Acumatica xRP Framework Fundamentals

Acumatica
The Cloud ERP

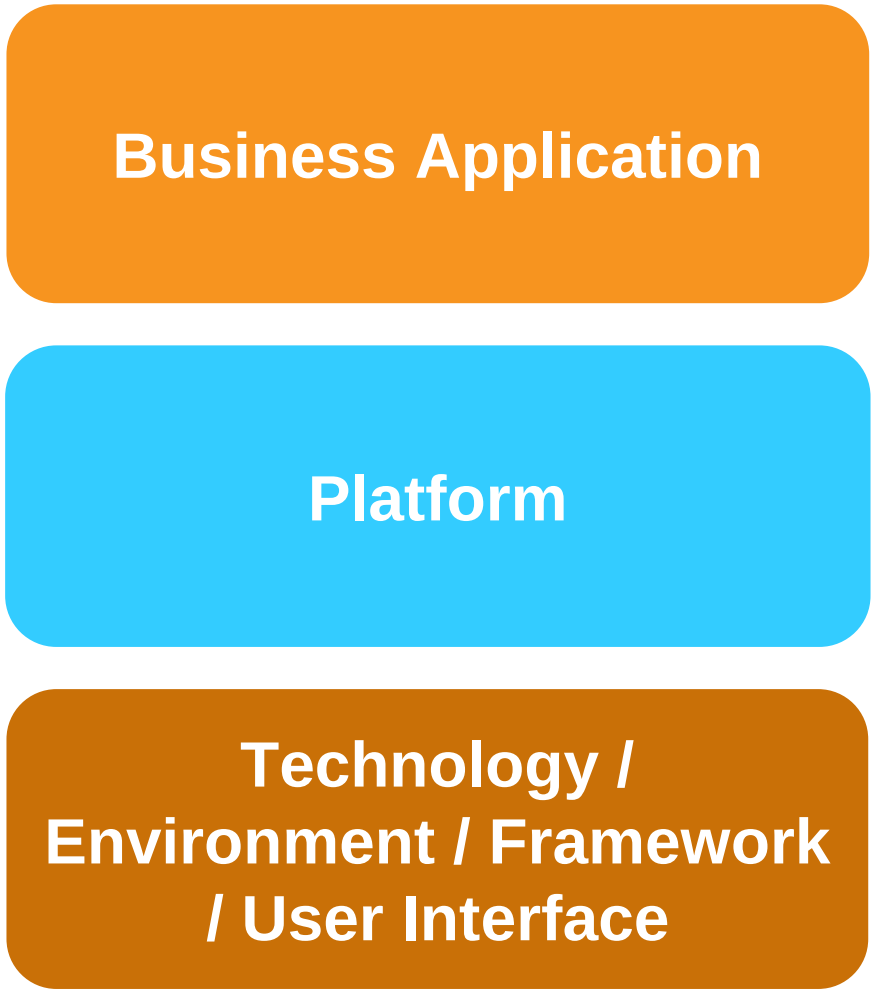
Nayan Mansinha
Lead – Developer Support
Acumatica

Agenda

- **Acumatica Platform Architecture**
- **Acumatica Framework Essentials**
  - Data Access Classes (DAC)
  - Business Logic Controllers (BLC or Graph)
  - Data Views
  - Business Query Language (BQL)
  - User Interface (ASP.NET)
  - Events
  - Customization (Extensions)
- **Next Steps!**

