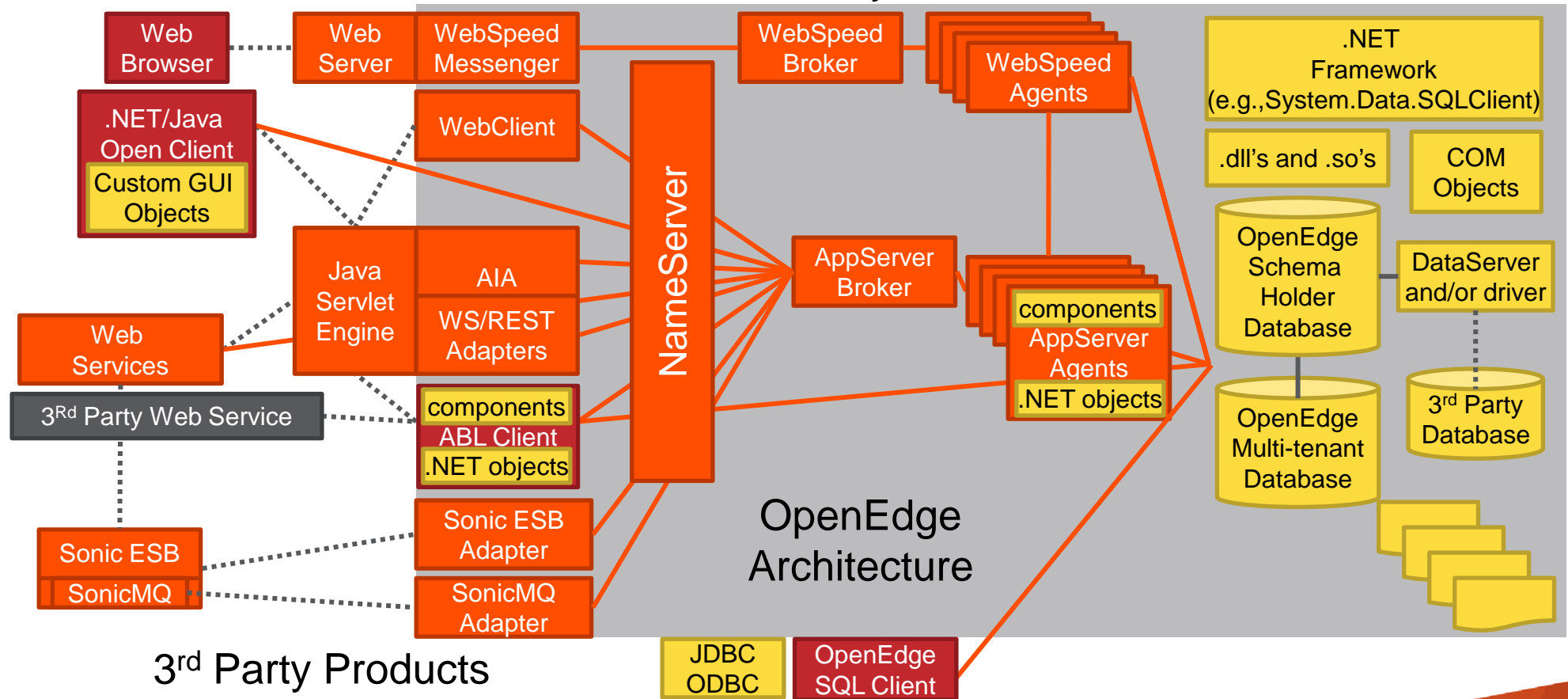
Traditional ABL Language Binding: Era of Web 1.0

- FIND
- DELETE
- CAN-FIND
- IMPORT
- COPY-BUFFER
- FOR EACH
- CREATE
- AVAILABLE
- EXPORT
- DEFINE QUERY/DATASET
- UPDATE
- IMPORT / EXPORT
- INPUT FROM / OUTPUT TO
- PUT
- INPUT THROUGH
- OUTPUT THROUGH
- COM-HANDLE – Automation Obj.
- COM-HANDLE – ActiveX Control
- CONTROL-FRAME
- (legacy DDE)



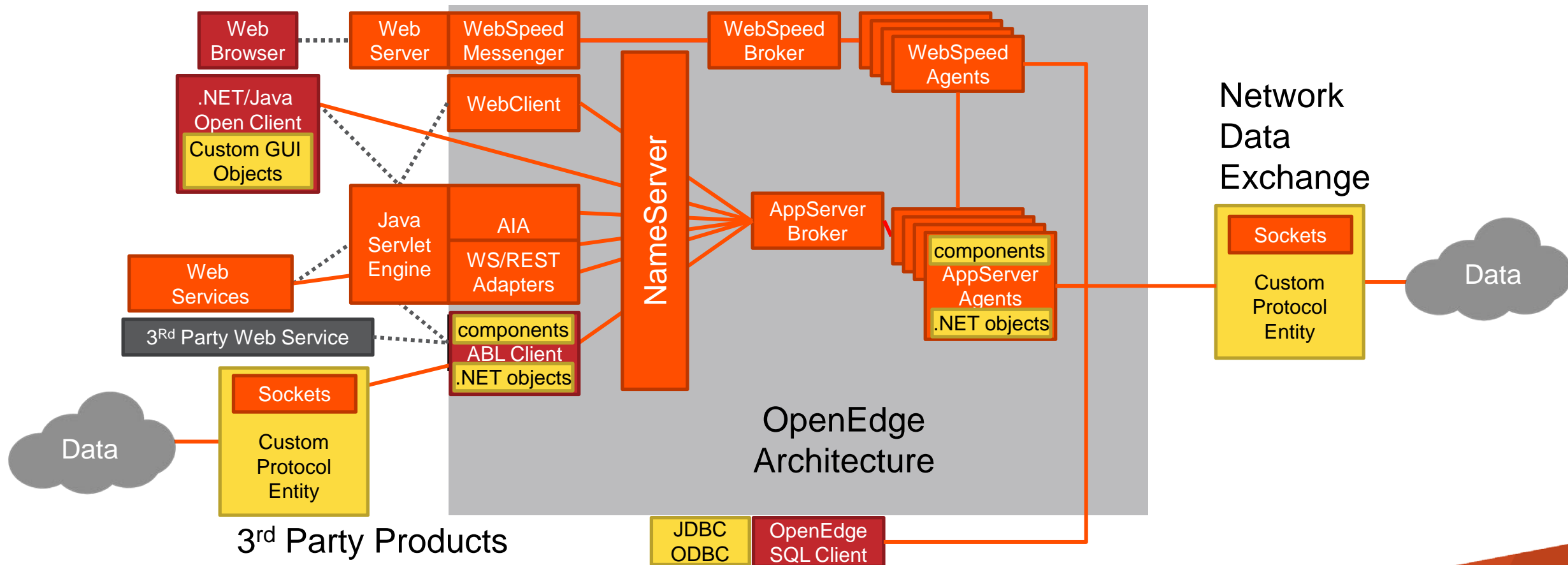
Non-Traditional, Non-Language Binding: Era of Web 2.0

- Progress.Data.BindingSource Object
- Progress.Windows.Form
- USING System.Windows.Forms
- DEFINE DATASET ... NESTED
- USING System.Data.SqlClient
- USING System.Data.BindSource

