

Digital Publishing on the Bus

Just a few war stories...



David Boike
Particular Software

@DavidBoike
make-awesome.com



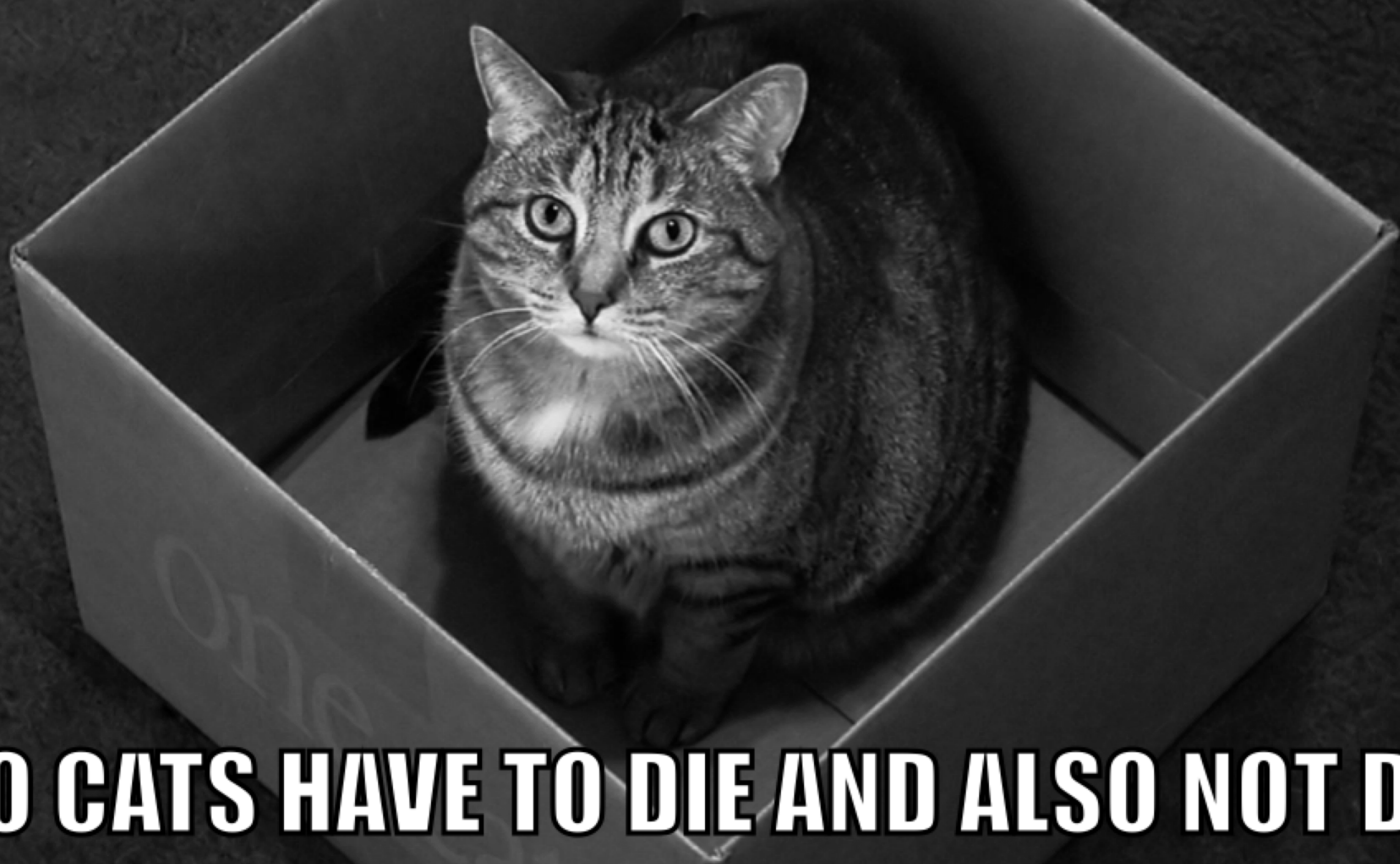




Particular
Software



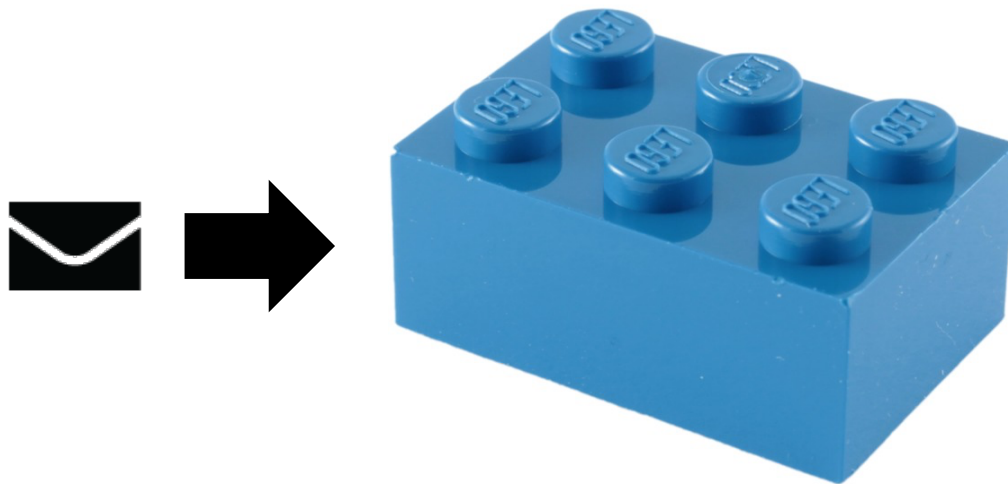
THIS IS NOT QUANTUM MECHANICS



NO CATS HAVE TO DIE AND ALSO NOT DIE



```
public class SomeHandler :  
    IHandleMessages<SomeMessage>  
{  
    public void Handle(SomeMessage message)  
    {  
        // Your turn!  
    }  
}
```



```
public class SomeHandler :
  IHandleMessages<SomeMessage>
{
    public IBus Bus { get; set; }

    public void Handle(SomeMessage message)
    {
        // Do some stuff

        Bus.Send(new NewCommand());
    }
}
```

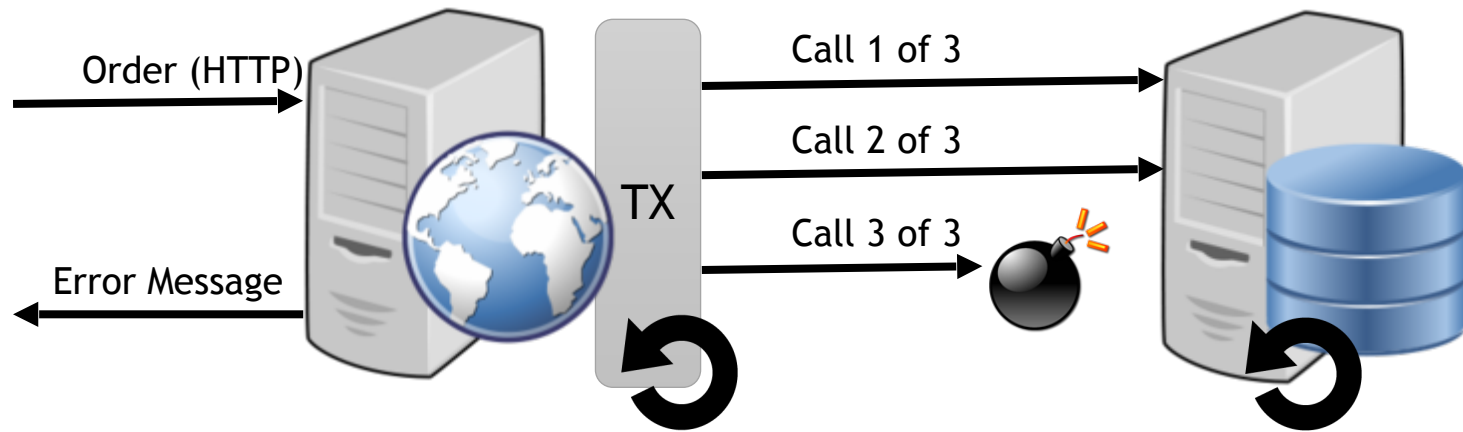


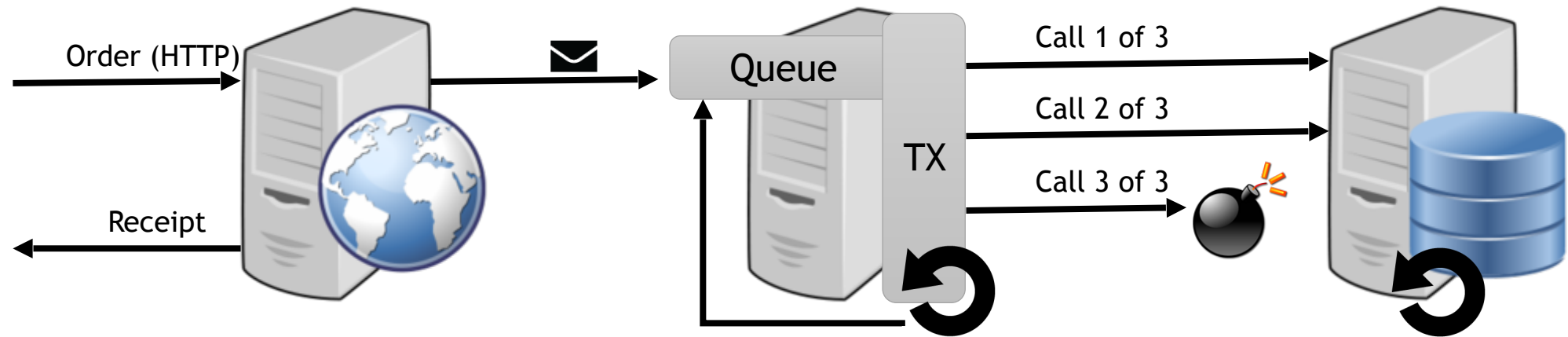

```
public class SomeHandler :
  IHandleMessages<SomeMessage>
{
  public IBus Bus { get; set; }

  public void Handle(SomeMessage message)
  {
    // Do some stuff

    Bus.Publish(new NewEvent());
  }
}
```