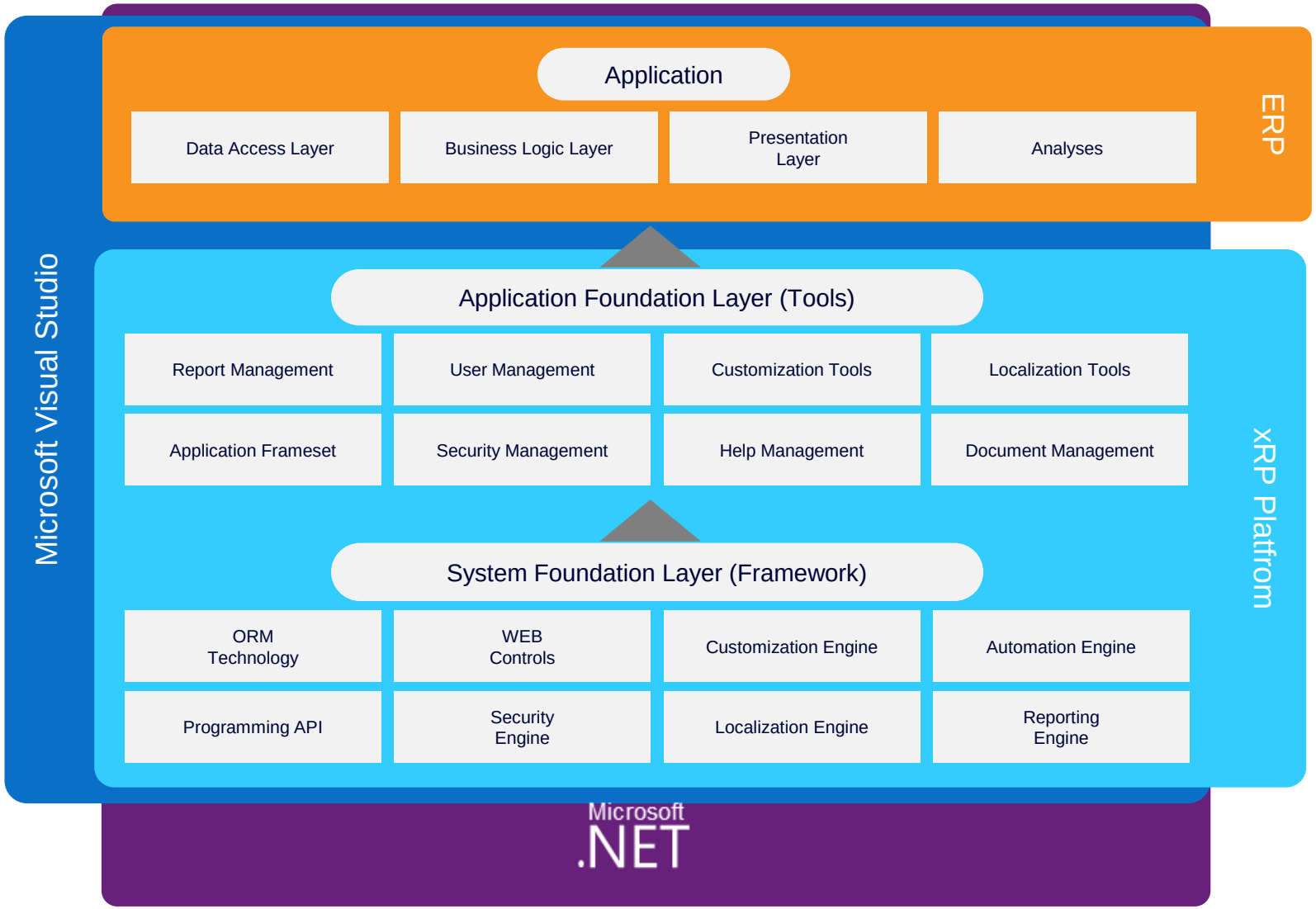
Acumatica Platform Architecture

xRP Platform



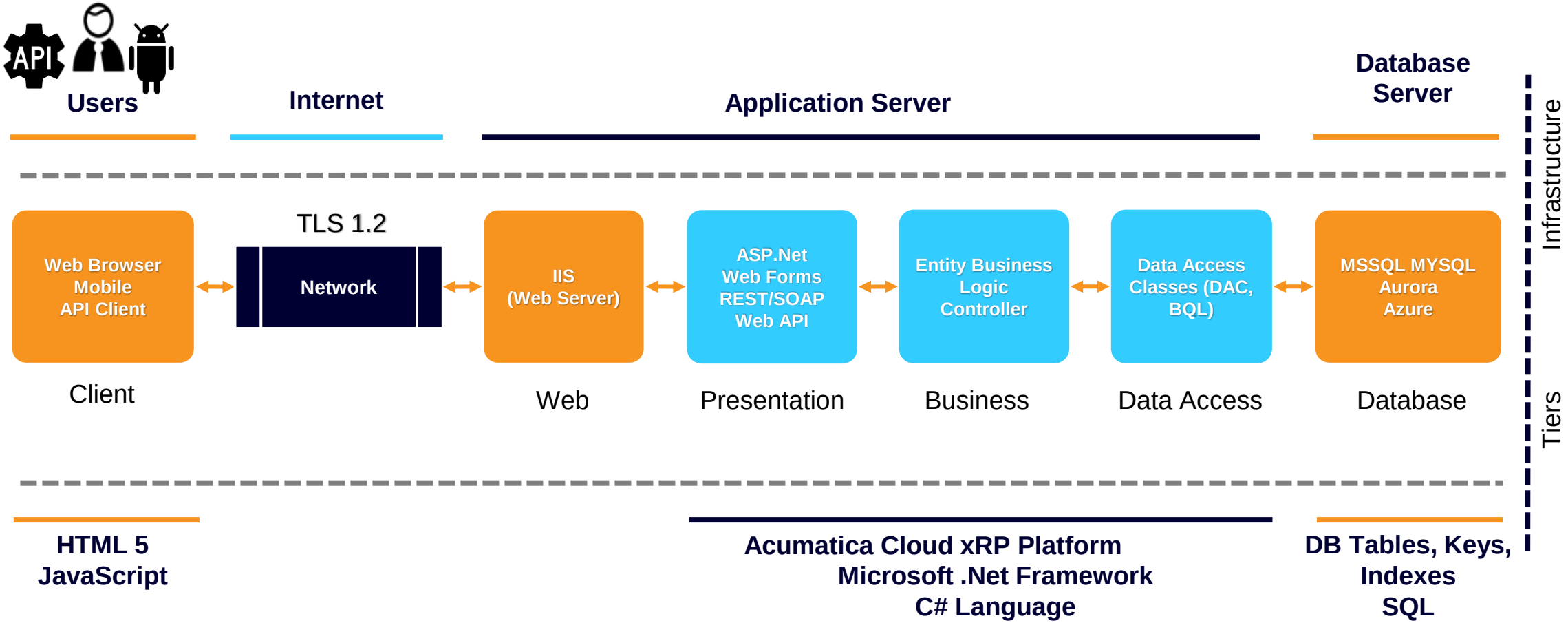
- **Code Standardization –** it's easier to support standardized code by any developer who knows it
- **Speedup Development –** with high level primitives and tools
- **Protect Form Technology Changes –** platform hides underlying technology and allows to change it without touching business logic code
- **Share Tools with ISVs and Partners –** involve other companies to build together

Acumatica Platform Architecture - Building Blocks

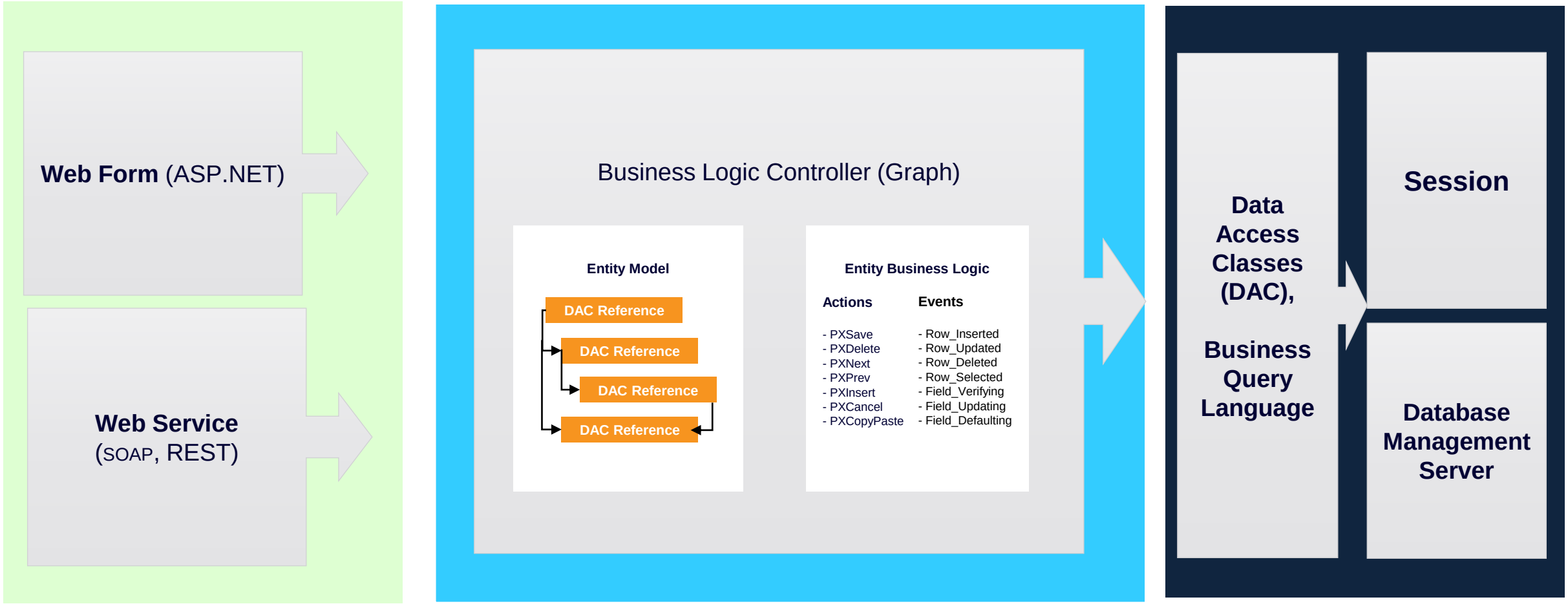


- **Single Runtime Environment** – Microsoft .NET
- **Single Development Environment** – Microsoft Visual Studio
- **System Foundation Layer** – Set of high-quality components and libraries used for the web application development
- **Application Foundation Layer** – Application frameset, development tools and set of pre-build web applications on top of a system foundation layer
- **Acumatica Application** – web application developed using the system foundation layer and integrated into the application foundation layer

Acumatica 6-tiers Web Architecture



Acumatica Framework Architecture



Presentation Layer

Business Logic Layer

Data Access Layer

Acumatica Platform Essentials

Acumatica Platform Essentials

- Data Access Classes (DAC)
- Business Logic Controllers (BLC or Graph)
- Data Views
- Business Query Language (BQL)
- User Interface (ASP.NET)
- Actions and Events
- Customizations (Extensions)

Data Access Classes (DAC)

DAC - Object-Relational Mapping

SQL

```
CREATE TABLE [dbo].[Product]
(
  //...
  [ProductCD] [nvarchar](15) NOT NULL,
  //...
  [Active] [bit] NOT NULL,
  //...
  CONSTRAINT [Product_PK]
    PRIMARY KEY CLUSTERED
  (
    [ProductCD] ASC
  )
)
```

DAC

```
public class Product : PX.Data.IBqlTable
{
    //...
    public abstract class productCD : PX.Data.BQL.BqlString.Field<productCD> {}
    [PXDBString(15, IsUnicode = true, IsKey = true)]
    [PXDefault]
    [PXUIField(DisplayName = "Product ID")]
    [PXSelector(typeof(Search<Product.productCD>),
                typeof(Product.productCD),
                typeof(Product.unitPrice))]
    public virtual string ProductCD { get; set; }

    //...
    public abstract class active : PX.Data.BQL.BqlBool.Field<active> { }
    [PXDBBool()]
    [PXDefault(true)]
    [PXUIField(DisplayName = "Active")]
    public virtual bool? Active { get; set; }
    //...
}
```

DAC - Definition

Data Access Class is a **type** that primarily represent **database table** in the application. A data access class consists of **data fields**.