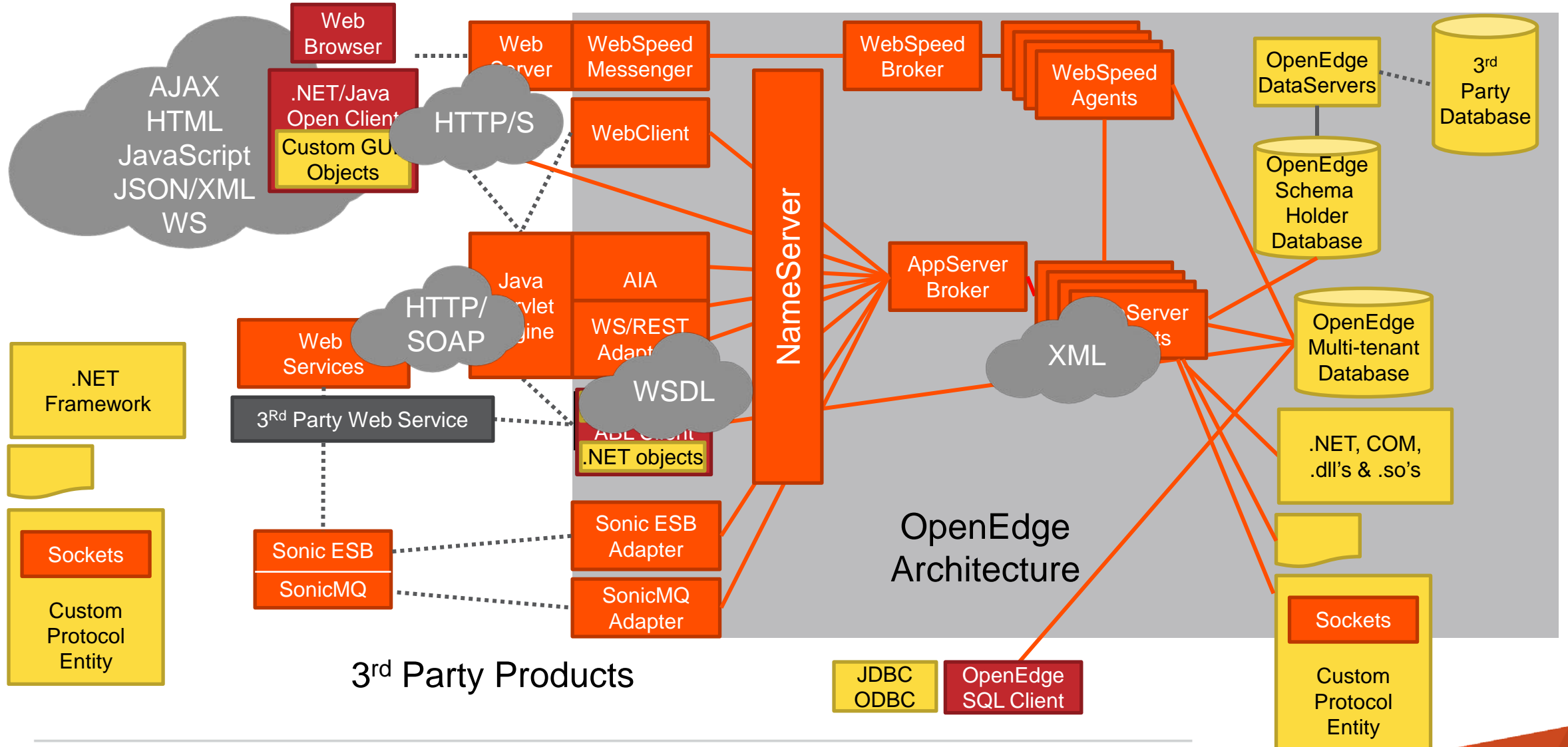


Language-Based Network Interface: Message-Based Data Access

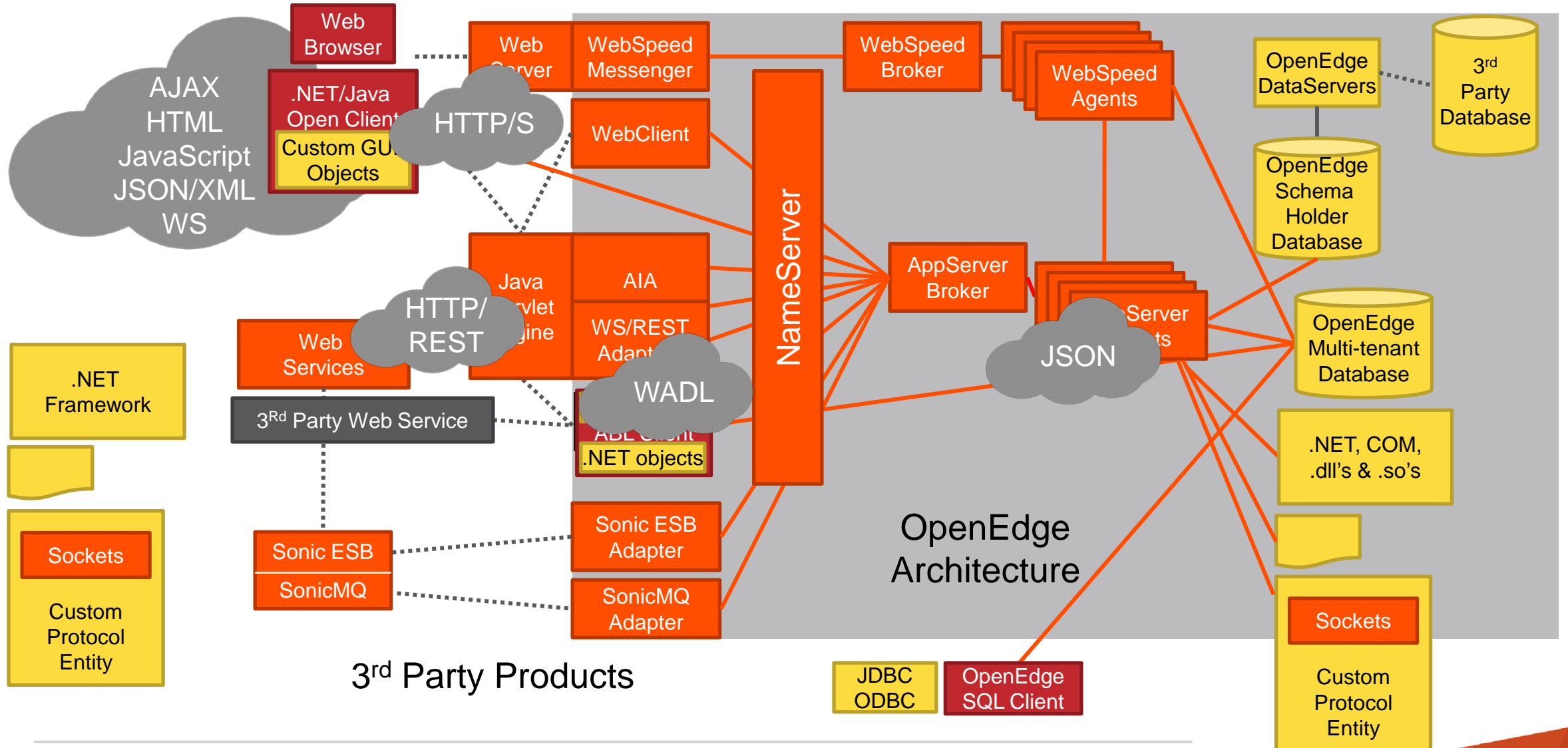
- CREATE SERVER-SOCKET
- CREATE SOCKET
- GET/SET-SOCKET-OPTION
- CONNECT/DISCONNECT
- READ/WRITE
- READ-RESPONSE



