xRP Framework Fundamentals & Best Practices

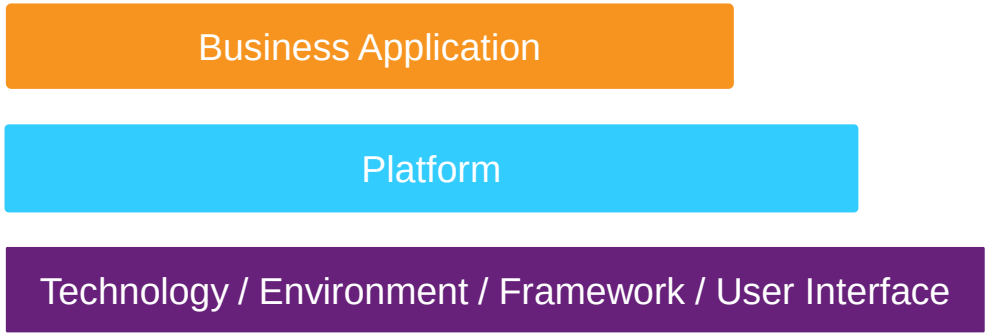
Dhiren Chhapgar
Principal Software Engineer
Acumatica

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
  - Customizations (Extensions)
- **What Next?**

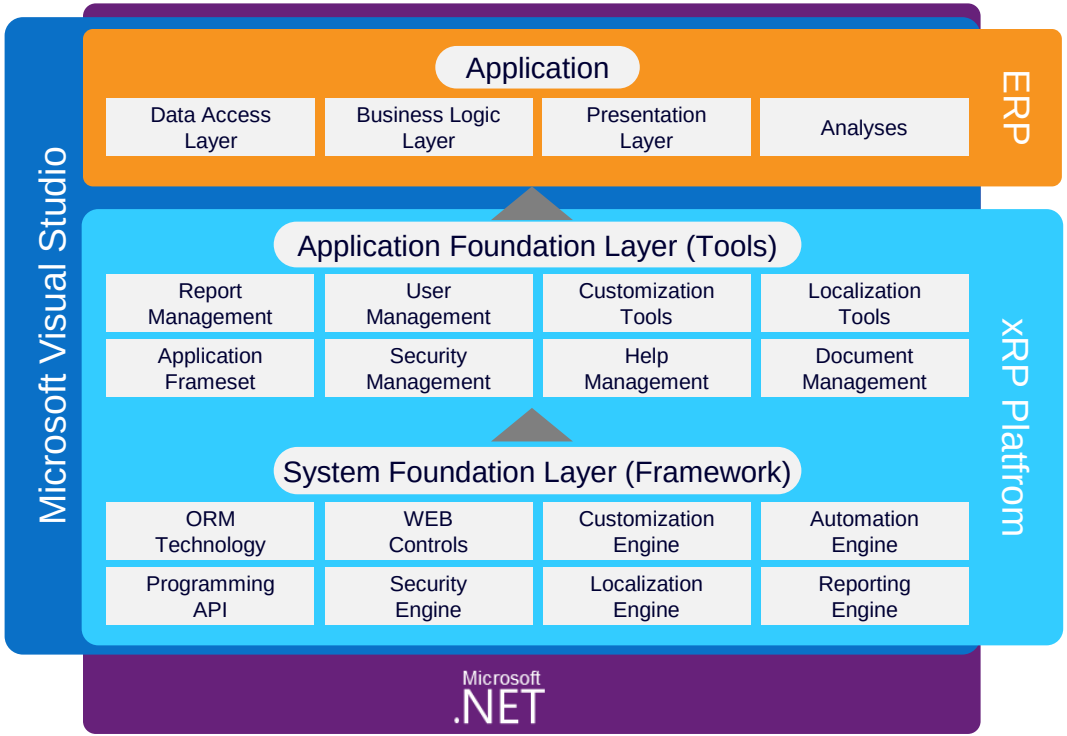
Acumatica Platform Architecture

xRP Platform



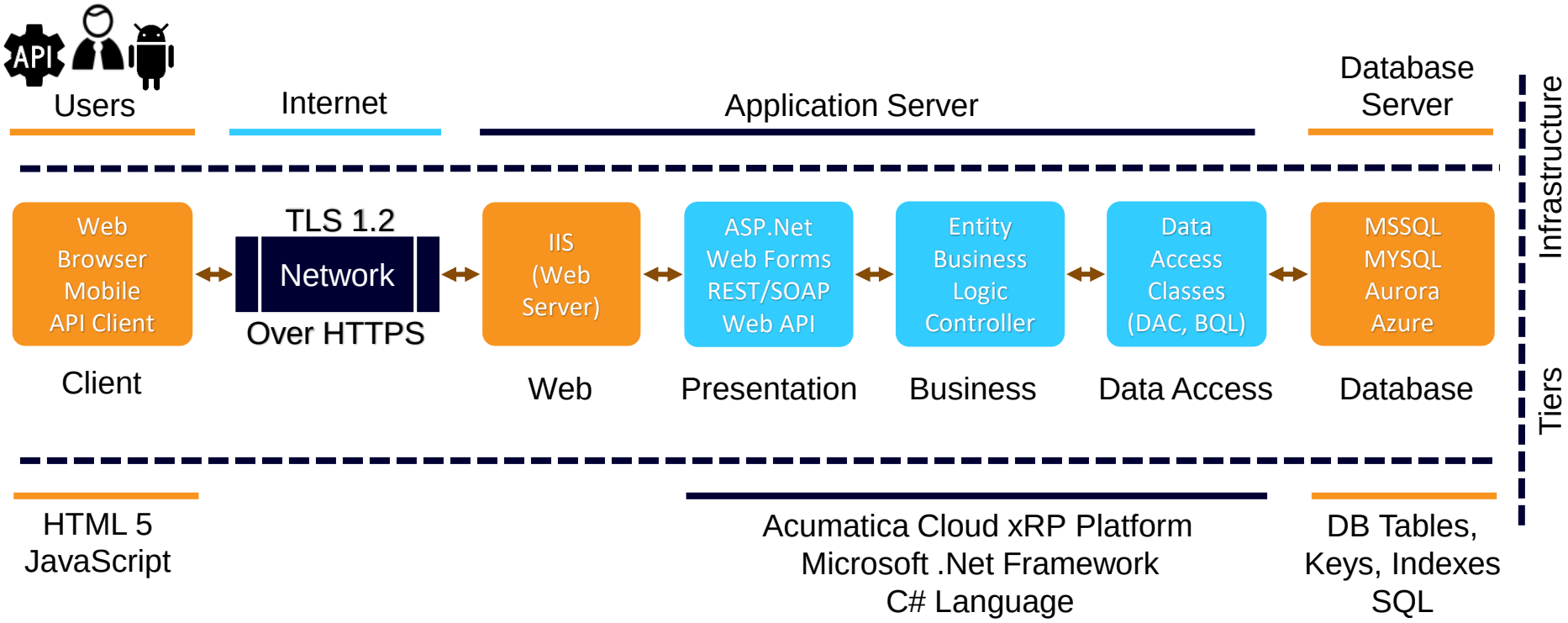
- **Code Standardization –** it is easier to support standardized code by any developer who knows it
- **Speedup Development –** with high level primitives and tools
- **Protect form Technology changes –** platform hides underlaying technology and allows to change it without touching business logic code.
- **Share Tools with Partners –** involve all other companies to build new world together

