Trends in Database Development: XML, .NET, WinFS

Alexander Vaschillo
Microsoft

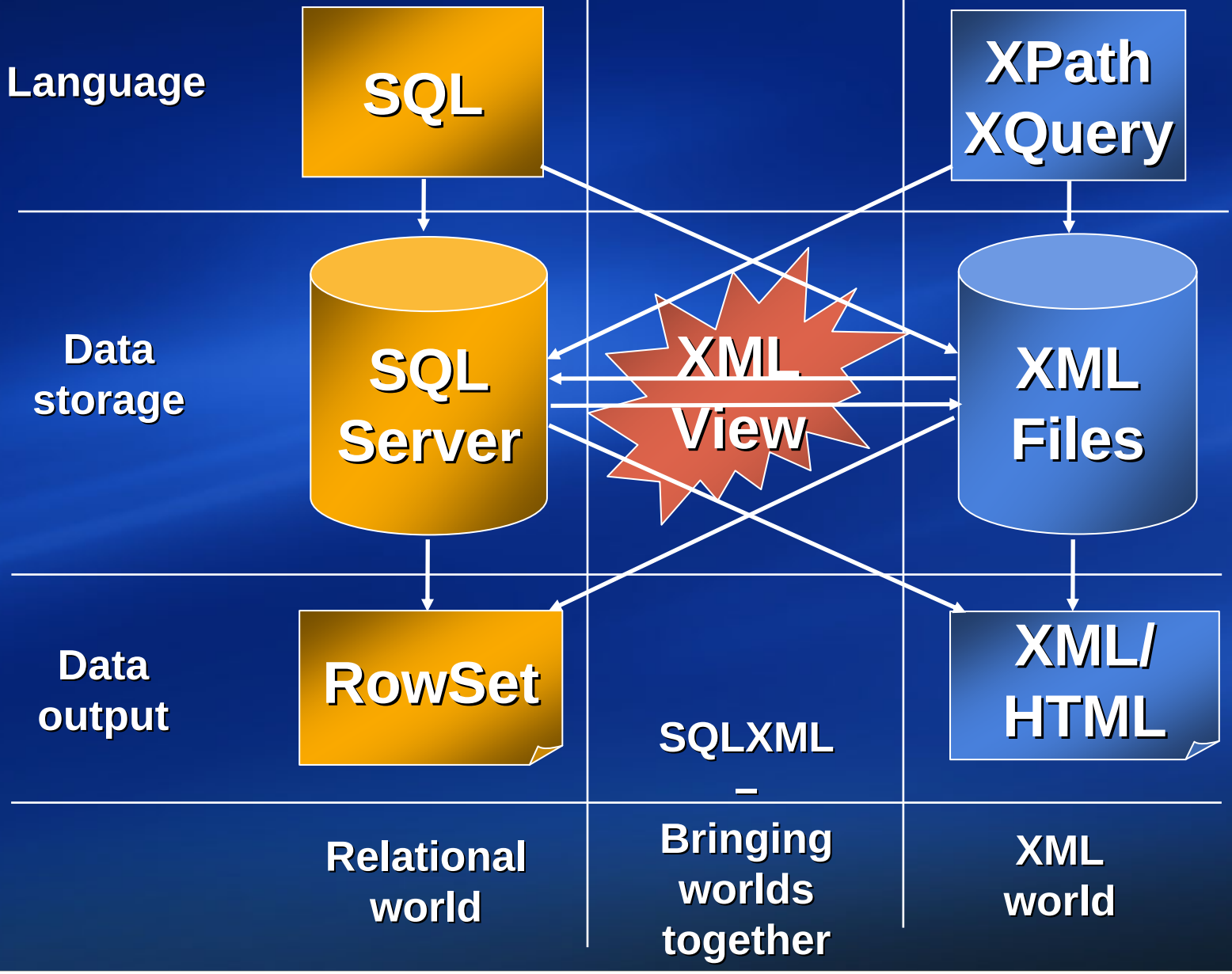
SQL Server Stores Everything

- Overall direction: Storing stuff whatever this stuff is
- Different data models
  - Relational
  - Hierarchical (XML)
  - Object (Graphs)
  - Files

Latest in SQL Server

- XML
  - Mapping to relational (SQLXML)
  - Native (XML Datatype)
- Objects
  - Mapping to relational (ObjectSpaces)
  - Native (CLR UDT)
- .NET integration
  - Server
  - Client
- Using all of the above
  - WinFS

