ADO.NET Data Services

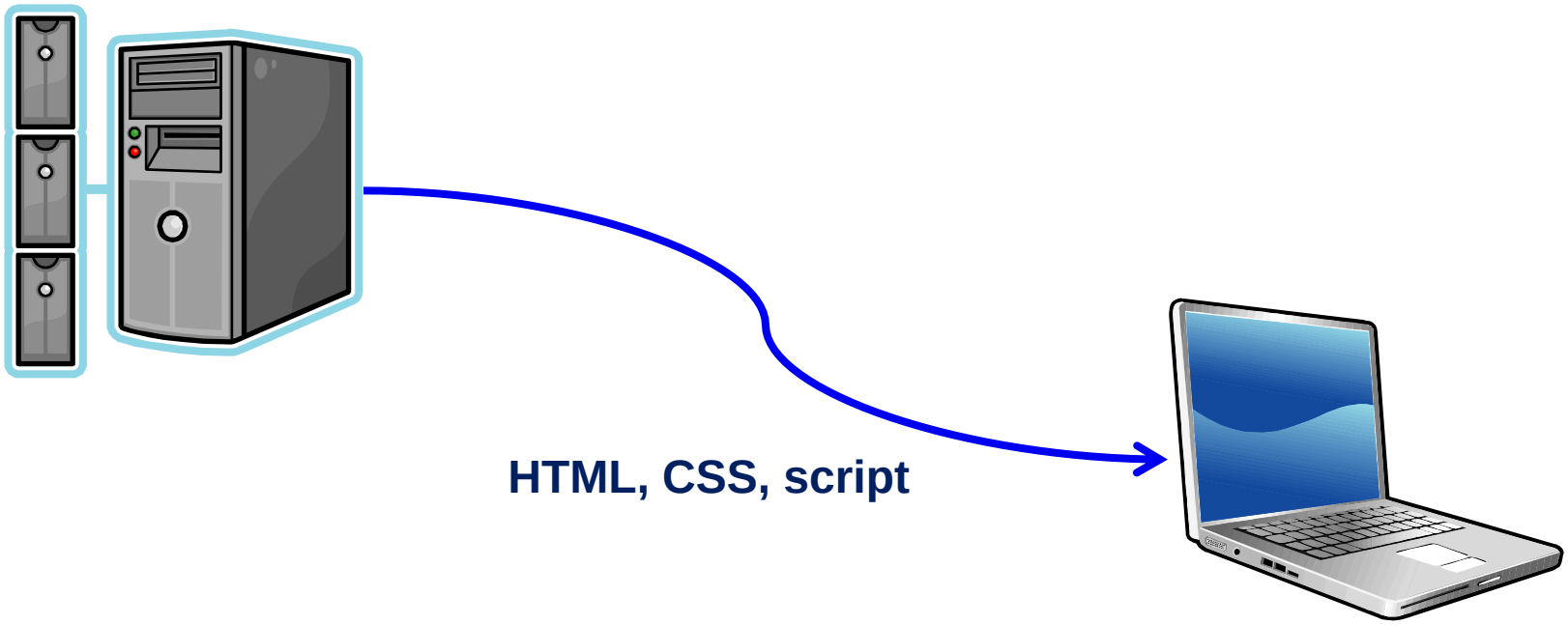
Pushing Data Over the Web

Overview

- Why Data Services?
- ADO.NET Data Services and REST
- Building and configuring a Data Service
- Consuming Data Services
- Query Interceptors
- Service Operations