POPULAR CONTENT



HIGH TRAFFIC WRITES

Popular Content

- INSERT statements do not scale with high traffic
- UPDATE is marginally better
- Either option couples web performance to database performance
- When database is backed up...
 - Tracking using `<script />` blocks page rendering!
 - Better as transparent tracking pixel
 - Too many threads busy serving tracking pixels still reduces overall performance of website



Popular Content

```
public class TrackingController
{
    public ActionResult TrackRead(Guid contentId)
    {
        StoryTracking.InsertInDb(contentId,
DateTime.UtcNow);

        return new TransparentPixelResult();
    }
}
```



Popular Content

```
public class TrackingController
{
    public IBus Bus { get; set; }

    public ActionResult TrackRead(Guid contentId)
    {
        // StoryTracking.InsertInDb(contentId,
        DateTime.UtcNow);

        Bus.Send(new TrackStoryRead
        {
            ContentId = contentId,
            Time = DateTime.UtcNow
        });

        return new TransparentPixelResult();
    }
}
```



Popular Content

```
public class StoryReadTracking :  
    IHandleMessages<TrackStoryRead>  
{  
    public void Handle(TrackStoryRead msg)  
    {  
        StoryTracking.InsertInDb(msg.ContentId, msg.Time);  
    }  
}
```



Popular Content

If for some reason handler is down for an extended period, information older than one hour is useless.

```
[TimeToBeReceived("1:00:00")]  
public class TrackStoryRead  
{  
    public Guid ContentId { get;  
set; }  
    public DateTime Time {get; set;}  
}
```



What Happened?

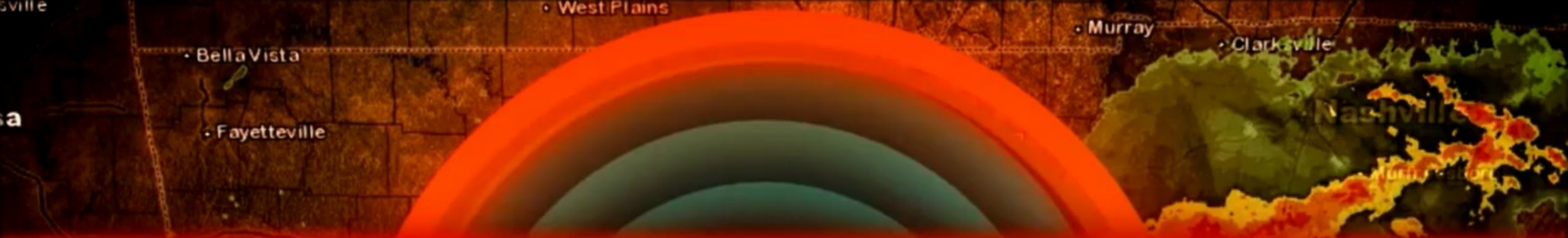
- Transparent pixel to avoid blocking page render
- Web request drops message in MSMQ Outgoing Queue
- MSMQ handles store & forward
- Centralized endpoint processes messages
- *WARNING: This will not work for non-MSMQ!*



Benefits

- Coupling between Web & DB removed
- Web response is near instant
- Queue smoothes over spikes in traffic
- Outdated messages automatically removed





SEVERE WEATHER



ALERT

Making lists is hard

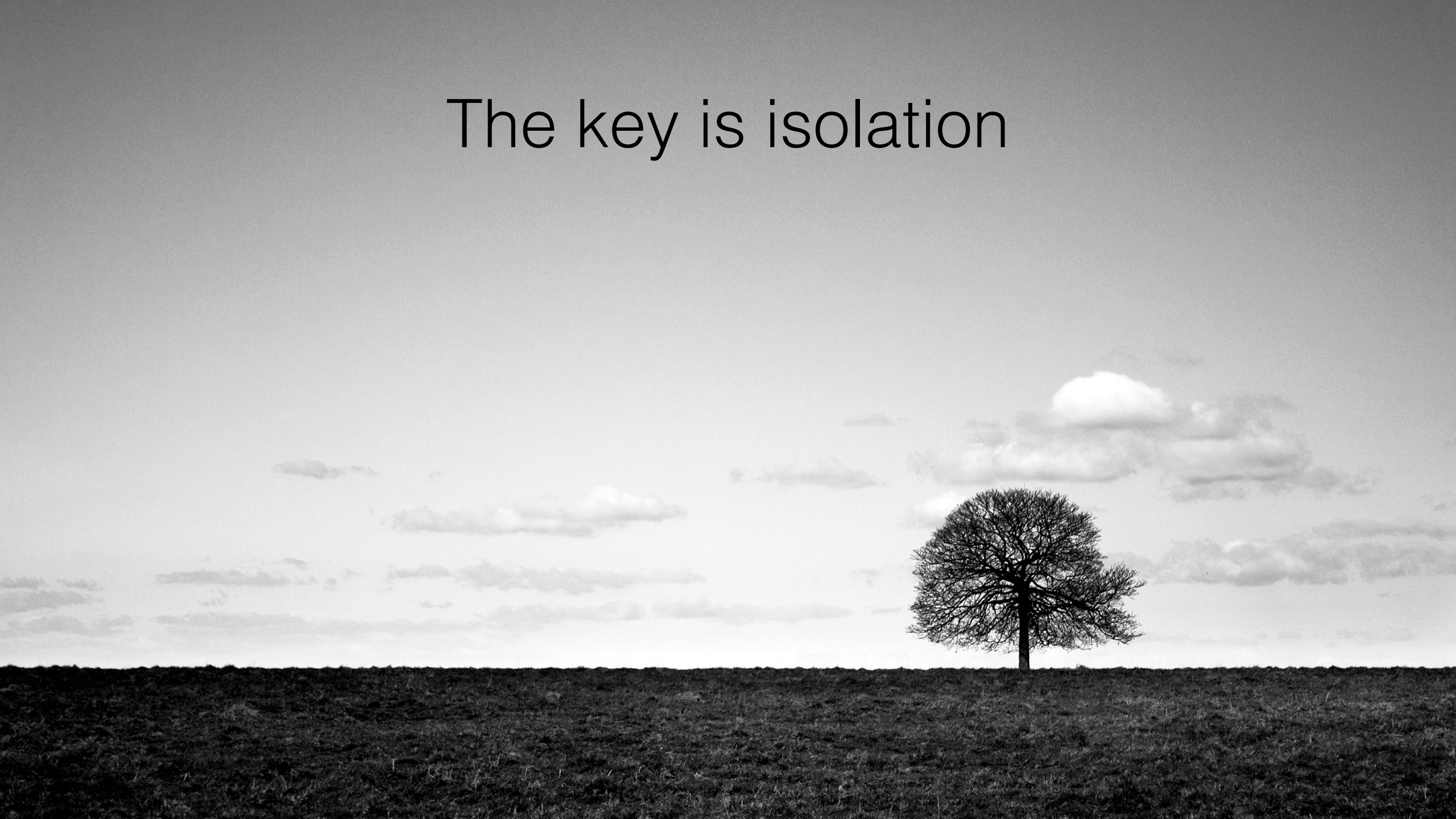
1. Save preferences in local database
2. Notify email service
3. Notify 3rd party SMS provider

