Caching

Outline

- Output caching
- Post-cache substitution
- Page fragment caching
- Data caching
- Data source caching
- Cache dependencies
- SQL cache dependencies

Output Caching

- **For relatively static pages, rendered content can be cached**
  - Add OutputCache directive to page with cache duration

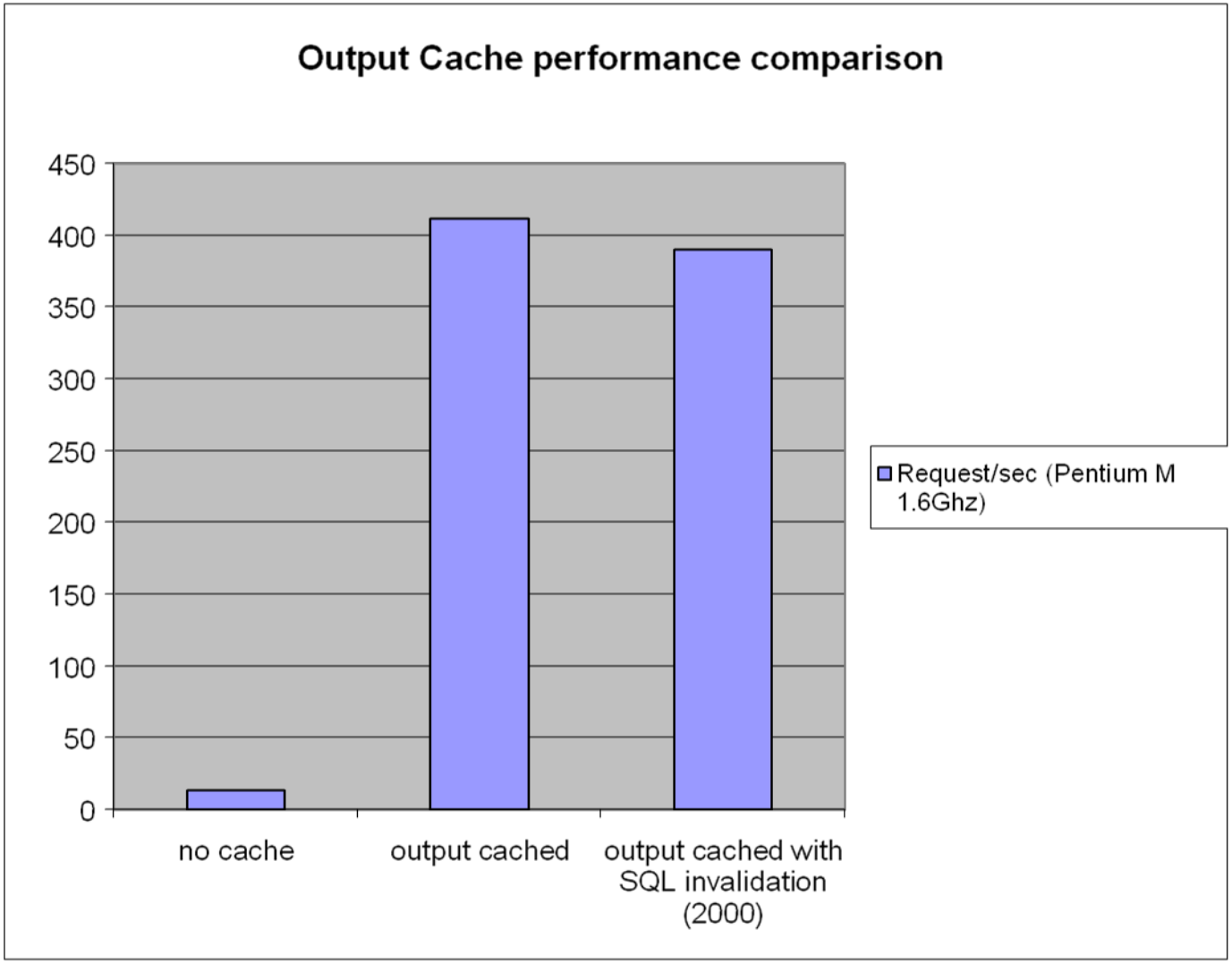
```
<%@ Page Language="C#" %>
<%@ OutputCache Duration="3600" VaryByParam="none" %>
<html>

  <script runat="server">
    void Page_Load(object sender, EventArgs e) {
         msgLabel.Text = DateTime.Now.ToString();
    }
  </script>

  <body>
    <h3>Output Cache example</font></h3>
    <p>Last generated on:
       <asp:label id="msgLabel" runat="server"/></p>
  </body>

</html>
```