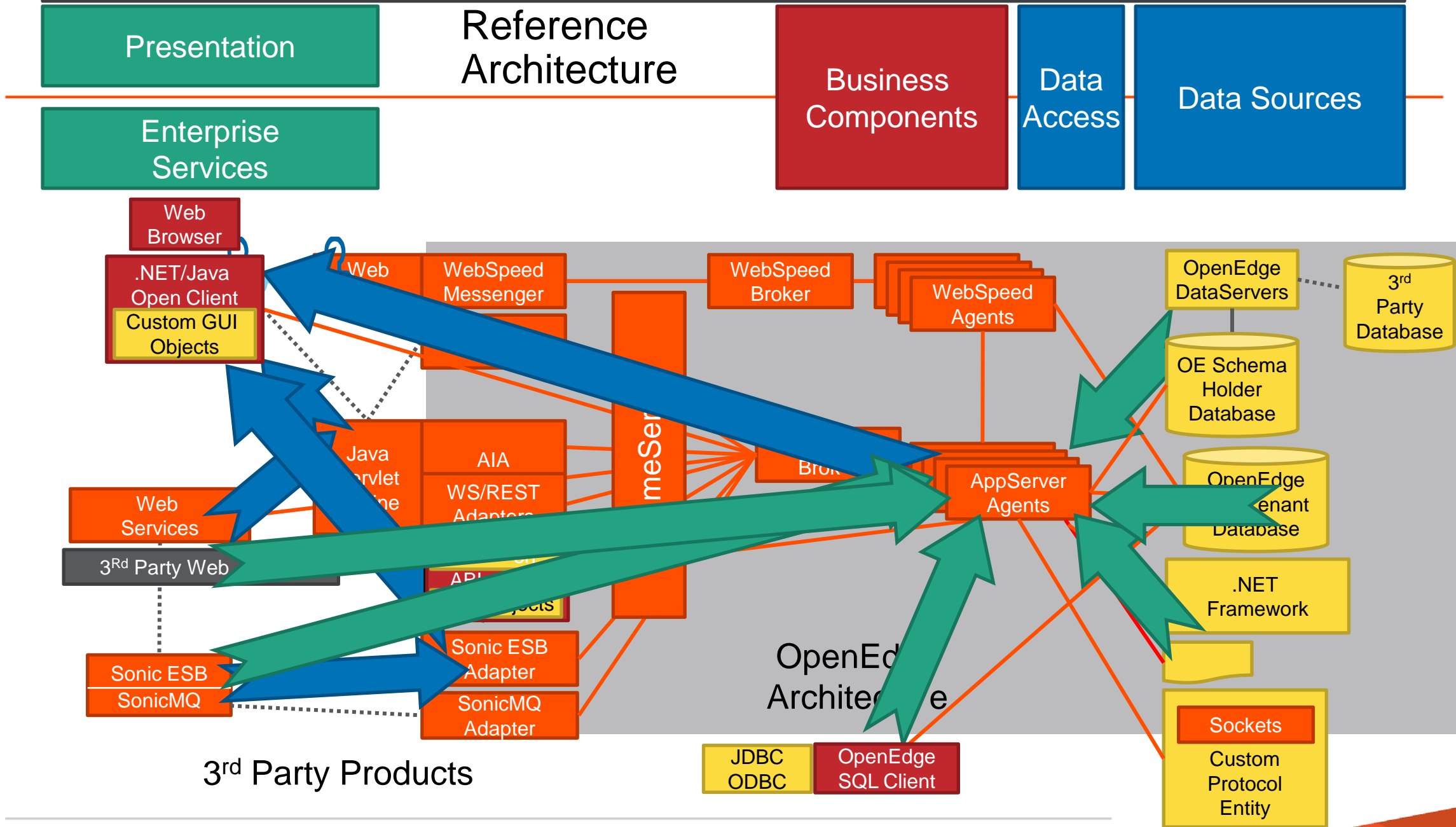
Web Services Data Transport (Heavyweight)



REST Services Data Transport (Lightweight)