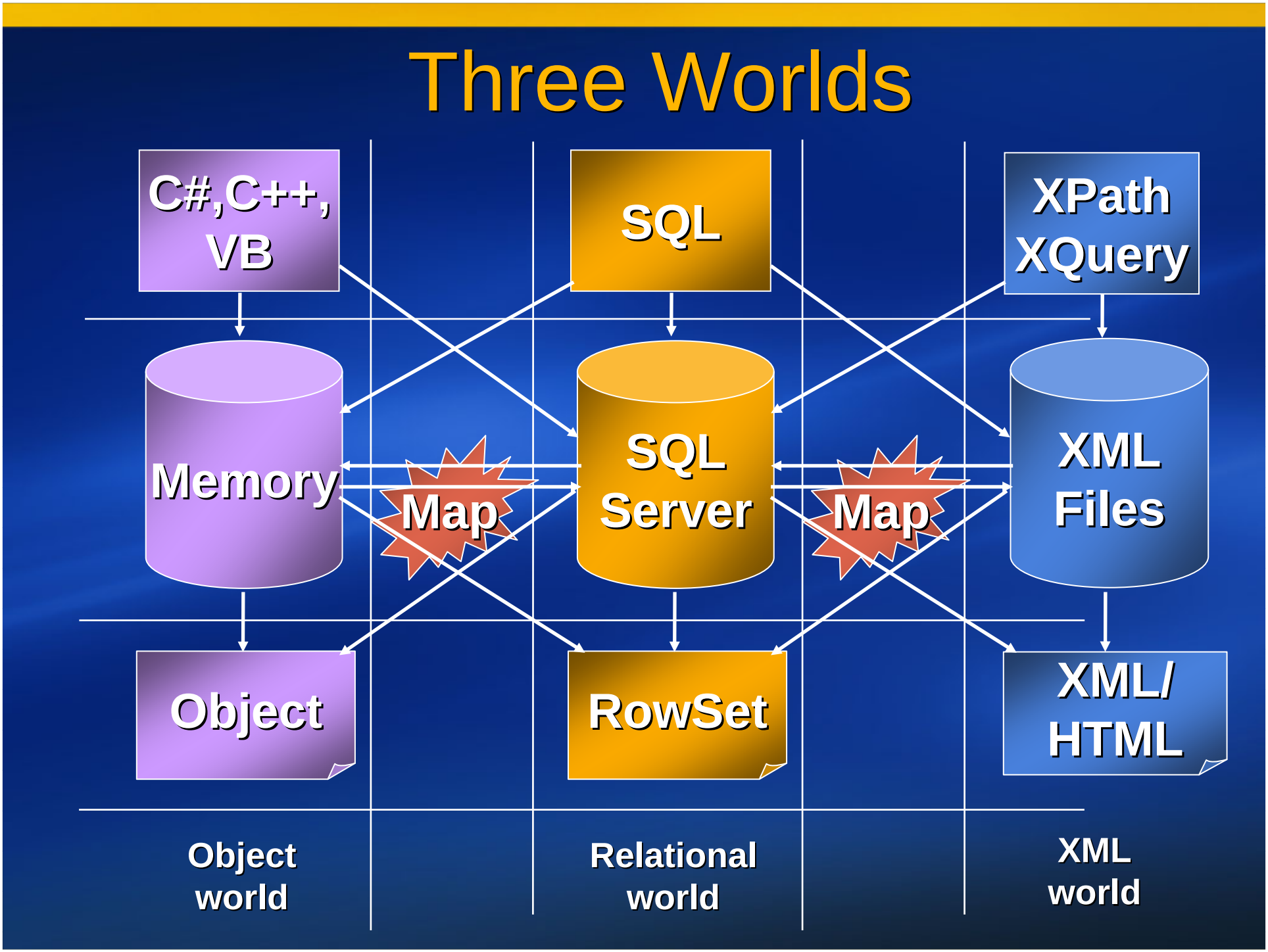
The Two Worlds



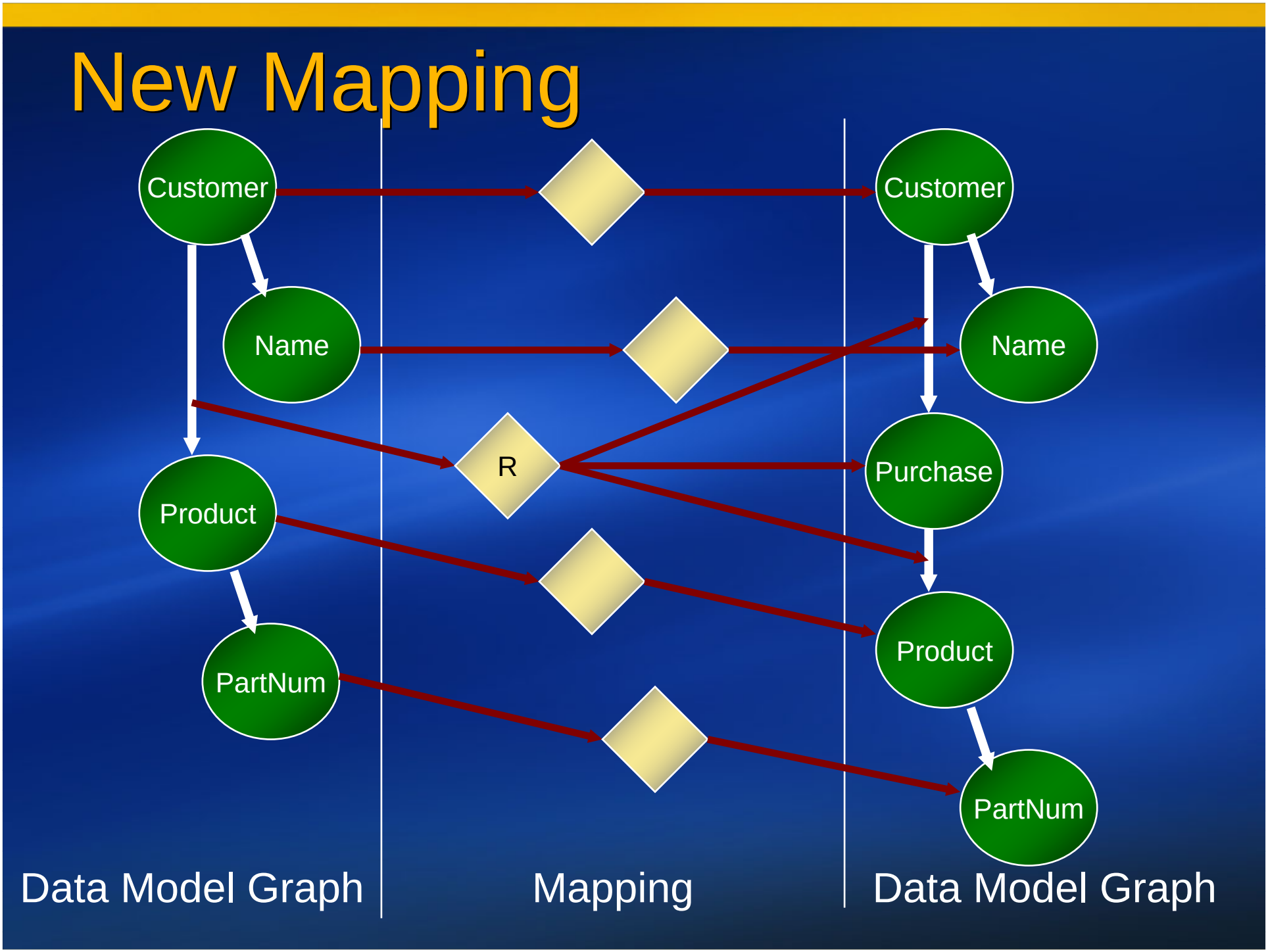
XSD Mapping Example

```
<xsd:schema  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
     xmlns:msdata="urn:schemas-microsoft-com:mapping-schema">
  <xsd:element name="Customer" msdata:relation="Customers">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="Order" msdata:relation="Orders">
            <xsd:annotation><xsd:appinfo>
              <msdata:relationship
                parent="Customers" parent-key="CustomerID"
                 child="Orders" child-key="CustomerID" />
            </xsd:appinfo></xsd:annotation>
          <xsd:complexType>
              <xsd:attribute name="OrderDate" type="xsd:dateTime"/>
          </xsd:complexType>
        </xsd:element>
      </xsd:sequence>
      <xsd:attribute name="CustomerID" />
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

Three Worlds



New Mapping