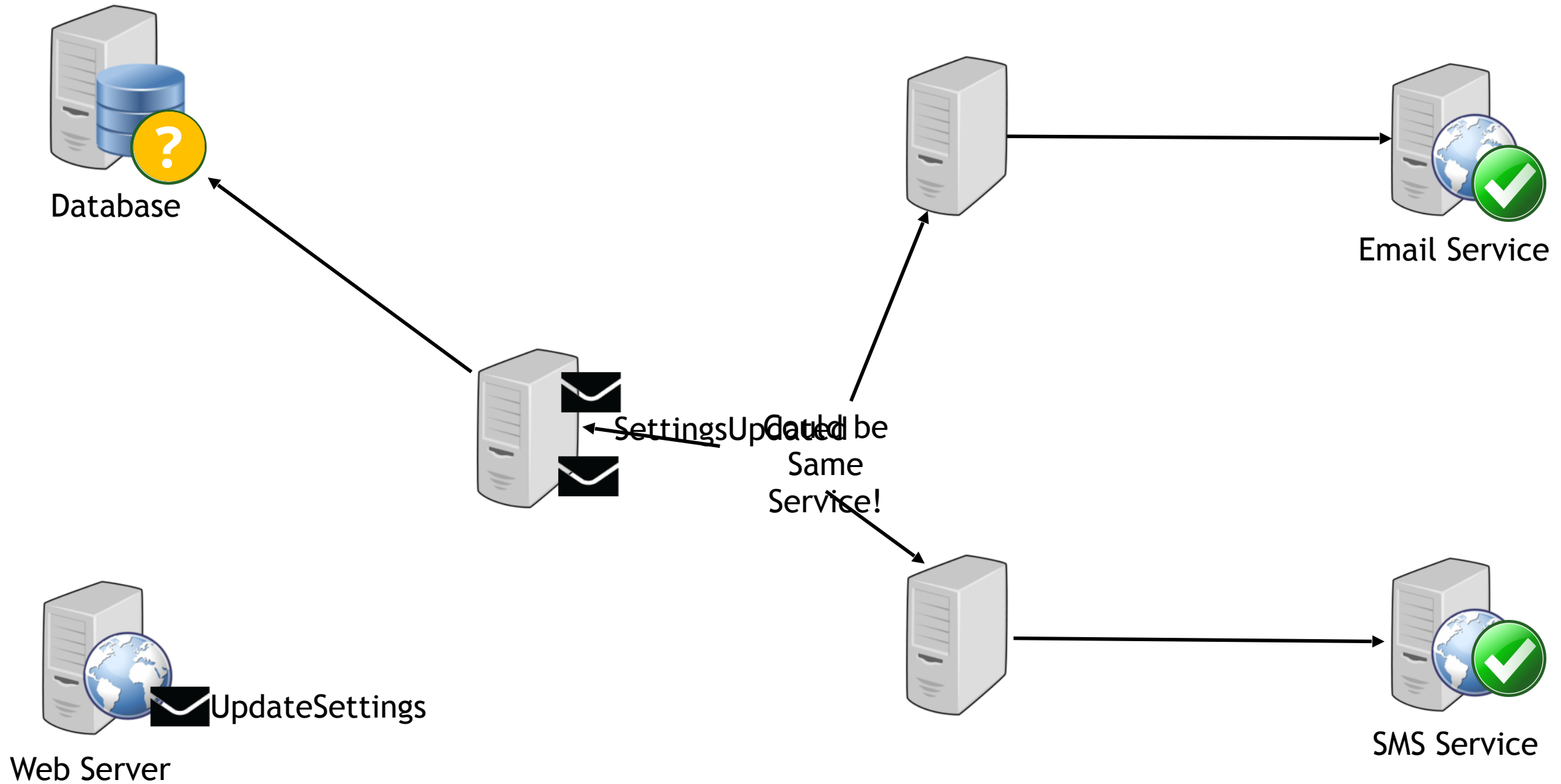




MEXICAN STANDOFF

The key is isolation





What Happened?

- Divided database & web service calls into separate handlers
- Web sends command to update database
- Database updated & event published within ambient transaction
- Web service adapters subscribe to event and interact with web services in isolation

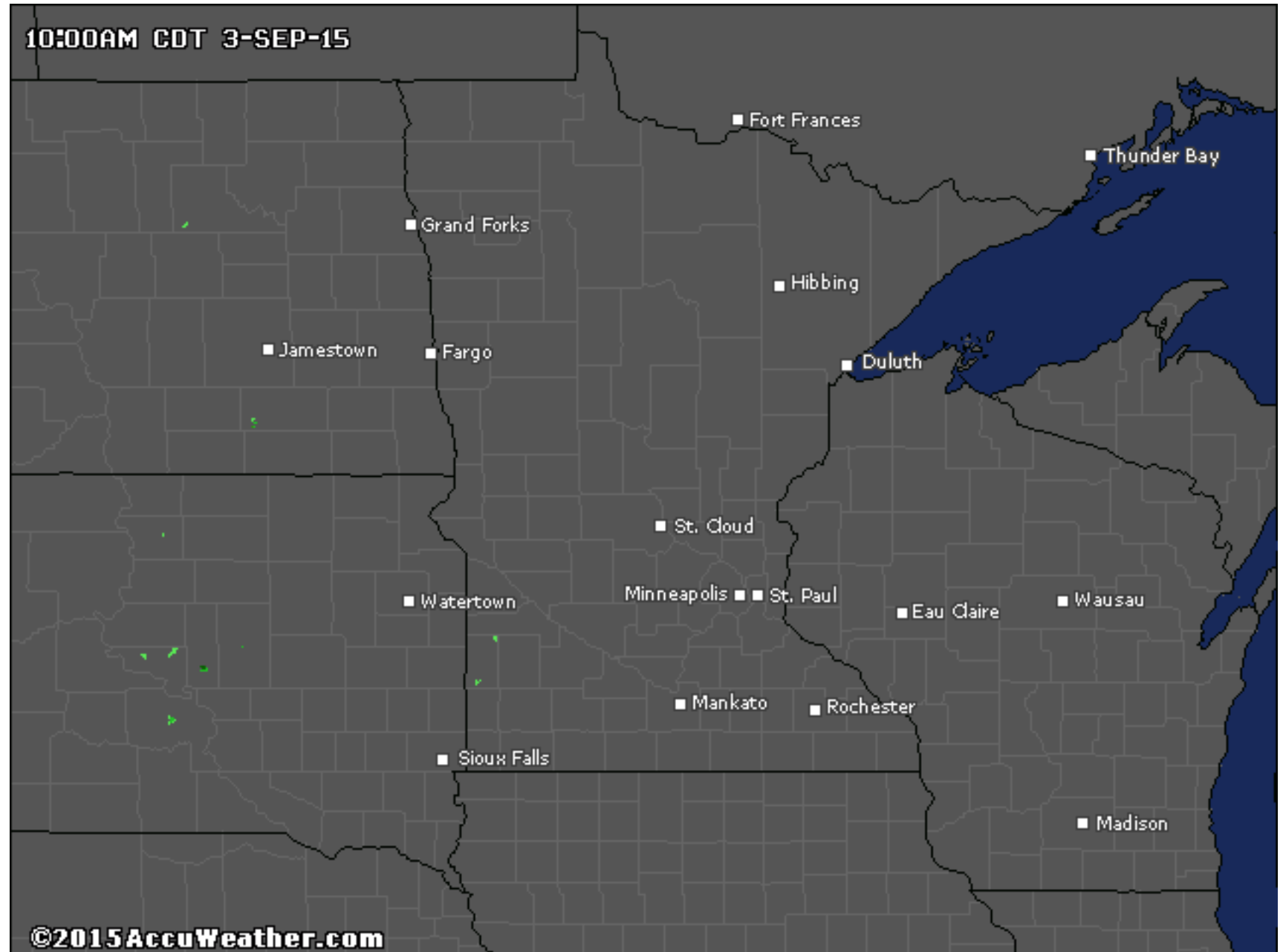


Benefits

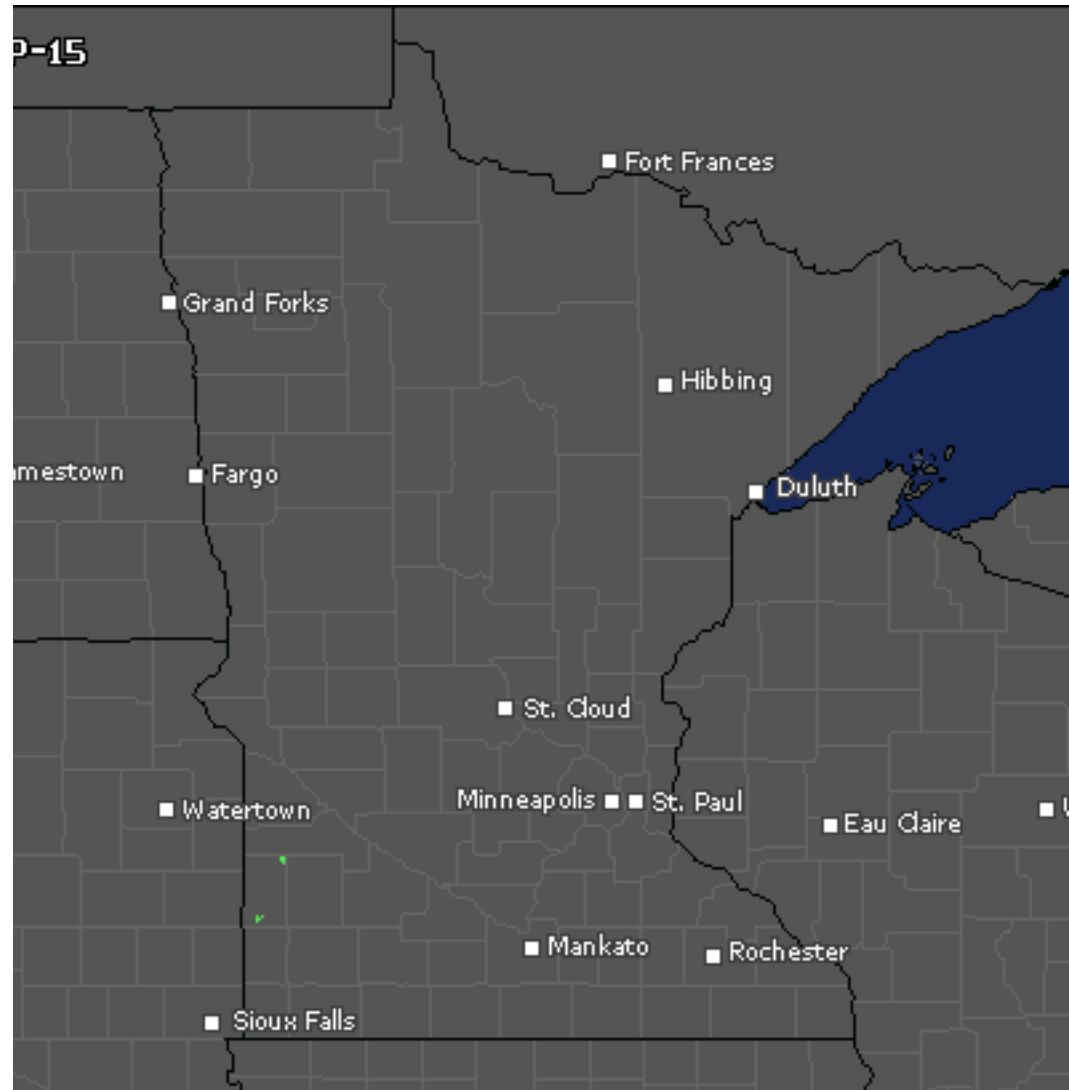
- Isolation of non-transactional web service calls
 - Luckily web service calls were idempotent
- Automatic retries when a web service is unavailable
- Return success to user immediately