Output caching performance



Attributes and their meaning for the OutputCache directive

OutputCache attribute	Values	Description
`Duration`	`number`	`Time, in seconds, that the page or user control is cached`

`Location`	`'Any'`
	`'Client'`
	`'Downstream'`
	`'Server'`
	`'None'`
		`Controls the header and meta tags sent to clients indicating where this page can be cached.`
		`'Any' means that the page can be cached on the browser client, a downstream server, or the server.`
		`'Client' means that the page will be cached on the client browser only.`
		`'Downstream' means that the page will be cached on a downstream server and the client.`
		`'Server' means that the page will be cached on the server only.`
		`'None' disables output caching for this page.`

`VaryByCustom`	`'Browser'` `custom string`	`Either vary the output cache by browser name and version, or by a custom string, which must be handled in an overridden version of GetVaryByCustomString().`
`VaryByHeader`	`'*'` `header names`	`A semi-colon separated list of strings representing headers submitted by a client.`

`VaryByParam`	`'none'`
	`'*'`
	`param name`
		`A semi-colon separated list of strings representing query string values in a GET request, or variables in a POST request.`

HttpCachePolicy

- **The attributes of the OutputCache directive are used to populate an instance of HttpCachePolicy**
  - Initialized in Page class' InitOutputCache() method
  - Can modify through Response.Cache
  - Only way to set sliding expiration or an explicit expiration time

HttpCachePolicy class

```
public sealed class HttpCachePolicy
{
  public HttpCacheVaryByHeaders VaryByHeaders {get;}
  public HttpCacheVaryByParams VaryByParams {get;}
  public void AppendCacheExtension(string extension);
  public void SetCacheability(HttpCacheability cacheability);
  public void SetExpires(DateTime date);
  public void SetLastModified(DateTime date);
  public void SetMaxAge(TimeSpan delta);
  public void SetNoServerCaching();
  public void SetSlidingExpiration(bool slide);
  //...
}

public sealed class HttpResponse
{
  public HttpCachePolicy Cache {get;}
  //...
}
```

Programmatically Settings Page Caching

```
<%@ Page Language="C#" %>
<html>
<script runat="server">
void Page_Load(object sender, EventArgs e)
{
  Response.Cache.SetMaxAge(3600);
  Response.Cache.SetCacheability(HttpCacheability.Server);
  Response.Cache.SetSlidingExpiration(true);
  msgLabel.Text = DateTime.Now.ToString();
}
</script>

<body>
<h3>Output Cache example</font></h3>
<p>Last generated on:
<asp:label id="msgLabel" runat="server"/>
</body>
</html>
```

Output caching location

- **Specifying OutputCache enables 3 types of caching:**
  - server - pages are cached in the worker process
  - client - pages are cached in client browsers that support it
  - proxy - pages are potentially cached by downstream proxies
- **You can control where a page is cached with the 'location' attribute**

Value of Location	Cache-Control header	Expires header?	Page cached on server?
`'Any'`	`public`	`Yes`	`Yes`
`'Client'`	`private`	`Yes`	`No`
`'Downstream'`	`public`	`Yes`	`No`
`'Server'`	`no-cache`	`No`	`Yes`
`'None'`	`no-cache`	`No`	`No`

Caching multiple versions of a page

- **VaryByParam='none' means a unique instance of a page is cached per verb**
  - One for GET, one for POST
- **If your page changes rendering based on query string or post body, you can use VaryByParam to cache additional versions**
  - "*" means cache a unique page for each different post body and/or query string (beware)
  - "varname" means cache a unique page for different values of this variable (semi-colon delimited list for multiple variables)