Customer
Name
Product
PartNum
R
Customer
Name
Purchase
Product
PartNum
Data Model Graph
Mapping
Data Model Graph

Native XML Store
XML Data Type

- XML data type
  - Native SQL type
  - Use for column, variable or parameter

```
CREATE TABLE docs (id INT PRIMARY KEY, xDoc XML NOT NULL)
```

- Store un-typed or typed XML instances
- Well-formed and validation checks
- Optional XML Schema enforcement
- XML instances stored as LOB (2GB)
  - Efficient binary representation

Native XML Store
XML Index

- Create XML index on XML column

```
CREATE XML INDEX idx_1 ON docs (xDoc)
```

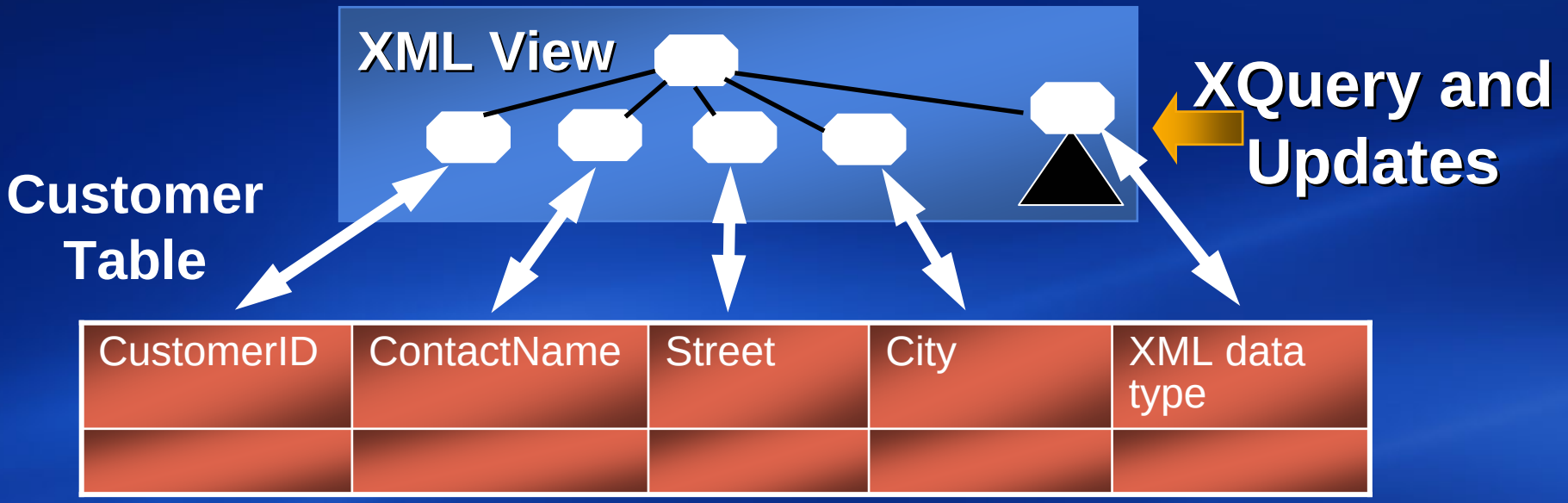
- Creates indexes on tags, values & paths
- Speeds up queries
  - Entire query is optimized
    - Same industry leading cost based optimizer
  - Indexes are used as available

XML Query

- XQuery: query XML documents and data
  - Standards-based: W3C
- In document 123, return section heading of section 3 and later

