# Building change-friendly large scale systems

# Where's the problem?

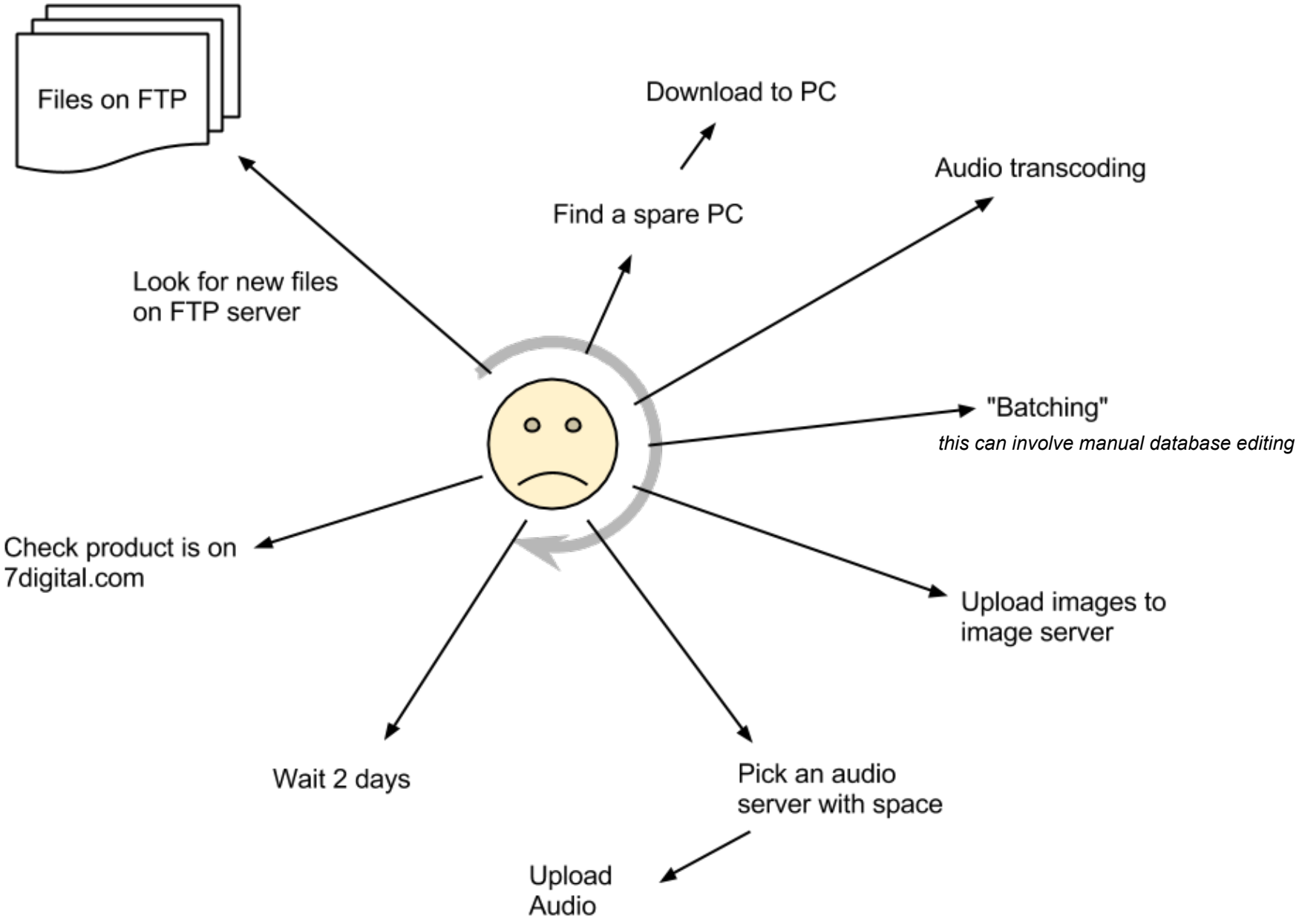
More tests ≅ Less change

Less *unexpected* change ≅ Fewer bugs

+ Better sharing of understanding
- More that's wrong when behaviour changes

We're always learning and things change all the time

# The Status-Quo



# The Challenge

121'466 lines of VB,
535 database tables,
1'296 stored procs, 395 views
25 million tracks, billions of rights entries.