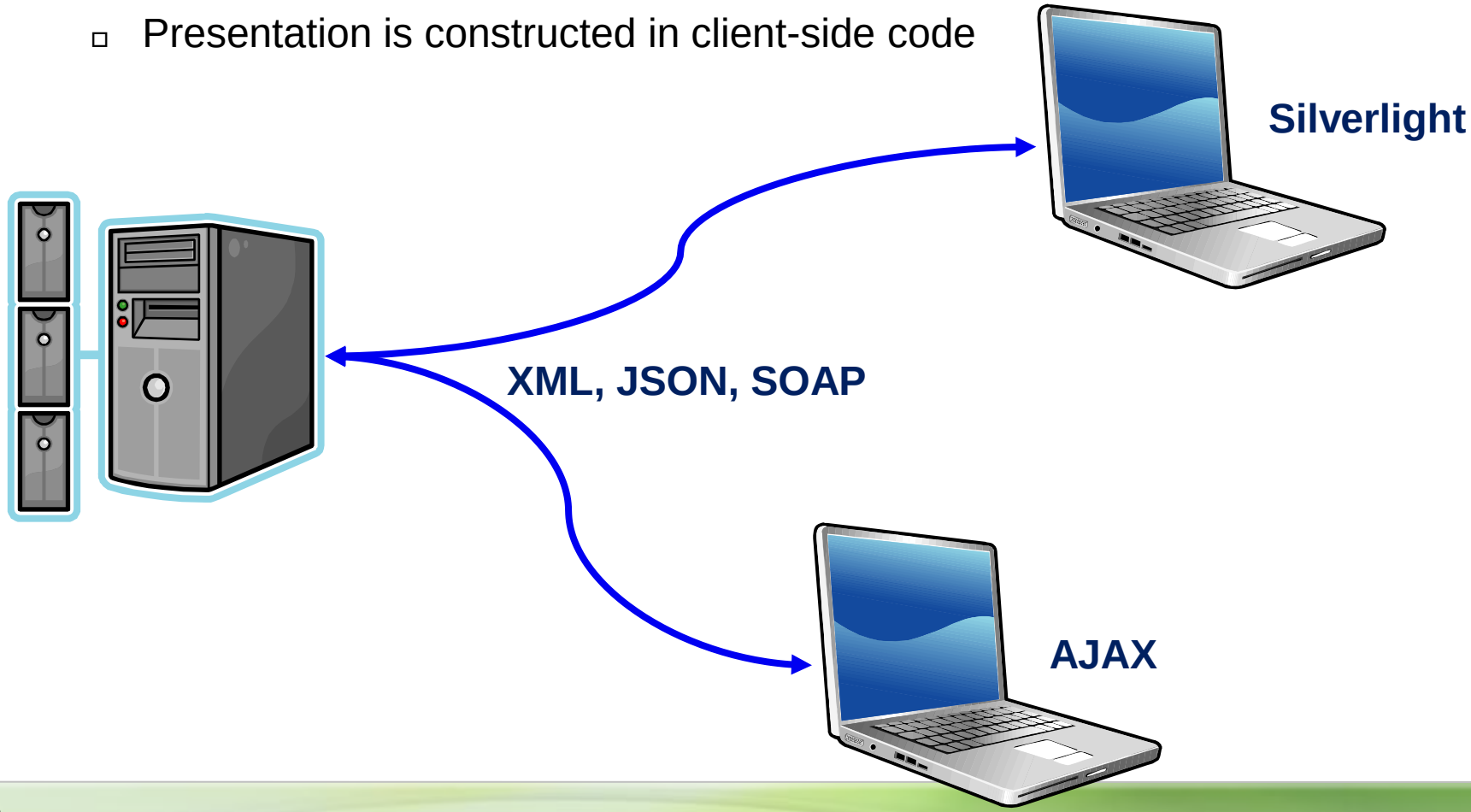
Motivation

- **Traditional web applications consume data on the server**
  - Send only HTML, CSS, and some script to the client



Enter the Rich Internet Application

- **Client wants to consume raw data**
  - Presentation is constructed in client-side code



Challenges

- **Need to expose raw data to the web**
- **Traditional SOAP based web services have some drawbacks**
  - Have an obsession with HTTP POST
  - Focus on operations (verbs) - not data (nouns)
  - Rely on XML, WSDL, WS-*, and SOAP tooling