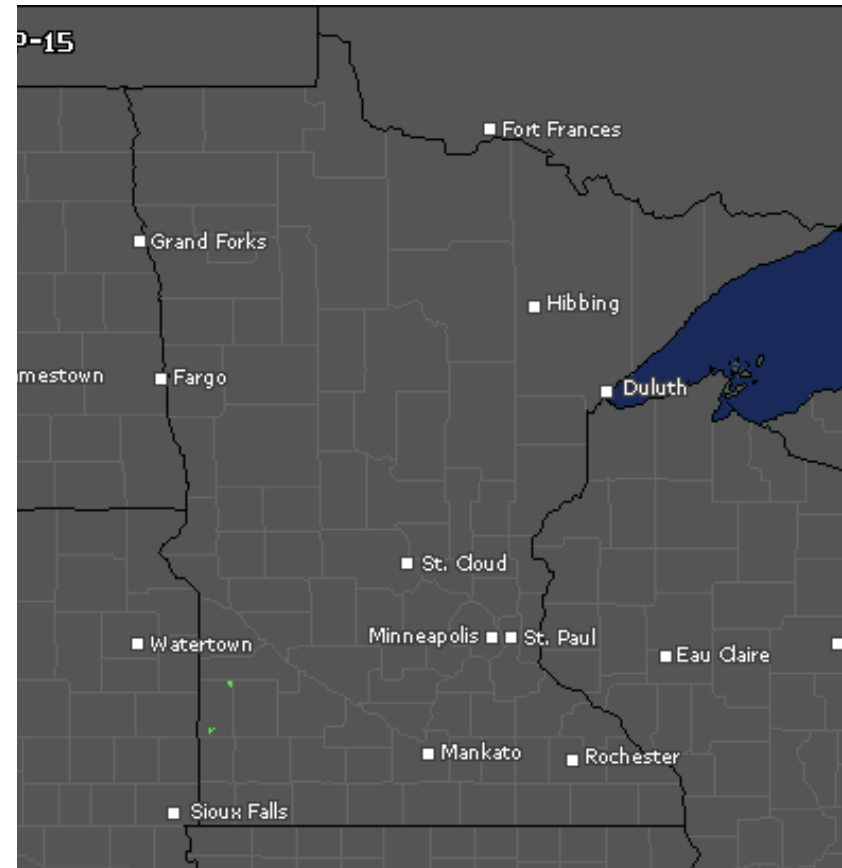
Weather Maps



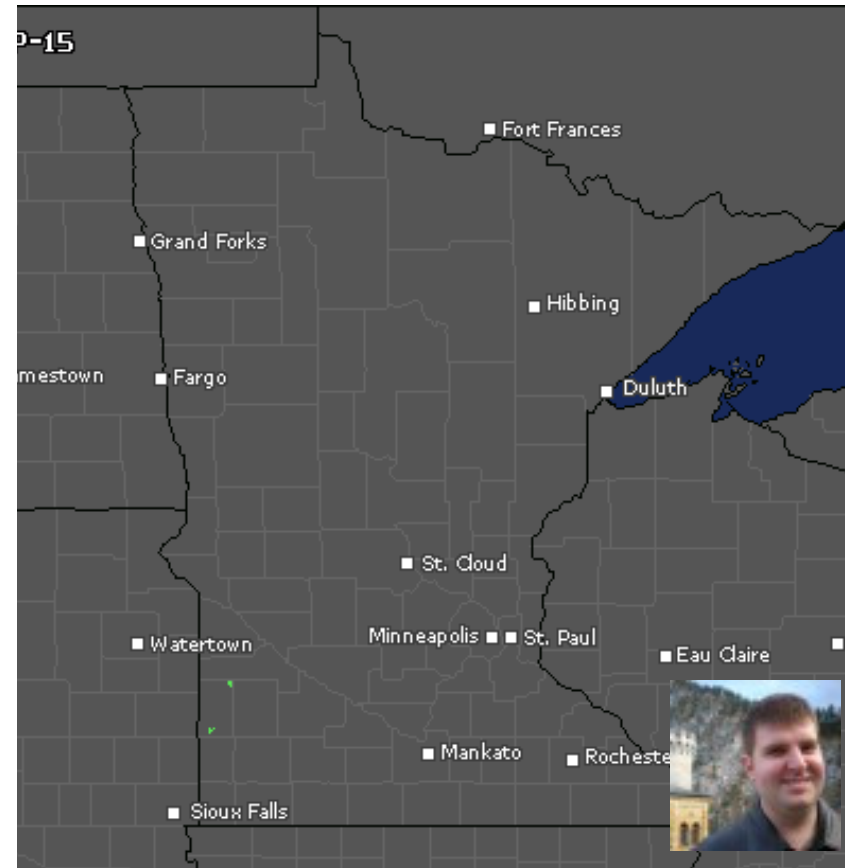
Weather Maps



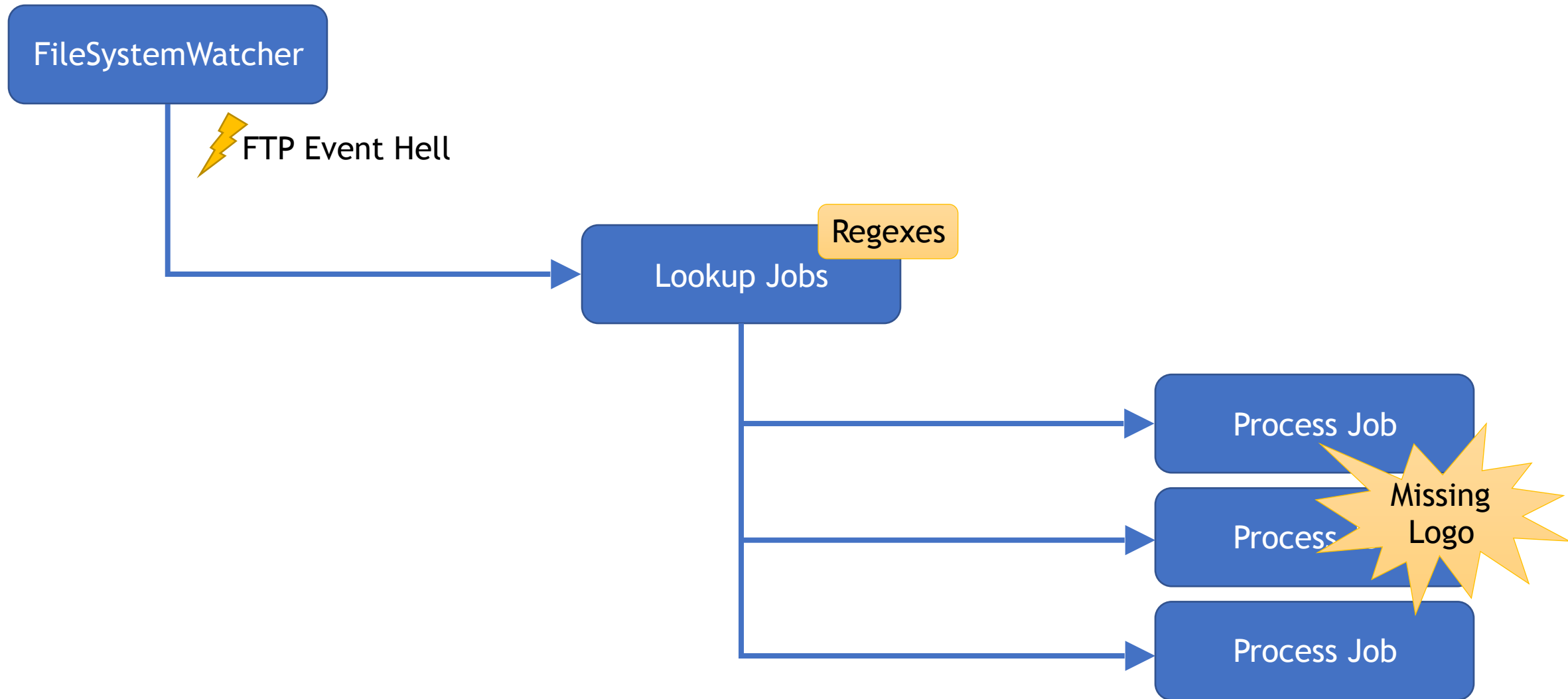
Weather Maps



Weather Maps

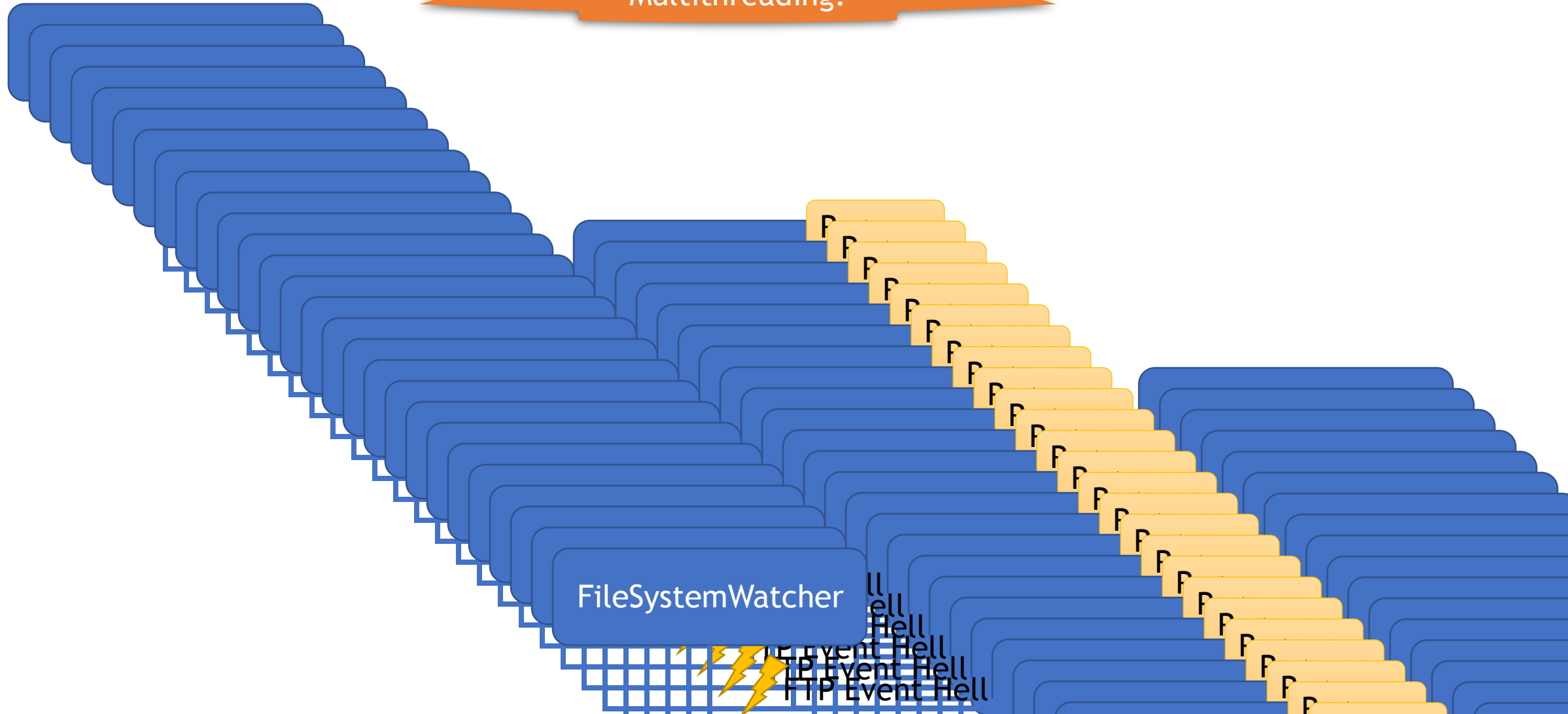


Weather Maps



Weather Maps

Multithreading!



Mutex
Lock/Monitor/
Semaphore
Spinlock

Weather Maps

Multithreading!

.Set()
.WaitOne()
.WaitAll()
.Reset()

FileSystemWatcher

FTP Event Hell

.WaitOne()

.WaitOne(int
n)
SafeWaitHandl

Gates

Lookup Jobs

Regexes

.Dispose()

ManualResetEvent

AutoResetEvent

Process Job

Process

Process Job



A man with a bloody forehead and a screaming expression is shown in a server room. He is wearing a light-colored shirt and has his arms raised. The background is filled with server racks and cables.

MULTITHREADING

THIS IS NOT A SITUATION YOU WANT TO BE IN

App Server

Thread

Thread

Thread

Thread

Thread

Thread

Thread

Thread

Thread

App Server

Thread

Thread

Thread

Thread

Thread