Common Infrastructure



Presentation

Reference Architecture

Enterprise Services

Web Browser

.NET/Java Open Client
Custom GUI Objects

Web Server

Java Servlet Engine

Web Services

3rd Party Web Service

Sonic ESB
SonicMQ

Business Components

Data Access

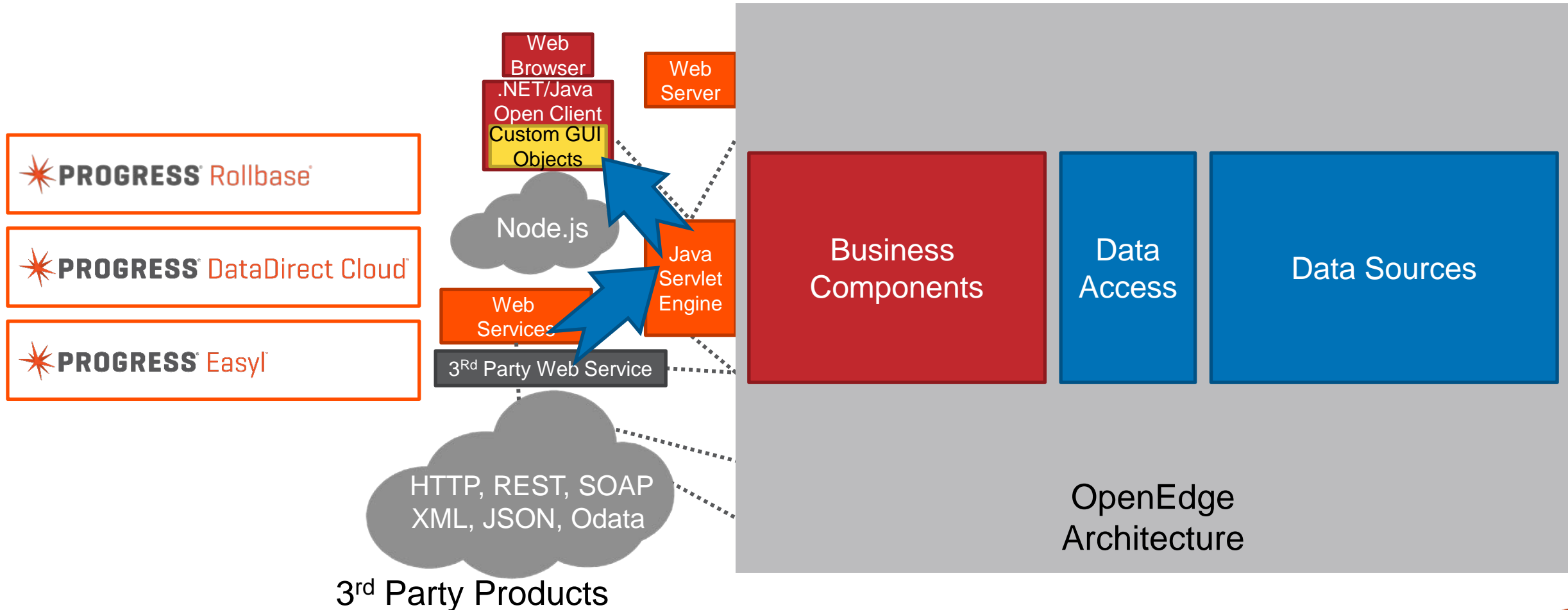
Data Sources

OpenEdge Architecture

3rd Party Products

Common Infrastructure

Reference Architecture

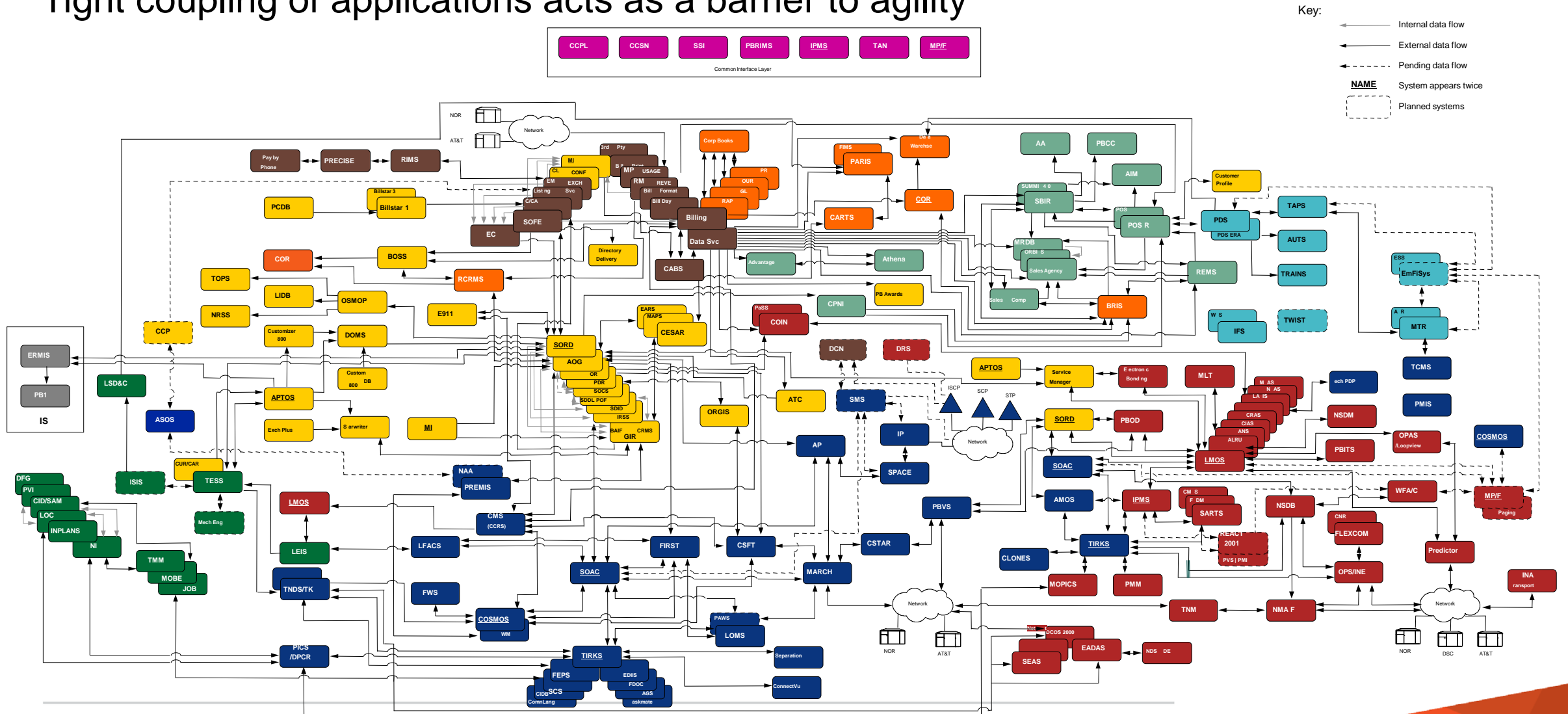


Loose vs. Tight Coupling at the Point of Integration

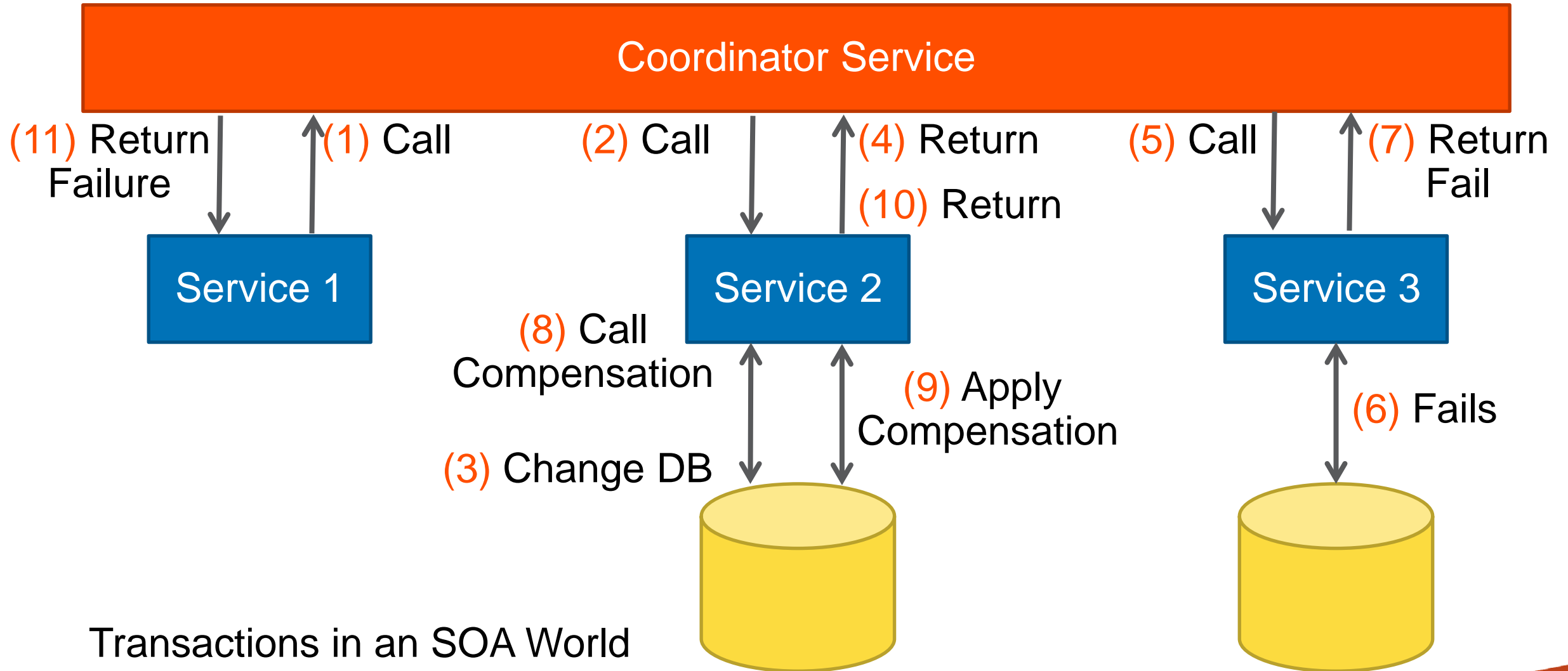
PROGRESS
EXCHANGE 2014

The SaaS Argument to Application Integration

Tight coupling of applications acts as a barrier to agility



Coordination Service



Transactions in an SOA World

My Contention...