Acumatica Platform Architecture

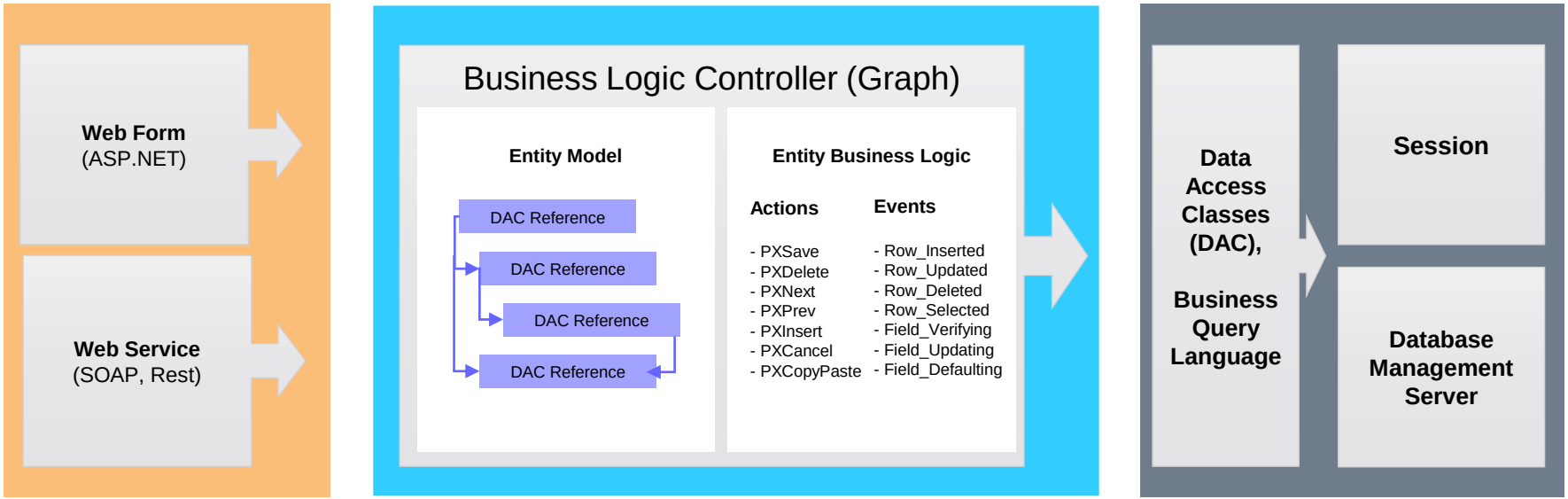


- **Single Runtime Environment** – Microsoft .NET
- **Single Development Environment** – Microsoft Visual Studio.
- **System Foundation Layer** – Set of high-quality components and libraries used for the web application development.
- **Application Foundation Layer** – Application frameset, development 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**

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)
- 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.IBqlField {}
        [PXDBString(15, IsUnicode = true, IsKey = true)]
        [PXDefault]
        [PXUIField(DisplayName = "Product ID")]
        [PXSelector(
                typeof(Search<Product.productCD>),
                typeof(Product.productCD),
                typeof(Product.unitPrice))]
        public string ProductCD { get; set; }
        //...
        public abstract class active : PX.Data.IBqlField { }
        [PXDBBool()]
        [PXDefault(true)]
        [PXUIField(DisplayName = "Active")]
        public bool? Active { get; set; }
        //...
}
```

DAC - Definition

Every DAC:

- [Serializable]
- Mapped to table
- Has field properties
- Has field abstract classes

Fields:

- Have keys “IsKey = true”
- Have attributes

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

public string ProductCD { get; set; }

public abstract class productCD : PX.Data.IBqlField {}
```

```
[PXDBString(15, IsUnicode = true, IsKey = true)]
[PXDefault(true)]
[PXUIField(DisplayName = "Active")]
```