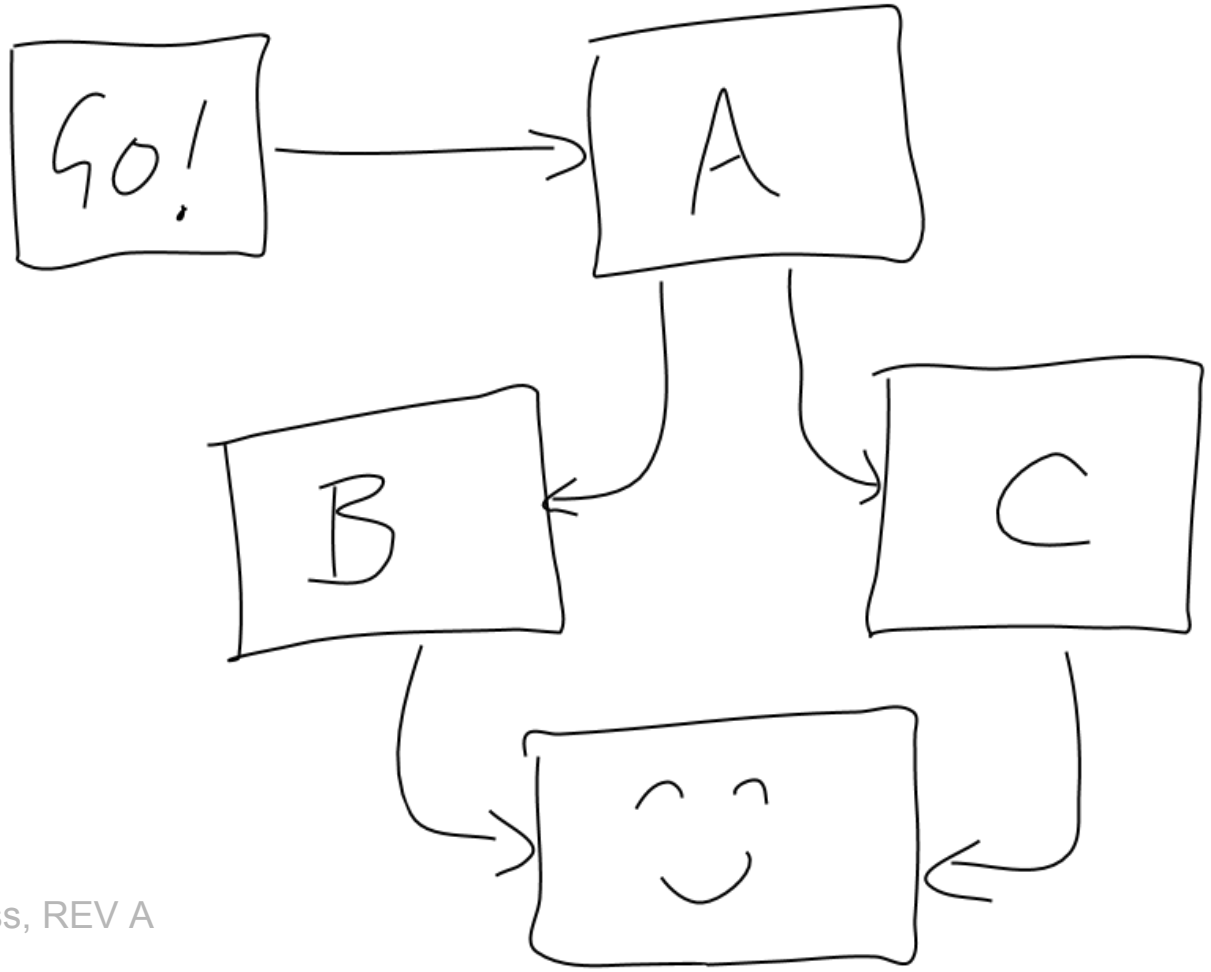
0 employees with end-to-end knowledge.