Key Question:

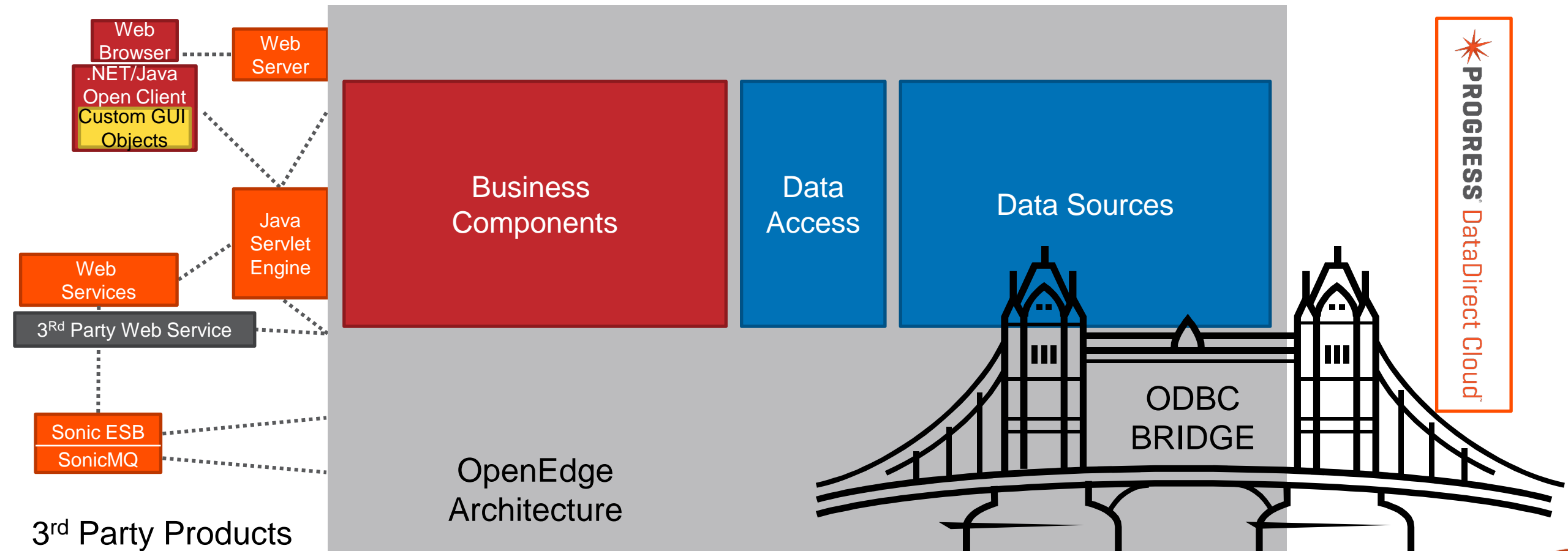
1. How will you organize your SOA environment to reach your goals?
 - How will you distribute consumers and providers of your services
2. What “services” and non-services operate against your key business components?
 - Data Integration
 - Data Transformation
 - Data Persistence
 - Data aggregation
 - Data Storage/Access

Essential business resources and drivers of operational decisions need tight binding

