Scaling ASP.NET websites from thousands to millions of users

Lessons learnt from scaling Web 2.0 apps to millions of users

Omar AL Zabir

Chief Architect, SaaS Platform, BT

omaralzabir.com

omaralzabir@gmail.com

Twitter.com/omaralzabir

Book "Building Web 2.0 portal using ASP.NET 3.5" from O'Reilly

How to blow up a website

- Get some super fast fiber broadband connection.
- Run this on couple of quad core desktops:

```
for (int j = 0; j < 100; j ++)
{
  for (int i = 0; i < 100; i ++)
  {
    var client = new WebClient();
    client.DownloadStringAsync(
          new Uri("http://www.microsoft.com/MISReport.aspx");
  }
  Thread.Sleep(500);
}
```

Prevent App Level DOS attack

- Application Level DOS attacks are attempts to hit your expensive pages too frequently so that you have 100% CPU and your site goes down. For ex, hitting some expensive Report page continuously.
- You can run out of ASP.NET Threads and stop responding completely to any request.
- I'm not talking about Network level DOS attacks like TCP SYN flood or DDOS attacks that hardware firewalls can prevent.
- Firewalls don't block legitimate requests.

Prevent App level DOS attacks

- Protect only expensive pages. Pages that are unlikely to get hit too many times within a short duration.
- Build a HttpModule and hook on OnInit event.
- Store which IP is making how many number of hits in last 10 minutes. Store the table in some in-memory cache.
- If the threshold has exceeded, stop responding to that IP. Call *Response.End()*
- Solution is here:
  http://tinyurl.com/omarDOS

IP	Hits in last 10 mins
128.0.0.1	500
128.0.0.2	345

ASP.NET ProcessModel Optimization

- ASP.NET ProcessModel Defaults
  - maxWorkerThreads = 20
  - maxIOThreads = 20
  - memoryLimit = 60
- Nowadays, servers are way too powerful. You don't need to be conservative.
- Change default process model setting in machine.config to make best use of CPU power.

```
<processModel
   enable="true"
   timeout="Infinite"
   idleTimeout="Infinite"
   shutdownTimeout="00:00:05"
   requestLimit="Infinite"
   requestQueueLimit="5000"
   restartQueueLimit="10"
   memoryLimit="60"
   webGarden="false"
   cpuMask="0xffffffff"
   userName="machine"
   password="AutoGenerate"
   logLevel="Errors"
   clientConnectedCheck="00:00:05"
   comAuthenticationLevel="Connect"
   comImpersonationLevel="Impersonate"
   responseDeadlockInterval="00:03:00"
   responseRestartDeadlockInterval="00:03:00"
   autoConfig="false"
   maxWorkerThreads="100"
   maxIoThreads="100"
   minWorkerThreads="40"
   minIoThreads="30"
   serverErrorMessageFile=""
   pingFrequency="Infinite"
   pingTimeout="Infinite"
   asyncOption="20"
   maxAppDomains="2000"
/>
```

ASP.NET Pipeline Optimization

- Default ASP.NET Pipeline has several components that intercept each and every request. Thus they add extra processing overhead on every request.
- Multiply the overhead by hundreds of requests per second – you get a significant overhead.

```
<httpModules>
  <add name="OutputCache" type="System.Web.Caching.OutputCacheModule" />
  <add name="Session" type="System.Web.SessionState.SessionStateModule" />
  <add name="WindowsAuthentication"
       type="System.Web.Security.WindowsAuthenticationModule" />
  <add name="FormsAuthentication"
       type="System.Web.Security.FormsAuthenticationModule" />
  <add name="PassportAuthentication"
       type="System.Web.Security.PassportAuthenticationModule" />
  <add name="UrlAuthorization" type="System.Web.Security.UrlAuthorizationMo
  <add name="FileAuthorization" type="System.Web.Security.FileAuthorizationM
  <add name="ErrorHandlerModule" type="System.Web.Mobile.ErrorHandlerModule
                           System.Web.Mobile, Version=1.0.5000.0,
                           Culture=neutral, PublicKeyToken=b03f5f7f11d50a
</httpModules>
```

ASP.NET Pipeline Optimization

- Remove what you don't need.
- If you are using Forms Authentication, SQL Server storage, no web.config based role based permission then you can remove most of them:

```
<httpModules>
        <!-- Remove unnecessary Http Modules for faster pipeline -->
        <remove name="Session" />
        <remove name="WindowsAuthentication" />
        <remove name="PassportAuthentication" />
        <remove name="AnonymousIdentification" />
        <remove name="UrlAuthorization" />
        <remove name="FileAuthorization" />
</httpModules>
```

Prevent large ASP.NET cookies on static content

- Each and every request, even static files, get the ASP.NET cookies sent.
- 517 bytes of worthless data per request.
- Avg pages have 40 resources. 40 x 517 = 20 KB.
- 1M page view = 20 GB of data upload to server.

Cookie:

.DBANON=w3kYczsH8Wvzs6MgryS4JYEF0N-8ZR6aLRSTU9KwVaGaydD6WwUHD7X9tN8vBgjgzKf3r3SJHusTYFjU85yYfnunyCeuExcZs895JK9Fk1HS68ksGwm3QpxnRZvpDBAfJKEUKee2OTlND0gi43qwwtIPLeY1; ASP.NET_SessionId=bmnbp155wilotk45gjhitoqg; .DBAUTH12=2A848A8C200CB0E8E05C6EBA8059A0DBA228FC5F6EDD29401C249D237812344C15B3C5C57D6B776037FAA8F14017880E57BDC14A7963C58B0A0B30229AF0123A6DF56601D814E75525E7DCA9AD4A0EF200832B39A1F35A5111092F0805B0A8CD3D2FD5E3AB6176893D86AFBEB68F7EA42BE61E89537DEAA3279F3B576D0C44BA00B9FA1D9DD3EE985F37B0A5A134ADC0EA9C548D