```
SELECT id, xDoc::query('
    for $s in
            /doc[@id = 123]//sec[@num >= 3]
    return <topic>{data($s/heading)}</topic>
    ')
FROM docs
```

XML View: Unification Model



- SQL Server “Yukon” XML data type
  - Use XQuery
- Relational columns
  - Use SQL
- XML View hides representation
  - Use XQuery against any data

.NET Integration

- Server side: SQLCLR
  - .NET hosted inside the database
  - Write stored procedures in C#
  - Use ADO programming model on the server the same way as on the client side
  - Create UDTs
- Client side
  - Web Services
  - Dataset
  - Objectspaces

SQLCLR

- Component reuse
- Mainstream development experience
- Familiar choice of programming languages and constructs
- Leverage existing libraries and components
- Seamless debugging and deployment
- Deep integration with the engine

SQLCLR Development

VB, C#, …

VS .NET Project

Build

Assembly: “TaxLib.dll”

SQL Data Definition:

```
create assembly …
create function …
create procedure …
create trigger …
create type …
```

Runtime hosted inside SQL

SQL Server

SQL Queries:

```
select
sum(tax(sal,state) )
from Emp where county
= ‘King’
```

Web Services Overview

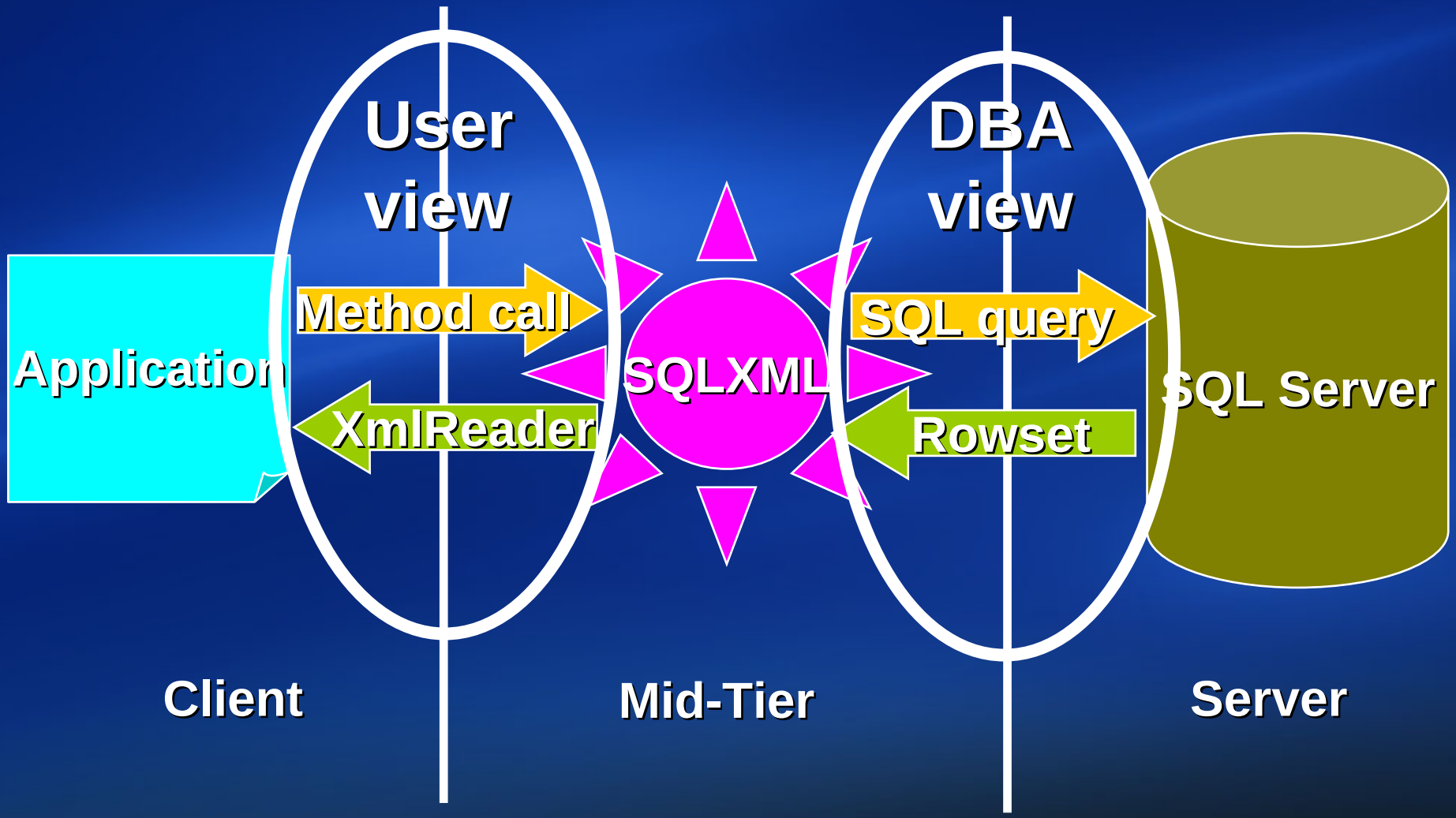
- Natural client side programming model
- Turn your existing Stored Procedures into web Services
- Messaging done according to SOAP 1.1 standard
- Choose how to model results
  - XML
  - Objects
  - Dataset
- Can run on database server or mid-tier
- Integrated with Visual Studio

Easy Programming Model

- SQLXML generates WSDL automatically
- Visual Studio.NET recognizes a Dataset
- Retrieve results of a Stored Procedure and load into a Dataset in 1 line of code!