Thread

Thread

Thread

Thread

Thread

Thread

Thread

Thread

Thread

Thread

Thread

Thread

Thread

Thread

Thread

Thread

Thread

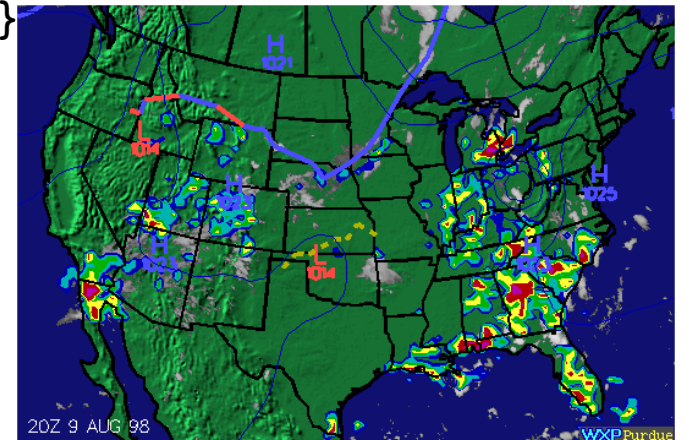
Thread

Thread

Messages

```
public class ImageFileUpdated : IEvent
{
    public DateTime UpdatedUtc { get;
set; }
    public string FilePath { get; set; }
}
```

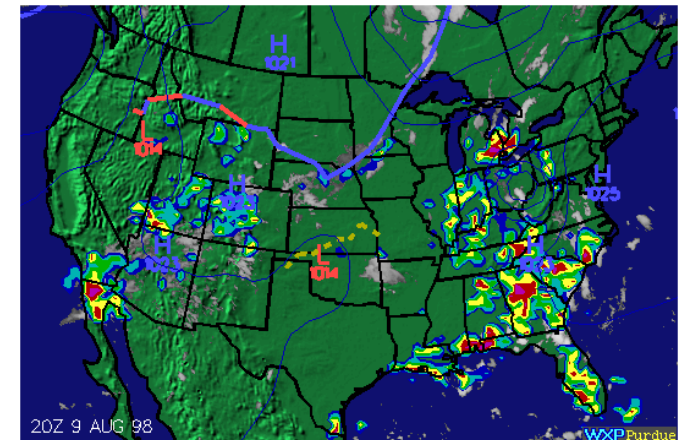
```
public class ProcessImageJob : ICommand
{
    public string FilePath { get; set; }
    public int JobId { get; set; }
}
```



Handlers

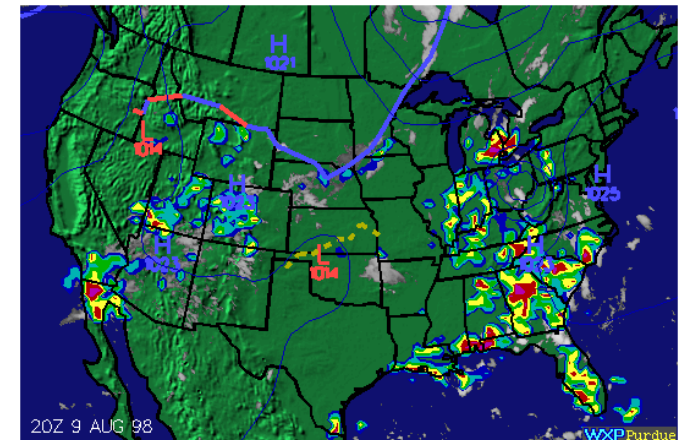
```
public class FindProcessingJobs :
  IHandleMessages<ImageFileUpdated>
{
    public IBus Bus { get; set; }

    public void Handle(ImageFileUpdated message)
    {
        foreach (var job in GetJobsForImage(message.FilePath))
        {
            Bus.Send(new ProcessImageJob
            {
                FilePath = message.FilePath,
                JobId = job.JobId
            });
        }
    }
}
```