Every DAC:

- **Mapped to table**

```
public class Product : PX.Data.IBqlTable
```

- **Has field properties**

```
public string ProductCD { get; set; }
```

- **Has field abstract classes**

```
public abstract class productCD : PX.Data.BQL.BqlString.Field<productCD> {}
```

Fields:

- **Has keys “IsKey = true”**

```
[PXDBString(15, IsUnicode = true, IsKey = true)]
```

- **Has attributes**

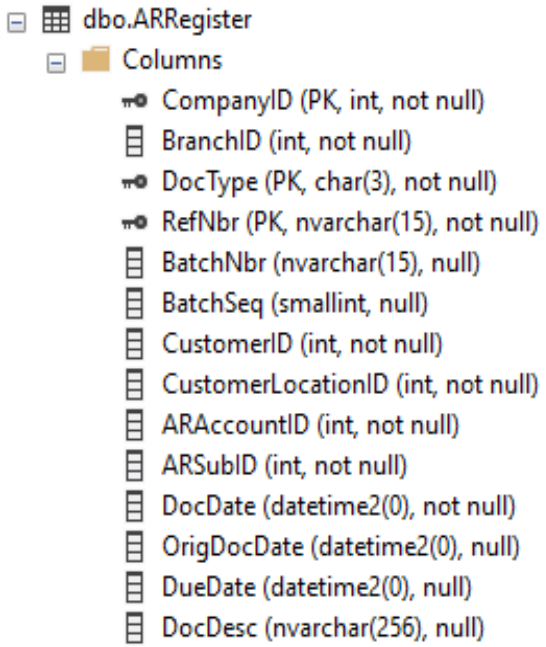
```
[PXDefault(true)]
[PXUIField(DisplayName = "Active")]
```

Attributes – Declarative Programming Model

Attribute	Meaning
[PXDBString(3, IsKey = true, InputMask="CCC")]	Database mapped field of type string
[PXBool()]	Virtual/unbound field of type bool
[PXDefault("Open")]	Provides default value for field
[PXDefault(PersistingCheck = PXPersistingCheck.NullOrBlank)]	Makes field required
[PXUIField(DisplayName = "Product ID", Enabled = true)]	Defines caption of the field in UI
[PXUIField(Enabled = false)]	Makes field disabled in UI
[PXSelector(typeof(Search<Product.productCD>))]	Defines foreign key reference and selector
[PXStringList(new string[] { "H", "C" }, new string[] { "On Hold", "Canceled" })]	Defines list of predefined values (drop-down)
[PXFormula(typeof(Mult<Transaction.qty, Transaction.unitPrice>))]	Apply calculated value to the field automatically.

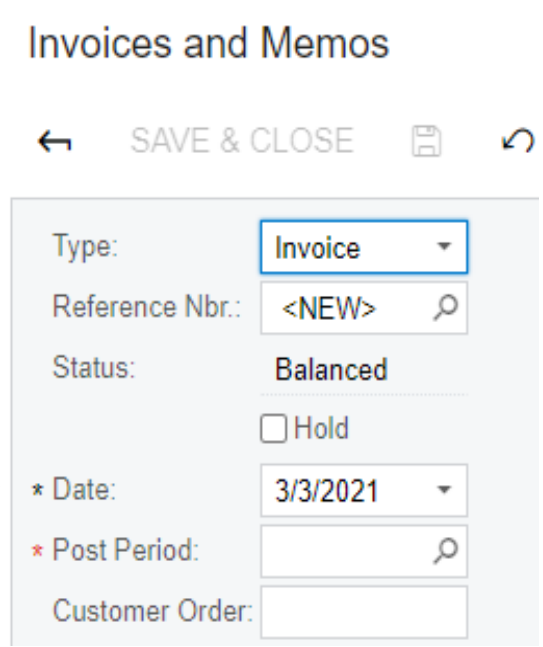
Attributes – Declarative Programming (cont.)

Database table (ARRegister) ⟷ **Data Access Class (ARRegister)** ⟷ **Acumatica Screen (Invoices and Memos)**




```
public partial class ARRegister:
   IBqlTable
{
   public abstract class docDate :
      BqlDateTime.Field<docDate> { }

   [PXDBDate]
   [PXUIField(DisplayName = "Date"]
   [PXDefault(typeof(
      AccessInfo.businessDate))]
   public virtual DateTime? DocDate
   ...
```



Business Logic Controller (PXGraph)

Business Logic Controller (BLC or Graph)

- **Controller encapsulate Business Logic and Data Views**
- **There is 1:1 correspondence between graph and a page**
- **Graph controls Business logic by virtue of:**
  - **Actions (Buttons)**
  - **Events**

```
public class ProductMaint : PXGraph<ProductMaint, Product>
{
    public PXSelect<Product> Products;
}
```