```
Dim Service As New MyHost.MyWebService()
Dim retval As Integer
DataSet ds = Service.GetCustomer(Name)
```

Web Services – Decoupled Architecture



Universal Storage

- SQL server
  - Abstracts data model from the user
  - Abstracts data access programming model from the user
  - Abstracts its existence from the user
- Can it replace file system?

User Files

- Unstructured data
  - Not really unstructured – proprietary structure
- Data broken into files
  - One level of granularity (HTML, Powerpoint)
  - Easy manipulation?
- Proprietary formats
  - Need particular application to interpret files
  - No Sharing (Import/Export)
  - No relationships
- Duplication of Data
- Compatibility of data (Emails, Contacts,…)

WinFS

- Database
  - Reliability, Concurrency, Speed, query optimization
- Understanding schemas
- Uniform Search
- New APIs
  - SQL
  - Objects
- Old APIs
  - Will be supported
- Old files still work
  - Want to enable richer integration – provide translations mechanisms

WinFS Schemas

- Unification on some level
  - Base schemas shipped with Windows
- Play by the rules – all applications will be enabled with your data
  - Use extensions for your proprietary data
- Convenient programming model
- Shell supports libraries
- Navigation (relationships)
- Integration (Email body is a document)

The Windows Schemas

User Data

- Principals
- Locations
- Calendar Events
- Core
- Message (Email)
- Documents
- Annotations
- Media
- Notes
- Person Tasks
- Audio
- Videos
- Images
- Games
- . . .

System

- System Tasks
- Explorer
- Config
- NaturalUI
- Programs
- Services
- Security
- Help
- Device
- . . .

Infrastructure

- WinFSTypes
- Meta
- Base
- File
- Sync
- ShellSubscriptions
- . . .

WinFs Data Model

- Common data model
- Simple but rich
- Common programming model for all applications
- Unified store - unified access
- Subsume relational concepts
- Rich object abstraction for data
- Semantic relationships
- Align with CLR

Common Data Model

- Map to CLR - framework
- Map to SQL – add some abstractions
- Express data in XML
- Combine SQL 99, CLR, XSD, UML
  - Nested tables
  - Relationships
  - Scalar types
- Provide mapping to other data models

SQL 99 Data Model

- Too implementation specific – not a good application level abstraction
- Tables, not types are first class
  - Identity
  - Operational semantics (copy, etc.)
- Integrate SQL with Programming language rather than map from objects to DB
- No high-level relationship support
  - Can use low-level concepts (foreign keys)

CLR Data Model

- Framework on top
  - Constrain
  - Enhance with additional functionality
- Not designed for persistence
  - References are not durable
  - Foreign keys can be persisted
- No set–oriented query capabilities
- No high-level relationship concepts
  - Can use collections to implement some
- No support for BLOBs

XML/XSD Data Model

- Use XML syntax, but not semantics
- No Relationships in XSD Data Model
- Too many concepts not useful for us and hard to map to CLR and SQL
- Different set of scalar types
- Complex mechanisms for type extension
  - Not compatible with CLR

WinFS Data Model

- Items (Entities)
- Scalar types
- Inline Types
- Inheritance
- Extensions
- Relationships

Items

- Have identity
- Smallest referenceable unit
- Have Properties and Behaviors (methods)
- Can exist independently, can be copied, etc.
- Person, Message, Document, Audio,...