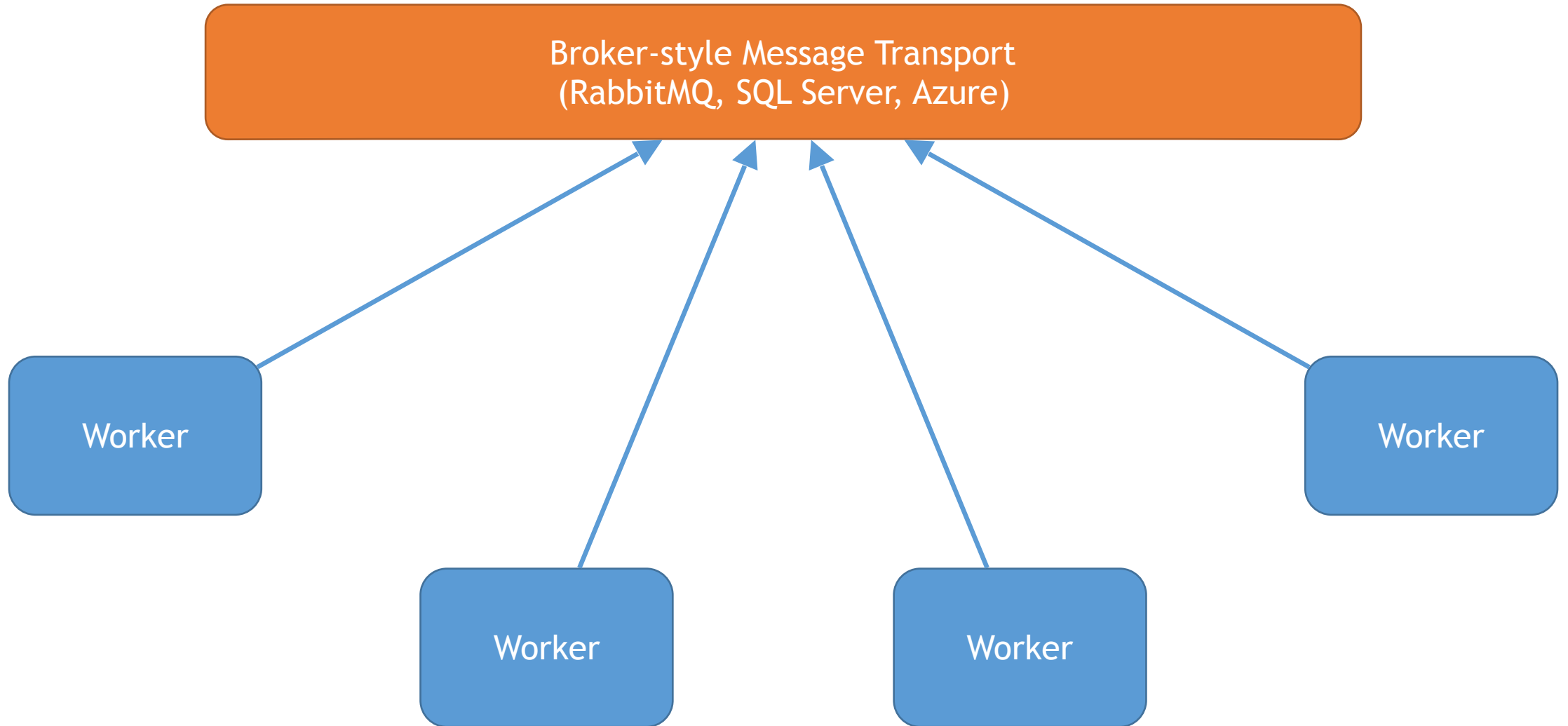


Handlers

```
public class ImageProcessor :  
    IHandleMessages<ProcessImageJob>  
{  
    public void Handle(ProcessImageJob message)  
    {  
        Job job = GetJobDetails(message.JobId);  
        ProcessImage(job, message.FilePath);  
    }  
}
```

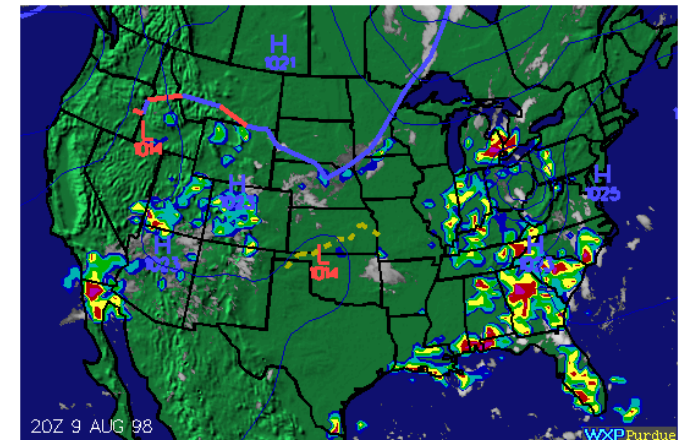


Scaling Out: Competing Consumers



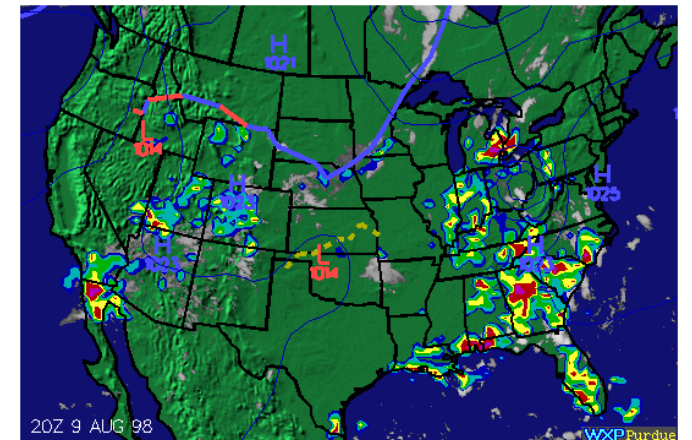
Back to the message...

```
public class ProcessImageJob :  
    ICommand  
{  
    public string FilePath { get; set; }  
}  
    public int JobId { get; set; }  
}
```



What about this?

```
public class ProcessImageJob :  
    ICommand  
{  
    public string FilePath { get; set; }  
}  
    public byte[] FileBytes { get;  
set; }  
    public int JobId { get; set; }  
}
```



Message Size Limitations

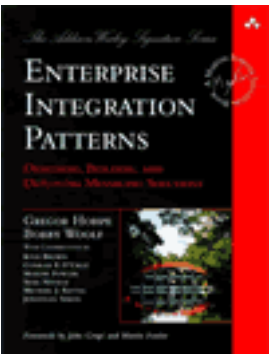
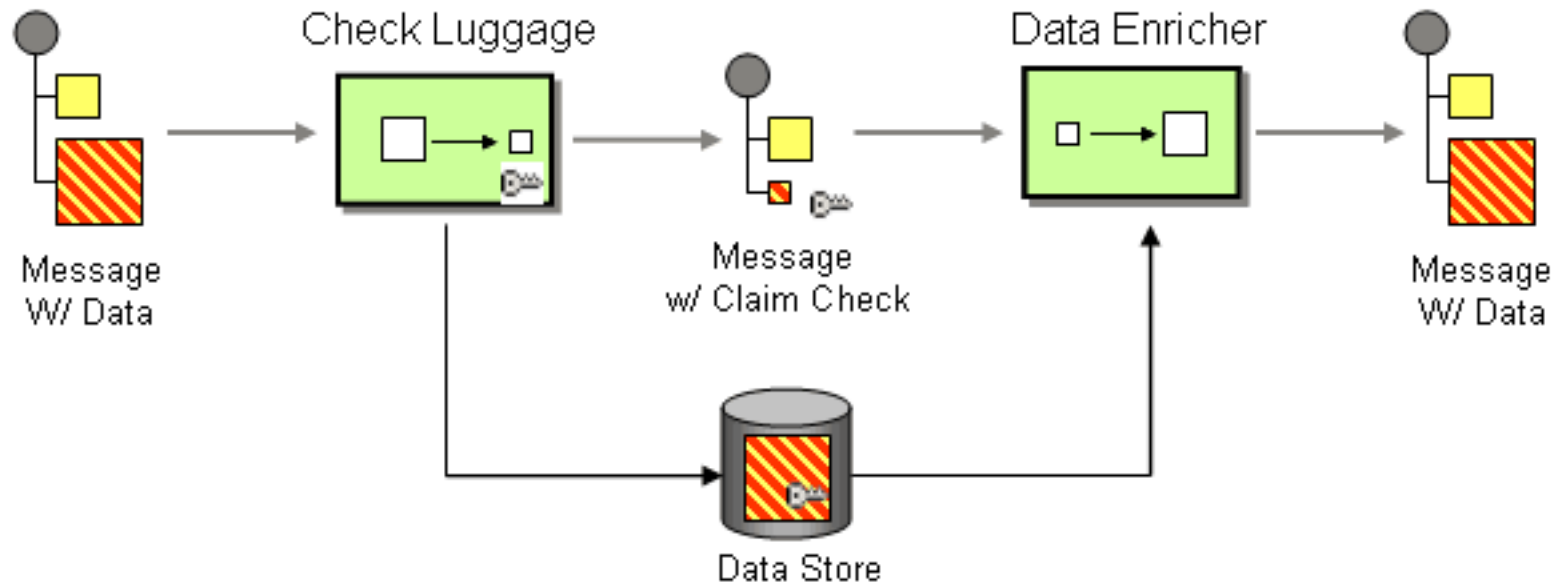
NServiceBus
Supported
Transports

Transport	Message Size Limit
MSMQ	4 MB
RabbitMQ	Unlimited ¹
SQL Server	Unlimited ¹
Azure Storage Queues ²	64 KB
Azure Service Bus ²	256 KB
Amazon SQS/SNS ²	256 KB
IronMQ	256 KB
ActiveMQ	Unlimited ¹
ZeroMQ	Configurable ¹
IBM WebSphere MQ	Configurable ¹ (4 MB Default)

¹ But seriously, don't go there!