Prevent ASP.NET cookies on static content

- Setup a new website in IIS, map to the same code folder.
- Map static.yoursite.com host header to that website.
- Prefix all css, js, image links with http://static.yoursite.com/
- Add a Global.asax's EndResponse event on the new website.

```
HttpContext context = HttpContext.Current;
if (context.Request.Url.Host == "static.yoursite.com")
{
    List<string> cookiesToClear = new List<string>();
    foreach (string cookieName in context.Request.Cookies)
    {
      HttpCookie cookie =
    context.Request.Cookies[cookieName];
      cookiesToClear.Add(cookie.Name);
    }
    foreach (string name in cookiesToClear)
    {
      HttpCookie cookie = new HttpCookie(name,
    string.Empty);
      cookie.Expires = DateTime.Today.AddYears(-1);
      context.Response.Cookies.Set(cookie);
```

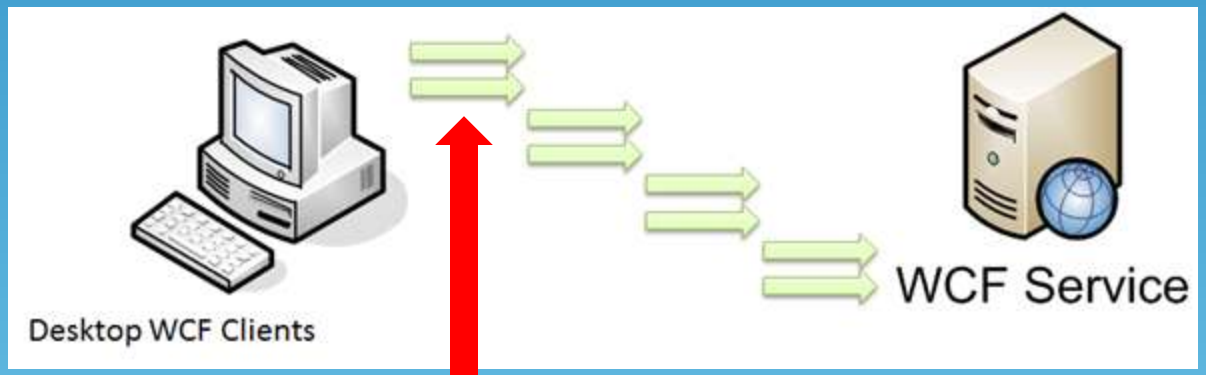
System.net optimization

- If you are using *HttpWebRequest* or *WebClient* or any other TCP/IP operation, increase the max connection limit.
- Default is 2 per IP.
- WCF service calls are limited by this setting.
- Unless you suspect rogue clients, set some reasonably high number on webservers, but moderate number on desktop clients.

```
<system.net>
  <connectionManagement>
    <add address="*" maxconnection="100" />
  </connectionManagement>
</system.net>
```

System.net default setting is suboptimal

Bottleneck



Max 2 concurrent calls

ASP.NET Profile Provider

- Anonymous Provider creates one anonymous user in database on every first hit to *Profile* object's Custom properties:

```
private GetPageflake(string source, string pageID, string userUniqueName)
{
  if( Profile.IsAnonymous )
  {
   using (new TimedLog(Profile.UserName,"GetPageflake"))
   {
```

- The SP *aspnet_Profile_GetProperties* gets called when you access *Profile* object for first time in a request. And this SP is slooooooooow!

ASP.NET Profile Provider

- The slow SP that gets fired when you access custom Profile properties:

```
CREATE PROCEDURE [dbo].[aspnet_Profile_GetProperties]
    @ApplicationName    nvarchar(256), @UserId …
AS
BEGIN
    DECLARE @ApplicationId uniqueidentifier
    SELECT @ApplicationId = NULL
    SELECT @ApplicationId = ApplicationId
            FROM dbo.aspnet_Applications
                WHERE LOWER(@ApplicationName) = LoweredApplicationName


…
…
…
    IF (@@ROWCOUNT > 0)
    BEGIN
          UPDATE dbo.aspnet_Users
        SET    LastActivityDate=@CurrentTimeUtc
        WHERE UserId = @UserId
    END
END
```

ASP.NET Profile Provider

- The slow SP's execution plan

```
Table 'aspnet_Applications'. Scan count 1, logical reads 2, physical reads 0,
                        read-ahead reads 0, lob logical reads 0, lob physical
                            reads 0, lob read-ahead reads 0.
(1 row(s) affected)
Table 'aspnet_Users'. Scan count 1, logical reads 4, physical reads 0,
                        read-ahead reads 0, lob logical reads 0, lob physical
                            reads 0, lob read-ahead reads 0.

(1 row(s) affected)
(1 row(s) affected)
Table 'aspnet_Profile'. Scan count 0, logical reads 3, physical reads 0,
                        read-ahead reads 0, lob logical reads 0, lob physical
                            reads 0, lob read-ahead reads 0.
(1 row(s) affected)
Table 'aspnet_Users'. Scan count 0, logical reads 27, physical reads 0,
                        read-ahead reads 0, lob logical reads 0, lob physical
                            reads 0, lob read-ahead reads 0.
(1 row(s) affected)
```

```
Query 4: Query cost (relative to the batch): 79%
UPDATE dbo.aspnet_Users SET LastActivityDate=@CurrentTimeUtc WHERE UserId =…
```

ASP.NET Profile Provider

- Don’t update *LastActivityDate* when Profile object is loaded. Do it only when Profile object is updated.
- Update once every hour or so. If *LastActivityDate* < *DateTime.Now.AddHours(-1)* then update. No need to do per request or too frequently.
- Hard code the Application ID to avoid one lookup inside the SP.
- tinyurl.com/omarGetProp

ASP.NET Membership

- ASP.NET Membership Provider Stored Procs use default transaction isolation level, which is Serializable.