```
<EntityType Name="Person">
  <Property Name="Name" Type="WinFS.String"/>
  <Property Name="Age" Type="WinFS.Int32" Default="1"/>
  <Property Name="Picture" Type="WinFS.Binary"/>
  <Property Name="Addresses" Type="Array(Address)"/>
</EntityType>
```

Scalar Types

- Used for properties on an Item (Relatonship)
- Carefully chosen subset of SQL and CLR types
- String, Boolean, Binary, Byte, Int16, Int32, Int64, Single, Double, Decimal, DateTime, Guid, XML, Stream.
- Enumeration

```
<Enumeration Name="Gender" >
  <EnumerationMember Name="Male" />
  <EnumerationMember Name="Female" />
</Enumeration>
```

- Arrays and Sets

Inline type

- A structure without identity
- Not referenceable
- Has to be contained in an Entity (or Relationship)
- Example: Address

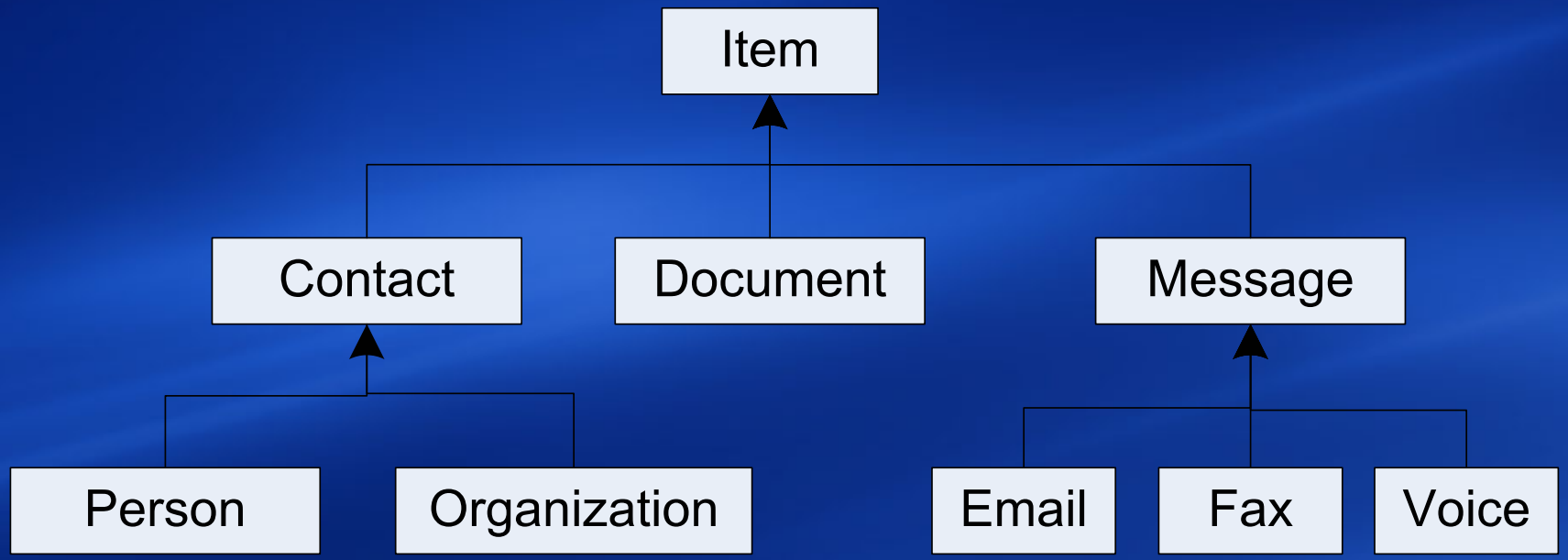
```
<InlineType Name="Address">
  <Property Name="Street" Type="String" Nullable="false"/>
  <Property Name="City" Type="String" Nullable="false"/>
  ...
</InlineType>
```

Inheritance

- Single inheritance (Items, Inline types)
- Substitutability

```
<Type Name="Name" >
  <Property Name="FirstName" Type="WinFS.String" />
  <Property Name="LastName" Type="WinFS.String" />
</Type>
<Type Name="NameWithMiddleInitial" BaseType="Name" >
  <Property Name="MiddleInitial" Type=“WinFS.String" />
</Type>
<Type Name="Person" BaseType="System.Storage.Item" >
  <Property Name=" PersonalName" Type="Array(Name)" />
</Type>
```

Inheritance Hierarchy



Extensions

- Non-invasively adds more structures to an existing Item
- Multiple extensions can be added independently
- Must be attached to an Item and live with it

```
<EntityExtensionType Name=”MSNData”
    ExtendsType=”PersonItem” >
   <Property Name="Passport" Type="PassportData" />
   <Property Name="MSNId" Type="Guid" />
</EntityExtentionType>
```

Relationships

- Document-Author, Message-Participant, Album-Record
- Association and Composition
- Relate two Items
- May have properties
- Support cardinalities (1:1, m:1, 1:m, m:m)
- May control lifetime
- Based on common values or identity

Relationship Example

```
<EntityType Name="Customer" ...>   ...
</EntityType>

<EntityType Name="Order" ...>
  <Property Name="CustRef" Type="Ref(Customer)"/>  ...
</EntityType>