² Subject to change from time to time

Claim Check Pattern



Enterprise Integration Patterns
Gregor Hohpe and Bobby Woolf

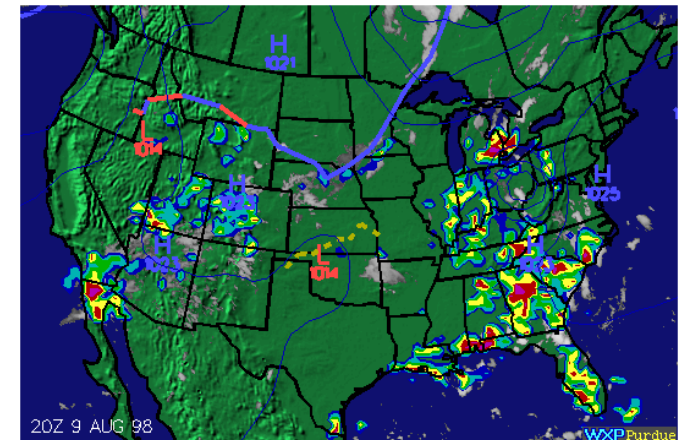
<http://www.enterpriseintegrationpatterns.com/patterns/messaging/StoreInLibrary.html>

Data Bus

```
public class ProcessImageJob : ICommand
{
    public DataBusProperty<byte[]> FileBytes { get;
set; }
    public int JobId { get; set; }
}
```

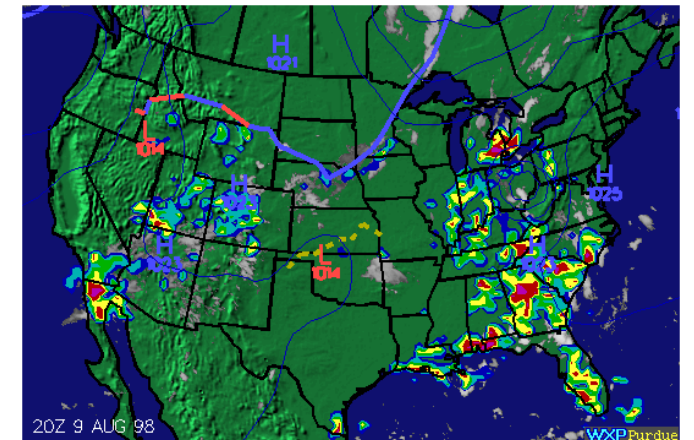
What Happened?

- Move image processing logic to message handler
- From FileSystemWatcher, send 1 message per updated file
- Add workers on additional machines to process increased load on an as-needed basis
- Refer to image data either by file path, or use claim check pattern with Data Bus



Benefits

- Separate FileSystemWatcher and Processing processes
- Isolate each processing job, with retries
- No complex threading code
- Easily increase number of processing threads
- Easily scale out to multiple machines