*and a rapidly growing company*

# How we think

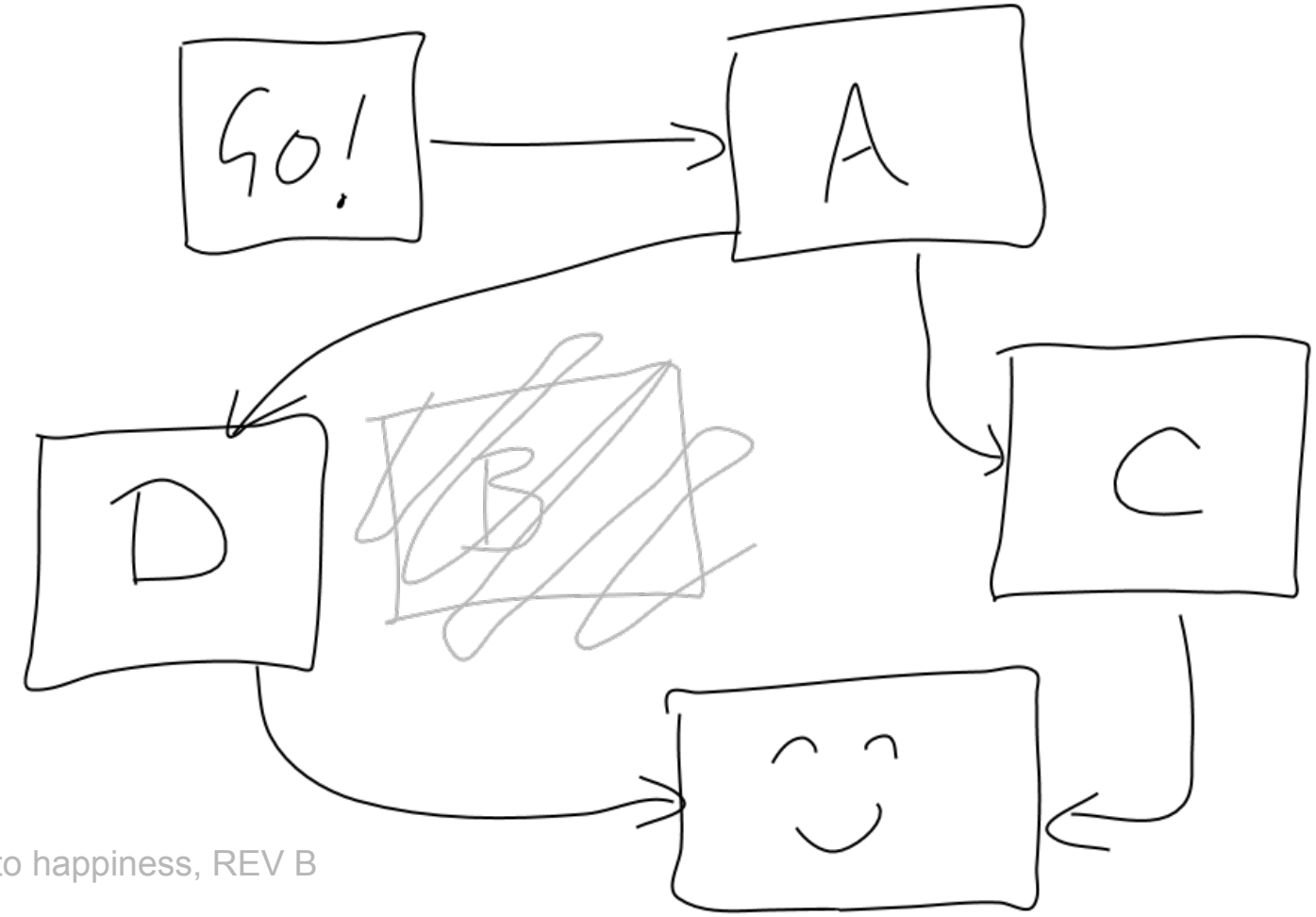
designing for low mental friction

# What it looks like when we rough out a design...




Path to happiness, REV A

# ...and when we change our minds




Path to happiness, REV B

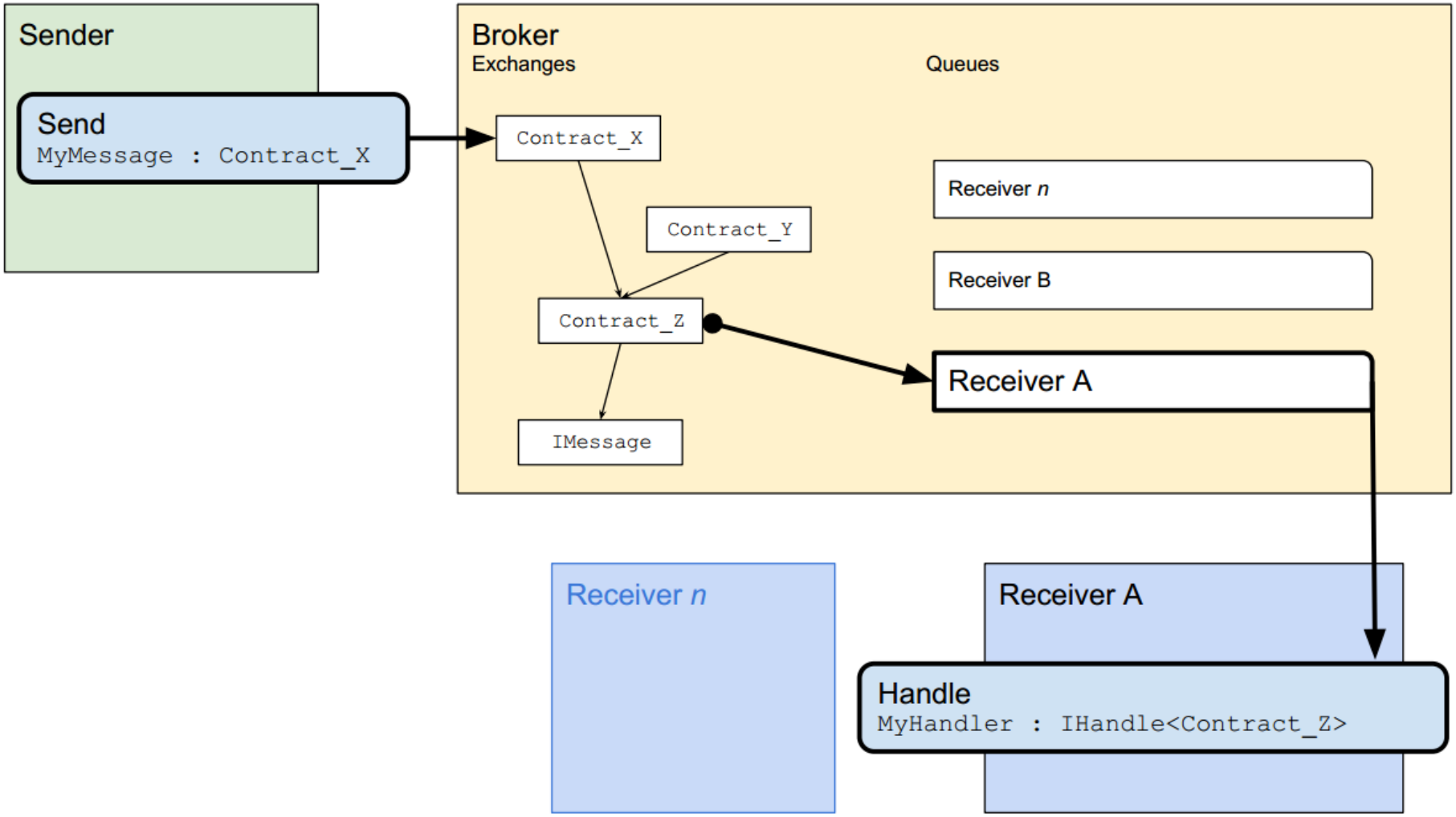
# things we want

- Small components
  - can be changed or discarded and rewritten
  - have a single, well defined purpose
- Adaptable 'plumbing'
- Meaningful tests that don't get in the way