Attributes – Declarative Programming

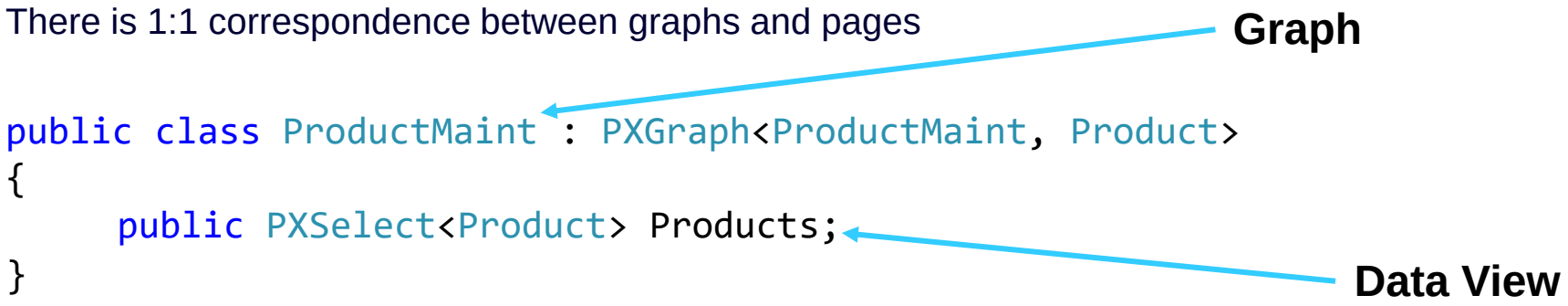
Attribute	Meaning
[PXDBString(3, IsKey = true, InputMask="CCC")]	Database mapped field of type string
[PXBool()]	Virtual field of type bool
[PXDefault("Open")]	Provides default value for field
[PXDefault(PersistingCheck = PXPersistingCheck.NullOrBlank)]	Makes field required
[PXUIField(DisplayName = "Product ID", Enabled = true)]	Defines name 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>))]	Brings auto-calculation rules that will be executed automatically.

Business Logic Controllers (PXGraph)

Business Logic Controllers (BLC or Graph)

Controllers encapsulating Business Logic and Data Views

There is 1:1 correspondence between graphs and pages



Graphs controls Business logic with:

- Actions (Buttons)
- Events

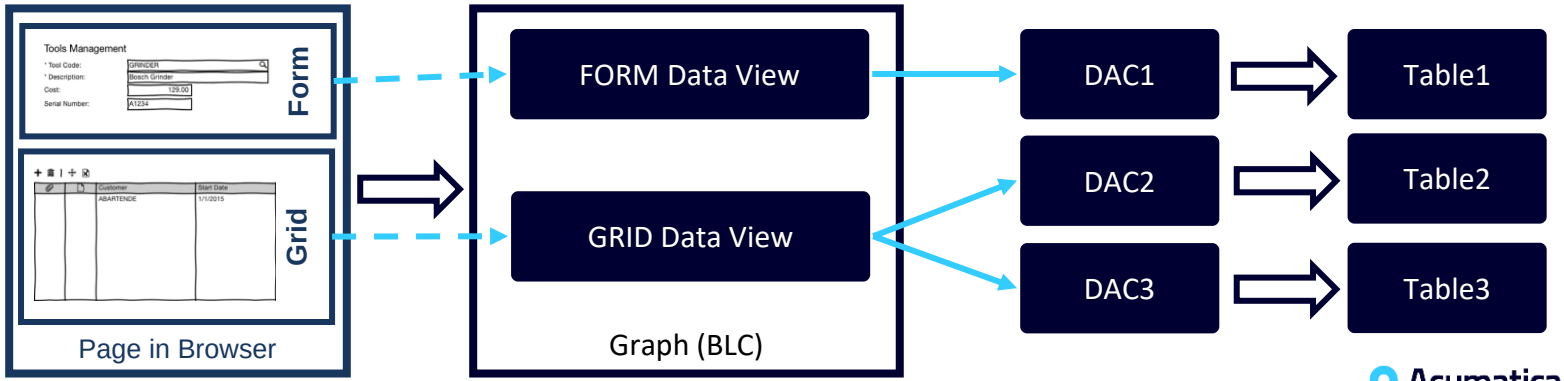
Data Views

Data Views

Data Views:

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

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

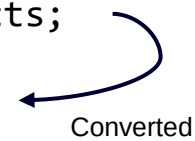


Business Query Language (BQL)

Business Query Language (BQL)

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

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



- 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)

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)

```
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

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
```

BQL – Fluent BQL

```
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

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 controls by ASPX files and LayoutRules