Content Publishing

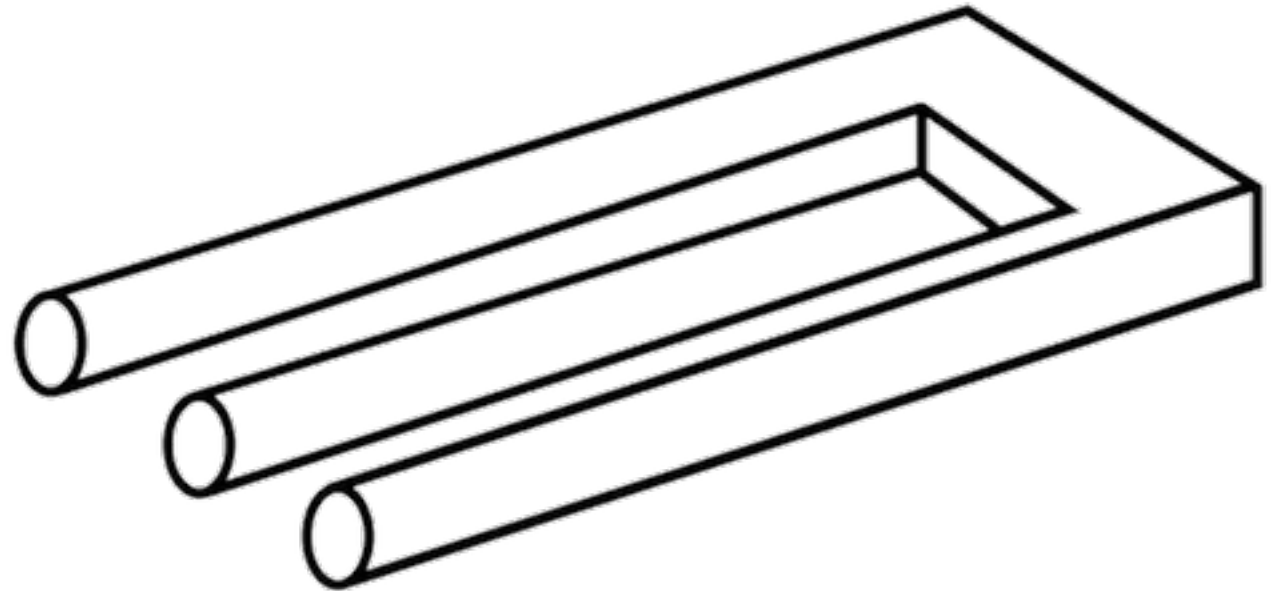




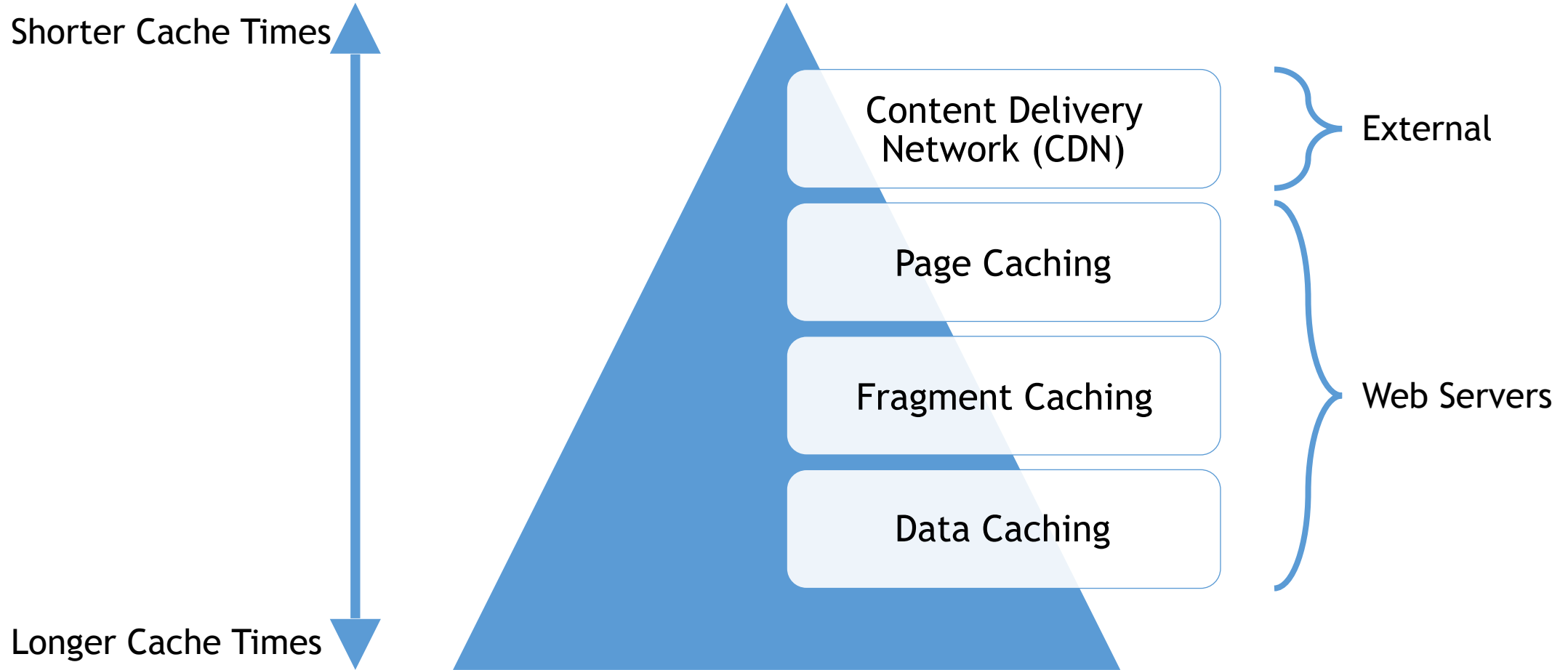
Enter the News Director

2 Hard Problems in Computer Science

1. Cache invalidation
2. Naming things
3. Off by one errors



Caching Levels





One publisher to rule them all

One publisher to rule them all

With NServiceBus.SqlServer Transport

```
-- Create unique message id
DECLARE @MsgId uniqueidentifier = NEWID()

-- Create Message Contents
DECLARE @Msg varchar(255) =
    '{ $type: "Messages.ContentUpdated", ContentId: "' + @ContentdId +
    "' }'

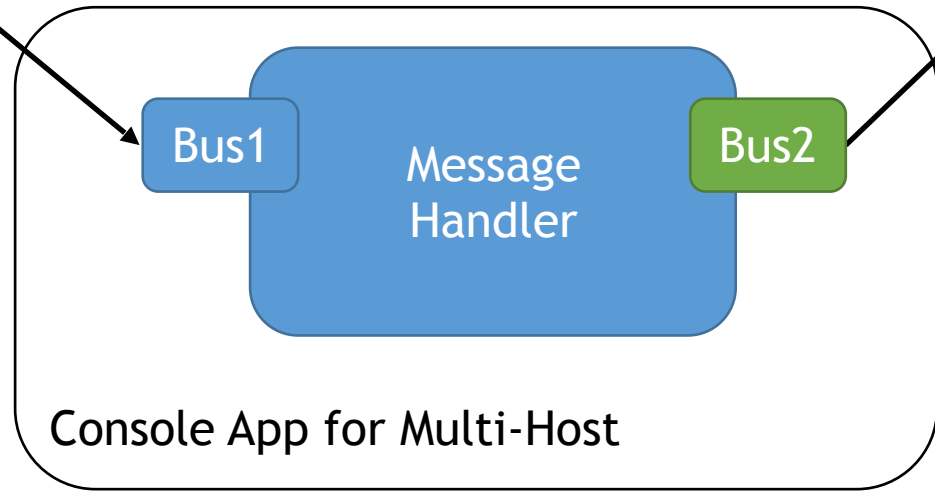
-- Send command straight from the DB
INSERT INTO ContentPublishQueue ([Id],[Recoverable],[Headers],
[Body])
VALUES
(
    @MsgId,
    'true',
    '',
    CONVERT(varbinary(255), @Msg)
```

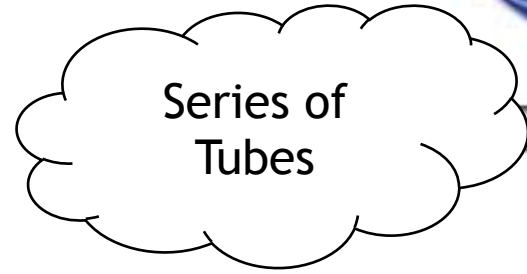


One publisher to rule them all

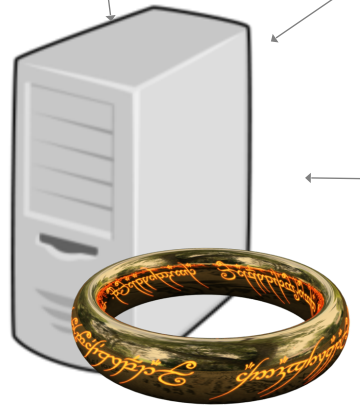
With NServiceBus.SqlServer Transport

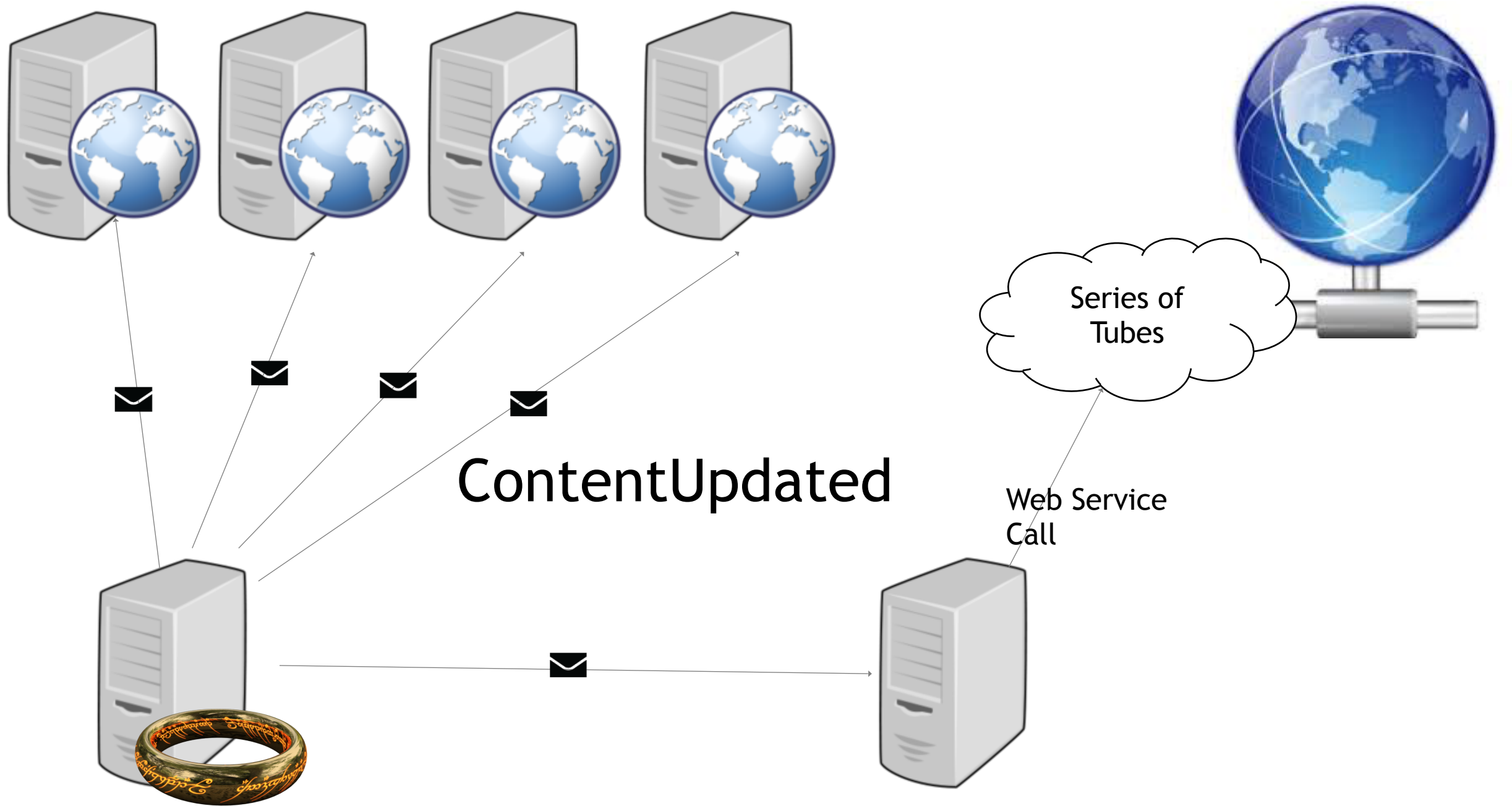
Great way to integrate legacy SQL systems!





Subscribe!





What Happened?

- Publish content within a message handler, not a sproc
- Changed various sproc executions to in-database enqueues
- Publish an event after publishing is complete
 - Subscribe to event in web tier and removed cached items
 - Subscribe to event on app server, and call CDN cache drop API
- Next web request encounters empty caches and regenerates HTML from scratch



Benefits

- Provide “instant gratification” in cached environment
- Manage various caches in a decoupled manner
 - Much easier to switch CDN providers
- Integrate with legacy SQL-bound systems





**WRAP
IT
UP**

Lessons

- Use messaging to maintain response times under high load
- Use messaging to split processes into more manageable bits
 - Each handler should do the bare minimum possible within a transaction
- Use publish/subscribe to decouple complex processes
- Rely on automatic retries to ensure processes are completed
 - This means you can report success to the user before completion!
- Use messaging to scale processing of discrete tasks
 - Avoid ugly threading code!

A sword with a tattered banner on a grassy hill under a cloudy sky. The sword is positioned vertically on the right side of the frame, with a dark hilt and a silver blade. A piece of dark, tattered fabric is attached to the crossguard and is blowing in the wind. The background consists of a green grassy hill in the foreground and a blue, cloudy sky in the background.

Thank You

David Boike
Particular Software

@DavidBoike
make-awesome.com

Free Stuff

2 full days of Udi Dahan's Advanced Distributed System Design Course!

<http://go.particular.net/MDC>

Access Code: MDC15 Expires Oct 21

David Boike
Particular Software

@DavidBoike
make-awesome.com