Transaction (Process ID ##) was deadlocked on lock resources with another process and has been chosen as the deadlock victim. Rerun the transaction.

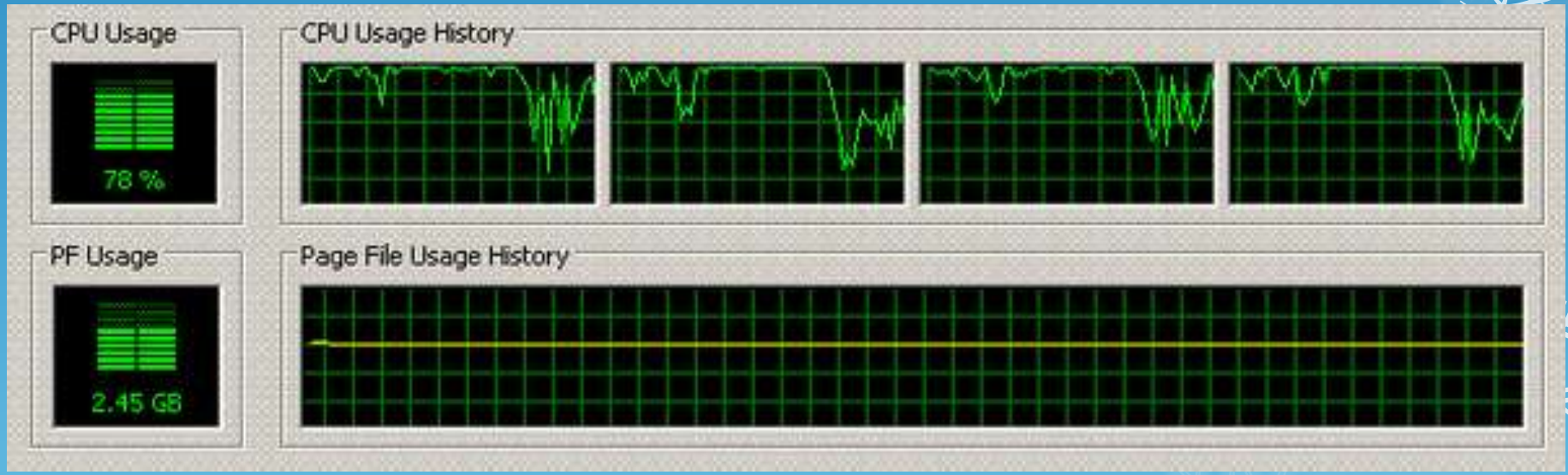
Timeout Expired. The Timeout Period Elapsed Prior To Completion Of The Operation Or The Server Is Not Responding.

```
ALTER PROCEDURE [dbo].[aspnet_Profile_GetProperties]
@ApplicationName nvarchar(256),
@UserName nvarchar(256),
@CurrentTimeUtc datetime
AS
BEGIN

SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;
```

ASP.NET Membership Query

- Common queries that brings down hell:
  - Select * from aspnet_users where UserName = 'blabla'
  - Select * from aspnet_membership where Email = "someone@somewhere.com"
- What's wrong with these queries?