<Association Name="OrderCustomer" >
  <End Role="OrderRole" Type="Order" Multiplicity="*" />
  <End Role="CustomerRole" Type="Customer"
    OnDelete="Cascade” Multiplicity="1" />
  <Reference FromRole=”OrderRole”
    ToRole=”CustomerRole” Property=”CustRef”/>
</Association>
```

Relationship Example

```
<Association Name="DocumentAuthor" >
  <End Role="DocumentRole" Type="Document"
     Multiplicity="*" />
  <End Role="ContactRole" Type="Contact" Multiplicity="1"/>
  <Condition>
    DocumentRole.Author = ContactRole.Name
  </Condition >
</Association>
```

Data Model Mapping

- A WinFS schema is mapped to a SQL schema
- Types are mapped to CRL classes in the storage (UDT), and CLR API classes
- Classes are automatically created based on type definition
- Views are generated for each type
- Relationships: UDT vs. metadata
- Schema becomes a namespace in the API

WinFS API

- Natural programming model
- Language integration
- Collections vs. SQL Queries
- Database operations are hidden from a developer

Querying WinFS

```
StorageContext sc = new StorageContext();

StorageSearcher<PersonItem> searcher =
    sc.Items.FilterByType<PersonItem>().
        Filter("Exists(Names[LastName='Smith']");

PersonItem p1 = searcher.GetFirst();
    or
foreach (PersonItem p in searcher)
{  ...
}
```

Query Composition

```
StorageSearcher<MessageItem> messages =
    sc.Items.FilterByType<MessageItem>().
        Filter("Subject LIKE '%Academic Days%'");

StorageSearcher<StorageRecord> myMessages
    =  messages.Project("Subject, Size ").
        Sort("Size desc");

foreach( StorageRecord m in myMessages)
{
   string s = m["Subject"];
   int size = m["Size"];
}
```

Item Creation

```
Item root = sc.GetRootItem();

PersonItem person = new PersonItem();
person.DateOfBirth = “11/01/1960";

FullName fullName = new FullName();
fullName.FirstName = “John”;
fullName.LastName = “Smith”;
person.Names.Add(fullName);

root.Children.Add(person);
sc.SaveChanges();
```

Relationship Navigation

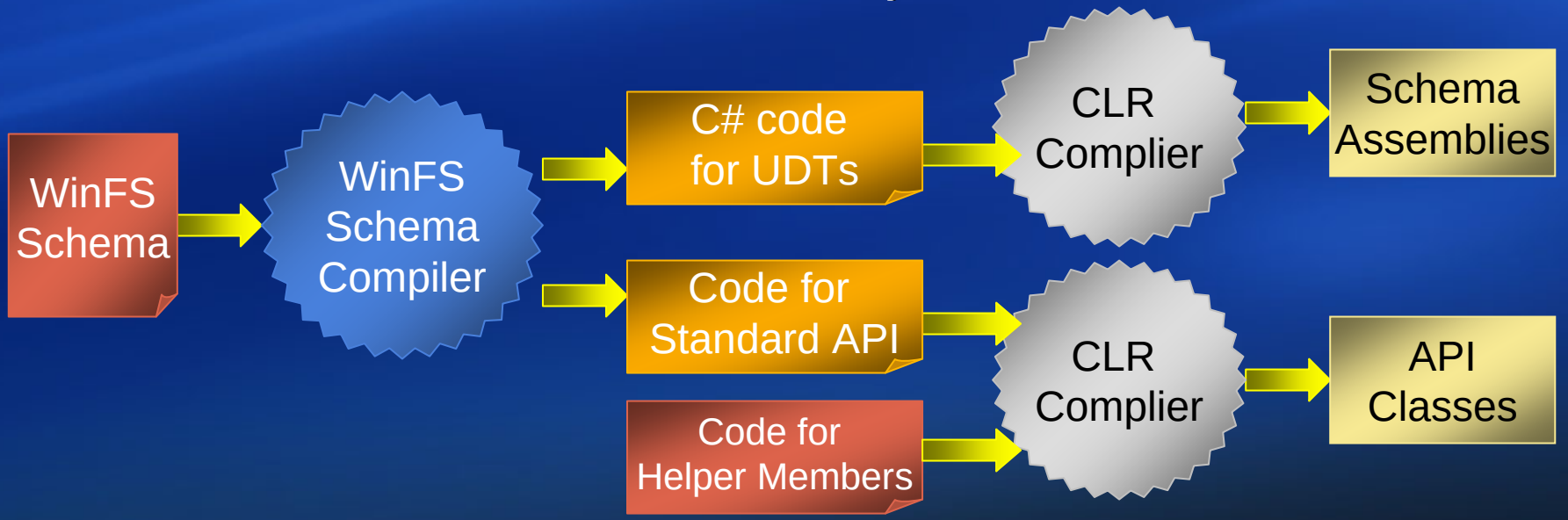
```
StorageSearcher<OrganizationItem> organizations =
    sc.Items.WithType<OrganizationItem>().
        Filter("Keywords[Value='Financial']");