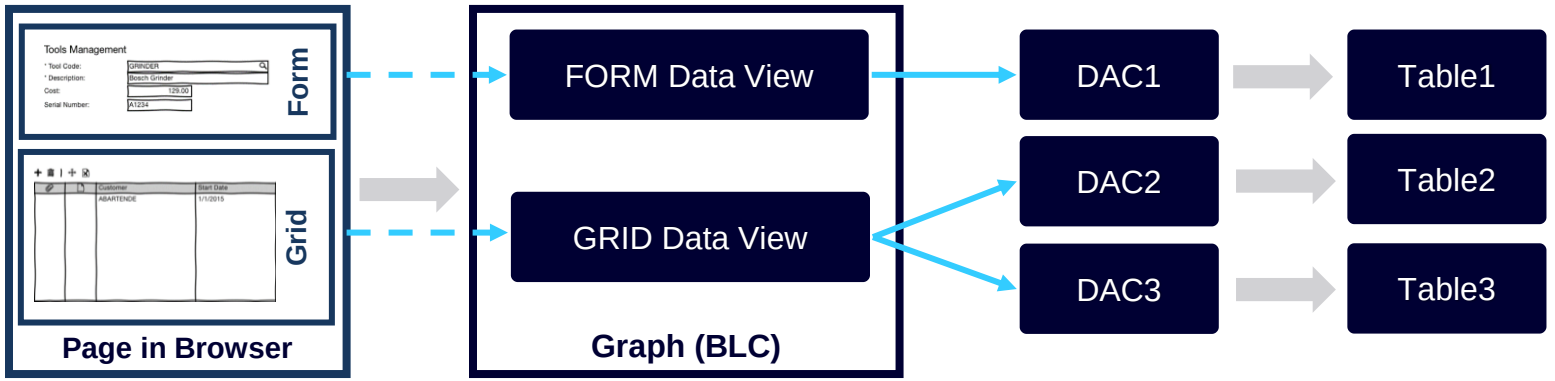
Data Views

Data Views

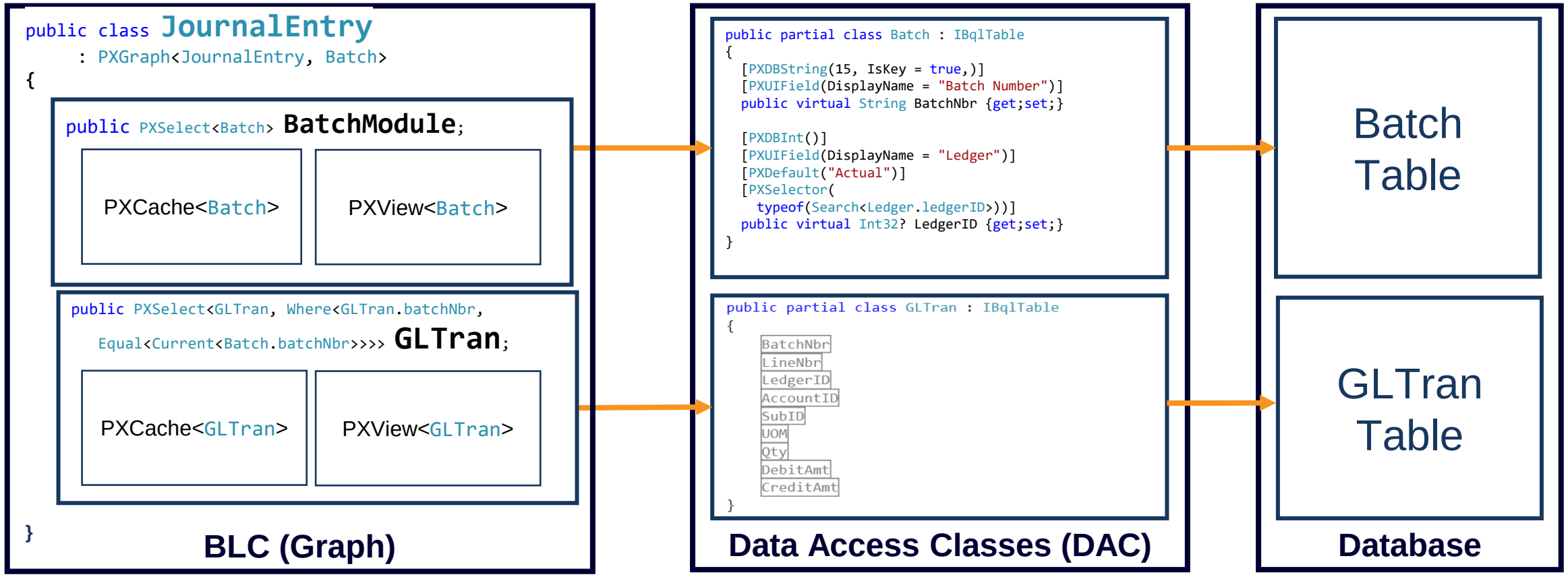
- Maintain changes in the Session with PXCache<DAC>
- Selects data from the Database with PXView<DAC>
- Provides data for User Interface

Examples

```
public PXSelect<Product> Products;
public PXSelect<DocTran, InnerJoin<Product, On<...>>> Trans;
```



Data Access Level Architecture



Business Query Language (BQL)

Business Query Language (BQL)

Data View –

```
PXSelect<Product, Where<Product.active, Equals<True>>> Products;
```

BQL Query –

```
typeof(Select<Product, Where<Product.active, Equals<True>>>)
```

Converted

T-SQL –

```
select * from Product
insert Product ( ...fields... ) values (...values....)
update Product set ...fields... where ...conditions...
delete from Product where ...conditions...
```

- **Is a part of Acumatica Data Access Layer**
- **Is mapped to SQL queries**
- **Hides the underlying database engine**
- **Is checked at compile time**
- **Comes with a variety of clauses, allowing to express most DB queries**

Business Query Language (BQL cont.)

A simple query can look like:

```
public PXSelect<Product> Products;
```

Result SQL:

```
SELECT Product.ProductCD, Product.Active, ...
FROM Product Product
ORDER BY Product.ProductCD
```

Business Query Language (BQL cont.)

```
public PXSelectJoin<SupplierProduct,
    InnerJoin<Supplier,
    On<Supplier.supplierID, Equal<SupplierProduct.supplierID>>>,
    Where2<
        Where<Current<SupplierFilter.countryCD>, IsNull,
    Or<Supplier.countryCD, Equal<Current<SupplierFilter.countryCD>>>>,
        And<Where<Current<SupplierFilter.minOrderQty>, IsNull,
            Or<SupplierProduct.minOrderQty,
                GreaterEqual<Current<SupplierFilter.minOrderQty>>>>>>,
    OrderBy<Asc<SupplierProduct.productID,
        Asc<SupplierProduct.supplierPrice,
        Desc<SupplierProduct.lastPurchaseDate>>>>> Products;
```

SELECT … FROM …

JOIN … ON …

WHERE (… OR …) AND (… OR …)

(…)

… > "VALUE"

ORDER BY … ASC, … DESC

Fluent - Business Query Language (BQL cont.)

```
SelectFrom<Detail>.
InnerJoin<Document>.
    On<Detail.docType.IsEqual<Document.docType>.
    And<Detail.docNbr.IsEqual<Document.docNbr>>>.
Where<Document.bAccountID.IsEqual<@P.AsInt>.
      And<AccessInfo.businessDate.FromCurrent.
      Diff<Document.date>.Days.IsGreater<TwentyEight>>>.
AggregateTo<GroupBy<Detail.itemID>, Sum<Detail.amount>>.
      Having<Detail.amount.Summarized.IsGreater<Zero>>.
OrderBy<Detail.itemID.Asc, Document.date.Desc>
```

Less brackets in declaration

Type validations on comparisons

Easier access to functions and parameters

Having clause

https://www.acumatica.com/blog/fluent-bql-increase-the-readability-maintainability-of-your-queries/

BQL – Querying Data

Passing Parameters from the code – Required:

```
foreach(Product record in PXSelect<Product,
                        Where<Product.isActive,
Equal<Required<Product.isActive>>>>>
                                .Select(this, true));
```

Parameter value from context – Current:

```
Product record = PXSelect<Product,
                    Where<Product.productID,
                      Equal<Current<Tran.productID>>>>>.Select(this));
```

BQL - LINQ 2 BQL

Inline mode:

```
var results = graph
    .Select<CRCase>()
    .FirstOrDefault(c
     => c.CaseCD == “000123”);
```

```
SELECT TOP 1 *
FROM CRCase
WHERE CaseCD = ‘000123’
```

Parameter mode:

```
var results = graph
    .Select<CRCase>()
.FirstOrDefault(c
     => c.CaseCD ==“000123”.AsParam());
```

```
DECLARE @P0 NVARCHAR(6)
DECLARE @P0 = ‘000123’
SELECT TOP 1 *
FROM CRCase
WHERE CaseCD = @P0
```

Acuminator – Your Magic Wand

- Validation
- Formatting
- Colorizing

```
public PXSelectJoin
    BCSyncStatus,
    InnerJoin<BCEntity,
        On<BCSyncStatus.connectorType, Equal<BCEntity.connectorType>,
        And<BCSyncStatus.bindingID, Equal<BCEntity.bindingID>,
        And<BCSyncStatus.entityType, Equal<BCEntity.entityType>>>>>,
    Where<BCSyncStatus.connectorType, Equal<Current<BCEntity.connectorType>>,
        And<BCSyncStatus.bindingID, Equal<Current<BCEntity.bindingID>>,
        And<BCSyncStatus.entityType, Equal<Current<BCEntity.entityType>>,
        And<BCSyncStatus.pendingSync, Equal<True>,
        And<BCSyncStatus.deleted, NotEqual<True>,
        And<BCSyncStatus.lastOperation, NotEqual<BCSyncOperationAttribute.skipped>,
        And2<
            Where<BCEntity.maxAttemptCount, IsNull,
                Or<BCSyncStatus.attemptCount, Less<BCEntity.maxAttemptCount>>>,
            And<Where<BCSyncStatus.lastOperation, Equal<BCSyncOperationAttribute.externChanged>,
                Or<BCSyncStatus.lastOperation, Equal<BCSyncOperationAttribute.localChanged>,
                Or<BCSyncStatus.lastOperation, Equal<BCSyncOperationAttribute.forcedToResync>,
                Or<BCSyncStatus.lastErrorMessage, IsNotNull>>>>
        >>>>>>>>,
    OrderBy<
        Asc<BCSyncStatus.sortOrder,
        Asc<BCSyncStatus.lastOperationTS>>>
    StatusSelectPendingTemplate;
```