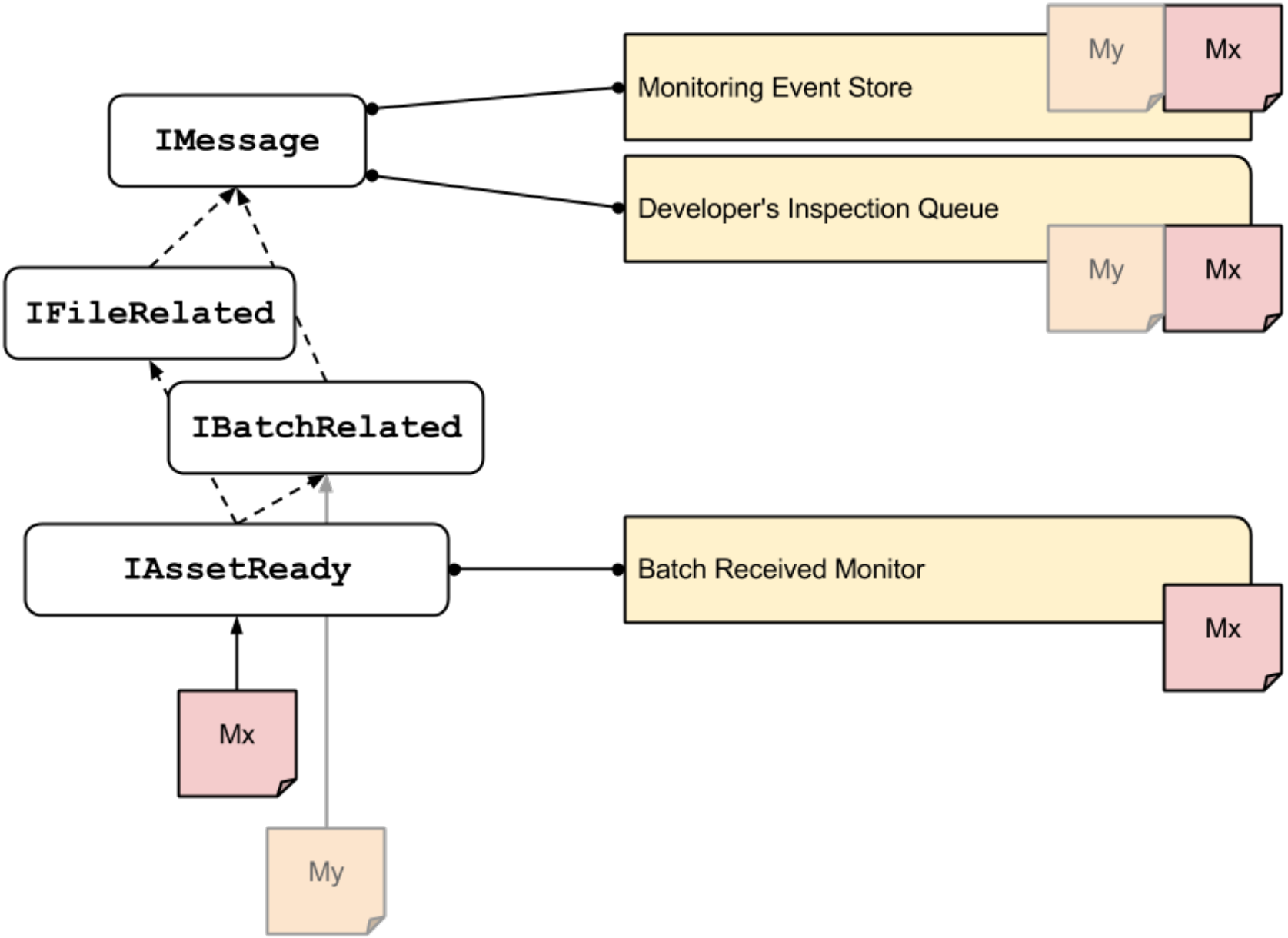
# Making it happen

tools and techniques

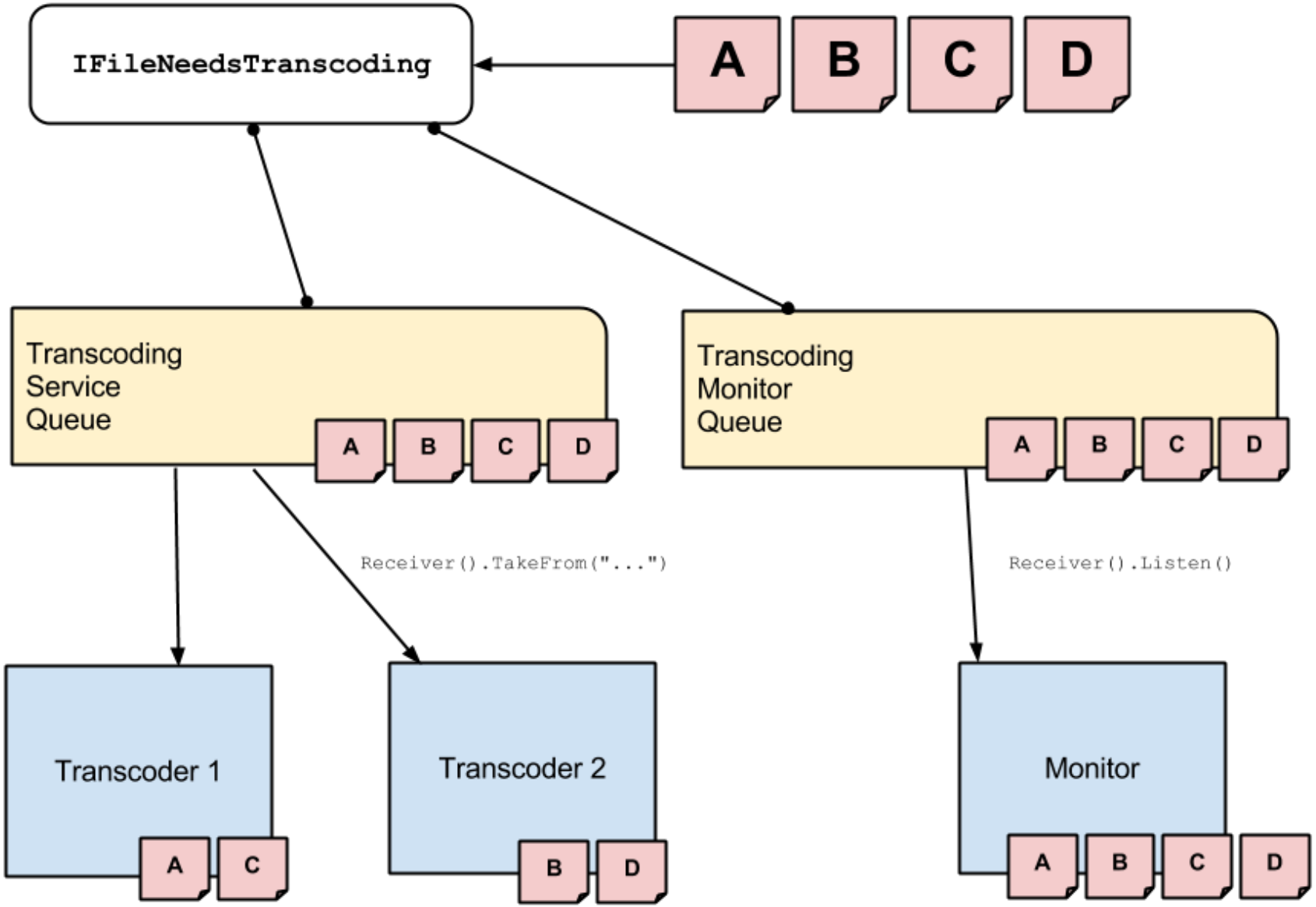
# SevenDigital.Messaging at a glance



# exchange and binding



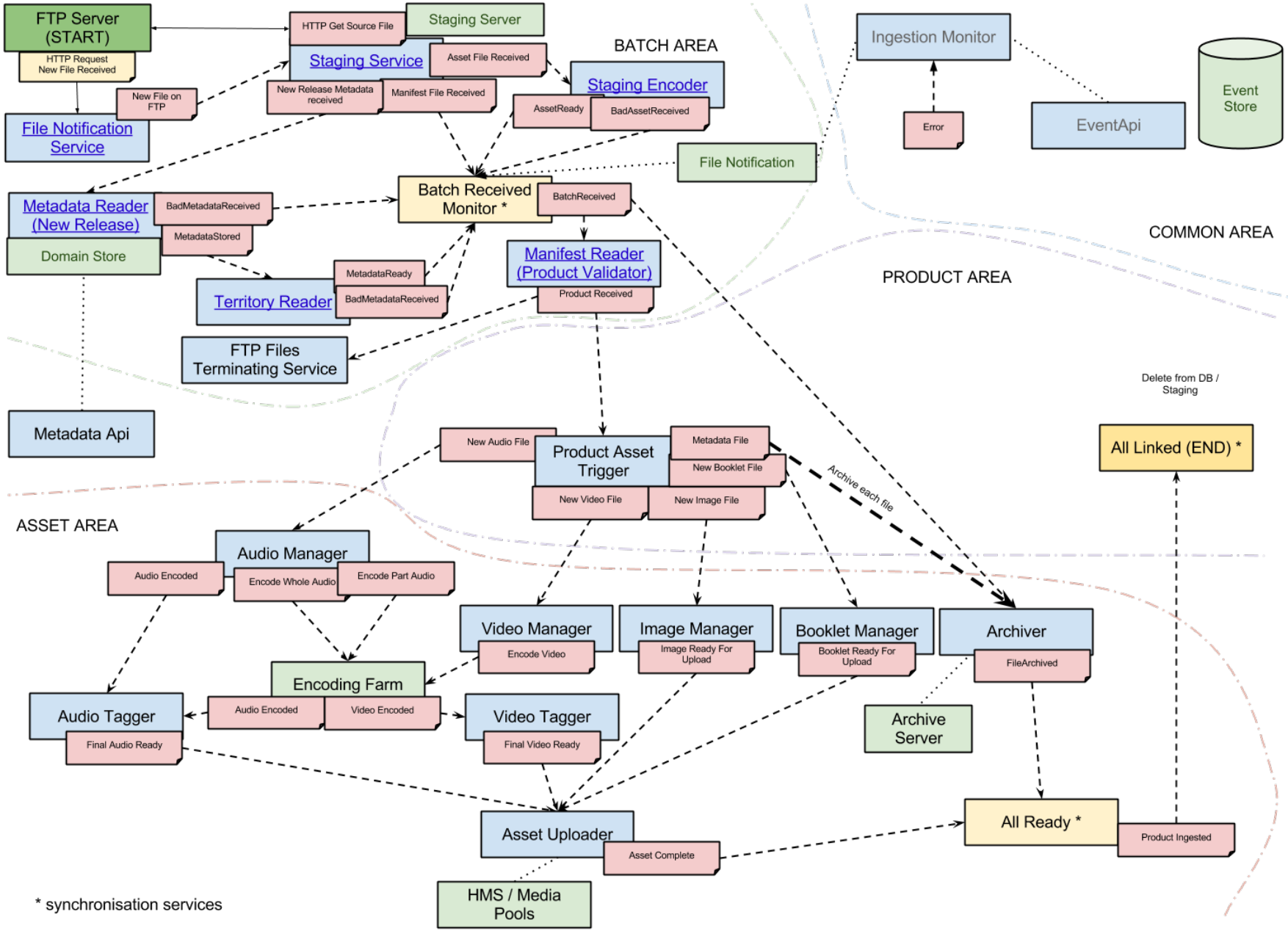
# consuming



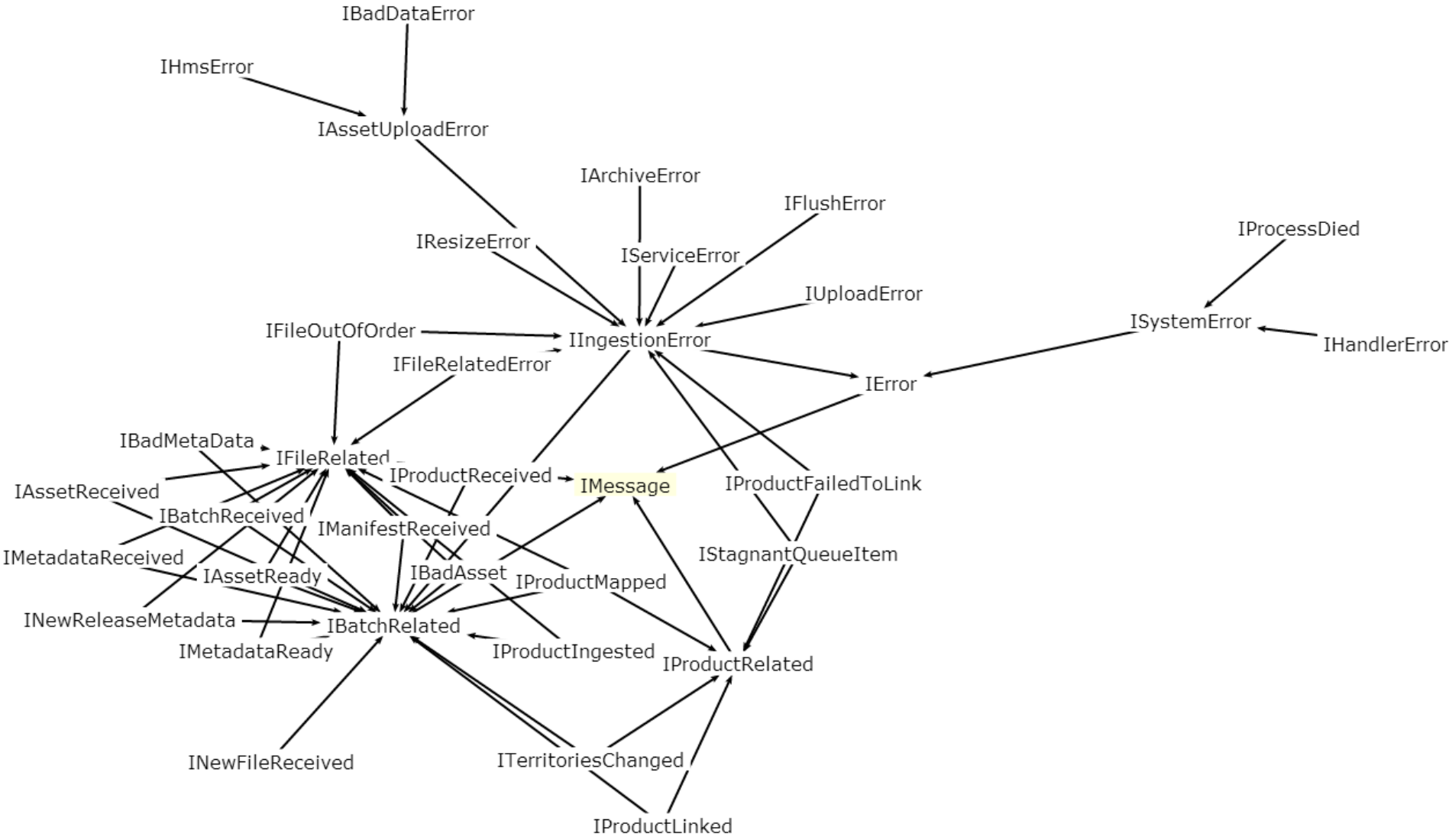

**Load Balancing across nodes**

**Listen to all**

# architecture in reality



# routing in reality



# Deployment and interaction

- Each component is a separate executable running on a range of different machines
  - Multithreading for free
- All interaction between services is over HTTP or AMQP (RabbitMq)
- Routing is all automatically generated from code

The edges of the system are wire protocols, not code

# Making it happen in software

Nothing to break the purity of separation:

- Everything goes "over-the-wire"
- No "Sagas"
- No transactions
- No rollbacks

Generally, we don't allow anything that makes one component dependent on any other.