VaryByParam values

VaryByParam value	Description
`'none'`	One version of page cached per request type (GET, POST, HEAD)
`'*'`	*n* versions of page cached based on query string and/or POST body variables
'v1'	*n* versions of page cached based on value of *v1* variable in query string or POST body
'v1;v2'	*n* versions of page cached based on value of *v1* and *v2* variables in query string or POST body

```
<%@ OutputCache Duration="60" VaryByParam="none" %>
```

```
<%@ OutputCache Duration="60" VaryByParam="*" %>
```

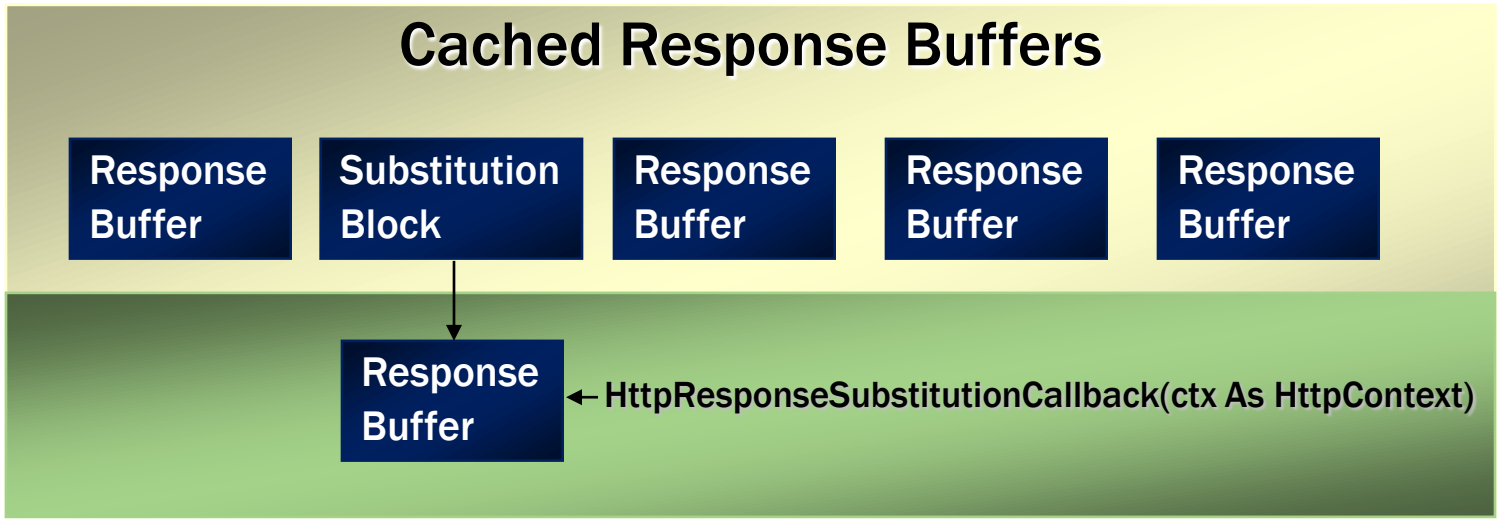
```
<%@ OutputCache Duration="60" VaryByParam="name;age" %>
```

VaryByHeader & VaryByCustom

- **Other options for creating additional cached pages**
  - VaryByHeader - any HTTP header
  - VaryByCustom - specify 'Browser' for unique cache entry for each browser type
  - VaryByCustom with GetVaryByCustomString method in HttpApplication derivative
    - Return your own string indicating whether it's a separate cache entry or not

Post-cache substitution

- **Output cache an entire page *except* xxx**
  - xxx must consist of a simple string
  - Use Response.WriteSubstitution with callback
  - Or use Substitution control with associated callback method



Post-cache substitution

- **Can elect to output cache an entire page *except* for specifically generated dynamic elements**

```
<%@ Page Language="c#" Debug="true" %>
<%@ OutputCache Duration="60" VaryByParam = "none" %>

<script runat="server">
  protected static string GetCurrentTime(HttpContext ctx) {
    return DateTime.Now.ToString();
  }
</script>

<html><body><form runat='server'>
<%= DateTime.Now %>
<br />
<% Response.WriteSubstitution(
   new HttpResponseSubstitutionCallback(GetCurrentTime)); %>

</form></body></html>
```