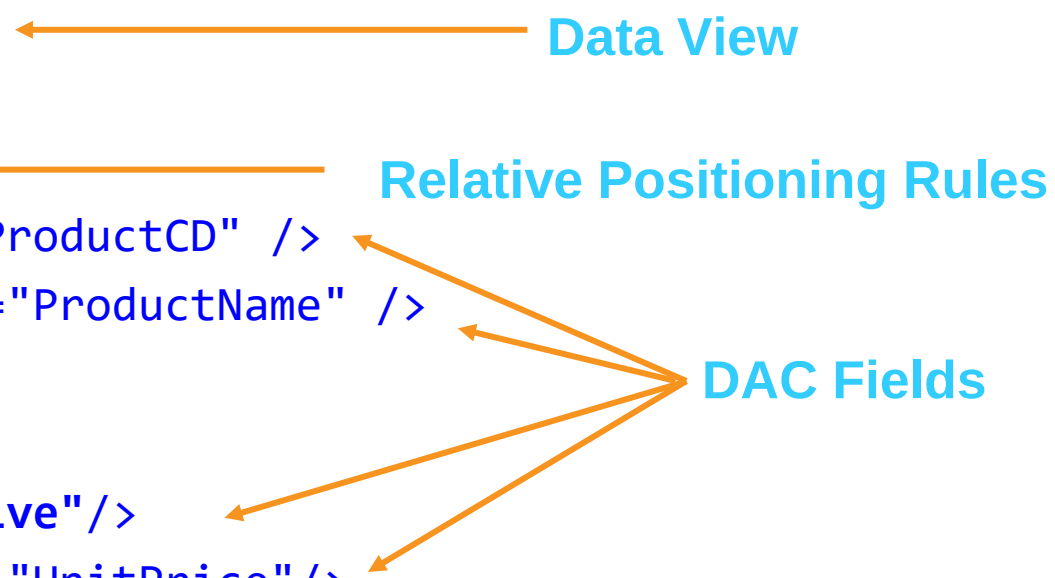
User Interface (ASP.NET)

User Interface - ASPX

User Interface in Acumatica is done using ASPX custom controls

```
<px:PXFormView ID="form" DataMember="Products">
<Template>
     <px:PXLayoutRule StartRow="True"/>
     <px:PXSelector ID="edProductCD" DataField="ProductCD" />
     <px:PXTextEdit ID="edProductName" DataField="ProductName" />

     <px:PXLayoutRule StartColumn="True"/>
     <px:PXCheckBox ID="edActive" DataField="Active"/>
     <px:PXNumberEdit ID="edUnitPrice" DataField="UnitPrice"/>
</Template>
</px:PXFormView>
```



Attributes defines UI Controls

Control	Attributes on the DAC Field	ASPX Definition
Text box	[PX(DB)String]	<px:PXTextEdit ID= ... />
Number edit box	[PX(DB)Int] or [PX(DB)Decimal]	<px:PXNumberEdit ID=... />
Mask edit box	[PX(DB)String(InputMask =...)]	<px:PXMaskEdit ID=... />
Drop-down list	[PXStringList] or [PXIntList]	<px:PXDropDown ID=... / >
Selector	[PXSelector]	<px:PXSelector ID=... />
Check box	[PX(DB)Bool]	<px:PXCheckBox ID=... />
Date-time picker	[PX(DB)Date]	<px:PXDateTimeEdit ID=... />
Time span edit box	[PXDBTimeSpan]	<px:PXDateTimeEdit ID=... TimeMode="True" />

Commit Changes

Normally server doesn’t get notified of every change to data made by user
You can make certain controls post changes to server once they occur:

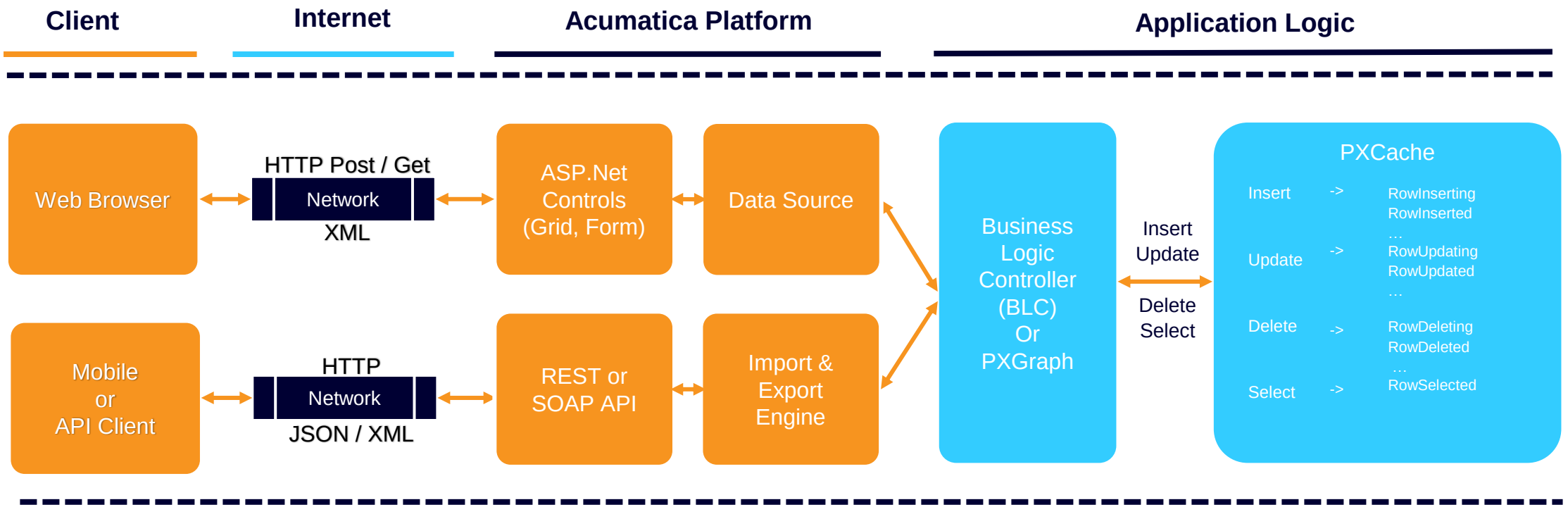
```
<px:PXFormView ID="form" ... >
     <Template>
          <px:PXCheckBox ID="chkHold"
                         runat="server"
                         DataField="Hold"
                         CommitChanges="True" />
```

The same for grid columns:

```
<px:PXGridLevel ... >
     <RowTemplate>
          <px:PXGridColumn DataField="ProductID"
                           CommitChanges="True" />
```

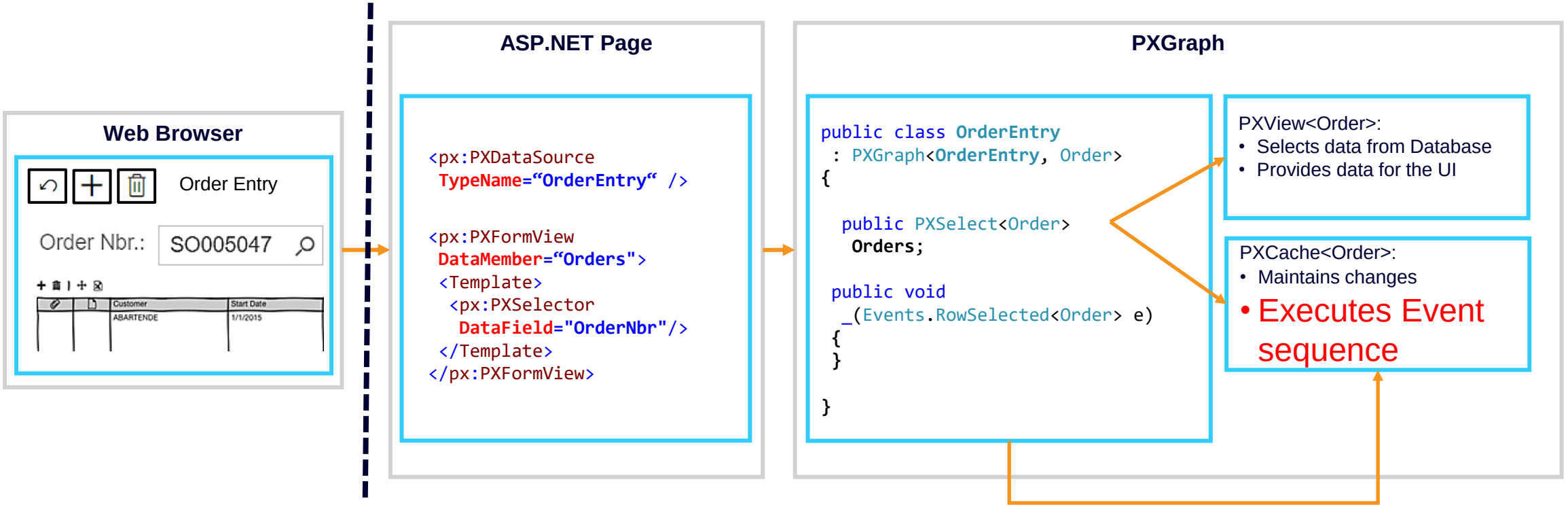
Event Model

Where do Events come from?