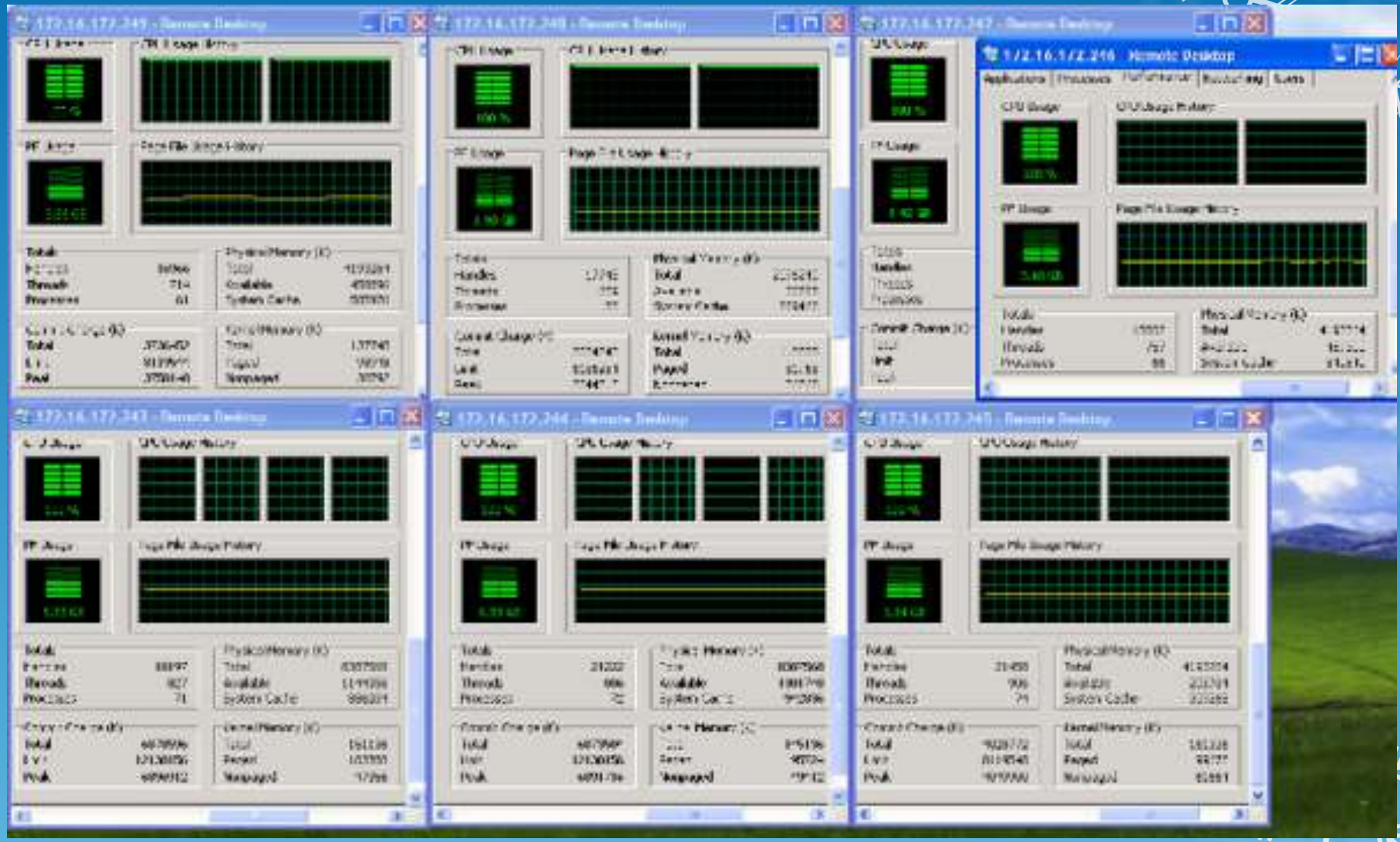
ASP.NET Membership Queries

- Let's look at the indexes:
- Table: *aspnet_users*
  - Clustered Index = ApplicationID, LoweredUserName
  - NonClustered Index = ApplicationID, LastActivityDate
  - Primary Key = UserID
- Table: *aspnet_membership*
  - Clustered Index = ApplicationID, LoweredEmail
  - NonClustered = UserID
- Table: *aspnet_Profile*
  - Clustered Index = UserID

- DO NOT use Email or UserName fields in WHERE clause. They are not part of the index instead LoweredUserName and LoweredEmail fields are in conjunction with ApplicationID field. All queries must have ApplicationID in the WHERE clause.

A love story

- .NET 3.0 was released, woohoo!
- WCF!
- Lambda Expressions!!
- Linq to SQL!!!
  - Upgraded to .NET 3.0, top to bottom.
  - Major deployment over the weekend.
  - Monday 9 AM, peak traffic.
  - No response from site.



~100% on all web servers

Linq to SQL is not suitable for high volume web applications

- Linq to SQL is not optimized for web application.
- No disconnected entity support.
- Entities are not serializable.
- Linq to sql expressions consume high CPU when compiled to SQL.

```
var query = from widget in dc.Widgets
where widget.ID == id && widget.PageID == pageId
select widget;

var widget = query.SingleOrDefault();
```

How bad Linq to SQL is?

So I dug into the call graph a bit and found out the code causing by far the most damage was the creation of the LINQ query object for every call! The actual round trip to the database paled in comparison. Now that was, again, a *huge* surprise. Check out the hit counts on this call – holy cow!

	Time (sec.)	Time with ...	Hit count
	106.2794	129.8808	1176879
	0.0004	8.1200	417
	0.0001	3.5834	417
	0.0001	2.0620	417
	0.0002	1.4915	417
TypeSyste...	0.0005	1.3781	417
	0.0002	1.0414	417
	0.0001	0.8036	834
	0.0001	0.6159	417

1,176,879 calls to SqlVisitor.Visit. Over a million calls to anything for loading a page three times can't be good.

Source: JD Conley's blog

Fixing Linq to SQL

- Convert all queries to Compiled Queries.
- tinyurl.com/omarLINQ

Linq to SQL transaction deadlocks

- Large table, high read and medium write, causes query timeouts, high locks, transaction deadlock because of SERIALIZATION isolation level.

Transaction (Process ID ##) was deadlocked on lock resources with another process and has been chosen as the deadlock victim. Rerun the transaction.

Timeout Expired. The Timeout Period Elapsed Prior To Completion Of The Operation Or The Server Is Not Responding.

```
using (var db = new YourDataContext2())
{
   db.Connection.Open();
   db.ExecuteCommand("SET TRANSACTION ISOLATION LEVEL READ
UNCOMMITTED;");

   var user = db.aspnet_Users.First();
   var pages = user.Pages.ToList();
}
```