```
<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>
```

Data View

Relative Positioning Rules

DAC Field

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:

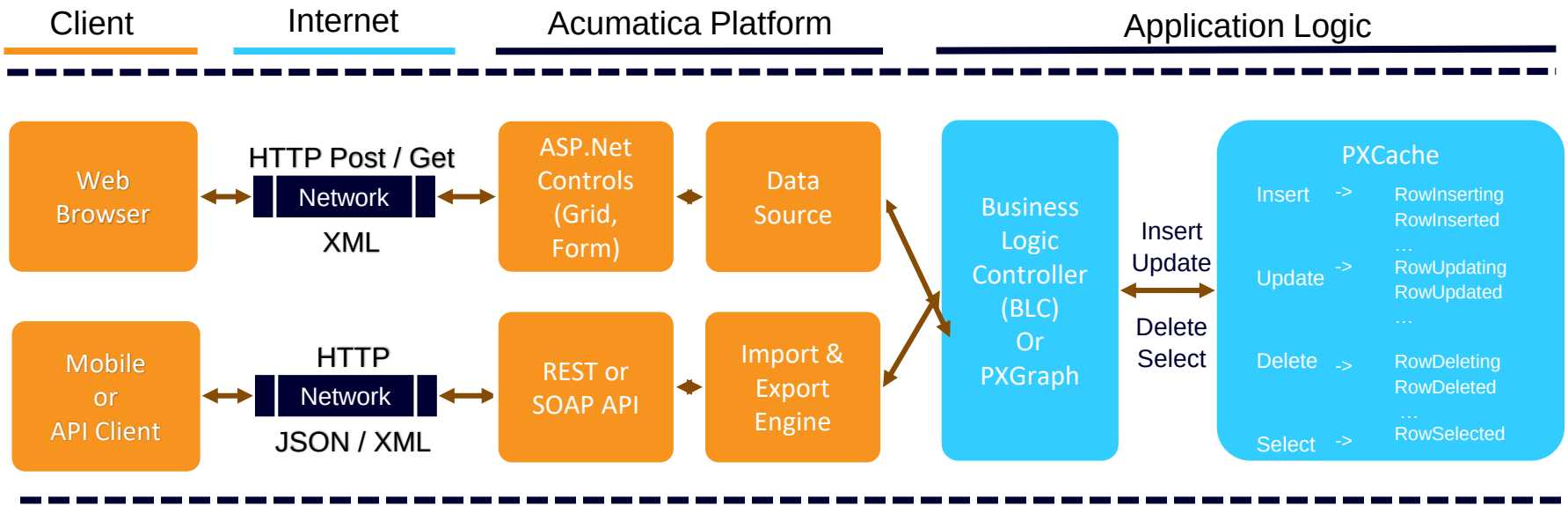
```
<px:PXFormView ID="form" ... >
      <Template>
            <px:PXCheckBox ID="chkHold"
                           runat="server"
                           DataField="Hold"
                           CommitChanges="True" />
```

The same for grid columns:

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

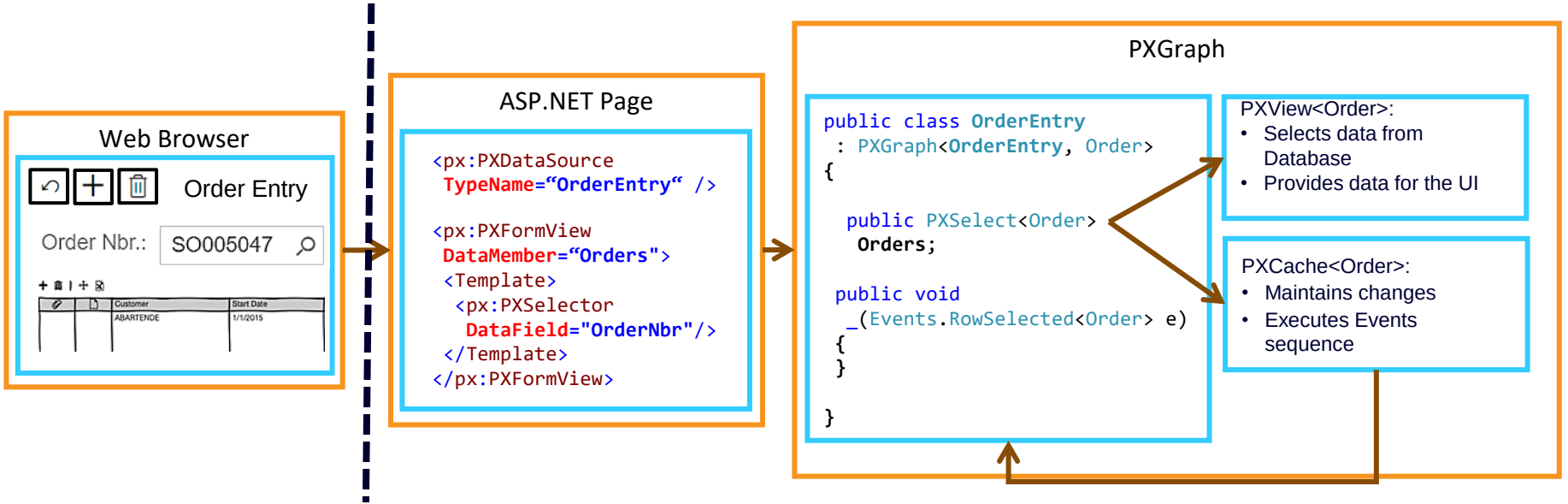
Event Model

Where do events come from?



Where do events come from?

Example:



Events 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) {}
```