ODBC Bridge

PROGRESS
EXCHANGE 2014

Common Infrastructure



Reference Architecture

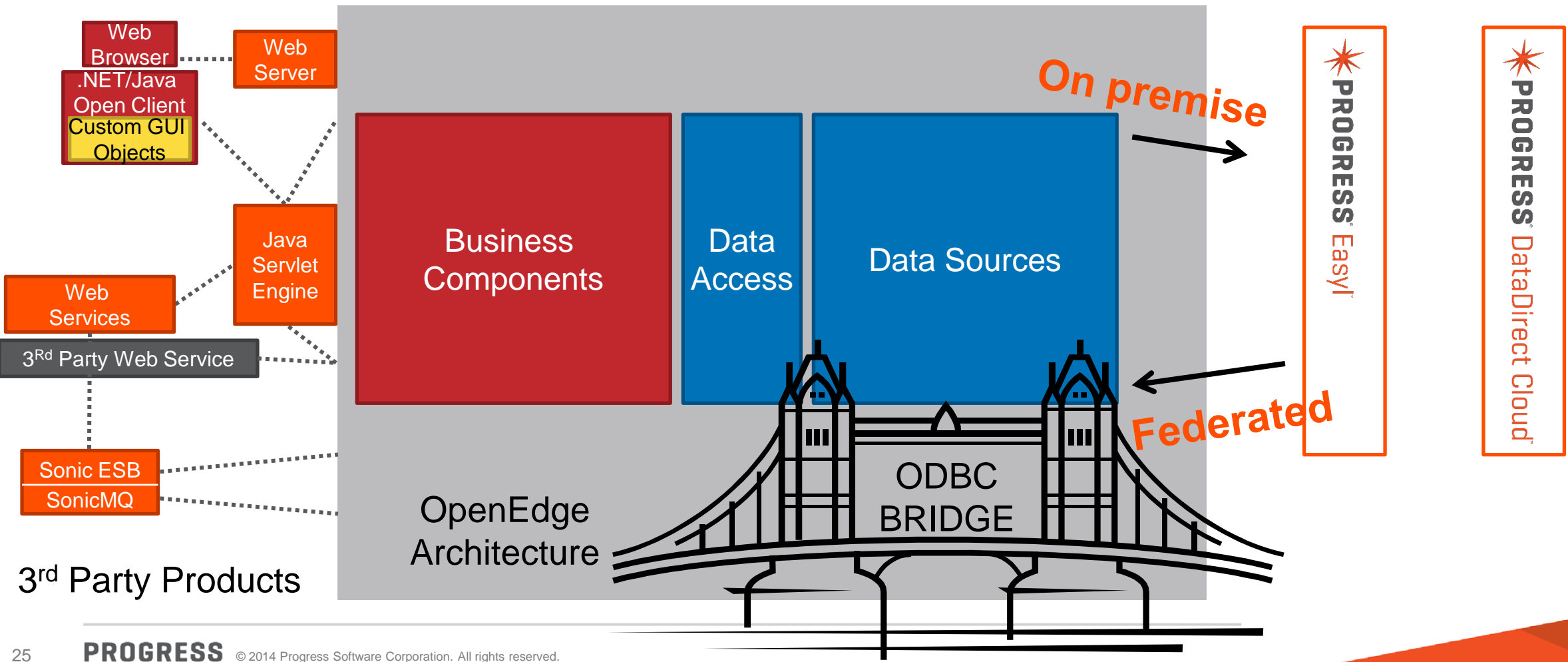
Presentation

Enterprise Services

Business Components

Data Access

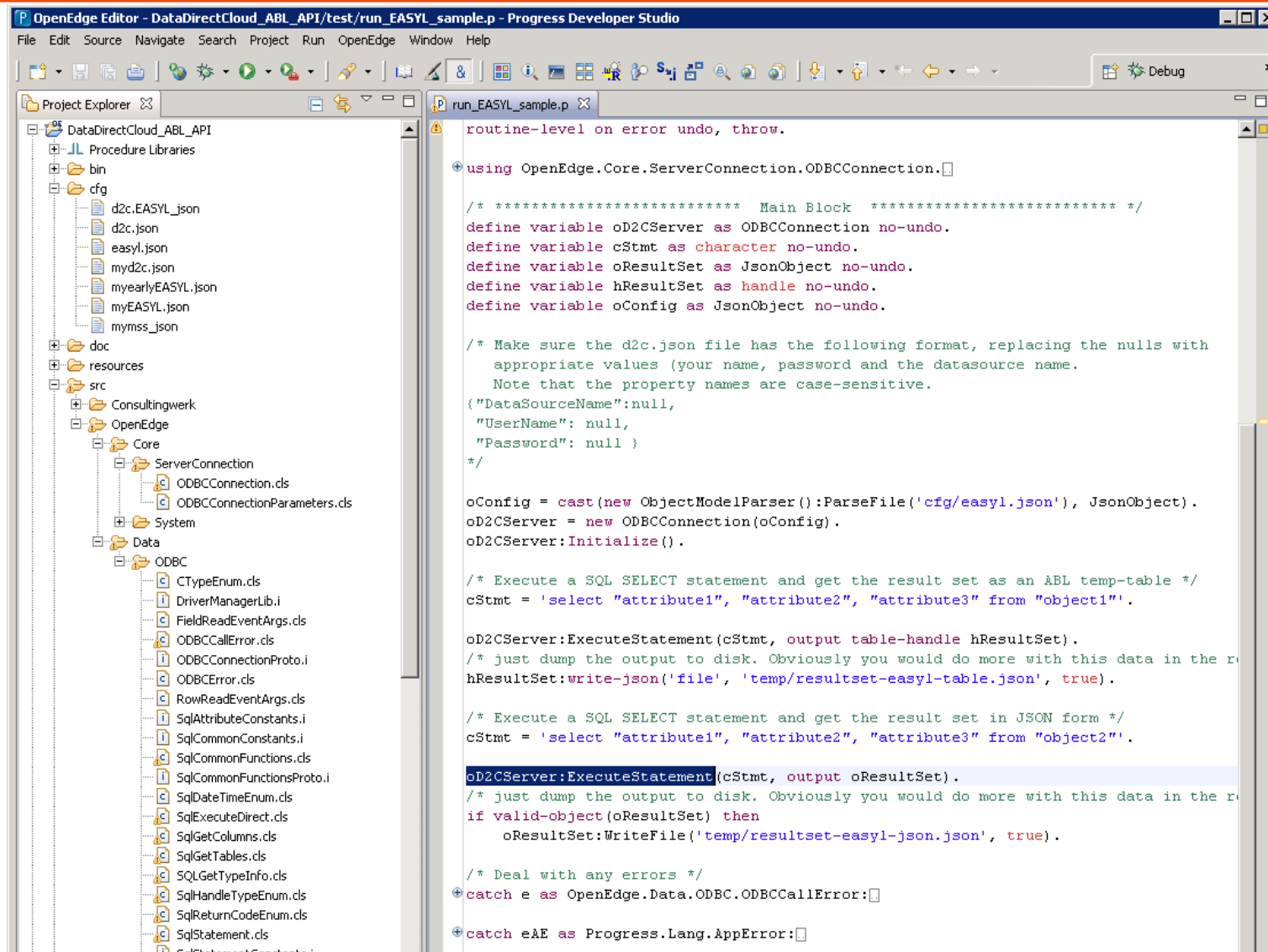
Data Sources



ODBC Bridge to Progress DataDirect Cloud and Progress EasyL

- Code Samples
 - Available August 2013 in OE 11.3.0 for D2C
 - Available March 2014 in OE 11.3.2 for EASYL
- ODBC API linked/mapped; API functions in include file: “SqlStatementProto.i”
- ODBC processes are object-ized into OOABL
 - SELECT queries return schema that is evaluated
 - Full CRUD possible but not provided in sample
- Results use default data type mappings
- Can assign results to temp tables or JSON objects
- No specific internationalization, LOB processing, or native ODBC extensions, etc.
- All API’s “available”; Only API’s needed by same are “implemented”

ODBC Bridge Code in PDSOE



```
OpenEdge Editor - DataDirectCloud_ABL_API/test/run_EASYL_sample.p - Progress Developer Studio
File Edit Source Navigate Search Project Run OpenEdge Window Help

Project Explorer
DataDirectCloud_ABL_API
  Procedure Libraries
  bin
  cfg
    d2c.EASYL_json
    d2c.json
    easy1.json
    myd2c.json
    myearlyEASYL.json
    myEASYL.json
    mymss_json
  doc
  resources
  src
    Consultingwerk
    OpenEdge
      Core
        ServerConnection
          ODBCConnection.cls
          ODBCConnectionParameters.cls
        System
      Data
        ODBC
          CTypeEnum.cls
          DriverManagerLib.i
          FieldReadEventArgs.cls
          ODBCError.cls
          ODBCConnectionProto.i
          ODBCError.cls
          RowReadEventArgs.cls
          SqlAttributeConstants.i
          SqlCommonConstants.i
          SqlCommonFunctions.cls
          SqlCommonFunctionsProto.i
          SqlDateTimeEnum.cls
          SqlExecuteDirect.cls
          SqlGetColumns.cls
          SqlGetTables.cls
          SQLGetTypeInfo.cls
          SqlHandleTypeEnum.cls
          SqlReturnCodeEnum.cls
          SqlStatement.cls
          SqlStatementConstants.i

run_EASYL_sample.p
  routine-level on error undo, throw.
  using OpenEdge.Core.ServerConnection.ODBCConnection.
  /* ***** Main Block ***** */
  define variable oD2CServer as ODBCConnection no-undo.
  define variable cStmt as character no-undo.
  define variable oResultSet as JsonObject no-undo.
  define variable hResultSet as handle no-undo.
  define variable oConfig as JsonObject no-undo.

  /* Make sure the d2c.json file has the following format, replacing the nulls with
  appropriate values (your name, password and the datasource name.
  Note that the property names are case-sensitive.
  {"DataSourceName":null,
  "UserName": null,
  "Password": null }
  */

  oConfig = cast(new ObjectModelParser():ParseFile('cfg/easy1.json'), JsonObject).
  oD2CServer = new ODBCConnection(oConfig).
  oD2CServer:Initialize().

  /* Execute a SQL SELECT statement and get the result set as an ABL temp-table */
  cStmt = 'select "attribute1", "attribute2", "attribute3" from "object1"'.

  oD2CServer:ExecuteStatement(cStmt, output table-handle hResultSet).
  /* just dump the output to disk. Obviously you would do more with this data in the r
  hResultSet:write-json('file', 'temp/resultset-easy1-table.json', true).

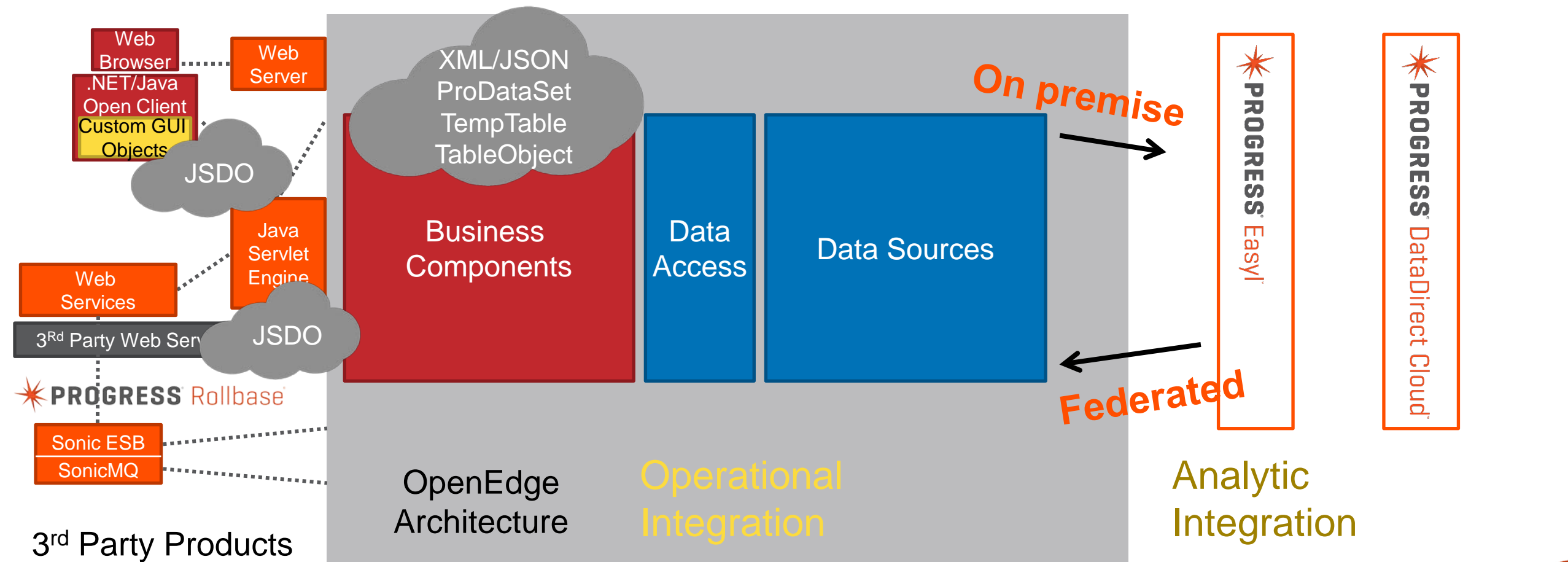
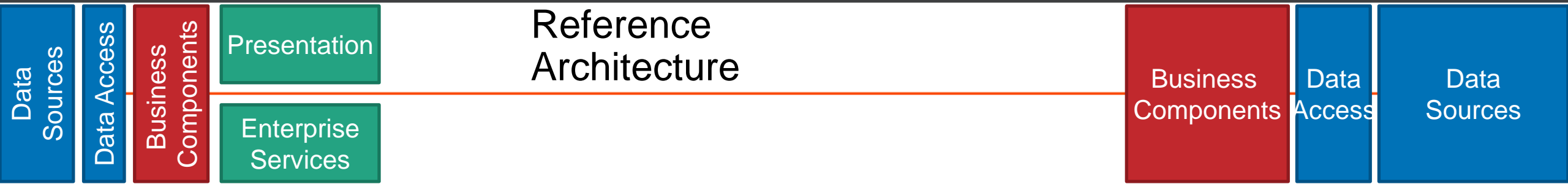
  /* Execute a SQL SELECT statement and get the result set in JSON form */
  cStmt = 'select "attribute1", "attribute2", "attribute3" from "object2"'.

  oD2CServer:ExecuteStatement(cStmt, output oResultSet).
  /* just dump the output to disk. Obviously you would do more with this data in the r
  if valid-object(oResultSet) then
    oResultSet:WriteFile('temp/resultset-easy1-json.json', true).

  /* Deal with any errors */
  catch e as OpenEdge.Data.ODBC.ODBCError.
  catch eAE as Progress.Lang.AppError.
```