# Test Driving

outside in for distributed software

# Test-driving outside in

Things that are test friendly:

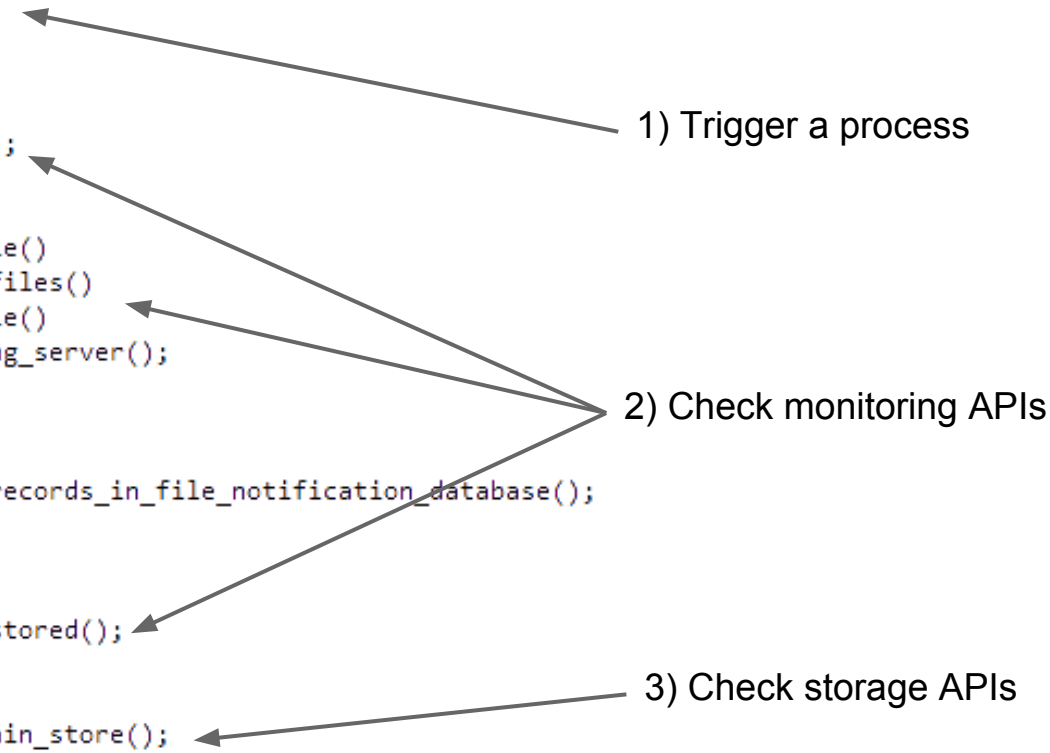
- Well defined edges
- Exposing system flow
- Avoiding logging-as-a-separate-thing
- Tiered scopes of behaviour

# Test driving

1. End-to-end (test the arrows)
2. Message-to-message acceptance test
3. Integration testing (minimise)
4. Unit testing components

# End-to-end

```
If_everything_is_correct_should_process_completely()
{
    Given.there_is_a_batch_with_product_having_mp3_and_flac_files();

    When
        .batch_is_uploaded_to_ftp_server();

    Then
        .File_notification
            .sends_messages_for_all_files();
    Then
        .File_staging_service
            .sends_message_for_metadata_file()
            .sends_messages_for_all_asset_files()
            .sends_message_for_manifest_file()
            .all_files_are_stored_on_staging_server();

    Then
        .File_notification
            .all_files_have_corresponding_records_in_file_notification_database();

    Then
        .Metadata_reader
            .sends_a_message_for_metadata_stored();

    Then
        .the_metadata_is_stored_in_the_domain_store();

    Then
        .Territory_reader
            .sends_a_message_for_metadata_ready();

    Then
        .the_metadata_is_stored_in_the_territory_pricing_database();

    Then
        .Staging_encoder
            .sends_a_message_for_each_asset()
            .stores_all_delivery_formats_on_staging_server_for_each_lossless_asset();

    Then
```



# Acceptance

```
[Test]
public void A_new_release_metadata_XML_file_is_staged_correctly()
{
    Write.Scenario();

    Given.a_new_release_metadata_XML_file_has_been_received();
    When.the_staging_service_is_notified_about_the_new_file_on_the_FTP_server();
    Then.the_file_is_downloaded_from_the_FTP_server();
    Then.the_file_is_uploaded_to_the_staging_server();
    Then.the_file_details_are_recorded_in_the_database();
    Then.the_new_release_metadata_received_message_is_sent_by_staging_service();
}
```

```
public void the_new_release_metadata_received_message_is_sent_by_staging_service()
{
    Write.Step();

    Assert.That(SentMessages<INewReleaseMetadataReceived>().Count(), Is.EqualTo(1));

    Assert.That(SentMessages<IManifestReceived>().Count(), Is.EqualTo(0));
    Assert.That(SentMessages<IAssetReceived>().Count(), Is.EqualTo(0));
}
```