Cache profiles

- **Can specify output cache settings in config file and apply collections of settings to multiple pages**

web.config

```
<configuration>

  <system.web>
    <caching>
    <outputCacheSettings>
      <outputCacheProfiles>
        <add name="Default" duration="60" varyByParam="user" />
        <add name="Aggressive" duration="3600" varyByParam="*" />
        <add name="Timid" duration="5" varyByParam="none" />
      </outputCacheProfiles>
    </outputCacheSettings>
  </system.web>
</configuration>
```

pg2.aspx

```
<%@ OutputCache
    CacheProfile="Default"  %>
```

pg1.aspx

```
<%@ OutputCache
    CacheProfile="Aggressive"  %>
```

Config control over cache settings

- Application-wide cache settings now configurable
- Disk caching allows output cache entries to be written temporarily to disk if necessary

```
<caching>
  <cache disableMemoryCollection="false"
         disableExpiration="false"
         percentagePhysicalMemoryUsedLimit="90" />

  <outputCache enabled="true"
               enableFragmentCache="true"
               sendCacheControlHeader="true"
               omitVaryStar="false">
      <diskCache enabled="true" maxSizePerApp="10" />
  </outputCache>
</caching>
```

Page fragment caching

- **Common for portions of a page to remain static**
  - Menus, navigation bars, footers, headers, etc.
- **To cache a page fragment, encapsulate in .ascx and mark with OutputCache directive**
  - Shared property indicates whether one instance is cached to share across pages
  - VaryByControl indicates a control value influences unique cache entries

Page fragment caching

MyUserControl.ascx

```
<%@ Control Language="C#" %>
<%@ OutputCache Duration="60" VaryByParam="none" Shared="true" %>

<h3>User control generated at <%= DateTime.Now %></h3>
```

Client.aspx

```
<%@ Page Language="C#" %>
<%@ Register TagPrefix="PS" TagName="Myuc" Src="MyUserControl.ascx" %>
<html>
  <body>
  <form runat="server">
    <PS:Myuc id="myUcControl" runat="server" /> <br/>
    <h3>This page rendered at: <%= DateTime.Now %> </h3>
  </form>
  </body>
</html>
```

Programmatic fragment caching control

- `UserControl.CachePolicy` **provides programmatic access**
  - Call within control to influence caching

```
public sealed class ControlCachePolicy
 {
   public void SetExpires(DateTime expirationTime);
   public void SetSlidingExpiration(bool useSlidingExpiration);
   public void SetVaryByCustom(string varyByCustom);

   public bool Cached { get; set; }
   public CacheDependency Dependency { get; set; }
   public TimeSpan Duration { get; set; }
   public bool SupportsCaching { get; }
   public string VaryByControl { get; set; }
   public HttpCacheVaryByParams VaryByParams { get; }
 }
```