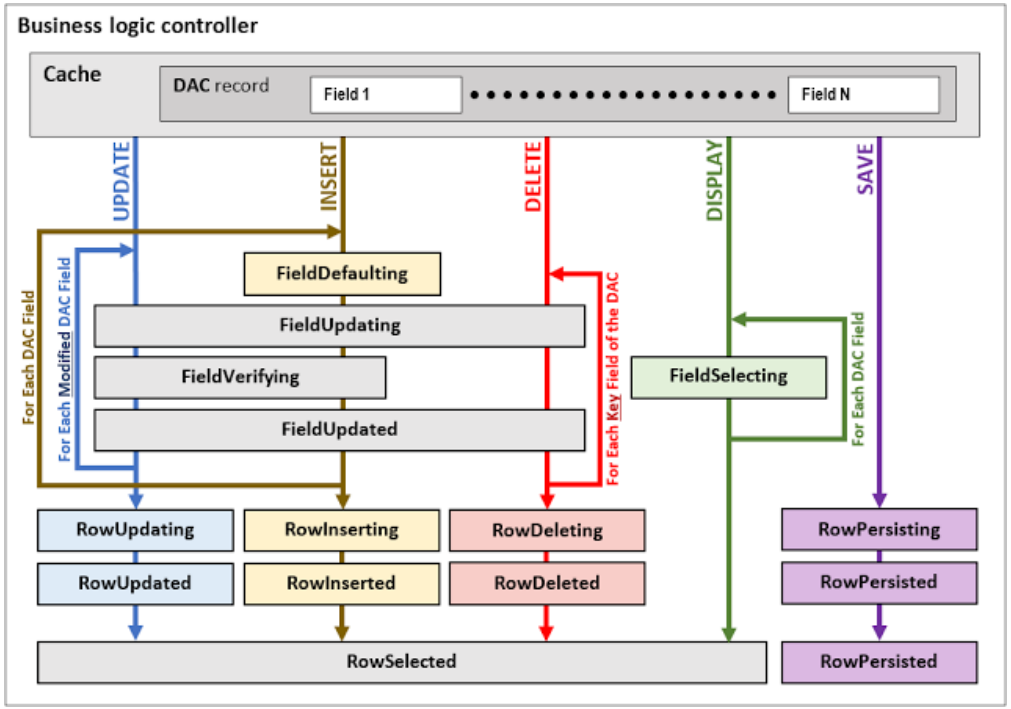
Attributes:

```
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



Customization

How to Customize

User Interface

- Add / Remove / Rearrange controls – Customization Browser Tools

Business Logic

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

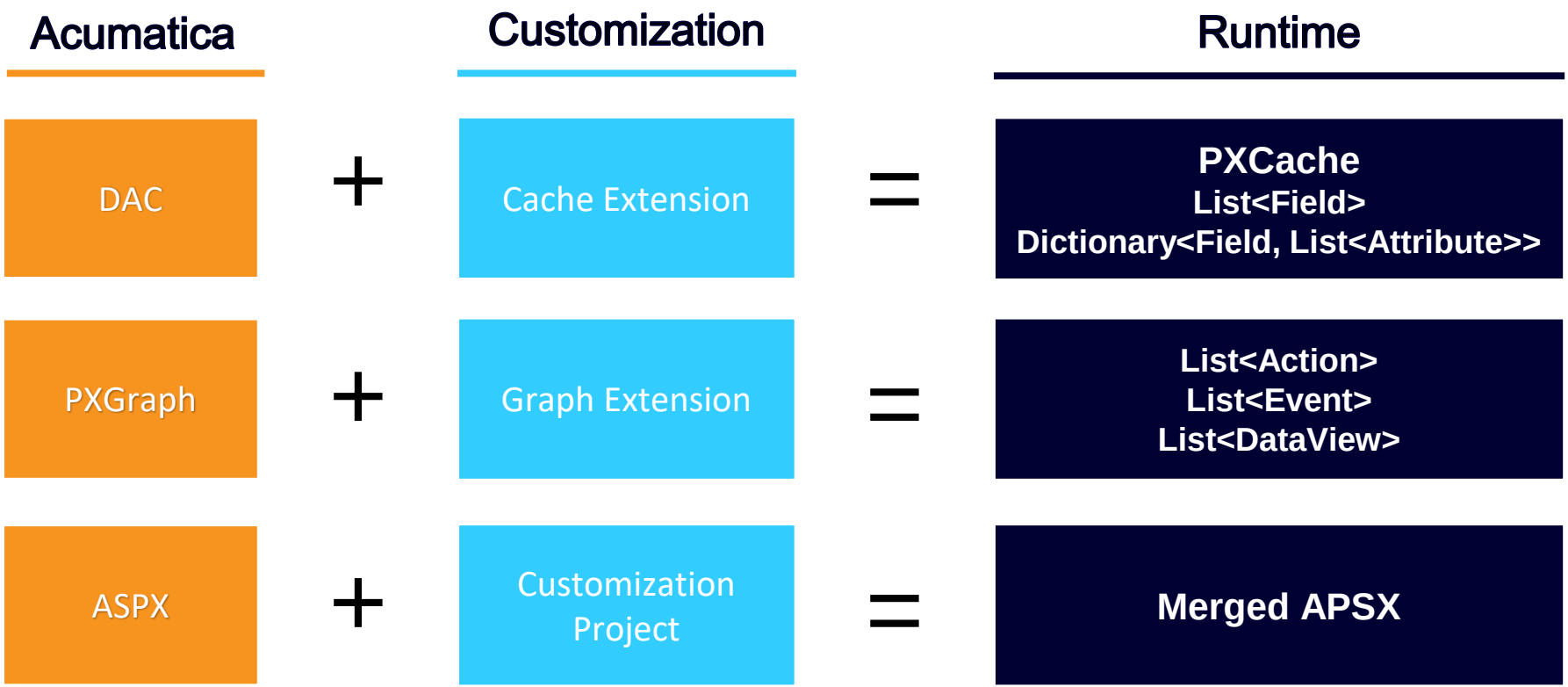
Database

- Add Columns and Tables – Customization Browser Tools

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



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)
    {
…
    }
}
```

What next?

What to do Next?

Learn Acumatica Framework and Tools:

- https://www.acumatica.com/acumatica-developer-training/
- Takes about 2-3 weeks for a developer

Customization Guide:

- https://help.acumatica.com/Main?ScreenId=ShowWiki&pageid=316b14fa-f406-4788-993c-7b043b1c5bd9

Learn from others, participate!

- https://github.com/Acumatica/
- https://stackoverflow.com/questions/tagged/acumatica
- https://asiablog.acumatica.com/

Acumatica
The Cloud ERP

Thank You

Dhiren Chhapgar

Dchhapgar@acumatica.com