# Integration tests from a service

```
using NUnit.Framework;
using SevenDigital.Messaging;
using StructureMap;

namespace SevenDigital.Jester.FileStaging.Service.Integration.Tests
{
    [TestFixture]
    public class DependencyInjectionTests
    {
        [SetUp]
        public void SetUp()
        {
            MessagingSystem.Configure.WithLoopbackMode();
            ConfigureFileStaging.WithDefaults();
        }

        [TearDown]
        public void teardown()
        {
            MessagingSystem.Control.Shutdown();
        }

        [Test]
        public void Should_be_able_to_resolve_handler()
        {
            var instance = ObjectFactory.GetInstance<NewFileReceivedHandler>();

            Assert.That(instance, Is.Not.Null);
        }
    }
}
```

... and that's it

# IoC in little chunks

- Avoiding god objects with message passing
- Handlers are wired up by messaging
- All parts are replaceable
- Lots of testable edges

# Examples of handling big change

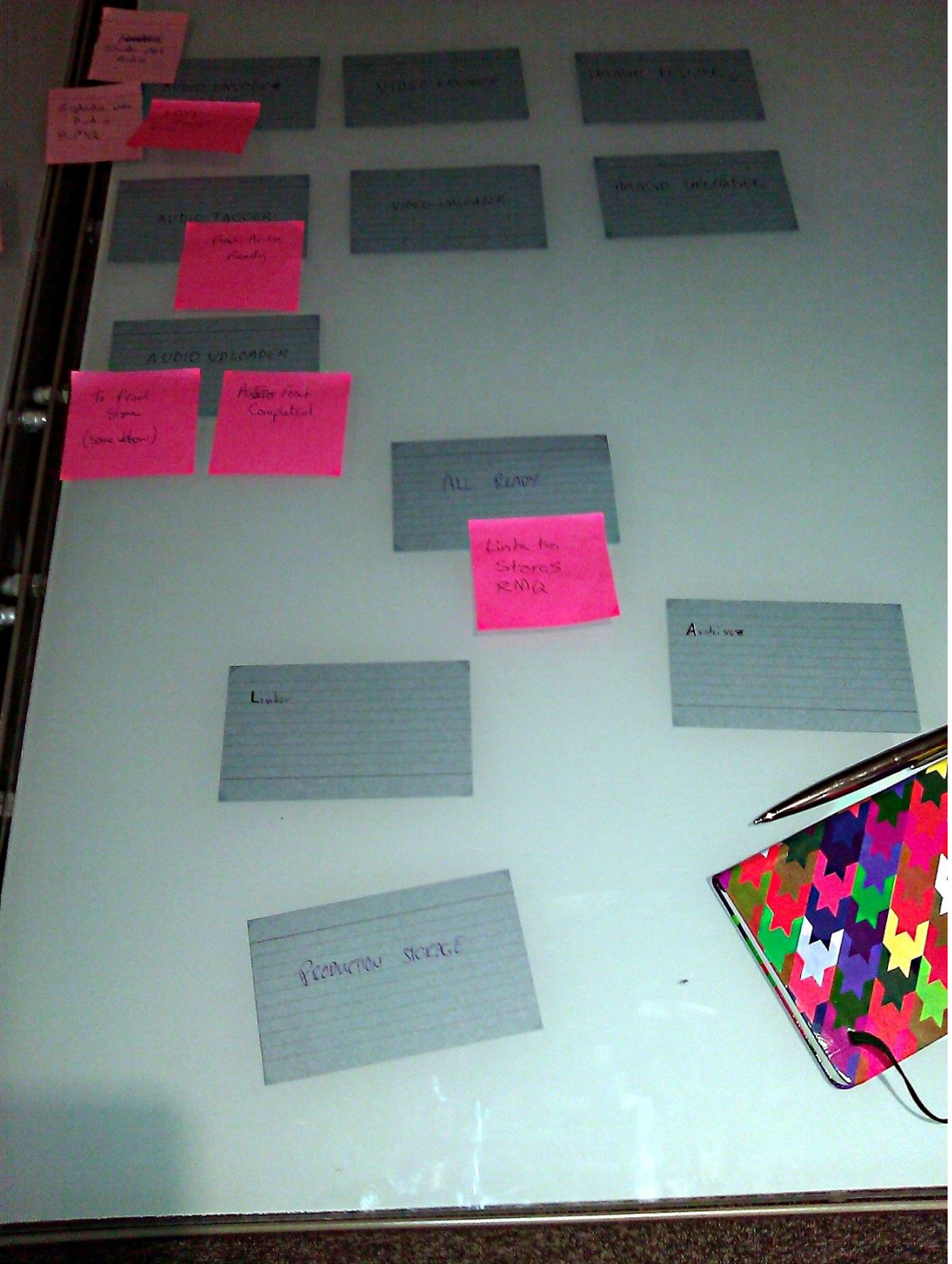
- Adding staging encoder
- routing to linker with manual and auto flows
- integrating with ftp over RMQ

# Challenges

- Serialisation in .Net is a *PITA*
- Finding the sweet spot for data persistence
- Fan-in and synchronisation

# Big wins

- Easy for stakeholders to understand the real layout of the code, and influence software design in a useful way
- Distribution, load balancing and multi-platform support for free
- *SoS* allows day to day focus to be on small components, without worrying about integration




# Why the colour scheme?

# Resources

https://nuget.org/packages/SevenDigital.Messaging/

https://github.com/i-e-b/SevenDigital.Messaging