Where do Events come from? (cont.)

Example:



Event Declaration

Classic:

```
public virtual void DAC_Field_FieldUpdated(PXCache cache, PXFieldUpdatedEventArgs e) {}
public virtual void DAC_RowInserting(PXCache cache, PXRowInsertingEventArgs e) {}
```

Generic:

```
public virtual void _(Events.FieldUpdated<DAC, DAC.field> e) {}
public virtual void _(Events.RowInserting<DAC> e) {}
```

Override event approach with 2 parameters

```
protected void _(Events.RowInserting<Batch> e, PXRowInserting baseHandler) {}
```

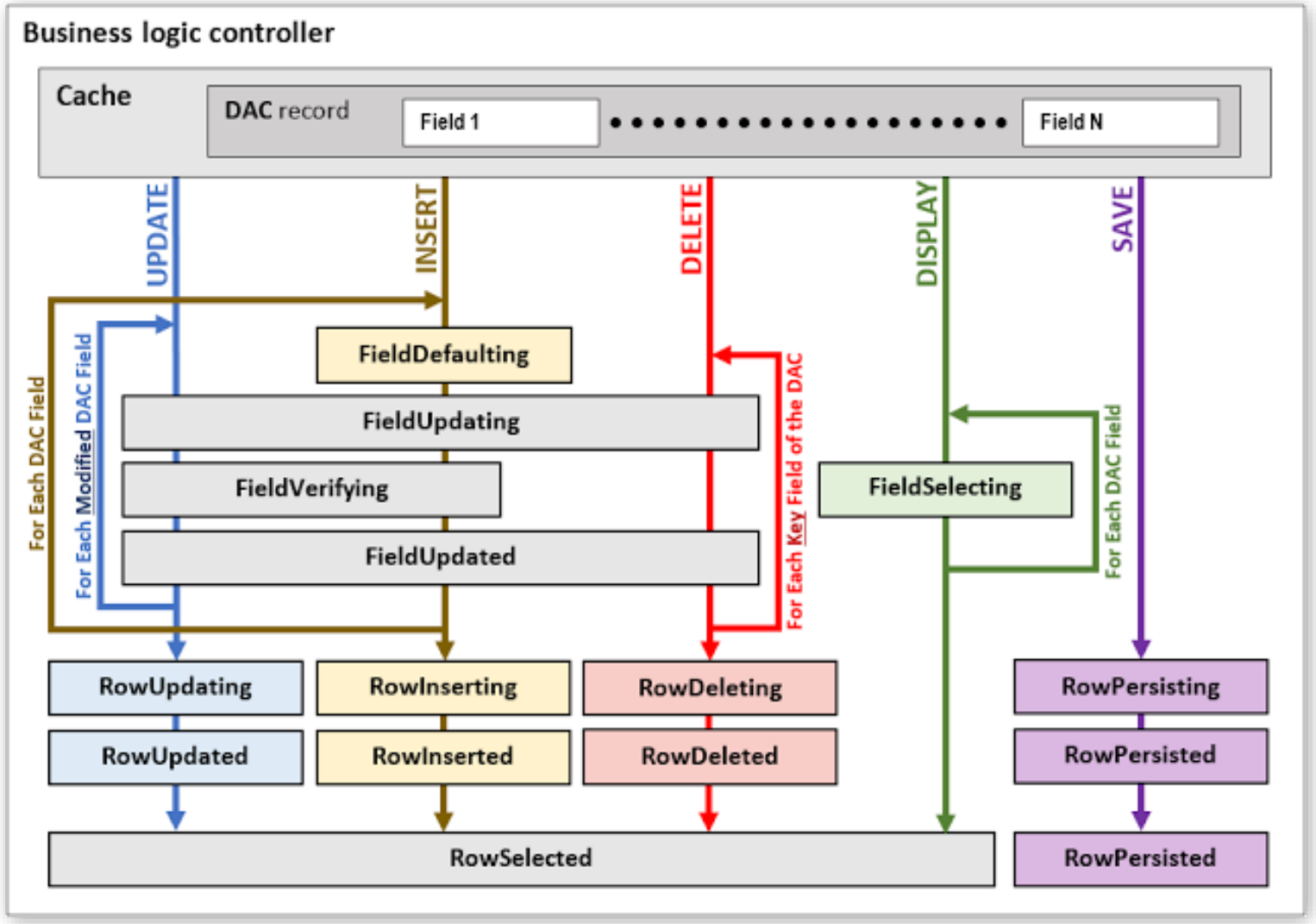
Attributes:

```
public class PXAwesomeAttribute: …, IPXFieldUpdatedSubscriber, IPXRowInsertingSubscriber,
{
    public virtual void FieldUpdated(PXCache cache, PXFieldUpdatedEventArgs e) {}
    public virtual void RowInserting(PXCache cache, PXRowInsertingEventArgs e) {}
}
```

Understanding The Event Model

Acumatica Events

- 12 Row-level events
- 5 Field-level events



Commonly Used Events

RowInserting	occurs before new data record is inserted into PXCache
RowInserted	occurs after a new data record has been successfully into PXCache
RowUpdating	occurs before updating of cached record
RowUpdated	occurs after updating of cached record
RowDeleting	occurs for a data record that is being deleted from the PXCache but can still be reverted to previous state
RowDeleted	occurs for a data record that is being deleted from the PXCache and status is marked for deletion
RowSelecting	occurs when system reads record from database reader
RowSelected	occurs when system should show record to the UI
FieldDefaulting	occurs when system calculates default value for the field.
FieldVerifiyng	occurs when need to validate input value.
FieldUpdated	occurs when the field value is changed