Data Caching

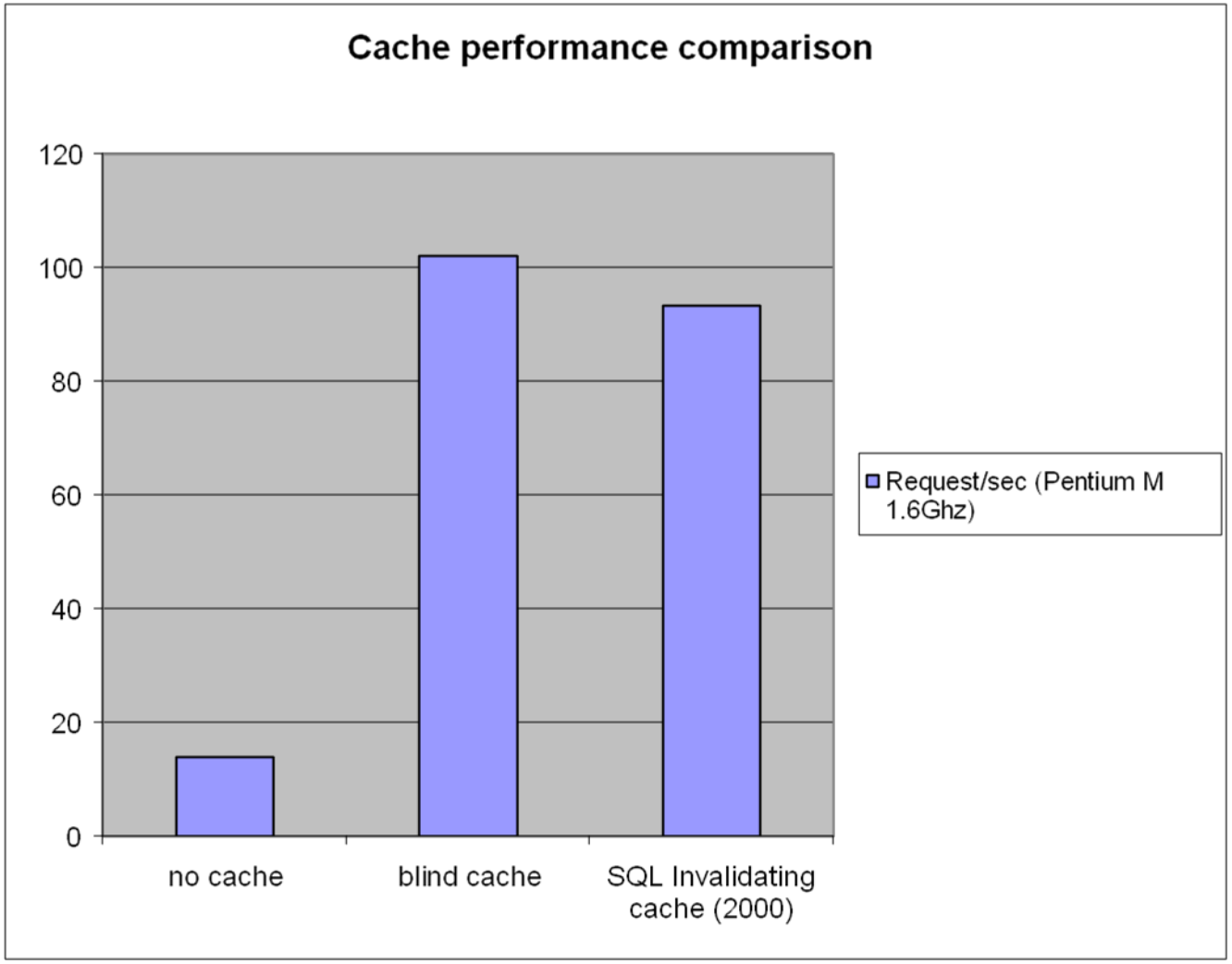
- **Internally, OutputCaching is built using a sophisticated data cache**
  - Available to you directly through Page.Cache for your own caching needs
- **Caching advantages**
  - Improve response times
  - Reduce DB round trips
  - Level of caching granularity up to you
  - Cache dependencies and timeout

Using the data cache

```
protected void Page_Load(Object src, EventArgs e)
{
  // Look in the data cache first
  DataView dv = (DataView)Cache["AuthorsDataView"];
  if (dv == null)  // wasn't there
  {
    string dsn = "server=.;trusted_connection=yes;database=pubs";
    SqlDataAdapter da =
              new SqlDataAdapter("select * from Authors", dsn);
    DataSet ds = new DataSet();
    da.Fill(ds, "Authors");
    dv = ds.Tables["Authors"].DefaultView;
    dv.AllowEdit   = false;
    dv.AllowDelete = false;
    dv.AllowNew    = false;
      // Save authors table in cache
    Cache["AuthorsDataView"] = dv;
  }

  //use dv here ...
}
```

Caching performance increase



Cache entry attributes

- When adding cache entries, several attributes can be specified
- Each insertion into the cache creates a new `CacheEntry` instance initialized with the specified attributes