Linq to SQL transaction deadlocks

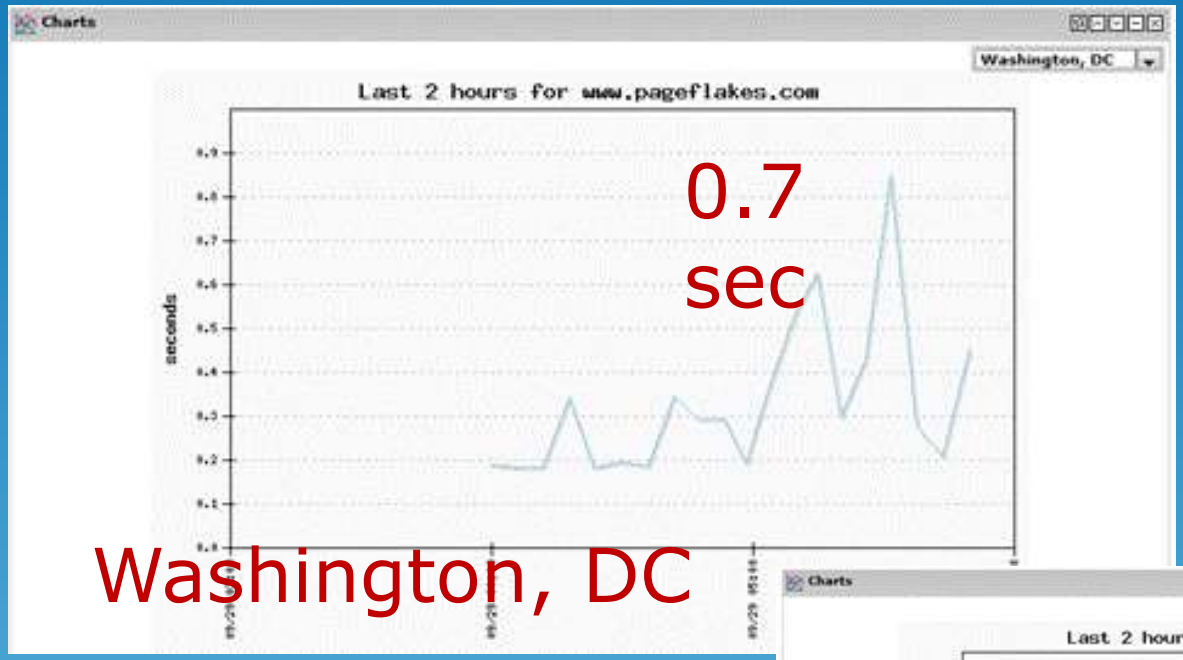
- Connection does not close!
- Bug in Data Context code.
- Solution is to override the Dispose.

```
class YourDataContext2 : YourDataContext, IDisposable {
 public new void Dispose() {
   if (base.Connection != null)
    if (base.Connection.State != System.Data.ConnectionState.Closed) {
       base.Connection.Close();
       base.Connection.Dispose();
    }
   base.Dispose();
 }
}
```

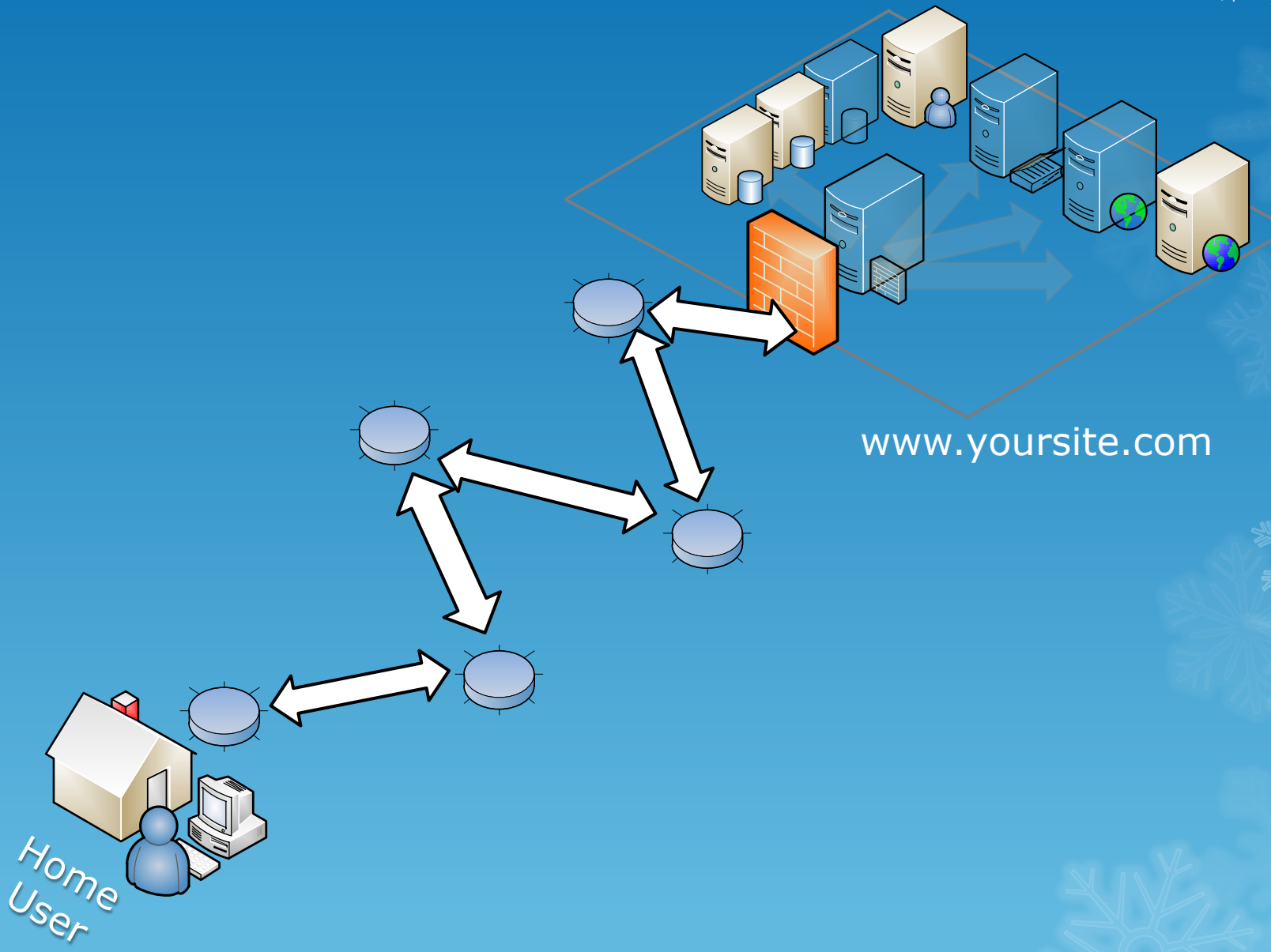
Content Delivery Network (CDN)

- CDN cache and deliver content from their servers that are closest to users' computers. The closer servers are to end user, the shorter roundtrip time for every request. For ex, Akamai has servers almost every city in the world.
- Content from CDN nodes get served faster with lower latency than coming from your servers.
- CDN Nodes have better caching and compression algorithms.
- CDN nodes can offload your server and network from delivering static files. Thus better throughput for dynamic content.