Customization

Customization Capabilities

User Interface

- Add / Remove / Rearrange controls – Customization Project Editor

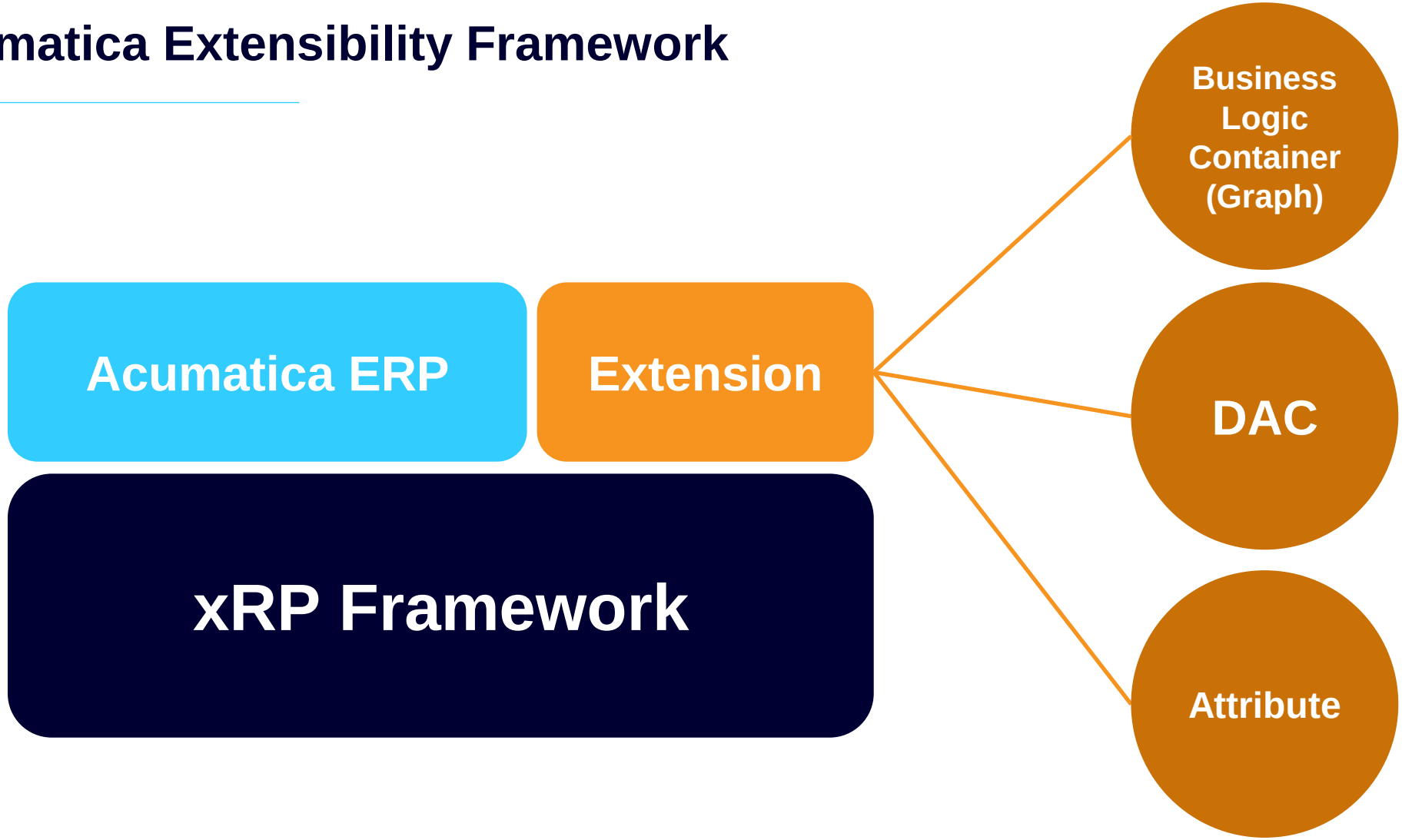
Business Logic

- Business Logic Containers (Graphs) – PXGraphExtension
  - DataMembers
  - Events
  - Actions
- Data Access Classes (DAC) – PXCacheExtensions
  - Attributes
  - Events

Database

- Add Columns and Tables – Customization Project Editor

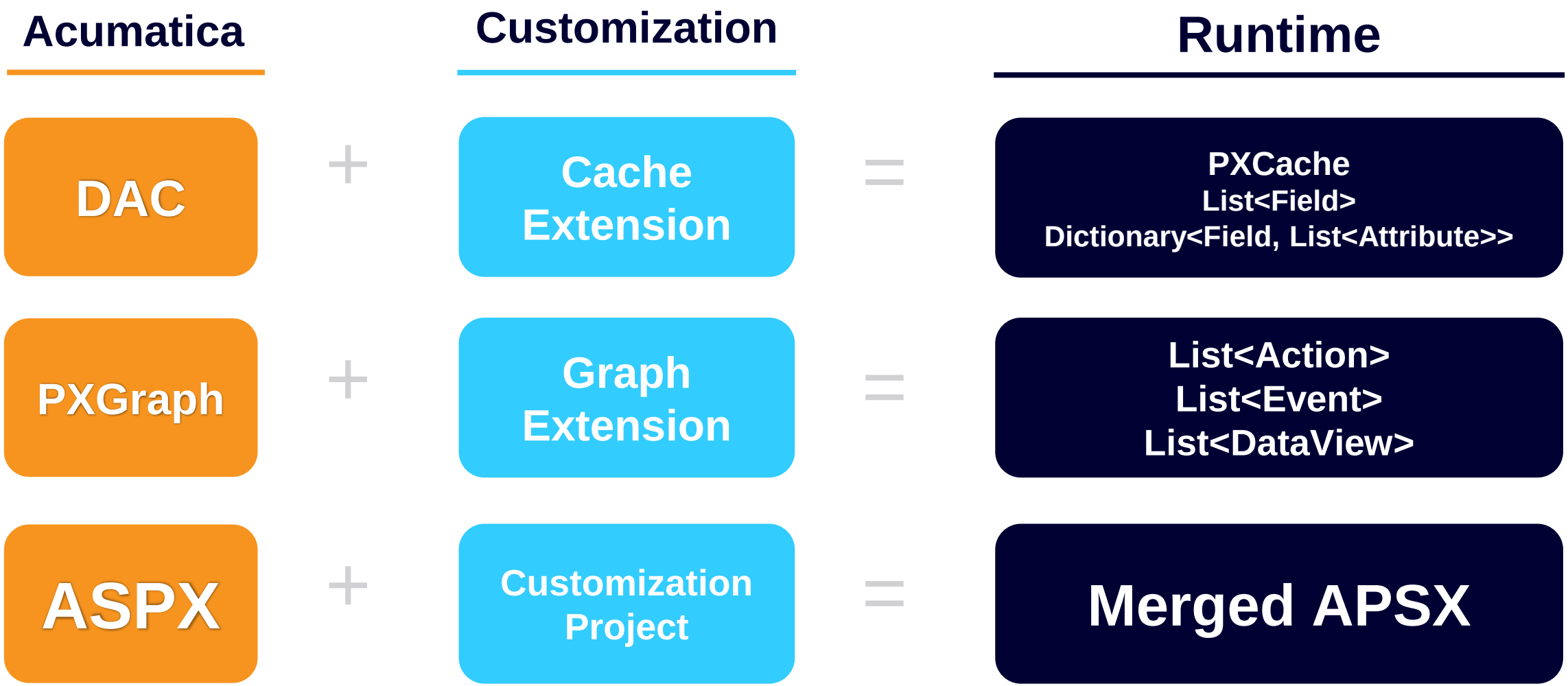
Acumatica Extensibility Framework



Acumatica Extensibility Framework

```
public class DACExtension : PXCacheExtension<BaseDAC>
{
     //Put new fields definition here
     //Customize existing attributes and fields
}
public class BLCExtension : PXGraphExtension<BaseBLC>
{
     //Put new event handlers, actions, data views or methods here
     //Customize existing logic with defining new one with the same name
}
```