StorageSearcher<PersonItem> employees =
    EmploymentRelation.GetEmployees(organizations);

StorageSearcher<DocumentItem> documents =
    DocumentAuthorRelation.GetDocuments(employees);

foreach( DocumentItem document in documents)
{   ...
}
```

Notifications

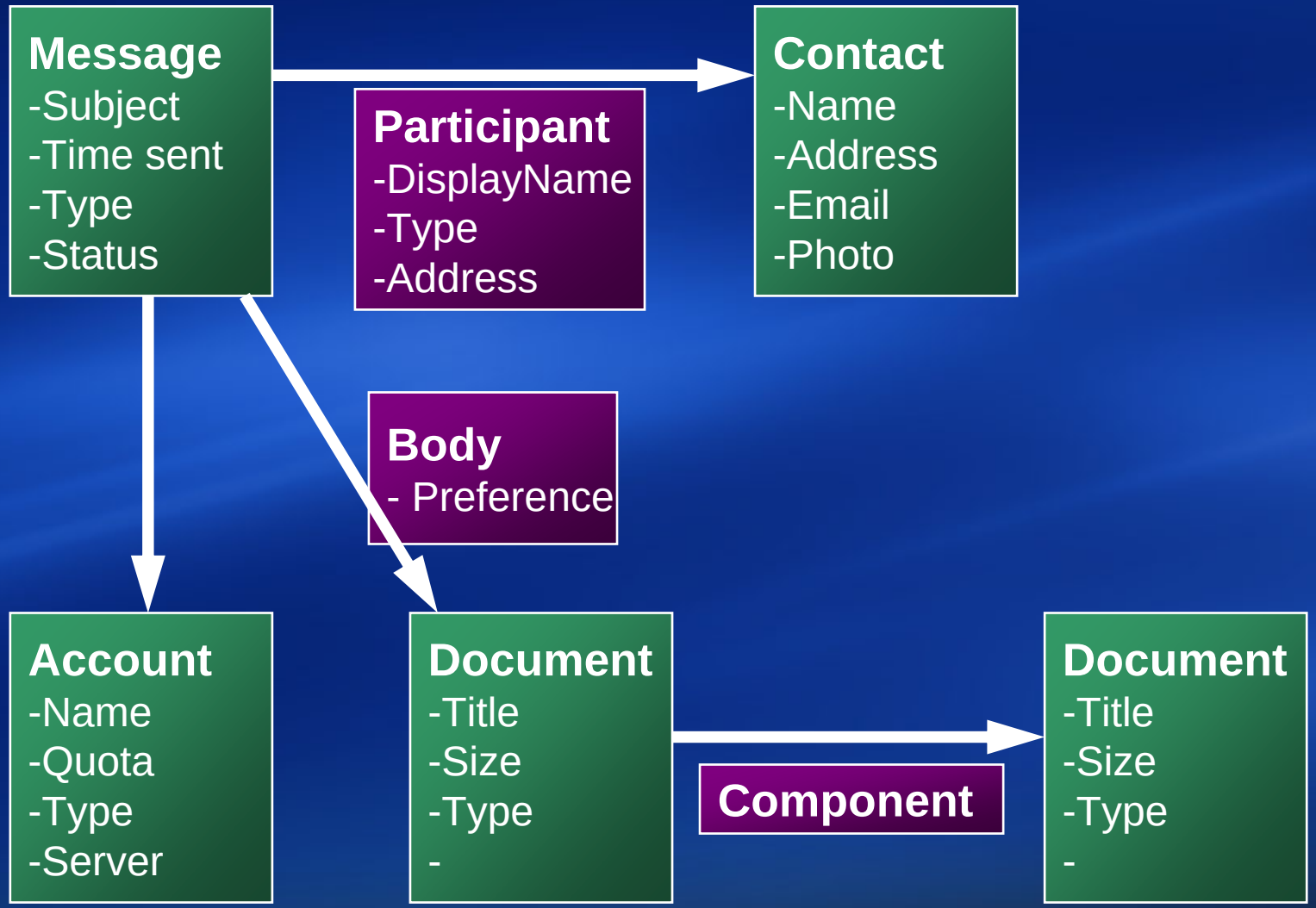
```
PersonItem person =
    sc.Items.FilterByType<PersonItem>().
        Filter("Exists(Names[LastName='Smith']").GetFirst();

StoreWatcherOptions Opts=new StoreWatcherOptions();
Opts.NotifyModified = true;

StoreWatcher w = person.GetWatcher( Opts );
w.StoreObjectChanged += new StoreEventHandler(
    MyHandler );

void MyHandler( Object sender, StoreEventArgs e )
{
    ...
}
```

Creating API for a Schema

- Create WinFS schema in XML format
- Schema compiler generates API assembly
- You can add your own “helper” members
- The assemblies are installed into a WinFS store
  - WinFS types are registered as UDTs
  - Views and other database objects are created



WinFS Message Schema (Example)



My Favorite Query

- What do I know about “John Smith”
- Documents by/about him
- Emails from him
- His address
- Phone calls from him
- Annotations he added to my papers
- Meetings with him


Microsoft®