ADO.NET Data Services Are RESTful

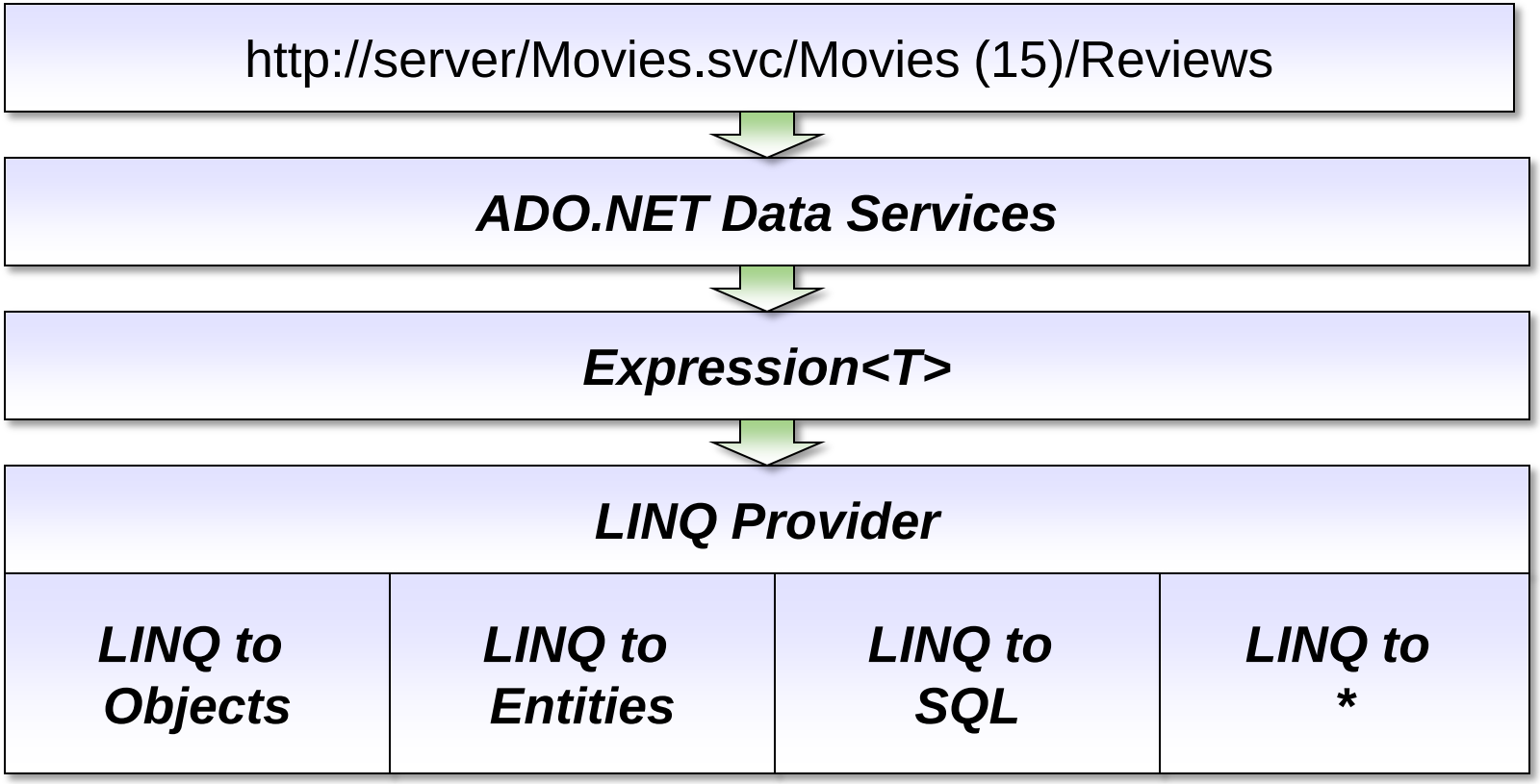
- **REpresentational State Transfer**
  - Uses only HTTP and HTTPS
  - Defines 4 operations with the HTTP verbs GET, POST, PUT, DELETE
  - Treat entities as resources – entities are addressable by URL

http://server/Movies.svc/Movies

http://server/Movies.svc/Movies (15)/Reviews

LINQ's Role in ADO.NET Data Services

- **LINQ's server side role is to provide storage independence**



Response Formats

- **Negotiated via the HTTP Accept header**
- **Can choose from Atom (default) and JSON**

```
GET /movieservice.svc/Movies(1)
Accept: application/atom+xml
```

```
<?xml version="1.0“ ?>
...
   <d:movie_id>1</d:movie_id>
   <d:title>
      Where the Wild Things Are
   </d:title>
...
```

```
GET /movieservice.svc/Movies(1)
Accept: application/json
```

```
{ "d" : {
  ...
   "movie_id": 1,
   "title": "Where the Wild Things Are",
   ...
} }
```