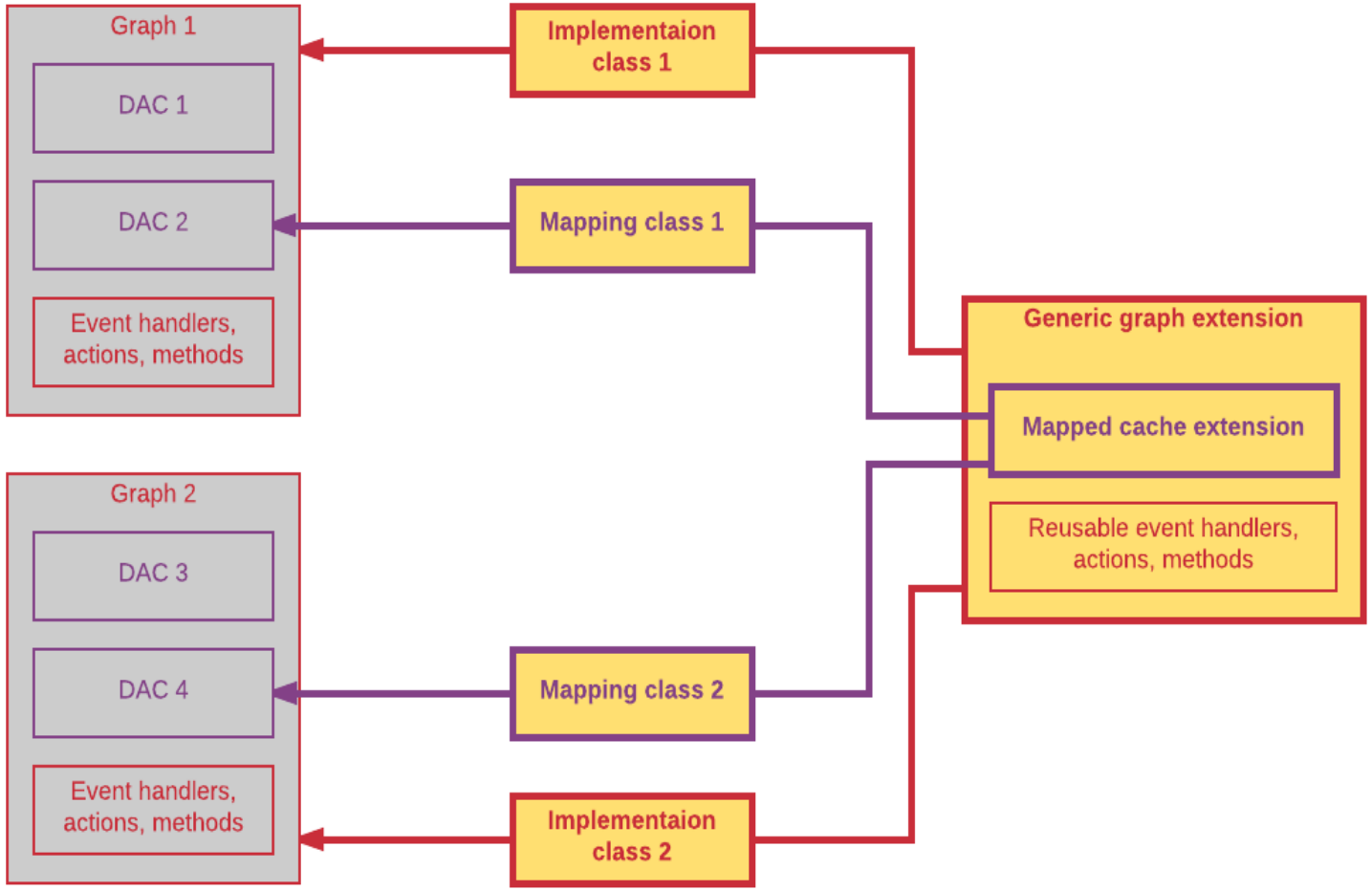
Acumatica Extensibility Framework



Reusable Business Objects

Examples in ERP

- Discount
- MultiCurrency
- SalesTax
- SalesPrice
- Contact
- Address
- Barcode scanning engine(WMS)



Access to Protected members in Graph Extensions

Ability to call Protected Members of Graph in Graph Extensions

```
[PXProtectedAccess]
public abstract class SOOrderEntryPXExt : PXGraphExtension<SOOrderEntry>
{
    [PXProtectedAccess]
    protected abstract void SetReadOnly(bool isReadOnly);

    protected virtual void _(Events.RowSelected<SOLine> e, PXRowSelected BaseInvoke)
    {
        SetReadOnly(false);

        if (BaseInvoke != null) { BaseInvoke(e.Cache, e.Args); }
    }
}
```

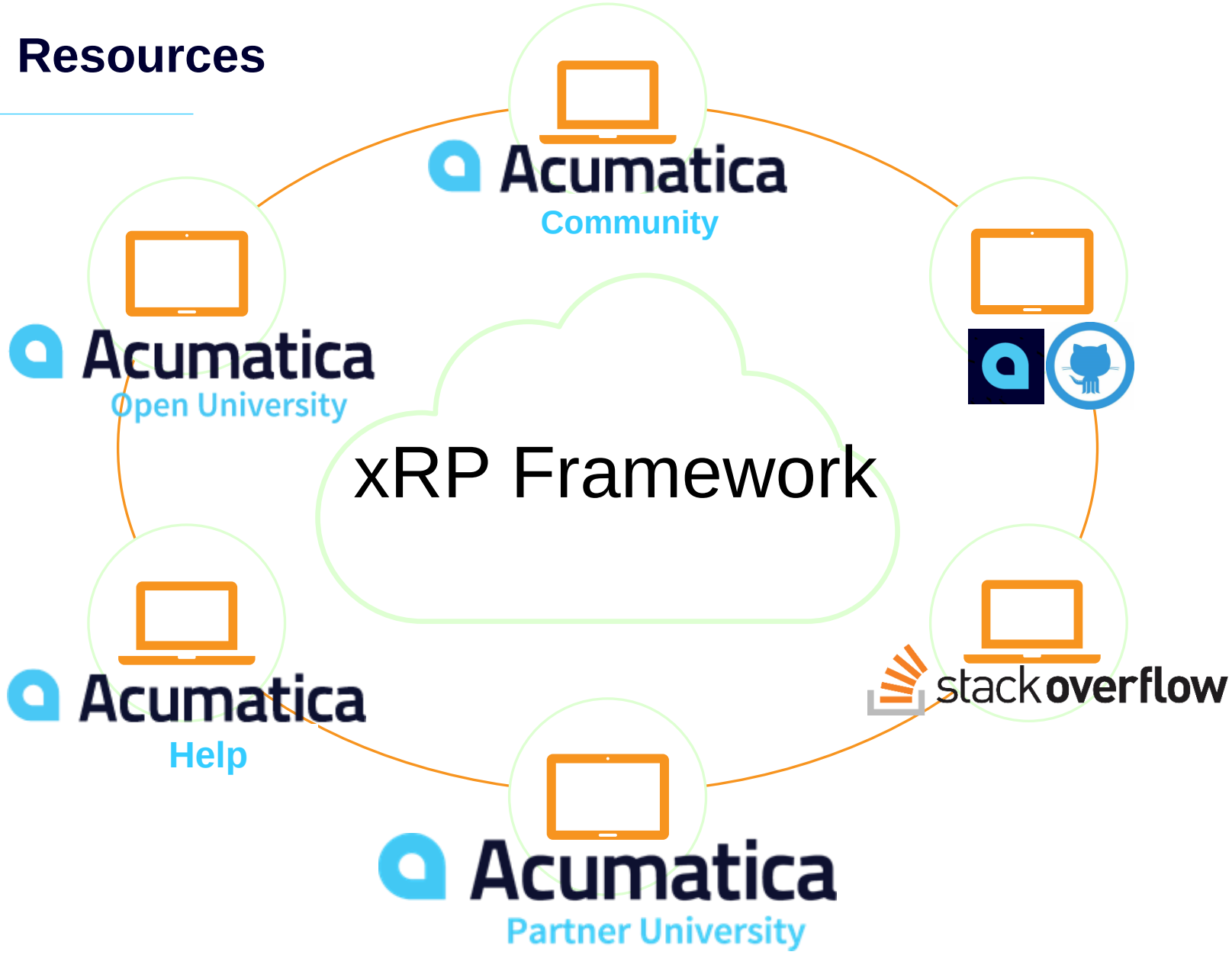
Webhooks

Support for setting up Acumatica as a listener via Webhooks

```
public class MyWebhookHandler : IWebhookHandler
{
    public async Task<System.Web.Http.IHttpActionResult> ProcessRequestAsync(
        HttpRequestMessage request, CancellationToken cancellationToken)
    {
      …
    }
}
```

Next Steps!

Learning Resources



Resources

Discussions, Best Practices, and Product Feedback | Community (acumatica.com)

Acumatica · GitHub

Acumatica Open University

Stack Overflow - Where Developers Learn, Share, & Build Careers

Acumatica - Help

Home (litmos.com) Partner University

Acumatica Developers Blog - Acumatica Platform and ERP

Developers Archives - Acumatica Cloud ERP

Questions?

Acumatica
The Cloud ERP

Thank You!

www.acumatica.com/developers