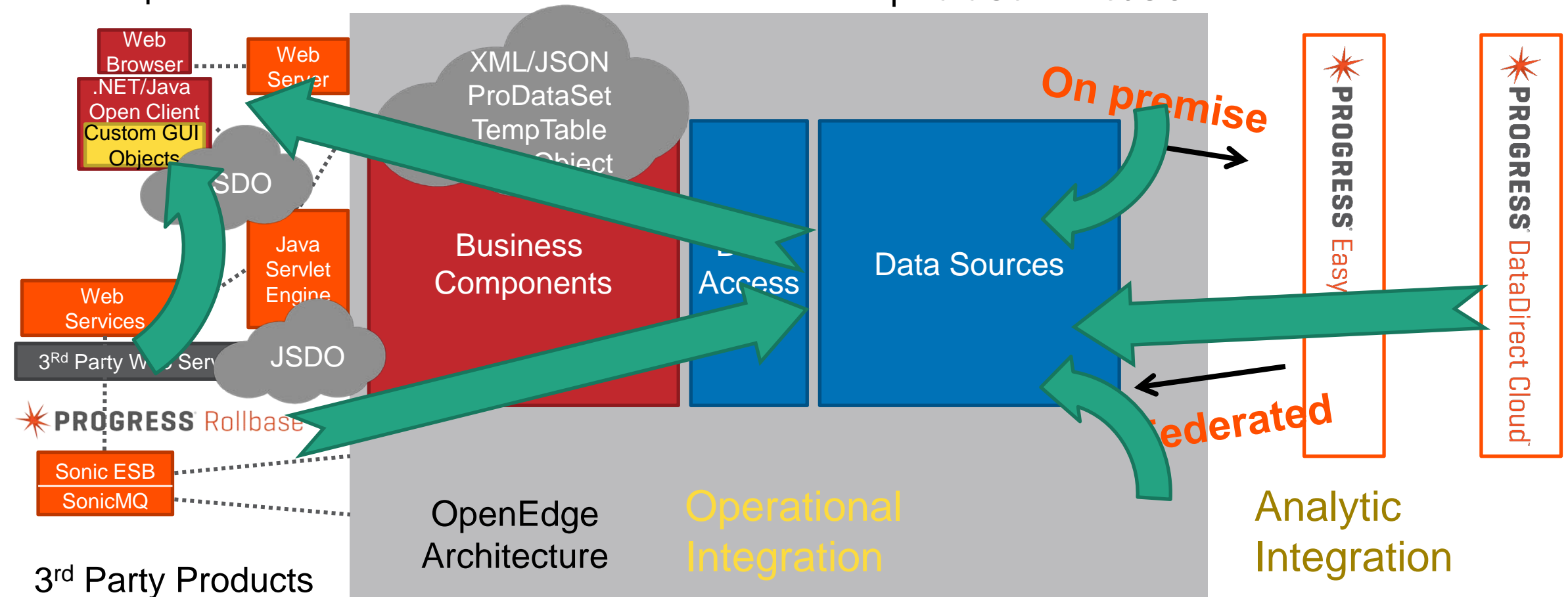
ODBC Bridge Code in PDSOE

```
oD2CServer = New ODBCConnection(<connect data>). /* conn obj */  
oD2CServer:ExecuteStatement(<stmt>, <result-obj>). /* run SQL */  
If <result-obj> then  
    <result-obj>:WriteFile(<json-file>). /* or temp-table */
```



Web Services Front End and Business Entities Behind the Firewall

- DEFINE DATASET-HANDLE pDataSet
- pDataSet:ReadXML
- pDataSet:WriteXML
- pDataSet:NameSpace-Uri
- pDataSet:ReadJSON
- pDataSet:WriteJSON



Future Data Abstractions,
Embedded Objects,
and Tight Integration

PROGRESS
EXCHANGE 2014

Disclaimer

- This presentation is for informational purposes only. You are cautioned that any information contained in this presentation may change in the course of product development.
- This presentation may not be interpreted as any commitment on behalf of Progress, and future development, timing and release of any products, features or functionality described in this presentation remains at the sole discretion of Progress.

Next Steps in “OpenEdge-Centric” Data Integration

1. Do Nothing
2. Improve Samples
3. OOABL built-in object supplements & embedded API
 - Expose Abstractions as built-ins (using internal OTM)
4. OOABL built-in objects with language integration & embedded API
 - Modernized, flexible, DataServer-equivalency + more

Object Model Representation of ODBC Base Classes

- **DataDirect.DO.ODBC.API**
 - Open Standards, Data Access Methods
- **DataDirect.DO.ODBC.Defs**
 - Open Standards, Data Access Properties
- **Progress.Lang.SysError**
 - **Progress.DO.ODBC.Error**
 - includes getSQLState() method

CLASS myAPIObj INHERITS (or IMPLEMENTS) API:

API:SQLSetStmtAttr(...).

.....

END CLASS.

Abstraction Layer on Top of Base Classes (Similar to Sample Program's)

- Embedded API's as built-in objects
 - DataDirect.DO.RecordSet – Any result set definition
 - DataDirect.DO.Connection – Any connected data source
 - DataDirect.DO.Command – Any SQL call or statement(s)

ConnObj:Open(User, Pwd, DSN).