URL $ Options

Option	Description	Example
$value	Retrieve a value without any surrounding metadata	/Movies(2)/Title/$value
$expand	Eager loading of specified elements	/Movies(1)?$expand=Reviews
$filter		/Movies?$filter=Title eq ‘Star Wars’
$orderby	Sort the target resources	/Movies?$orderby=Title desc
$top	Return only the top n resources	/Movies?$top=10 /Movies?$orderby=Title&$top=5
$skip	Skip the first n resources	/Movies?$skip=100&$top=10

Basic Ingredients For a Data Service

- **One assembly reference**
  - System.Data.Services
- **One WCF service endpoint**
  - Use Factory=DataServiceHostFactory in the @ Service attribute
  - Service class derives from DataService<T>
- **One data source**
  - Entity Framework EDM Model (ObjectContext derived class)
  - Any CLR type with one or more public IQueryable<T> properties

Setting Up A CLR Model Data Source

- **Resources must have a primary key**
  - ID property, or a [DataServiceKey] decoration
- **Implement IUpdateable if CUD support is required**
  - Entity Framework provides it's own IUpdateable implementation

```
public class Movie
{
  public int ID {get; set;}
  public string Title {get; set;}
  public List<Review> Reviews
         {get; set;}
}
```

```
public class MovieDataSource
{
  ...
  public IQueryable<Movie> Movies
  {
      get { return _movies.AsQueryable(); }
  }

  List<Movie> _movies;
}
```

Configuration

- **Entity Access Rules**
  - Everything is off by default
  - SetEntityAccessRule uses property names to specify access rights
  - Wildcard (*) allows access to all public, queryable properties

```
public class InMemoryMovies :
                DataService<MovieDataSource>
{
    public static void InitializeService(
                        IDataServiceConfiguration config)
    {
        config.SetEntitySetAccessRule("Movies",
                                      EntitySetRights.All);
    }
}
```

A Basic .NET Client

- **Use any class that can send an HTTP request**
  - System.Net.WebRequest
  - System.Linq.Xml.XDocument

```
string url = "http://server/InMemoryMovies.svc/Movies/";
string orderby = "$orderby=Title desc";

XDocument result = XDocument.Load(String.Format("{0}?{1}", url, orderby));

XNamespace atom = "http://www.w3.org/2005/Atom";
XNamespace ds = "http://schemas.microsoft.com/ado/2007/08/dataservices";
var movies = from m in result.Descendants(atom + "content")
             select new  {
                 ID = m.Descendants(ds + "ID").First().Value,
                 Title = m.Descendants(ds + "Title").First().Value
             };
```

Using the Data Services Client

- **DataServiceContext class – designed to work with Data Services**
  - Lives in the System.Data.Services.Client assembly
  - Converts LINQ queries to REST requests
  - Converts response to types defined in client assembly via reflection

```
Uri serviceRoot = new Uri("http://server/InMemoryMovies.svc");
DataServiceContext ctx = new DataServiceContext(serviceRoot);

var result = from m in ctx.CreateQuery<Movie>("Movies")
             orderby m.Title descending
             select m;
```

```
GET /MovieSite/InMemoryMovies.svc/Movies()?$orderby=Title%20desc
HTTP/1.1
User-Agent: Microsoft ADO.NET Data Services
Accept: application/atom+xml,application/xml
Connection: Keep-Alive
```

Strongly Typed Client

- **Command line code generation - Datasvcutil.exe**
  - Generate codes for a DataServiceContext derived class
  - Generates code client side resource representations
  - Similar to sqlmetal.exe and Entity Framework ObjectContext derivations

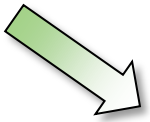
```
>datasvcutil /uri:http://server/moviesite/movies.svc /out:Movies.cs
```

```
Uri uri = new Uri("http://localhost:8080/InMemoryMovies.svc/");

MovieDataSource ctx = new MovieDataSource(uri);
var result = from m in ctx.Movies
             orderby m.Title
             select m;
```

Creating an Entity

- **Pass the entity set name and the new object to AddObject**
  - Server will respond with fresh representation
  - DataService context will update client side object
  - No data sent to server until you invoke SaveChanges

```
MovieDataSource ctx = new MovieDataSource(uri);
Movie movie = new Movie {
    Title = "No Country For Young Men"
};

ctx.AddToMovies(movie);
ctx.SaveChanges();
```



```
POST /MovieSite/InMemoryMovies.svc/Movies HTTP/1.1
...
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
  ...
  <content type="application/xml">
    <m:properties>
        <d:Title>No Country For Young Men</d:Title>
    </m:properties>
  </content>
```