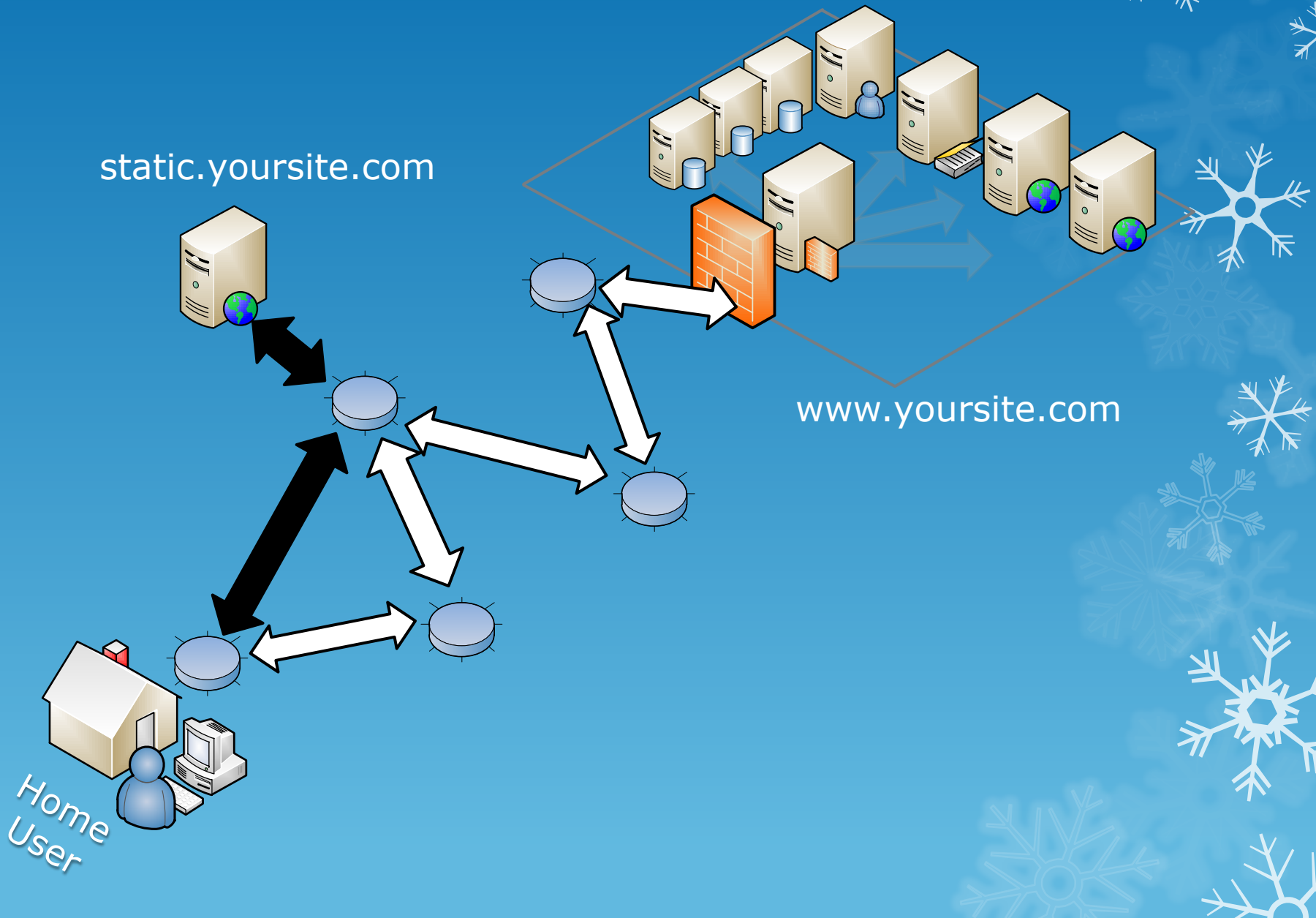
Content Delivery Network



Without CDN



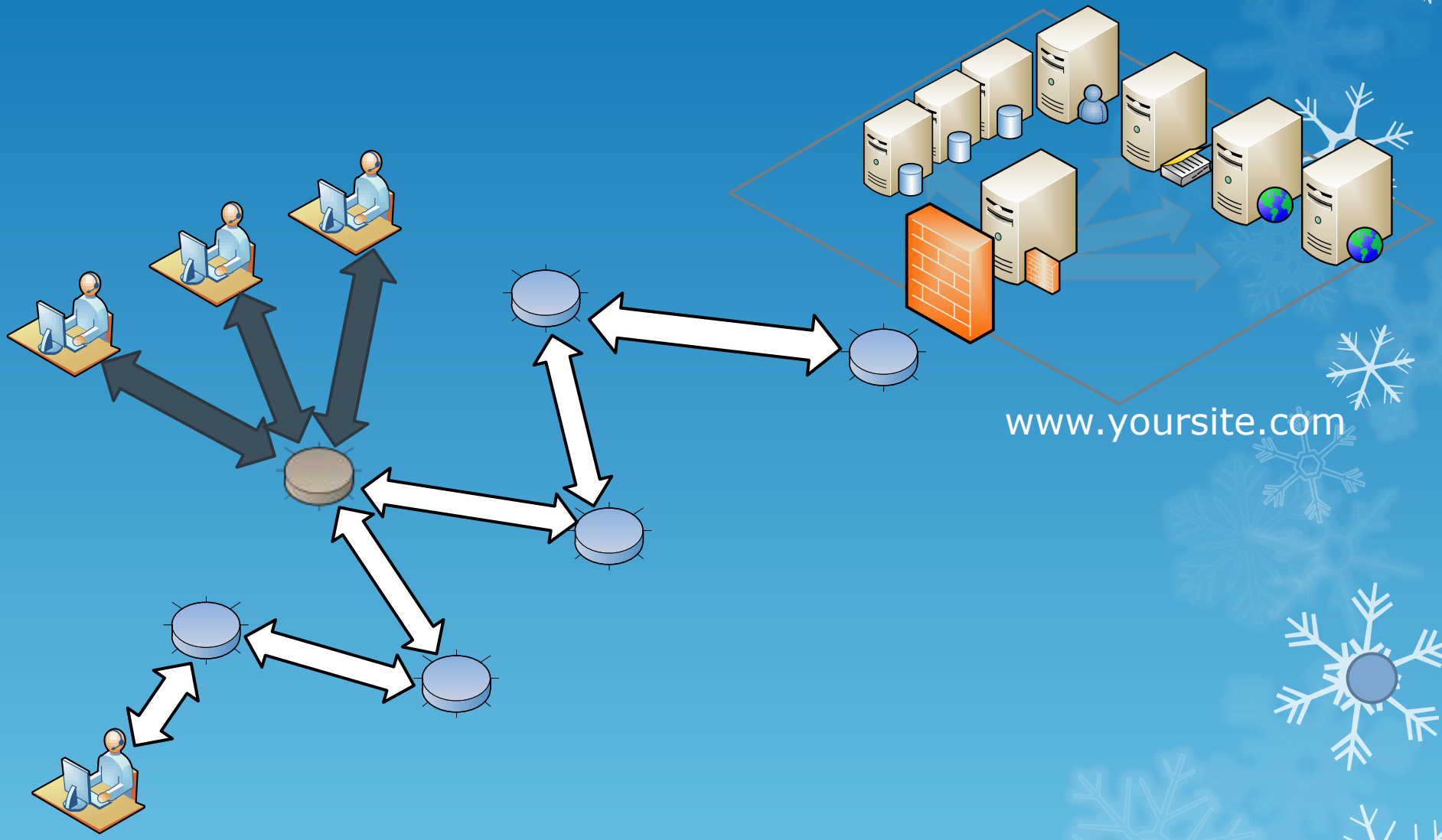
How CDN works



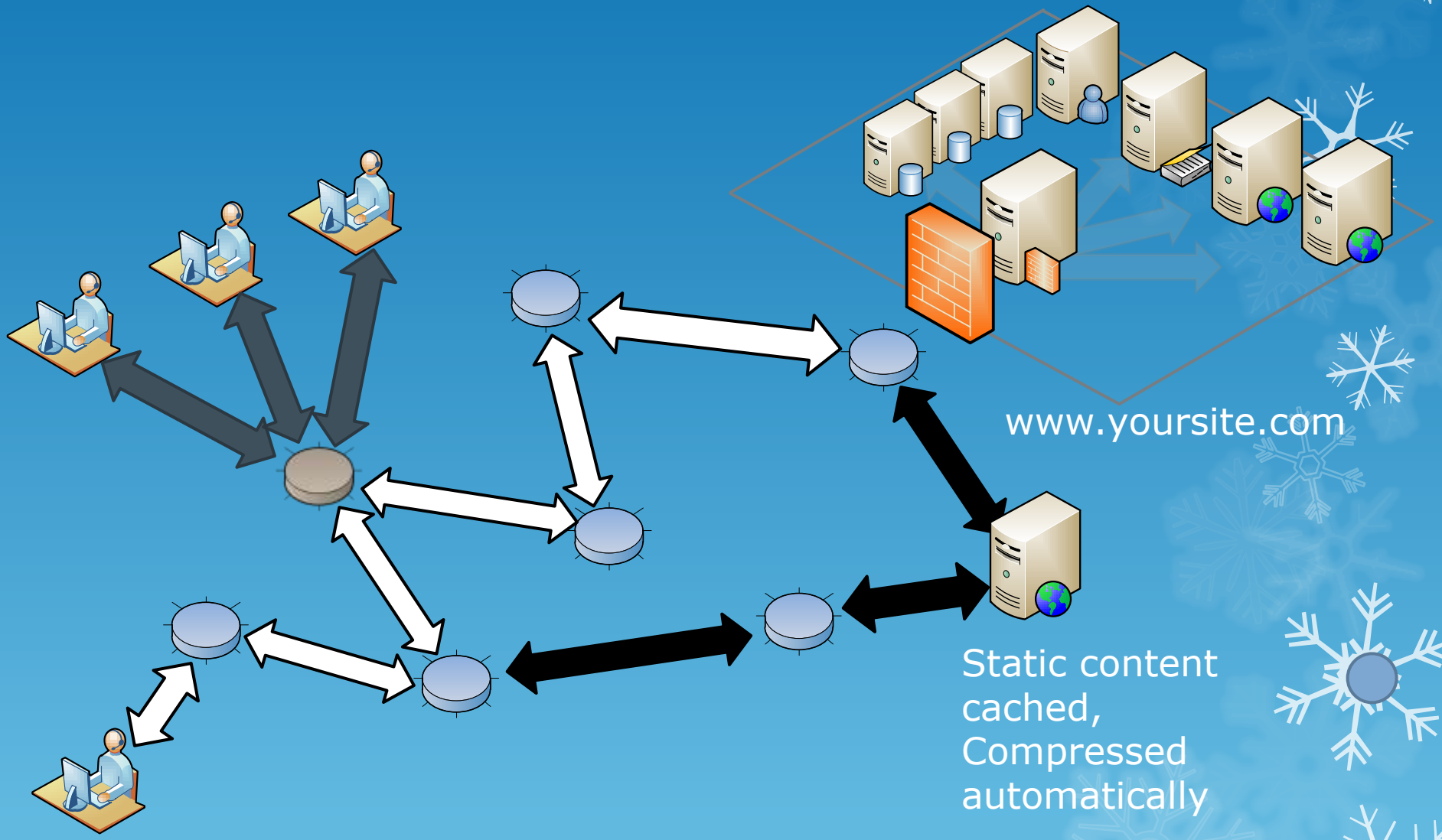
Two types of CDN

- Static – you upload the files to CDN and they give you an URL. E.g. yoursite.cachefly.net/logo.gif
- Dynamic – Host your dynamic application behind the CDN. For ex, Edgecast and Panther Express.
  - Very cheap - $0.2/GB