RecordSetObj = CommandObj:ExecuteStatement("Select * from Customer").

Do WHILE RecordSetObj:HasRecs:

 CREATE tt.

 ASSIGN tt.name = RecordSetObj:column("name"):VALUE.

 RecordSetObj:MoveNext NO-ERROR

END.

Disadvantages: Opt.1: Samples Thru Opt. 3: OOABL Built-In Object Supplements

- Code practices on embedded object → no correlation to existing code or practices
- Foreign schema references → not tied to the language
 - No schema/reference checks; mismatches result in corruption?
- Integration capabilities → data bound copies;
 - All CRUD by copy
- Purposed Objects → provide varying degrees of abstraction & function
 - Roll your own ODBC would be available but would require foundation work from you
- No Integration → Transactions, record scope, cursor management, etc.
- Only default mappings → for derived temp-table, JSON, other targets
 - No data conversion on results
- Internationalization, LOB types, native extensions, language hooks → loosely possible over time; must be “hacked” into ABL language.

Foundational Abstractions for Built-In Objects With Language Binding

- `DataDirect.DO.DB` – logical database equivalent
- `DataDirect.DO.Table` – file equivalent
- `DataDirect.DO.Column` – field equivalent
- `DataDirect.DO.Index` – index equivalent
- `DataDirect.DO.DS` – DATA-SOURCE equivalent

FULL
DISCLAIMER

Query Example Using Dynamic Schema Object Definitions

```
DEFINE PRIVATE VARIABLE myCloudDbObj AS CLASS DataDirect.DO.DB NO-UNDO.  
DEFINE PRIVATE VARIABLE custObj AS CLASS DataDirect.DO.Table NO-UNDO.
```

```
myCloudDbObj:GET-EMPTY("CloudDSN");  
myCloudDbObj:SET-CONNECT(("CloudDSN", "sports", UserID, Pwd).  
myCloudDbObj:LOAD-SCHEMA()  
custObj = NEW DO.TABLE(myCloudDbObj:GET-TABLE-SCHEMA("ora-cust")). /* foreign name */  
  
CREATE BUFFER bh FOR TABLE (custObj.GET-LOCAL-NAME()).  
CREATE QUERY qh.  
  
qh.SET-BUFFERS(bh).  
qh:QUERY-PREPARE("FOR EACH " + custObj.GET-LOCAL-NAME()).  
qh:QUERY-OPEN().  
qh:GET-FIRST().  
qh:QUERY-CLOSE().  
Bh:BUFFER-RELEASE().  
DELETE OBJECT bh.  
DELETE OBJECT qh.
```

Query Example Using Static Object Definitions & Database Persistence

```
DEFINE PRIVATE VARIABLE mydbObj1 AS CLASS DataDirect.DO.DB NO-UNDO.  
DEFINE PRIVATE VARIABLE mydbObj2 AS CLASS DataDirect.DO.DB NO-UNDO.  
myClouddbObj1 = new DB("CloudDSN1", "sports1", UserID, Pwd).  
myClouddbObj2 = new DB("CloudDSN2", "sports2", UserID, Pwd).  
myClouddbObj1:LOAD-SCHEMA("Cust*").  
myClouddbObj2:LOAD-SCHEMA("Ord*").  
myClouddbObj1:CREATE-DB().  
myClouddbObj2:CREATE-DB().
```

```
prodb -db CloudDSN1 -ld "sports1" -db CloudDSN2 -ld "sports2"
```

```
DEFINE QUERY CustOrd FOR sports1.customer, sports2.order  
OPEN QUERY q FOR EACH sports1.customer, EACH sports2.order OF sports1.customer.  
GET FIRST q  
DO WHILE NOT QUERY-OFF-END('q').  
    GET NEXT q  
END  
CLOSE QUERY q
```

Transaction Example Using Static Schema Object Definitions

```
DEFINE PRIVATE VARIABLE myClouddbObj AS CLASS DataDirect.DO.DB NO-UNDO.
```

```
myClouddbObj = new DB("CloudDSN", "sports", UserID, Pwd).
```

```
myClouddbObj:LOAD-SCHEMA("Customer")
```

```
SAVE CACHE COMPLETE myClouddbObj:ALIAS-NAME  
    TO VALUE(myClouddbObj:DB-NAME) + ".csh") NO-ERROR.
```

```
IF ERROR-STATUS:ERROR THEN MESSAGE "Not Saved".
```

```
prodb -cache CloudDSN.csh
```

```
/* proutil <database> -C load schema <filename> - db */
```

TransBlock:

```
DO TRANSACTION ON ERROR UNDO, LEAVE
```

```
DO
```

```
    FIND customer.
```

```
    ASSIGN customer.cust-num = 12.
```

```
    UPDATE customer.
```

```
END.
```

```
END.
```


Additional Integration Possibilities

```
DEF VAR custObj AS CLASS DataDirect.DO.Table.
```

```
custObj:Update(SET cust-num = 12").
```

```
custObj:FILL.
```

```
DEF VAR cmdObj AS CLASS DataDirect.DO.Command.
```

```
cmdObj:SET("UPDATE customer SET cust-num = 12").  
cmdObj:EXECUTE().
```

```
cmdObj:SET("SELECT * from customer").  
cmdObj:EXECUTE().
```

```
DEFINE QUERY Cust FOR sports.customer.
```

```
OPEN QUERY q FOR EACH sports.customer
```

```
DEFINE QUERY Cust FOR sports.customer.
```

```
OPEN QUERY q Select * from sports.customer
```

```
DEFINE QUERY qCust FOR sports.customer.
```

```
DEFINE DATA-SOURCE srcCust FOR QUERY qCust.
```

```
DEFINE QUERY qCust FOR sports.customer.
```

```
DEFINE DATA-SOURCE srcCust Select * from sports.customer.
```

```
DEFINE VARIABLE mydbObj AS CLASS DataDirect.DO.DB NO-UNDO.
```

```
myClouddbObj = new DB("CloudDSN", "sports", UserID, Pwd).
```

```
myClouddbObj:DIALECT = "Oracle".
```

Additional Integration Possibilities (ProDataSet CloudServer)

```
DEFINE VARIABLE mydbObj AS CLASS DataDirect.DO.DB NO-UNDO.
```

```
myClouddbObj = new DB("CloudDSN", "sports", UserID, Pwd).
```

```
DEFINE QUERY qCust FOR sports.customer.
```

```
DEFINE DATA-SOURCE srcCust Select * from sports.customer.
```

```
DEFINE DATASET ds
```

```
  DATA-RELATION FOR ttCust, ttOrd
```

```
    RELEATION-FIELDS(ttCust.Cust-Num, ttOrd.Cust-Num). /* Could be derived from foreign constraints */
```

FULL
DISCLAIMER

Potential Advantages of OOABL Built-In Objects With Language Integration

- Flexible schema usage (no migrations or porting)
- Minimal data source management
 - Management, configuration and other supplementary objects possible
- Access standardized on D2C relational model and SQL compliance
- Underlying objects are extensible
 - Feature mapping possible (e.g., word indexes, sequences, etc.)
 - Feature integration possible (e.g., data links, bulk operations, etc.)
- Tight Language Integration
 - Transactions (commit/rollback), record blocks, error handling, etc.
 - Same code, different data source target is possible
 - Schema exposure (back and front)
 - Data federations and multi-source combinations
 - Language extensions for SQL possible

Who's interested?

PROGRESS EXCHANGE²⁰¹⁴

Visit the Resource Portal

- **Get session details & presentation downloads**
- **Complete a survey**
- **Access the latest Progress product literature**

www.progress.com/exchange2014



PROGRESS