Updating An Entity

- **Query for the object, make changes, then invoke UpdateObject**

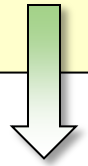
```
MovieDataSource ctx = new MovieDataSource(uri);
Movie movie = (from m in ctx.Movies
               where m.ID == 2
               select m).First();

movie.Title = "Monsters, Inc.";
ctx.UpdateObject(movie);
ctx.SaveChanges();
```

```
PUT /MovieSite/InMemoryMovies.svc/Movies(2) HTTP/1.1
...

<?xml version="1.0" encoding="utf-8" standalone="yes"?>
...
  <content type="application/xml">
    <m:properties>
      <d:ID>2</d:ID>
      <d:Title>Monsters, Inc.</d:Title>
    </m:properties>
  </content>
```

Delete Entity

- **Query for the entity, then pass the entity to Delete object**

```
MovieDataSource ctx = new MovieDataSource(uri);
Movie movie = (from m in ctx.Movies
               where m.ID == 2
               select m).First();
ctx.DeleteObject(movie);
ctx.SaveChanges();
```



```
DELETE /MovieSite/InMemoryMovies.svc/Movies(2) HTTP/1.1
Content-Length: 0
```

Silverlight Client

- **Can "Add Service Reference" for a strongly typed client**
  - All network requests in Silverlight are asynch

```
var ctx = new MovieReviews.MovieReviewEntities(
            new Uri("MovieReviewService.svc",
            UriKind.Relative));

DataServiceQuery<Movie> query =
   ctx.Movies.OrderBy(m => m.Title)
              .Take(100) as DataServiceQuery<Movie>;
query.BeginExecute((result) =>
  _grid.ItemsSource = query.EndExecute(result).ToList(),
   null
);
```

AJAX Clients

- **ADO.NET Data Service AJAX Client Library**
  - http://www.codeplex.com/aspnet/Release/ProjectReleases.aspx?ReleaseId=13357
  - Provides query, insert, update, and delete methods
  - Parses results into columns and rows.

```
var service =
    new Sys.Data.DataService("http://localhost:8080/InMemoryMovies.svc/");
service.query("Movies?$orderby=Title desc",
              onQueryComplete, onQueryFailed);
```

```
function onQueryComplete(result) {
    var sb = new Sys.StringBuilder();
    for (row in result) {
        sb.append(
            String.format("ID = {0} Title = {1}<br />",
                          result[row].ID, result[row].Title));
    }
    $get("resultsDiv").innerHTML = sb.toString();
}
```

Query Interceptors

- **Interceptors are fired on a GET request for a particular resource**
  - Inject custom logic into processing pipeline on a per request basis
  - Uses: custom authorization, custom validation

```
public class InMemoryMovies : DataService<MovieDataSource>
{
    [QueryInterceptor("Movies")]
    public Expression<Func<Movie, bool>> OnQueryMovies()
    {
        return movie => movie.Title.StartsWith("Star");
    }

    // ...
```

Change Interceptors

- **Change Interceptors fired on PUT, POST, DELETE operations**

```
[ChangeInterceptor("Movies")]
public void OnChangeMovies(Movie m, UpdateOperations operations)
{
    if ((operations & UpdateOperations.Delete) == UpdateOperations.Delete
                    && Thread.CurrentPrincipal.Identity == null)
    {
        throw new DataServiceException(400, "YOU cannot delete movies");
    }
}
```

Service Operations

- **Expose a method of the data service class as a URI**
  - Method could include custom business logic or complicated query logic
- **Restrictions**
  - Method can only accept in parameters
  - Must return void, 'IQueryable<T>, or IEnumerable<T>
  - Must decorate with [WebGet] or [WebInvoke]

```
[WebGet]
public IQueryable<Movie> TopRatedMovie(int minReviews)
{
    var result = from m in CurrentDataSource.Movies
                 where m.Reviews.Count > minReviews
                 orderby m.Reviews.Average(r => r.Rating) ascending
                 select m;
    return result;
}
```

Summary

- **ADO.NET Data Services provides a RESTful interface to data**
  - A great option to expose data to the web
- **GET, PUT, POST, DELETE are the four basic operations**
- **Data Service response comes in ATOM or JSON**
- **DataServiceContext class enables LINQ to Data Services**