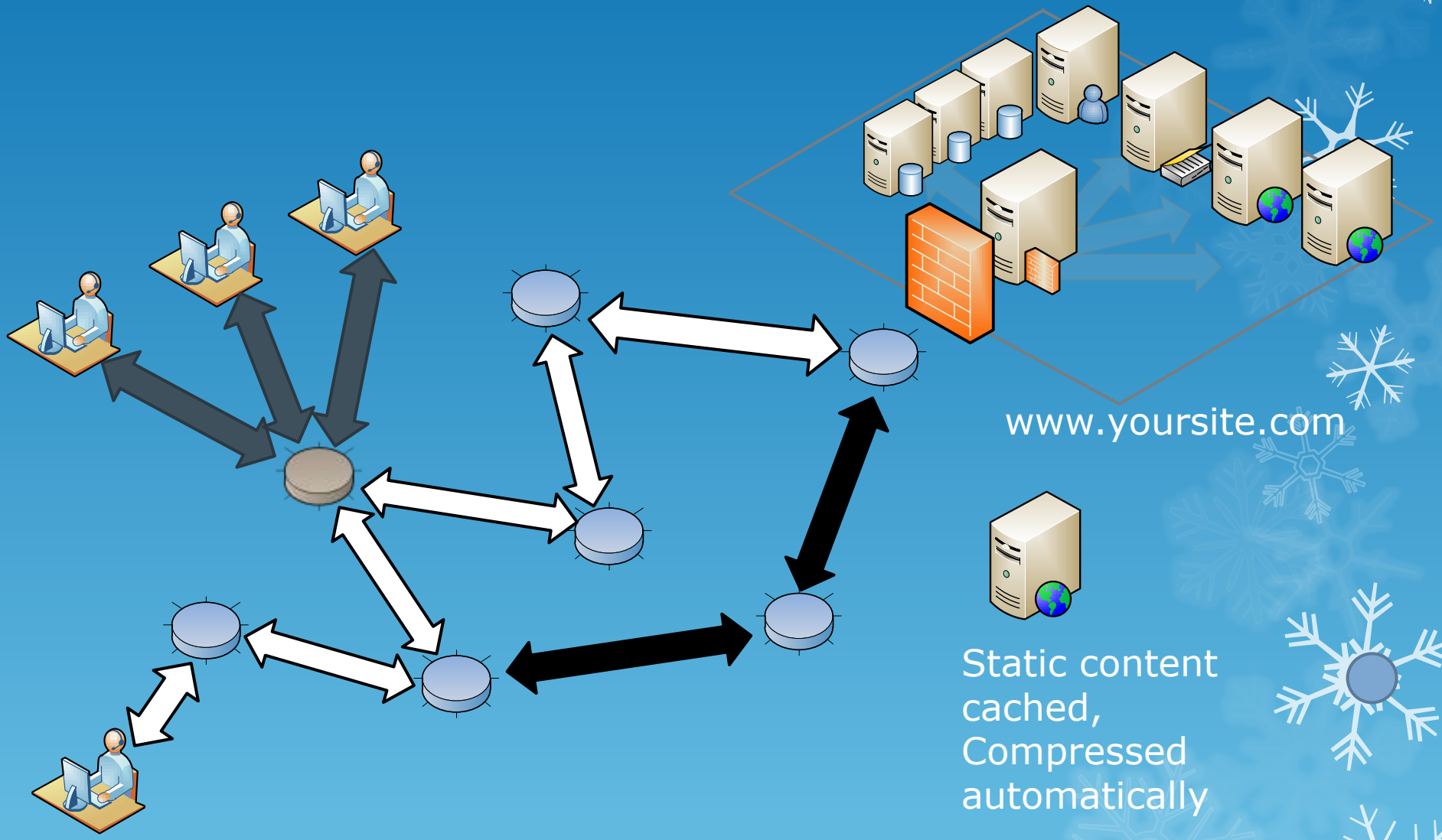
How Dynamic CDN works



How Dynamic CDN works



How Dynamic CDN works



Static content cached, Compressed automatically

13 disasters for production websites

1. **Faulty hard drive supplied by supplier, data corruption within weeks.**
2. **Controller malfunctions and corrupts all disks in the same controller.**
3. **RAID malfunction.**
4. **CPU overheated and burned out.**
5. **Firewall went down.**
6. **Remote Desktop stopped working after a patch installation.**
7. **Remote Desktop max connection exceeded. Cannot login anymore to servers.**
8. **Database got corrupted while we were moving the production database from one server to another over the network.**
9. **One developer deleted the production database accidentally while doing routine work.**
10. **Support crew at hosting service formatted our running production server instead of a corrupted server that we asked to format.**
11. **Windows got corrupted and was not working until we reinstalled.**
12. **DNS goes down. Don't get domain from GoDaddy.**
13. **Internet backbone goes down in different part of the world.**

http://tinyurl.com/omar13

Conclusion

- ASP.NET out of the box, does not scale for millions of hits.
- Must make the hacks at code, database and configuration level to get it to scale.
- That’s reality for any technology, not ASP.NET specific.