Property	Type	Description
`Key`	`string`	`Unique key used to identify this entry in the cache`
`Dependency`	`CacheDependency`	`A dependency this cache entry has - either on a file, a directory, or another cache entry - which when changed, should cause this entry to be flushed`
`Expires`	`DateTime`	`A fixed date and time after which this cache entry should be flushed`
`SlidingExpiration`	`TimeSpan`	`The time between which the object was last accessed and when the object should be flushed from the cache`
`Priority`	`CacheItemPriority`	`How important this item is to keep in the cache compared to other cache entries (used when deciding how to remove cache objects during scavenging)`
`OnRemoveCallback`	`CacheItemRemoved-Callback`	`A delegate which can be registered with a cache entry for invocation upon removal`

Setting cache entry attributes

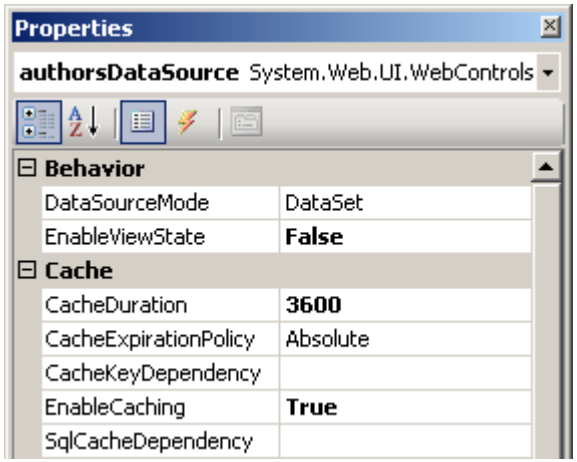
```
<script runat="server">
public void Application_OnStart()
{
  object obj = // retrieve obj to place in cache somehow
  DateTime expire = DateTime.Now.AddHours(1);
  Cache.Insert("MyVal", // key
               obj,     // object
               null,    // dependencies
               expire,  // absolute expiration
               Cache.NoSlidingExpiration, // sliding exp.
               CacheItemPriority.Default, // priority
               null);   // callback delegate
}
</script>
```

DataSource caching

- **DataSet-based data sources can easily be cached**
  - Typically combine with disabling view state for efficiency

au_id	au_lname	au_fname	phone	address	city	state	zip	contract
abc	abc	abc	abc	abc	abc	abc	abc	☐
abc	abc	abc	abc	abc	abc	abc	abc	☑
abc	abc	abc	abc	abc	abc	abc	abc	☐
abc	abc	abc	abc	abc	abc	abc	abc	☑
abc	abc	abc	abc	abc	abc	abc	abc	☐

SqlDataSource - authorsDataSource



Removing objects from the cache

- **Objects can be explicitly taken out of the cache...**
  - explicitly by calling `Remove`
  - Cache can remove item implicitly for a variety of reasons (Data expiration, Memory consumption)
  - Low priority data removed first
  - Can register for removal notification, including reason

Using removal notification callback

global.asax

```
<script runat="server">
public void Application_Start(Object src, EventArgs e)
{
  System.IO.StreamReader sr =
    new System.IO.StreamReader("pi.txt");
  string pi = sr.ReadToEnd();
  Context.Cache.Add("pi", pi, null,
                    Cache.NoAbsoluteExpiration,
                    TimeSpan.Zero,
                    CacheItemPriority.Normal,
                    new CacheItemRemovedCallback(this.OnRemove));
}

public void OnRemove(string key, object val,
                     CacheItemRemovedReason r)
{
        // respond to cache removal here
}
</script>
```

Using cache dependencies

- **Cache dependencies can ensure that data is not stale**
  - Cache entry can be flushed when a file changes
  - Cache entry can be flushed when a directory changes
  - Cache entry can be flushed when another cache entry is removed

Using cache dependencies

```
<%@ Application Language="C#" %>
<script runat="server">
public void Application_OnStart()
{
  System.IO.StreamReader sr =
    new System.IO.StreamReader(Server.MapPath("pi.txt"));
  string pi = sr.ReadToEnd();

  CacheDependency piDep =
    new CacheDependency(Server.MapPath("pi.txt"));
  Context.Cache.Add("pi", pi, piDep,
                    Cache.NoAbsoluteExpiration,
                    Cache.NoSlidingExpiration,
                    CacheItemPriority.Default,null);
}
</script>
```

SQL cache dependencies

- **Cache coherency problems easily introduced**
  - Avoid by flushing stale data immediately
- **SQL cache dependencies offer a compelling solution**
  - Cache entry tied to database table (or row)
  - Data flushed whenever change is detected
  - Supported in SQL 7/2000 with polling (requires additional setup)
  - Supported in SQL 2005/2008 with service broker
  - Extensible architecture with 'pluggable' cache dependency class

Using SQL cache dependencies

- **Fine-grained notification**
  - Can tie to changes in command results
  - "Push" model with service broker (not poll)

```
// Populate authors in cache with SQL dependency
using (SqlConnection conn = new SqlConnection(dsn))
{
  SqlCommand cmd = new SqlCommand(
         "SELECT au_id, au_fname, au_lname FROM dbo.Authors", conn);
  SqlCacheDependency scd = new SqlCacheDependency(cmd);
  SqlDataAdapter da = new SqlDataAdapter(cmd);
  DataSet authorsDs = new DataSet();
  da.Fill(authorsDs, "authors");

  Cache.Insert("au", authorsDs, scd);
}
```

Listening for SqlDependency notifications

- **Before you can use SqlCacheDependency you must open a listener for *each* connection string you use**
  - Typically done in Application_Start within global.asax
  - Call System.Data.SqlClient.SqlDependency.Start(dsn)

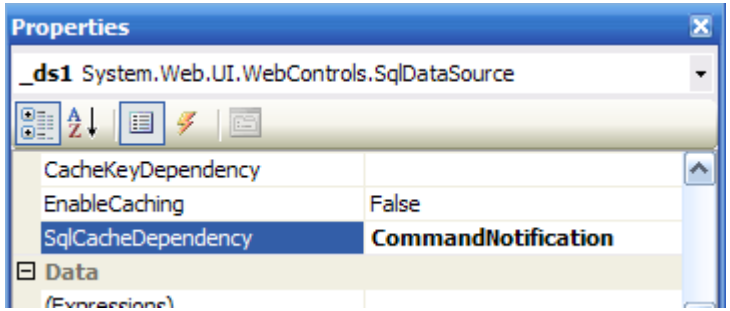
```
void Application_Start(object sender, EventArgs e)
{
    // Code that runs on application startup
    // Start sql dependency listener for sql 2005 / 2008

    string dsn = ConfigurationManager.ConnectionStrings["myDsn"]. ConnectionString;
    System.Data.SqlClient.SqlDependency.Start(dsn);
}
```

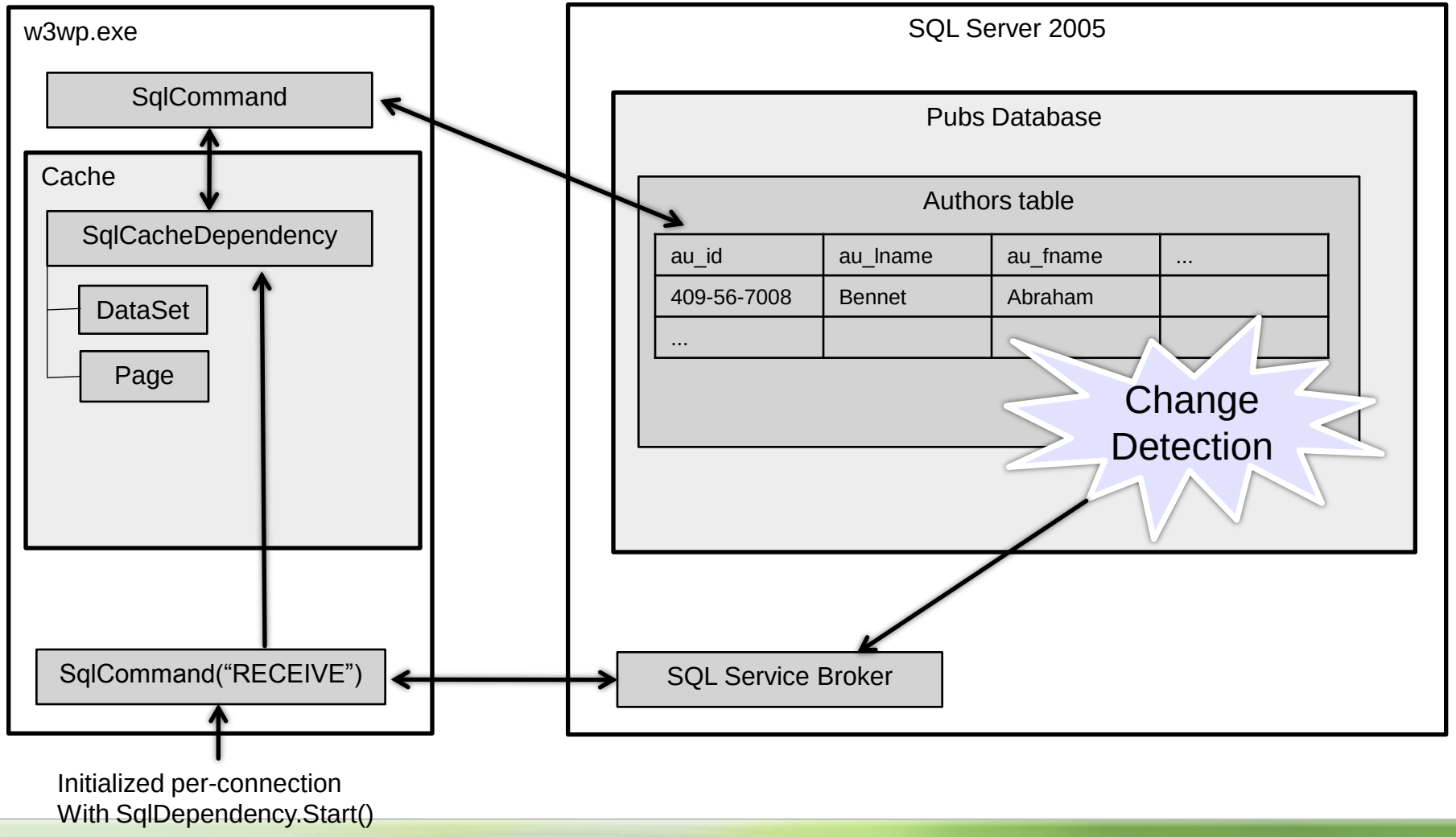
Declarative cache dependencies

- **Use special 'CommandNotification' string for SQL 2005/2008**
  - SQL queries somewhat restricted (no 'SELECT *' for example)

```
<%@ OutputCache Duration="3600" VaryByParam="none"
                SqlDependency="CommandNotification" %>
```



SQL cache dependence lifecycle



Diagnostic Checklist for SQL Cache Dependencies

- **Verify SqlDependency.Start() called for each connection string**
- **Verify service broker enabled**
  (ALTER DATABASE <db> SET ENABLE_BROKER **)**
- **Verify SQL identity has permission to register for notifications**
  (GRANT SUBSCRIBE QUERY NOTIFICATIONS TO <username>)
- **Query constraints:**
  - Column names explicit (no "*")
  - Full 2-part names of tables (schema.object) like dbo.authors
  - No aggregate functions (SUM, AVG, COUNT, …)
  - No windowing or ranking functions (like ROW_NUMBER)
  - No references to temporary tables or views
  - No subqueries, outer joins, or self joins
  - No *text, ntext,* or *image* column types
  - No DISTINCT, HAVING, CONTAINS, or FREETEXT
  - SProcs may not use SET NOCOUNT ON

Summary

- **Output caching**
  - Entire page caching
- **Post-cache substitution**
  - Inverse output caching
- **Data caching**
  - Fine-grained caching
- **Data source caching**
  - Simple, flexible way of caching declarative data sources
- **SQL cache dependencies**
  - Solution to very common cache coherency problem